CLOSER 2014

4th INTERNATIONAL CONFERENCE ON CLOUD COMPUTING AND SERVICES SCIENCE

PROCEEDINGS

3 - 5 APRIL, 2014

BARCELONA, SPAIN

WWW.CLOSER.SCITEVENTS.ORG

SPONSORED BY:

INSTICC

IN COOPERATION WITH:

acm In-Cooperation

sigmis

CLOSER 2014

Proceedings of the
4th International Conference on
Cloud Computing and Services Science

Barcelona, Spain

3 - 5 April, 2014

Sponsored by
INSTICC – Institute for Systems and Technologies of Information, Control and Communication

In Cooperation with
ACM SIGMIS – ACM Special Interest Group on Management Information Systems

Copyright © 2014 SCITEPRESS – Science and Technology Publications
All rights reserved

Edited by Markus Helfert, Frédéric Desprez, Donald Ferguson,
Frank Leymann and Víctor Méndez Muñoz

Printed in Portugal
ISBN: 978-989-758-019-2
Depósito Legal: 372331/14

http://closer.scitevents.org
closer.secretariat@scitevents.org

BRIEF CONTENTS

INVITED SPEAKERS IV

SPECIAL SESSIONS CHAIRS IV

ORGANIZING AND STEERING COMMITTEES V

PROGRAM COMMITTEE VI

AUXILIARY REVIEWERS XI

SPECIAL SESSIONS PROGRAM COMMITTEE XII

SELECTED PAPERS BOOK XII

FOREWORD XIII

CONTENTS XV

INVITED SPEAKERS

Helge Meinhard

CERN, IT Department

Switzerland

Paulo Gonçalves

INRIA-DANTE

France

Ignacio Martín Llorente

OpenNebula Director and CEO & Full Professor at UCM

Spain

SPECIAL SESSIONS CHAIRS

SPECIAL SESSION ON FEDERATED CLOUD GOVERNANCE

Víctor Méndez Muñoz, Universitat Autònoma de Barcelona, UAB, Spain

SPECIAL SESSION ON MULTI-CLOUDS

Dana Petcu, West University of Timisoara, Romania

ORGANIZING AND STEERING COMMITTEES

CONFERENCE CHAIR

Markus Helfert, Dublin City University, Ireland

PROGRAM CO-CHAIRS

Frédéric Desprez, LIP / INRIA, France

Donald Ferguson, Dell, U.S.A.

Frank Leymann, University of Stuttgart, Germany

Víctor Méndez Muñoz, Universitat Autònoma de Barcelona, UAB, Spain

PROCEEDINGS PRODUCTION

Marina Carvalho, INSTICC, Portugal

Helder Coelhas, INSTICC, Portugal

Ana Guerreiro, INSTICC, Portugal

Filipe Mariano, INSTICC, Portugal

Andreia Moita, INSTICC, Portugal

Raquel Pedrosa, INSTICC, Portugal

Vitor Pedrosa, INSTICC, Portugal

Cátia Pires, INSTICC, Portugal

Ana Ramalho, INSTICC, Portugal

Susana Ribeiro, INSTICC, Portugal

Rui Rodrigues, INSTICC, Portugal

André Santos, INSTICC, Portugal

Fábio Santos, INSTICC, Portugal

José Varela, INSTICC, Portugal

CD-ROM PRODUCTION

Pedro Varela, INSTICC, Portugal

GRAPHICS PRODUCTION AND WEBDESIGNER

André Lista, INSTICC, Portugal

Mara Silva, INSTICC, Portugal

SECRETARIAT

Carla Mota, INSTICC, Portugal

WEBMASTER

Susana Ribeiro, INSTICC, Portugal

PROGRAM COMMITTEE

Antonia Albani, University of St. Gallen, Switzerland

Vasilios Andrikopoulos, University of Stuttgart, Germany

Cosimo Anglano, Universita' del Piemonte Orientale "A. Avogadro", Italy

Ashiq Anjum, University of Derby, U.K.

Claudio Ardagna, Universita' degli Studi di Milano, Italy

Danilo Ardagna, Politecnico di Milano, Italy

Alvaro Arenas, Instituto de Empresa Business School, Spain

José Enrique Armendáriz-Iñigo, Universidad Pública de Navarra, Spain

Matthew Arrott, University of California San Diego, U.S.A.

Benjamin Aziz, University of Portsmouth, U.K.

Zeina Azmeh, I3S Laboratory, University of Nice - Sophia Antipolis, France

Amelia Badica, Faculty of Economics and Business Administration, University of Craiova, Romania

Henri E. Bal, Vrije Universiteit, The Netherlands

Janaka Balasooriya, Arizona State University, U.S.A.

Costas Bekas, IBM Zurich Research Lab, Switzerland

Karin Bernsmed, SINTEF ICT, Norway

Nik Bessis, University of Derby, U.K.

Luiz F. Bittencourt, IC/UNICAMP, Brazil

Stefano Bocconi, TU Delft, The Netherlands

Sergey Boldyrev, HERE Maps, Finland

Anne Boyer, Loria - Inria Lorraine, France

Francisco Brasileiro, Universidade Federal de Campina Grande, Brazil

Iris Braun, Dresden Technical University, Germany

Andrey Brito, Universidade Federal de Campina Grande, Brazil

Ralf Bruns, Hannover University of Applied Sciences and Arts, Germany

Anna Brunstrom, Karlstad University, Sweden

Rebecca Bulander, Pforzheim University of Applied Science, Germany

Tomas Bures, Charles University in Prague, Czech Republic

Massimo Cafaro, University of Salento, Italy

Manuel Isidoro Capel-Tuñón, University of Granada, Spain

Miriam Capretz, University of Western Ontario, Canada

Eddy Caron, École Normale Supérieure de Lyon, France

Noel Carroll, University of Limerick, Ireland

John Cartlidge, University of Bristol, U.K.

Humberto Castejon, Telenor, Norway

Rong N. Chang, IBM T. J. Watson Research Center, U.S.A.

Davide Cherubini, Alcatel-Lucent Ireland, Ireland

Augusto Ciuffoletti, Università di Pisa, Italy

Daniela Barreiro Claro, Universidade Federal da Bahia (UFBA), Brazil

Christine Collet, Grenoble Institute of Technology, France

Thierry Coupaye, Orange, France

António Miguel Rosado da Cruz, Instituto Politécnico de Viana do Castelo, Portugal

Tommaso Cucinotta, Alcatel-Lucent, Ireland

Eduardo Huedo Cuesta, Universidad Complutense de Madrid, Spain

Edward Curry, National University of Ireland, Galway, Ireland

Eliezer Dekel, IBM Research Haifa, Israel

PROGRAM COMMITTEE (CONT.)

Yuri Demchenko, University of Amsterdam, The Netherlands

Frédéric Desprez, LIP / INRIA, France

Khalil Drira, LAAS-CNRS, France

Michel Dubois, University of Southern California, U.S.A.

Ake Edlund, Royal Institute of Technology, Sweden

Erik Elmroth, Umeå university, Sweden

Mohamed Eltoweissy, Virginia Polytechnic Institute and State University, U.S.A.

Robert van Engelen, Florida State University, U.S.A.

Donald Ferguson, Dell, U.S.A.

Stefano Ferretti, University of Bologna, Italy

Mike Fisher, BT, U.K.

Ian T. Foster, Argonne National Laboratory and University of Chicago, U.S.A.

Geoffrey Charles Fox, Indiana University, U.S.A.

Chiara Francalanci, Politecnico di Milano, Italy

Maria Ganzha, SRI PAS and University of Gdansk, Poland

David Genest, Université d'Angers, France

Fekade Getahun, Addis Ababa University, Ethiopia

Chirine Ghedira, IAE - University Jean Moulin Lyon 3, France

Lee Gillam, University of Surrey, U.K.

Katja Gilly, Miguel Hernandez University, Spain

Andrzej Goscinski, Deakin University, Australia

Anna Goy, University of Torino, Italy

George A. Gravvanis, Democritus University of Thrace, Greece

Patrizia Grifoni, CNR, Italy

Stephan Groß, Technische Universität Dresden, Germany

Carmem Satie Hara, Universidade Federal do Paraná, Brazil

Benjamin Heckmann, University of Applied Sciences Darmstadt, Germany

Markus Helfert, Dublin City University, Ireland

Frans Henskens, University of Newcastle, Australia

Neil Chue Hong, University of Edinburgh, U.K.

Marianne Huchard, Lirmm Université Montpelier 2, France

Mohamed Hussien, Suez Canal University, Egypt

Sorin M. Iacob, Thales Nederland BV, The Netherlands

Ilian Ilkov, IBM Nederland B.V., The Netherlands

Anca Daniela Ionita, University Politehnica of Bucharest, Romania

Fuyuki Ishikawa, National Institute of Informatics, Japan

Hiroshi Ishikawa, Tokyo Metropolitan University, Japan

Ivan Ivanov, SUNY Empire State College, U.S.A.

Martin Gilje Jaatun, SINTEF ICT, Norway

Keith Jeffery, Independent Consultant (previously Science and Technology Facilities Council), U.K.

Meiko Jensen, Independent Centre for Privacy Protection Schleswig-Holstein, Germany

Yiming Ji, University of South Carolina Beaufort, U.S.A.

MING JIANG, University of Leeds, U.K.

Xiaolong Jin, Chinese Academy of Sciences, China

Jose Fernando Rodrigues Jr., Universidade de São Paulo, Brazil

Eric Jul, Alcatel Lucent, Ireland

David R. Kaeli, Northeastern University, U.S.A.

Yücel Karabulut, VMware, U.S.A.

Daniel S. Katz, University of Chicago, U.S.A.

Gabor Kecskemeti, MTA SZTAKI, Hungary

Attila Kertesz, MTA SZTAKI, Hungary

Claus-Peter Klas, FernUniversität in Hagen, Germany

PROGRAM COMMITTEE (CONT.)

Carsten Kleiner, University of Applied Sciences&Arts Hannover, Germany

Geir M. Køien, University of Agder, Norway

Dimitri Konstantas, University of Geneva, Switzerland

George Kousiouris, National Technical University of Athens, Greece

László Kovács, MTA SZTAKI, Hungary

Marcel Kunze, Karlsruhe Institute of Technology, Germany

Young Choon Lee, University of Sydney, Australia

Miguel Leitão, ISEP, Portugal

Wilfried Lemahieu, KU Leuven, Belgium

Fei Li, Vienna University of Technology, Austria

Kuan-ching Li, Providence University, Taiwan

Donghui Lin, Kyoto University, Japan

Shijun Liu, School of computer science and technology, Shandong University, China

Pedro Garcia Lopez, University Rovira i Virgili, Spain

Suksant Sae Lor, HP Labs, U.K.

Simone Ludwig, North Dakota State University, U.S.A.

Glenn Luecke, Iowa State University, U.S.A.

Theo Lynn, Dublin City University, Ireland

Shikharesh Majumdar, Carleton University, Canada

Elisa Marengo, Università degli Studi di Torino, Italy

Pierre Maret, Université de Saint Etienne, France

Ioannis Mavridis, University of Macedonia, Greece

Richard McClatchey, CCCS, University West of England, U.K.

Jose Ramon Gonzalez de Mendivil, Universidad Publica de Navarra, Spain

Andreas Menychtas, National Technical University of Athens, Greece

Mohamed Mohamed, TELECOM SudParis, France

Owen Molloy, National University of Ireland, Galway, Ireland

Marco Casassa Mont, Hewlett-Packard Laboratories, U.K.

Rubén Santiago Montero, Universidad Complutense de Madrid, Spain

Christine Morin, Institut national de recherche en informatique et en automatique (Inria), France

Kamran Munir, University of the West of England (Bristol, UK), U.K.

Víctor Méndez Muñoz, Universitat Autònoma de Barcelona, UAB, Spain

Hidemoto Nakada, National Institute of Advanced Industrial Science and Technology (AIST), Japan

Philippe Navaux, UFRGS - Federal University of Rio Grande Do Sul, Brazil

Lee Newcombe, Capgemini, U.K.

Jean-Marc Nicod, Institut FEMTO-ST, France

Bogdan Nicolae, IBM Research, Ireland

Christophe Nicolle, Laboratoire Le2i - University of Bourgogne, France

Mara Nikolaidou, Harokopio University of Athens, Greece

Karsten Oberle, Alcatel-Lucent Bell Labs, Germany

Alexander Paar, TWT GmbH Science and Innovation, Germany

David Padua, University of Illinois at Urbana-Champaign, U.S.A.

Federica Paganelli, CNIT - National Interuniversity Consortium for Telecommunications, Italy

Claus Pahl, Dublin City University, Ireland

Michael A. Palis, Rutgers University, U.S.A.

George Pallis, University of Cyprus, Cyprus

David Paul, The University of Newcastle, Australia

PROGRAM COMMITTEE (CONT.)

Juan Fernando Perez, Imperial College London, U.K.

Dana Petcu, West University of Timisoara, Romania

Giovanna Petrone, University of Torino, Italy

Jean-Marc Pierson, Université Paul Sabatier, Toulouse 3, France

Agostino Poggi, University of Parma, Italy

Antonio Puliafito, Università degli Studi di Messina, Italy

Juha Puustjärvi, University of Helsinki, Finland

Li Qi, Research Institute of Internet of Things, The Third Research Institute of MPS, China

Francesco Quaglia, Sapienza Università di Roma, Italy

Arcot Rajasekar, University of North Carolina at Chapel Hill, U.S.A.

Arkalgud Ramaprasad, University of Miami, U.S.A.

Manuel Ramos-Cabrer, University of Vigo, Spain

Nadia Ranaldo, University of Sannio, Italy

Andrew Rau-Chaplin, Dalhousie University, Canada

Christoph Reich, Hochschule Furtwangen University, Germany

Alexander Reinefeld, Zuse Institute Berlin, Germany

Norbert Ritter, University of Hamburg, Germany

Tarcísio da Rocha, Universidade Federal de Sergipe, Brazil

Luis Rodero-Merino, Galician Research and Development Center in Advanced Telecomunications (GRADIANT), Spain

Jonathan Rouzaud-Cornabas, CNRS / INRIA / Université de Lyon / ENS Lyon, France

Rizos Sakellariou, University of Manchester, U.K.

Elena Sanchez-Nielsen, Universidad De La Laguna, Spain

Patrizia Scandurra, University of Bergamo, Italy

Erich Schikuta, Universität Wien, Austria

Lutz Schubert, Ulm University, Germany

Michael Schumacher, University of Applied Sciences Western Switzerland, Switzerland

Wael Sellami, Faculty of Sciences Economics and management, Sfax, Tunisia

Giovanni Semeraro, University of Bari Aldo Moro, Italy

Carlos Serrao, ISCTE-IUL, Portugal

Armin Shams, Sharif University of Technology, Iran, Islamic Republic of

Keiichi Shima, IIJ Innovation Institute, Japan

Adenilso da Silva Simão, Universidade de São Paulo, Brazil

Marten van Sinderen, University of Twente, The Netherlands

Richard O. Sinnott, University of Melbourne, Australia

Frank Siqueira, Universidade Federal de Santa Catarina, Brazil

Cosmin Stoica Spahiu, University of Craiova - Faculty of Automation, Computers and Electronics, Romania

Josef Spillner, Technische Universität Dresden, Germany

Satish Srirama, University of Tartu, Estonia

Ralf Steinmetz, Technische Universität Darmstadt, Germany

Burkhard Stiller, University of Zürich, Switzerland

Yasuyuki Tahara, The University of Electro-Communications, Japan

Domenico Talia, University of Calabria and ICAR-CNR, Italy

Yoshio Tanaka, National Institute of Advanced Industrial Science & Technology, Japan

Cedric Tedeschi, IRISA - University of Rennes 1, France

Gilbert Tekli, Telecom St. Etienne, Lebanon

Joe Tekli, Antonine University (UPA), Lebanon

PROGRAM COMMITTEE (CONT.)

Guy Tel-Zur, Ben-Gurion University of the Negev (BGU), Israel

Vagan Terziyan, University of Jyväskylä, Finland

Maria Beatriz Toledo, State University of Campinas, Brazil

Orazio Tomarchio, University of Catania, Italy

Johan Tordsson, Umeå University, Sweden

Paolo Trunfio, Università della Calabria, Italy

Francesco Tusa, University College London, U.K.

Astrid Undheim, Telenor ASA, Norway

Geoffroy Vallee, Oak Ridge National Laboratory, U.S.A.

Kurt Vanmechelen, Universiteit Antwerpen, Belgium

Luis M. Vaquero, HP Labs, U.K.

Athanasios Vasilakos, University of Western Macedonia, Greece

Massimo Villari, University of Messina, Italy

Sabrina de Capitani di Vimercati, Università degli Studi di Milano, Italy

Bruno Volckaert, Ghent University, Belgium

Hiroshi Wada, NICTA, Australia

Maria Emilia M. T. Walter, University of Brasilia, Brazil

Chen Wang, CSIRO ICT Centre, Australia

Dadong Wang, CSIRO, Australia

Martijn Warnier, Delft University of Technology, The Netherlands

Marco Winckler, University Paul Sabatier (Toulouse 3), France

Jan-Jan Wu, Academia Sinica, Taiwan

Hany F. El Yamany, Suez Canal University, Egypt

Bo Yang, University of Electronic Science and Technology of China, China

Ustun Yildiz, University of California, San Diego, U.S.A.

Michael Zapf, Georg Simon Ohm University of Applied Sciences, Germany

Wolfgang Ziegler, Fraunhofer Institute SCAI, Germany

AUXILIARY REVIEWERS

Laeeq Ahmed, PDC, Royal Institute of Technology, Sweden

Christophe Bobineau, Grenoble Institute of Technology, France

Damien Borgetto, Institut de Recherche en Informatique de Toulouse (IRIT), France

Philip Church, Deakin University, Australia

José Cordeiro, Polytechnic Institute of Setúbal / INSTICC, Portugal

Alevtina Dubovitskaya, HES-SO Valais Wallis, EPFL, Switzerland

Eugen Feller, INRIA and Lawrence Berkeley National Lab, U.S.A.

Antonios Gouglidis, University of Macedonia, Greece

Katarina Grolinger, Western University, Canada

Christos Grompanopoulos, University of Macedonia, Greece

Leo Iaquinta, University of Milano-Bicocca, Italy

Sotiris Kotsiantis, Educational Software Development Laboratory, University of Patras, Greece

Wubin Li, Department of Computing Science, Umeå University, Sweden

Giovanni Maccani, National University of Ireland Maynooth, Italy

Ketan Maheshwari, Argonne National Laboratory, U.S.A.

Martin Meyer, Dublin City University, Ireland

Cataldo Musto, Universita degli studi di Bari, Italy

P-O Östberg, Umeå University, Sweden

Nikos Parlavantzas, IRISA, France

Plamen Petkov, Dublin City University, Ireland

Eduardo Roloff, UFRGS, Brazil

Petter Svärd, Umeå University, Sweden

Luis Tomas, Umeå University, Sweden

SPECIAL SESSIONS PROGRAM COMMITTEE

SPECIAL SESSION ON FEDERATED CLOUD GOVERNANCE

Tim Bell, CERN, Switzerland

Gonçalo Borges, LIP, Portugal

Karim Chine, Cloud Era Ltd, U.K.

Enol Fernández, IFCA, Spain

Tomás Fernández, Univ. Santiago de Compostela, Spain

Víctor Manuel Fernández, Santiago de Compostela University, Spain

Nuno Ferreira, European Grid Infrastructure, The Netherlands

Mario Úbeda Garcia, CERN, France

Álvaro López García, Spanish National Research Council, Spain

Tristan Glatard, CNRS, Canada

Jordi Guijarro, CESCA, Spain

Björn Hagemeier, Forschungszentrum Jülich GmbH, Germany

Fotis Karayannis, Independentf, Greece

Charles Loomis, LAL, Univ. Paris-Sud, CNRS/IN2P3, France

Gilles Mathieu, CNRS/IN2P3, France

Boris Parak, CESNET, z. s. p. o., Czech Republic

Satoshi Sekiguchi, AIST, Japan

Peter Szegedi, TERENA, The Netherlands

SPECIAL SESSION ON MULTI-CLOUDS

Marcos Almeida, Softeam, France

Danilo Ardagna, Politecnico di Milano, Italy

Juan Fernando Perez Bernal, Imperial College London, U.K.

Alysson Bessani, University of Lisbon, Portugal

Massimo Ficco, Second University of Naples, Italy

Elisabetta Di Nitto, Politecnico di Milano, Italy

Leire Orue-Echevarria, Tecnalia Research & Innovation, Spain

Massimiliano Rak, Second University of Napoli, Italy

Umberto Villano, Università degli Studi del Sannio, Italy

SELECTED PAPERS BOOK

A number of selected papers presented at CLOSER 2014 will be published by Springer-Verlag in a CCIS Series book. This selection will be done by the Conference Chair and Program Co-chairs, among the papers actually presented at the conference, based on a rigorous review by the CLOSER 2014 Program Committee members.

FOREWORD

This book contains the proceedings of the 4th International Conference on Cloud Computing and Services Science (CLOSER 2014) which was organized and sponsored by the Institute for Systems and Technologies of Information, Control and Communication (INSTICC). This conference was held in Cooperation with the Association for Computing Machinery - Special Interest Group on Management Information Systems.

The technical program of CLOSER 2014 covered areas like "Cloud Computing Fundamentals", "Services Science Foundation for Cloud Computing", "Cloud Computing Platforms and Applications", "Cloud Computing Enabling Technology" and "Mobile Cloud Computing and Services." We expect that these proceedings will appeal to a broad audience of engineers, scientists and business people interested in cloud computing and service systems. We further believe that the papers in these proceedings demonstrate new and innovative solutions, and highlight technical problems in the mentioned areas that are challenging and worthwhile.

The conference was also complemented with one workshop and two special sessions namely the Emerging Software as a Service and Analytics - ESaaSA 2014 (co-chaired by Victor Chang, Gary Wills and Robert Walters); Special Session on Federated Cloud Governance - FedCloudGov 2014 (chaired by Víctor Méndez Muñoz) and the Special Session on Multi-Clouds - MultiCloud 2014 (chaired by Dana Petcu).

CLOSER 2014 received 127 paper submissions from all continents. From these, 21 papers were published and presented as full papers, 40 were accepted as short papers and another 27 were presented as posters. These numbers, leading to a full-paper acceptance ratio of 16,54% and an oral paper acceptance ratio of 48%, show the intention of preserving a high quality forum for the next editions of this conference.

The high quality of the CLOSER 2014 programme was enhanced by three keynote lectures, delivered by experts in their fields, including (alphabetically): Helge Meinhard (CERN, IT Department, Switzerland), Ignacio Martín Llorente (OpenNebula Director and CEO & Full Professor at UCM, Spain) and Paulo Gonçalves (INRIA-DANTE, France).

The high number and high quality of the papers received imposed difficult choices in the review process. Each paper was reviewed by at least two experts. Those papers that, according to the reviews, were considered adequately balanced in terms of quality, originality and relevance to the conference themes were selected. We hope that these Conference Proceedings, submitted for indexation to Thomson Reuters Conference Proceedings Citation Index, INSPEC, DBLP, EI and Scopus, may help the Cloud Computing community to find interesting research work. All presented papers will soon be available at the SCITEPRESS digital library. Furthermore, a short list of presented papers will be selected and their authors will be invited to submit an extended version of their paper to be included in a forthcoming book of CLOSER Selected Papers to be published by Springer during 2014.

At the closing session, awards based on the best combined marks of paper reviewing, as assessed by the Program Committee, and the quality of the presentation, as assessed by session chairs at the conference venue, were conferred to the best papers and the best student papers submitted to the conference.

In conjunction with a set of related conferences, namely WEBIST and SMARTGREENS, CLOSER had a new kind of session, named The European Project Space (EPS). The EPS aimed to provide insights into the research projects that are currently going on in Europe within the fields of web-based information systems, Cloud Computing and Smart Grids and Green IT Systems. This initiative intended to facilitate opportunities for knowledge and technology sharing, and establish the basis for future collaboration networks involving current project partners and interested conference delegates.

Finally, we would like to express our thanks, first of all, to the authors of the technical papers, whose work and dedication make possible to put together a program that we believe is very exciting and of high technical quality. Next, we would like to thank all the members of the program committee and auxiliary reviewers, who helped us with their expertise and time. We would also like to thank the invited speakers for their invaluable contribution and for sharing their vision with the CLOSER 2014 audience. Special thanks should be addressed to the INSTICC Steering Committee whose invaluable work made this event possible.

Markus Helfert
Dublin City University, Ireland

Frédéric Desprez
LIP / INRIA, France

Donald Ferguson
Dell, U.S.A.

Frank Leymann
University of Stuttgart, Germany

Víctor Méndez Muñoz
Universitat Autònoma de Barcelona, UAB, Spain

CONTENTS

INVITED SPEAKERS

KEYNOTE SPEAKERS

Agile Infrastructure at CERN - Moving 9'000 Servers into a Private Cloud IS-5
Helge Meinhard

Probabilistic Resource Allocation based on a Large Deviations Principle IS-7
Paulo Gonçalves

OpenNebula - Latest Innovations in Private Cloud Computing IS-9
Ignacio Martín Llorente

CLOUD COMPUTING FUNDAMENTALS

FULL PAPERS

Transparent Access to Relational Databases in the Cloud using a Multi-tenant ESB 5
Steve Strauch, Vasilios Andrikopoulos, Santiago Goméz Sáez and Frank Leymann

Shadows on the Cloud: An Energy-aware, Profit Maximizing Resilience Framework for Cloud Computing 15
Xiaolong Cui, Bryan Mills, Taieb Znati and Rami Melhem

A SWRL Bridge to XACML for Clouds Privacy Compliant Policies 27
Hanene Boussi Rahmouni, Marco Casassa Mont, Kamran Munir and Tony Solomonides

Procurement Auctions to Maximize Players' Utility in Cloud Markets 38
Paolo Bonacquisto, Giuseppe Di Modica, Giuseppe Petralia and Orazio Tomarchio

SHORT PAPERS

Portable Green Cloud Services 53
Stephan Ulbricht, Wolfram Amme, Thomas Heinze, Simon Moser and Hans-Dieter Wehle

A Template Description Framework for Services as a Utility for Cloud Brokerage 60
Li Zhang, Frank Fowley and Claus Pahl

Performance Prediction for Unseen Virtual Machines 70
John O'Loughlin and Lee Gillam

Cloud Computing Adoption Factors and Processes for Enterprises - A Systematic Literature Review 78
Rania Fahim El-Gazzar

Essential Elements of an SME-specific Search of Trusted Cloud Services 88
Andrea Horch, Constantin Christmann, Holger Kett, Jürgen Falkner and Anette Weisbecker

Automatic Software Development as a Service (ASDaaS) 95
Hind Benfenatki, Catarina Ferreira da Silva, Nabila Benharkat and Parisa Ghodous

Iris: An Inter-cloud Resource Integration System for Elastic Cloud Data Centers 103
Ryousei Takano, Atsuko Takefusa, Hidemoto Nakada, Seiya Yanagita and Tomohiro Kudoh

A Survey on Approaches for Interoperability and Portability of Cloud Computing Services 112
Kostas Stravoskoufos, Alexandros Preventis, Stelios Sotiriadis and Euripides G. M. Petrakis

VM Live Migration Algorithm Based on Stable Matching Model to Improve Energy Consumption and Quality of Service 118
Abdelaziz Kella and Ghalem Belalem

An Open-Source Framework for Integrating Heterogeneous Resources in Private Clouds 129
Julio Proaño, Carmen Carrión and Blanca Caminero

A Green Model of Cloud Resources Provisioning 135
Meriem Azaiez, Walid Chainbi and Hanen Chihi

A Simple and Generic Interface for a Cloud Monitoring Service 143
Augusto Ciuffoletti

Model-based Approach to Automatic Software Deployment in Cloud 151
Franklin Magalhães Ribeiro Junior and Tarcísio da Rocha

Cost-efficient Capacitation of Cloud Data Centers for QoS-aware Multimedia Service Provision 158
Ronny Hans, Ulrich Lampe, Michael Pauly and Ralf Steinmetz

Internet of Things in the Cloud - Theory and Practice 164
Philip Wright and Andrea Manieri

A Framework for Ambient Computing 170
Mohammed Fethi Khalfi and Sidi Mohamed Benslimane

Construction and Verification of the Trusted Cloud Service 179
Dexian Chang, Hongqi Zhang and Yingjie Yang

Storage CloudSim - A Simulation Environment for Cloud Object Storage Infrastructures 186
Tobias Sturm, Foued Jrad and Achim Streit

Robust Cloud Monitor Placement for Availability Verification 193
Melanie Siebenhaar, Ulrich Lampe, Dieter Schuller and Ralf Steinmetz

Optimizing the Stretch for Virtual Screening Application on Pilot-agent Platforms on Grid/Cloud by using Multi-level Queue-based Scheduling 199
T. Q. Bui, E. Medernach, V. Breton and H. Q. Nguyen

Migrating Relational Databases to the Cloud - Rethinking the Necessity of Rapid Elasticity 205
Kevin Williams

Beyond Grid Portals and Towards SaaS - A New Access Model for Computational Grids, in the dMRI Brain Context 211
Tarik Zakaria Benmerar, Mouloud Kachouane and Fatima Oulebsir-Boumghar

SERVICES SCIENCE FOUNDATION FOR CLOUD COMPUTING

FULL PAPERS

Cloud Asset Pricing Tree (CAPT) - Elastic Economic Model for Cloud Service Providers 221
Soheil Qanbari, Fei Li, Schahram Dustdar and Tian-Shyr Dai

Creo: Reduced Complexity Service Development 230
Per-Olov Östberg and Niclas Lockner

Discovering Secure Service Compositions 242
Luca Pino, George Spanoudakis, Andreas Fuchs and Sigrid Gürgens

SHORT PAPERS

Proactive Adaptation in Service Composition using a Fuzzy Logic Based Optimization Mechanism 257
Silvana de Gyvés Avila and Karim Djemame

Preference-driven Refinement of Service Compositions 268
Cheikh Ba, Umberto Costa, Mirian Halfeld Ferrari, Rémy Ferré, Martin A. Musicante, Veronika Peralta and Sophie Robert

Collaborative, Dynamic & Complex Systems - Modeling, Provision & Execution 276
Vasilios Andrikopoulos, Santiago Goméz Sáez, Dimka Karastoyanova and Andreas Weiß

Choreography-based Consolidation of Multi-instance BPEL Processes 287
Sebastian Wagner, Oliver Kopp and Frank Leymann

A Cloud Application for Security Service Level Agreement Evaluation 299
Valentina Casola, Massimiliano Rak and Giuseppe Alfieri

Utility-based Decision Making in Collective Adaptive Systems 308
Vasilios Andrikopoulos, Marina Bitsaki, Santiago Goméz Sáez, Dimka Karastoyanova, Christos Nikolaou and Alina Psycharaki

A Web Service Discovery Approach Based on Hybrid Negotiation 315
Raja Bellakhal, Walid Chainbi and Khaled Ghédira

Task Placement in a Cloud with Case-based Reasoning 323
Eric Schulte-Zurhausen and Mirjam Minor

The Management System of INTEGRIS - Extending the Smart Grid to the Web of Energy 329
Joan Navarro, Andreu Sancho-Asensio, Agustín Zaballos, Virginia Jiménez-Ruano, David Vernet and José Enrique Armendáriz-Iñigo

CLOUD COMPUTING PLATFORMS AND APPLICATIONS

FULL PAPERS

Monitoring Large Cloud-Based Systems 341
Mauro Andreolini, Marcello Pietri, Stefania Tosi and Andrea Balboni

Robust Performance Control for Web Applications in the Cloud 352
Hector Fernandez, Corina Stratan and Guillaume Pierre

PAEAN4CLOUD - A Framework for Monitoring and Managing the SLA Violation of Cloud Service-based Applications 361
Yehia Taher, Rafiqul Haque, Dinh-Khoa Nguyen and Béatrice Finance

SHORT PAPERS

Methodology to Determine Relationships between Performance Factors in Hadoop Cloud Computing Applications 375
Luis Eduardo Bautista Villalpando, Alain April and Alain Abran

A Cloud-based GWAS Analysis Pipeline for Clinical Researchers 387
Paul Heinzlreiter, James Richard Perkins, Óscar Torreño, Johan Karlsson, Juan Antonio Ranea, Andreas Mitterecker, Miguel Blanca and Oswaldo Trelles

Architectural Design of a Deployment Platform to Provision Mixed-tenancy SaaS-Applications 395
Matthias Reinhardt, Stefan T. Ruehl, Stephan A. W. Verclas, Urs Andelfinger and Alois Schütte

DDSF: A Data Deduplication System Framework for Cloud Environments 403
Jianhua Gu, Chuang Zhang and Wenwei Zhang

Everest: A Cloud Platform for Computational Web Services 411
Oleg Sukhoroslov and Alexander Afanasiev

Economical Aspects of Database Sharding 417
Uwe Hohenstein and Michael C. Jaeger

Towards a Composition-based APIaaS Layer 425
Claudio Guidi, Saverio Giallorenzo and Maurizio Gabbrielli

Comparison of Request Admission Based Performance Isolation Approaches in Multi-tenant SaaS Applications 433
Rouven Kreb and Manuel Loesch

An Open Market of Cloud Data Services 439
Verena Kantere

Mjolnirr: A Hybrid Approach to Distributed Computing - Architecture and Implementation 445
Dmitry Savchenko and Gleb Radchenko

Distributed File System Based on Erasure Coding for I/O-Intensive Applications 451
Dimitri Pertin, Sylvain David, Pierre Evenou, Benoît Parrein and Nicolas Normand

Data Integrity in Cloud Transactions 457
Sachi Nishida and Yoshiyuki Shinkawa

CLOUD COMPUTING ENABLING TECHNOLOGY

FULL PAPERS

ALPS: The Accountable Cloud Protocol Stack 467
Wassim Itani, Ayman Kayssi and Ali Chehab

A User Data Location Control Model for Cloud Services 476
Kaniz Fatema, Philip Healy, Vincent C. Emeakaroha, John P. Morrison and Theo Lynn

A Cloud Accountability Policy Representation Framework 489
Walid Benghabrit, Hervé Grall, Jean-Claude Royer, Mohamed Sellami, Monir Azraoui, Kaoutar Elkhiyaoui, Melek Önen, Anderson Santana De Oliveira and Karin Bernsmed

Context-aware Cloud Application Management 499
Uwe Breitenbücher, Tobias Binz, Oliver Kopp, Frank Leymann and Matthias Wieland

SHORT PAPERS

Evaluating the Impact of Virtualization on Performance and Power Dissipation 513
Francisco J. Clemente-Castelló, Sonia Cervera, Rafael Mayo and Enrique S. Quintana-Ortí

Energy-aware VM Scheduling in IaaS Clouds using Pliant Logic 519
Attila Benyi, Jozsef Daniel Dombi and Attila Kertesz

Improving Resource Utilization in Cloud Environments using Application Placement Heuristics 527
Atakan Aral and Tolga Ovatman

On Optimizing Resource Allocation and Application Placement Costs in Cloud Systems 535
Cihan Seçinti and Tolga Ovatman

Challenges of Capacity Modelling in Complex IT Architectures 543
Andrea Kõ, Péter Fehér and Zoltán Szabó

Optimizing Access Control Performance for the Cloud 551
Slim Trabelsi, Adrien Ecuyer, Paul Cervera Y Alvarez and Francesco Di Cerbo

Unified Invocation of Scripts and Services for Provisioning, Deployment, and Management of Cloud Applications Based on TOSCA 559
Johannes Wettinger, Tobias Binz, Uwe Breitenbücher, Oliver Kopp, Frank Leymann and Michael Zimmermann

Analysing the Lifecycle of Future Autonomous Cloud Applications 569
Geir Horn, Keith Jeffery, Jörg Domaschka and Lutz Schubert

QoS- and Security-aware Composition of Cloud Collaborations 578
Olga Wenge, Ulrich Lampe and Ralf Steinmetz

A Requirements Analysis for IaaS Cloud Federation 584
Alfonso Panarello, Antonio Celesti, Maria Fazio, Massimo Villari and Antonio Puliafito

Analysing the Migration Time of Live Migration of Multiple Virtual Machines 590
Kateryna Rybina, Abhinandan Patni and Alexander Schill

Towards a Security SLA-based Cloud Monitoring Service 598
Dana Petcu and Ciprian Crăciun

Extending Hypervisor Architecture to Allow One Way Data Transfers from VMs to Hypervisors 604
Mustafa Aydin and Jeremy Jacob

Brokering SLAs for End-to-End QoS in Cloud Computing 610
Tommaso Cucinotta, Diego Lugones, Davide Cherubini and Karsten Oberle

Confidential Execution of Cloud Services 616
Tommaso Cucinotta, Davide Cherubini and Eric Jul

Key Completion Indicators - Minimizing the Effect of DoS Attacks on Elastic Cloud-based Applications Based on Application-level Markov Chain Checkpoints 622
George Kousiouris

MOBILE CLOUD COMPUTING AND SERVICES

SHORT PAPERS

Utilizing ICN/CCN for Service and VM Migration Support in Virtualized LTE Systems 633
Morteza Karimzadeh, Triadimas Satria and Georgios Karagiannis

Applying SDN/OpenFlow in Virtualized LTE to Support Distributed Mobility Management (DMM) 639
Morteza karimzadeh, Luca Valtulina and Georgios Karagiannis

Implementation of Asynchronous Mobile Web Services - Implementation and First Usage 645
Marc Jansen, Javier Miranda and Juan Manuel Murillo

Semantic Data Integration for Ubiquitous Logistics - An Approach Supporting Autonomous Logistics in Urban Environments 652
Stefan Wellsandt, Konstantin Klein, Marco Franke, Karl Hribernik and Klaus-Dieter Thoben

SPECIAL SESSION ON FEDERATED CLOUD GOVERNANCE

FULL PAPERS

On-demand Cloud Architecture for Academic Community Cloud - Another Approach to Inter-cloud Collaboration 661
Shigetoshi Yokoyama and Nobukazu Yoshioka

A New Cloud Computing Governance Framework 671
Ahmed Shaker Saidah and Nashwa Abdelbaki

Cloud Governance by a Credit Model with DIRAC 679
Víctor Méndez Muñoz, Adrian Casajús Ramo, Ricardo Granciani Diaz and Andrei Tsaregorodtsev

SPECIAL SESSION ON MULTI-CLOUDS

FULL PAPERS

Multi-dimensional Resource Allocation for Data-intensive Large-scale Cloud Applications 691
Foued Jrad, Jie Tao, Ivona Brandic and Achim Streit

Multi-cloud and Multi-data Stores - The Challenges Behind Heterogeneous Data Models 703
Marcos Aurélio Almeida da Silva and Andrey Sadovykh

A Multi-Cloud Framework for Measuring and Describing Performance Aspects of Cloud Services Across Different Application Types 714
G. Kousiouris, G. Giammatteo, A. Evangelinou, N. Galante, E. Kevani, C. Stampoltas, A. Menychtas, A. Kopaneli, K. Ramasamy Balraj, D. Kyriazis, T. Varvarigou, P. Stuer and L. Orue-Echevarria Arrieta

HS4MC - Hierarchical SLA-based Service Selection for Multi-Cloud Environments 722
Soodeh Farokhi, Foued Jrad, Ivona Brandic and Achim Streit

SHORT PAPERS

Cloudifying Applications with ARTIST - A Global Modernization Approach to Move Applications onto the Cloud 737
Leire Orue-Echevarria, Juncal Alonso, Hugo Brunelière, Andreas Menychtas, Philip Langer and Manuel Wimmer

Towards Bridging the Gap Between Scalability and Elasticity 746
Nicolas Ferry, Gunnar Brataas, Alessandro Rossini, Franck Chauvel and Arnor Solberg

Preliminary Design of a Platform-as-a-Service to Provide Security in Cloud 752
Valentina Casola, Alessandra De Benedictis, Massimiliano Rak and Umberto Villano

EU Project SeaClouds - Adaptive Management of Service-based Applications Across Multiple Clouds 758
Antonio Brogi, José Carrasco, Javier Cubo, Francesco D'Andria, Ahmad Ibrahim, Ernesto Pimentel and Jacopo Soldani

AUTHOR INDEX 765

INVITED SPEAKERS

KEYNOTE SPEAKERS

Agile Infrastructure at CERN
Moving 9'000 Servers into a Private Cloud

Helge Meinhard
CERN, IT Department, Switzerland

Abstract: CERN, the big particle physics lab close to Geneva (Switzerland), is home to the Large Hadron Collider (LHC) and its four data-intensive experiments. In 2012, a new particle was detected that was identified in 2013 as the long-sought Higgs boson. The experiments deliver some 30 Petabytes of data per year to the CERN computer centre that runs almost 10'000 servers in order to process them. For the next years, data rates will grow, and so will the number of servers. In order to cope with this increase without corresponding increase of personnel, CERN reviewed the way it runs its computer centre, and launched the "Agile Infrastructure" project with the aim of organising services into horizontal layers; moving to a private cloud is an integral part of the strategy. Major work areas are configuration, cloud computing, and monitoring. Whenever possible, existing free, open-source solutions have been selected as components for well-confined functionality, with CERN adding just a bit of glue to make the components work together, with the aim of being able to easily replace a component, should a better solution become available.
Cornerstones of the new system include Puppet and PuppetDB, Foreman, Openstack, ElasticSearch and Kibana, Flume and others. As of January 2014, some 7'000 servers (physical or virtual) are managed with Puppet, and some 2'100 physical servers have been moved into the private cloud. The presentation will discuss the motivation, describe the current choices of tools, report on experience made so far, and give an outlook of what is expected to come.

BRIEF BIOGRAPHY

Helge Meinhard studied physics and obtained his Ph. D. in experimental particle physics in 1991. After two years of research at CERN, he provided computing support to the CHORUS and ATLAS experiments at CERN. In 2001 he joined CERN's IT Department, where until 2009 he was responsible of server and storage procurements. Since 2010 he has been the head of the "Platform and Engineering Services" group; the group provides large-scale services to the physics and the engineering communities, most notably a batch computing service on about 4'500 servers.

Probabilistic Resource Allocation based on a Large Deviations Principle

Paulo Gonçalves[i]
INRIA-DANTE, France

EXTENDED ABSTRACT

Users of a Cloud Computing platform can have several numbers of choices regarding server selection (some are compute intensive, some provide better I/O performance, some are superior in networking). Cloud provider such as Amazon offers many different server instances that differ in many aspects with respect to CPU speed, network bandwidth and memory capacity. Each of these instances provides a certain amount of dedicated resource and charges per instance-hour consumed. A Service Provider finds it to be extremely difficult to optimise the best combination of servers to be deployed in a \emph{Cloud} for his business on a certain application. This problem differs from the concept of traditional distributed computing (like Grid), since the numbers of servers are virtually unlimited but bandwidth is limited. The choice of deployment of resources can be dynamically tuned using cloud virtualization, that abstracts the IT resources to allow communication and control on-line. Cost of resources varies significantly depending on server types and Cloud Service Providers.

In most applications, the amount of IT resource that is actually used, is a highly variable quantity that follows the instantaneous activity, and in particular the volume of exchanged traffic when network infrastructures are concerned. Depending on the type of application, the generated workload can be a highly varying process that turns difficult to find an acceptable trade-off between an expensive over-provisioning able to anticipate peak loads and a sub performing resource allocation that does not mobilise enough resources. To bypass this challenge, dynamic bandwidth allocation is an original approach that we chose to investigate in the context of network virtualization. We aim to demonstrate the proof of concept for the case of a Video on Demand (VoD) system by adaptively tuning the provisioned bandwidth to the current application workload. In this work we have resorted to probabilistic provisioning of resource management; however in some situations it can be used to anticipate resource requirements that can serve as inputs for dynamic resource allocation.

Our approach attempts to capture some properties that describe user behaviors or workload generating mechanism of the system and fits them to a mathematical model satisfying particular properties. We leverage these properties to derive a probabilistic assumption on the mean workload of the system at different time resolutions. Embedding the notion of time scale is very important since time scale is by essence intrinsic to dynamicity. In this study we build our system using epidemic models where Markovian models are widely used and happen to satisfy to the specific property mentioned above.

Epidemic information dissemination has been an active area of research in distributed systems, such as Peer-to-Peer (P2P) or VoD systems. In Eugster et al. (Eugster et al., 2004), it has been already demonstrated that the epidemic algorithms can be used as an effective solution for information dissemination in the P2P systems as deployed on Internet or ad-hoc networks. The authors of Bonald et al. (Bonald et al., 2008) studied random epidemic strategies like the random peer, latest useful chunk algorithm to achieve optimal information dissemination. However the most relevant work to our study is derived in Caron et al. (Caron et al., 2011) where the authors proposed an approach to predict workload for cloud clients. They used auto-scaling algorithm for resource provisioning and validated the result with real-world Cloud client application traces. Our approach encompasses both constructive Markovian model to reproduce epidemic information dissemination and workload provisioning aspects. However, we stress that its originality stems from the analysis of the Large Deviation property of the proposed Markovian model. The resulting characterization can be viewed as a multi-resolution extension of the classical

steady-state distribution for the observable mean value of the random process over different aggregated time scales.

After constructing the Markovian mathematical model, we propose two possible and generic ways to exploit these information in the context of probabilistic resource provisioning. They can serve as the input of resource management functionalities of the Cloud environment. It is evident that we can not define elasticity without the notion of a time scale. The Large Deviation Principle (LDP) is capable of automatically integrating the time resolution in automatic description of the system. It is to be noticed that Markovian processes do satisfy the LDP, but so do some other models as well. Hence, our proposed probabilistic approach is very generic and can adapt to address any provisioning issues, provided the resource volatility can be resiliently represented by a stochastic process for which the LDP holds true.

BIBLIOGRAPHY

T. Bonald, L. Massouli_e, F. Mathieu, D. Perino, and A. Twigg. Epidemic live streaming: Optimal performance trade-offs. *ACM SIGMETRICS Performance Evaluation Review - SIG-METRICS '08*, 36(1):325–336, June 2008.

E. Caron, F. Desprez, and A. Muresan. Pattern matching based forecast of non-periodic repetitive behavior for cloud clients. *Journal of Grid Computing*, 9(1):49–64, March 2011.

P. T Eugster, R. Guerraoui, A. Kermarrec, and L. Massoulie. Epidemic information dissemination in distributed systems. *IEEE Computer Society*, 37(5):60–67, May 2004.

BRIEF BIOGRAPHY

Paulo Gonçalves (PhD, HDR) is associate researcher at the *Institut National de Recherche en Informatique et Automatique* (INRIA). His PhD work was on the analysis of non-stationary processes with emphasis on time-frequency and time-scale representations of signals. After his post-doc at Rice University (1994-96) on closely related topics, he focused his research activity on fractal analysis and more specifically on scaling laws estimation from wavelet decompositions. Then, P. Gonçalves broaden up the scope of his investigation to more general wavelet-based statistical inference and also on graph based semi-supervised classification. In the course, he privileged network metrology as the main application domain for his methodological research and dealt with statistical characterisation and modelling of traffic, for protocol quality assessment and control. Recently, he returned to wavelet theory and more generally to multi-scale analysis of dynamic graph signals.

[i] This is a joint work with Thomas Begin (Université Lyon 1), Patrick Loiseau (EURECOM) and Shubhabrata Roy (ENS Lyon)

OpenNebula
Latest Innovations in Private Cloud Computing

Ignacio Martín Llorente
OpenNebula Director and CEO & Full Professor at UCM, Spain

Abstract: The OpenNebula Project is an open-source project delivering a simple but feature-rich solution to build enterprise clouds and virtualized data centers. OpenNebula includes advanced features for integration, management, scalability, reliability and accounting that many Enterprise IT shops need for cloud adoption. With thousands of deployments worldwide, OpenNebula has a very wide user base that includes leading companies in banking, technology, telecom and hosting, and research and supercomputing centers. The keynote will describe the latest developments incorporated in OpenNebula to address the high performance and scalability challenges in large scale production environments.

BRIEF BIOGRAPHY

Dr. Llorente is Director of the OpenNebula Project and CEO & co-founder at C12G Labs. He is an entrepreneur and researcher in the field of cloud and distributed computing, having managed several international projects and initiatives on Cloud Computing, and authored many articles in the leading journals and proceedings books. Dr. Llorente is one of the pioneers and world's leading authorities on Cloud Computing. He has held several appointments as independent expert and consultant for the European Commission and several companies and national governments. He has given many keynotes and invited talks in the main international events in cloud computing, has served on several Groups of Experts on Cloud Computing convened by international organizations, such as the European Commission and the World Economic Forum, has contributed to several Cloud Computing panels and roadmaps, and currently serves in the Editorial Board of IEEE Transactions on Cloud Computing and Journal of Grid Computing - From Grids to Cloud Federations. He founded and co-chaired the Open Grid Forum Working Group on Open Cloud Computing Interface, and has participated in the main European projects in Cloud Computing. Llorente holds a Ph.D in Computer Science (UCM) and an Executive MBA (IE Business School), and is a Full Professor (Catedratico) and the Head of the Distributed Systems Architecture Group at UCM.

CLOUD COMPUTING FUNDAMENTALS

FULL PAPERS

Transparent Access to Relational Databases in the Cloud using a Multi-tenant ESB

Steve Strauch, Vasilios Andrikopoulos, Santiago Gómez Sáez and Frank Leymann
Institute of Architecture of Application Systems, University of Stuttgart, Stuttgart, Germany
{*strauch, andrikopoulos, gomez-saez, leymann*}*@iaas.uni-stuttgart.de*

Keywords: Data Access Layer, Relational Databases, Enterprise Service Bus (ESB), Multi-tenancy.

Abstract: In the last years Cloud computing has become popular among IT organizations aiming to reduce their operational costs. Applications can be designed to run in the Cloud, or can be partially or completely migrated to it. Migrating the data layer of an application to the Cloud, however, implies that existing applications might need to be adapted in order to access their migrated to the Cloud databases. In this work we examine how we can use an existing ESB to enable transparent access to the relational data store running either in the Cloud or on-premise. The goal of our approach is to minimize the effort required to adapt the application. In particular, we discuss the requirements and prototype realization of a Cloud aware data access layer for transparent data access, using an existing open source and multi-tenant aware ESB as the basis. We then evaluate the performance of our proposed solution by considering different Cloud providers and using example data from an existing benchmark as application workload.

1 INTRODUCTION

With the advent of Cloud services beyond Infrastructure as a Service (IaaS) solutions (Mell and Grance, 2011), the migration of individual application layers or even individual architectural components to the Cloud, becomes possible. Applications are usually built using the three layers pattern: Presentation, Business Logic, and Data (Fowler et al., 2002). The migration of the Data layer to the Cloud is the focus of this work. Application data is typically moved to the Cloud for Cloud bursting, data analysis, or backup and archiving purposes. We distinguish between two different types of interaction with databases in the Cloud. On the one hand there is the direct interaction with a database hosted on the Cloud as a *data store* that takes place on a fine granular level, e.g., by using SQL commands. The second interaction type relies on a service interface to interact with the Cloud database, as in the case of e.g. Amazon SimpleDB[1], which becomes a *data service*. This requires interaction on the level of the service interface that is more coarse grained compared to the interaction when using SQL for instance. We therefore need to consider both types of interactions in the following. In addition to relational databases (Codd, 1970), widely established in industry and research *Not Only SQL* (NoSQL)[2] solutions have become increasingly popular in recent years (Sadalage and Fowler, 2012). For the scope of this work we focus only on relational databases.

Looking at the Data Layer in particular, the layer can be subdivided into the Data Access Layer (DAL) and the Database Layer (DBL). The DAL is an abstraction layer encapsulating the data access functionality, while the DBL provides persistence and retrieval capabilities. As discussed in (Andrikopoulos et al., 2013), the migration of the Data Layer to the Cloud includes two main steps: the migration of the DBL to the Cloud, and the adaptation of the DAL to enable Cloud data access. The DBL can either be migrated completely or partially to the Cloud. In the latter case, the DBL is distributed using both non-Cloud technologies (local databases), and data stores and services. Thus, the distribution of the DBL essentially changes the border of the application compared to traditional applications. The assumption in our work is that the DBL has already been migrated to the Cloud, and therefore we focus on the DAL. As a consequence of the tight coupling of the Business Logic layer with the DBL in traditional applications built without using any service-oriented or Cloud technology, the Business Logic layer must be explicitly aware of the location of the data and the data store it is interacting with.

[1] Amazon SimpleDB: http://aws.amazon.com/de/simpledb/

[2] List of NoSQL Databases: http://nosql-database.org

Therefore, any change to the location of the DBL results in the need for adapting the Business Logic layer, not only to rewire the connection to the DBL, but also to cope with potentially less fine granular interaction types when moving to data service solutions.

In order to minimize the impact of this adaptation, in this work we propose the use of the Enterprise Service Bus (ESB) technology (Chappell, 2004) as the means to decouple the Business Logic from the Database Layer by leveraging the Data Access Layer. The ESB technology as the messaging infrastructure between applications and services addresses the essential requirement of application integration while ensuring loose coupling. In the recent years it has become as a result ubiquitous in enterprise computing environments. ESBs control message handling during the interaction between applications and services, allowing for dynamic routing, and are at the core of each Service-Oriented Architecture (SOA) (Josuttis, 2007). It is only natural therefore to enable the distribution of the DBL in the Cloud by taking advantage of the dynamic routing features of ESBs, allowing for hosting the database both on-premise and off-premise. As to the extend of our knowledge there is no solution so far that supports this, we want to enable the communication between the Business Logic and the DBL via the ESB (DAL) using the native database communication protocol. For this purpose, we build on an existing open source ESB previously extended for multi-tenancy (Strauch et al., 2012a; Strauch et al., 2013b) to enable transparent access to multiple data sources both locally, and in the Cloud.

Our contribution therefore can be summarized by offering:

- The identification of requirements for enabling transparent data access to data stores and data services.
- The design and realization of CDASMix, a multi-tenant aware ESB solution that enables transparent data access to databases both on-premise and off-premise.
- A performance evaluation of our proposal, establishing the impact of introducing an ESB as the DAL between the application and the database across different deployment options.

The remaining of the paper is structured as follows: Section 2 identifies the requirements for supporting transparent data access for databases that have been migrated to the Cloud. Based on these requirements, Section 3 discusses the architecture and implementation of our proposal. Section 4 provides a performance evaluation for this realization for different database and deployment options considering different providers. The paper closes with Section 5 and Section 6 summarizing related work, and concluding with some future work, respectively.

2 TRANSPARENT CLOUD DATA ACCESS REQUIREMENTS

The set of *functional* and *non-functional* requirements for transparent Cloud data access we present in this section has been identified during our work in various EU research projects and especially during the collaboration with industry partners. The requirements are geared towards minimizing the impact to the Business Logic and Presentation application layers of adapting the application to cope with the migration of the database to the Cloud (Andrikopoulos et al., 2013).

2.1 Functional Requirements

The following functionalities must be offered by any DAL that provides transparent access to databases in the Cloud.

FR_1 *Interaction with Data Stores and Data Services*: The DAL must support both fine- and coarse-grained types of interactions, through SQL and service APIs, respectively.

FR_2 *Management and Configuration*: The DAL must provide management and configuration capabilities for both data stores and data services, e.g. registration of a new data store, including its configuration data, e.g. database schemas, database system endpoint URL, etc.

FR_3 *On-premise and Off-premise Support*: The DAL has to support data stores and data services that are either hosted on-premise or off-premise and using Cloud or non Cloud technologies.

FR_4 *Transparent Data Access*: As the database layer might be distributed between on-premise and off-premise using Cloud and non-Cloud technologies, the business logic is no longer aware where the data is located compared to traditional applications. Thus, the DAL must determine autonomously the data store or data service the request should be forwarded to, based on the request sent by the Business Logic layer.

FR_5 *Caching*: As the migration of the database to the Cloud introduces a potentially high distance between the different application layers measured in network hops, the DAL has to support caching in order to mitigate the consequences of network latency.

2.2 Non-functional Requirements

In addition to the required functionalities, a DAL enabling Cloud data access should also respect the following properties.

- NFR_1 *Loose Coupling*: The DAL must allow for Cloud data stores or data services to be changed without affecting the Business Logic layer, except from the necessary reconfiguration of the connection to the data store or data service.
- NFR_2 *Security*: As the DAL stores information on the available data stores and data services including their internal structure, e.g. database schemas and access credentials, third parties have to be prevented from gaining access to these data.
- NFR_3 *Backward Compatibility & Extensibility*: DAL components have to be compatible with directly wiring the business logic with a specific data store, as in traditional applications. Furthermore, extensibility has to be ensured in order to integrate the components with future technologies and solutions, e.g. new types of data services in the Cloud.

As discussed in the previous section, the proposed approach builds on the use of the ESB technology as the realization of these requirements. However, given the fact that multiple organizational domains may have access to the same data stores or services, it has to be ensured that each organization is confined to their own data. While modern Relational Database Systems (RDBMSs) have been offering this capability for a while, *multi-tenancy* on the level of middleware is still an open issue. For this purpose, in (Strauch et al., 2012b) we identify a set of requirements for multi-tenancy for ESB solutions. These requirements, summarized in the following, have also to be taken into consideration when designing and implementing a transparent DAL solution using an ESB.

More specifically, *tenant-based identification* and *hierarchical access contro*l have to be supported, as the ESB has to manage and identify multiple tenants (organizational domains) and their corresponding users. For the administration and management of both tenants and users, *customizable interfaces* have to be provided, including both Web service interfaces and GUIs. Through these interfaces, the tenant-based deployment and configuration of the ESB and the services available for a specific tenant should be managed in a transparent way. As other multi-tenant components in a PaaS environment request similar information, e.g. workflow engines, the ESB should offer a *shared registry* of services and of tenants and users for other PaaS components. Furthermore, services and applications that are not multi-tenant aware should still be able to seamlessly use and transparently interact with the ESB.

In addition, both *data isolation* and *performance isolation* have to be ensured, in order to avoid that tenants acquire access to data or computing resources of other tenants. In order to address the *security* aspect, corresponding confidentiality, authorization, authentication, and integrity mechanisms have to be realized on tenant and user level. Finally, the ESB components must be extensible and reusable by other components in the PaaS environment to ensure independence from a specific technology or solution. All of these requirements are considered in the following discussion.

3 CLOUD DATA ACCESS SUPPORT IN ServiceMix

In this section we discuss *CDASMix (Cloud Data Access Support in ServiceMix)*, an architectural and implementation approach which fulfills the requirements identified in the previous section. As the basis for the design and implementation of our approach, we reuse and extend the ESB^{MT} multi-tenant aware ESB solution[3], as presented in (Strauch et al., 2013a). ESB^{MT} enhances the Apache ServiceMix 4.3 JBI container[4] with multi-tenant communication support within service endpoints deployed in the ESB, and multi-tenant aware dynamic endpoint deployment capabilities. As shown in Fig. 1, the ESB^{MT} architecture contains three layers: Presentation, Business Logic, and Resources. The Presentation and Business Logic layers define a generic interface and logic design approach of a multi-tenant aware administration and management application for ESBs. The Resources layer contains the ESB-specific instances cluster, databases for tenant-related information, and an intermediary component to mediate the management communication between the upper layers and the ESB instance cluster. However, ESB^{MT} focuses on enabling multi-tenancy at the communication level between application services, rather than on application's data access. Therefore, the remaining of this section describes the extensions realized for CDASMix in each of the system's layers highlighted in Fig. 1 to enable transparent Cloud data access support, and provide a deeper insight on the realized components. An extended description of the architectural and implementation ap-

[3] ESB^{MT}: http://www.iaas.uni-stuttgart.de/esbmt/

[4] Apache ServiceMix: http://servicemix.apache.org

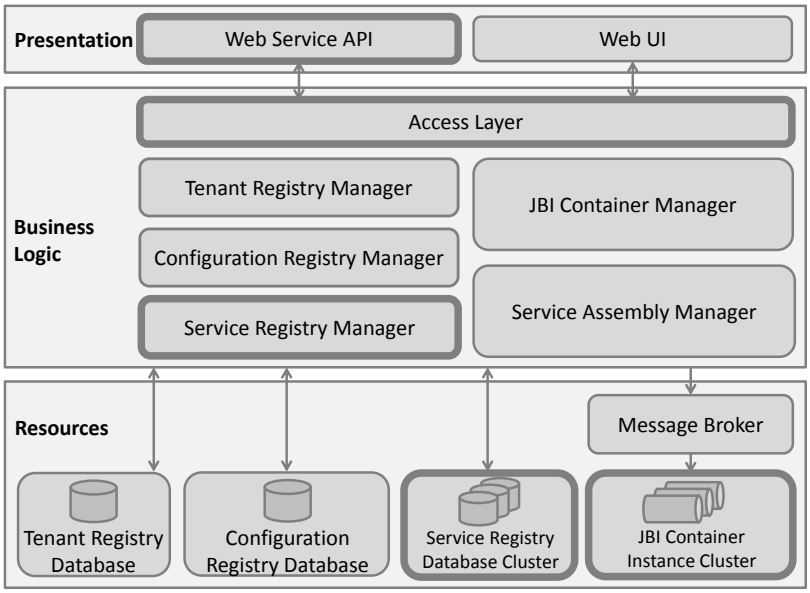


Figure 1: Architecture of CDASMix based on ESB^{MT}.

proaches used is provided in (Gómez Sáez, 2013).

3.1 Resources Layer

The Resources Layer of CDASMix consists of a set of registries and multiple JBI containers which constitute a *JBI Container Instance Cluster*, as described in Fig. 1. JBI containers provide support for message routing and transformation, as in traditional ESB solutions. ESB^{MT}, and by extension CDASMix, relies on the OSGi Framework [5] in order to allow components to be deployed in a loosely coupled way. The ServiceMix solution at the core of ESB^{MT} is equipped for this purpose with several OSGi components which realize the ESB functionality complying to the JBI specification. For example, the ServiceMix-http Binding Component (BC) supports routing of HTTP requests, and the Apache Camel [6] Service Engine (SE) realizes a set of known Enterprise Integration Patterns (Hohpe and Woolf, 2003). The Normalized Message Router (NMR) provides integration support between JBI and OSGi components within the ESB through a message-based interface. JBI components provided by ServiceMix support multiple communication protocols (e.g. HTTP, SMTP, JMS, etc.) by marshaling and demarshaling protocol-specific messages into a standardized internal Normalized Message Format (NMF) (Rademakers and Dirksen, 2008). Nevertheless, database system native communication protocols are not supported, as they are solution-specific. We focus our approach by providing communication support for a well known and widely used database system, MySQL [7]. The same approach however can be used to provide support for other systems too, e.g. PostgresSQL [8].

Figure 2 zooms in the architecture of one ESB instance in the cluster in order to illustrate how transparent message routing and transformation of application data requests was realized in CDASMix based on the ESB^{MT} architecture. More specifically, the developed `MySQL Proxy` is an OSGi component that implements an OSGi- and JBI-compliant version of a Java-based MySQL proxy (NFR$_1$). As the default MySQL Proxy shipped with MySQL distributions is implemented in the C programming language, the existing open source Java-based MySQL Proxy in Myosotis Tungsten Connector [9] was reused and extended. The MySQL OSGi bundle implements the native MySQL communication protocol and is fully integrated with the JBI container via the Normalized Message API, an OSGi bundle which exposes an interface for accessing the NMR endpoint registries and routing operations. In order to mitigate the latency introduced by using CDASMix as an intermediate communication component between the application architecture layers, the caching mediation pattern presented in (Rao et al., 2006) is realized within CDASMix. Thus, a caching component ensures high performance of SQL read op-

[5] OSGi Version 4.3: http://www.osgi.org/Download/Release4V43/

[6] Apache Camel: http://camel.apache.org

[7] MySQL: http://www.mysql.com

[8] PostgresSQL: http://www.postgresql.org

[9] Tungsten Replicator: http://myosotis.continuent.org

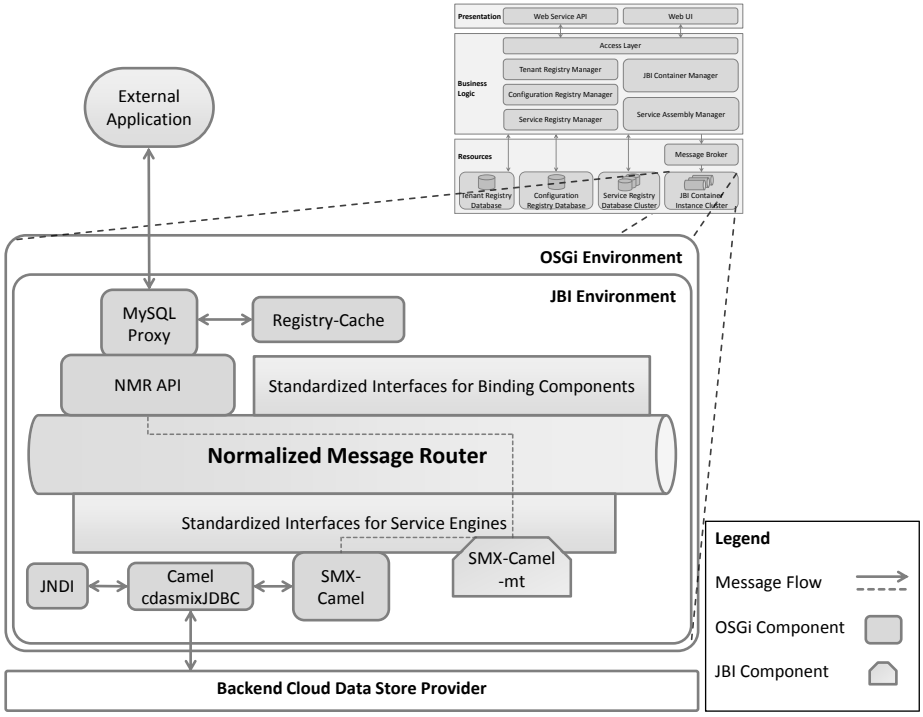


Figure 2: Architecture of an ESB Instance.

erations in CDASMix by utilizing a Least Recently Used (LRU) caching policy configured in Ehcache 2.6.0 [10], and by deleting cache records when SQL statements involve data modification (FR_5). Therefore, the caching strategy does not only aim to reduce the latency of accessing data through the ESB, but also intends to reduce the NMR system resources consumption.

The ESB^{MT} Camel SE 2011.11 (denoted as `SMX-Camel-mt` in Fig. 2) provides multi-tenancy and integration support between the JBI environment and the Enterprise Integration Patterns realizations of Apache Camel. In CDASMix, a custom Camel component named `CamelcdasmixJDBC` (see Fig. 2) was developed for dynamically connecting to backend data sources via multiple database system communication protocols (FR_4, NFR_1). This component is accessed through a camel endpoint in the `SMX-Camel` component (Fig. 2). Database connections are registered in the JNDI component of ServiceMix to decrease the latency produced by creating one database connection per user per received request. The JBI specification limits the inclusion of referenced packages in the SU during build time, rather than during runtime as in OSGi. Therefore, extensions realized on the `CamelcdasmixJDBC` component, e.g. supporting a new database system, require updating, building, and redeploying the service units referencing the `CamelcdasmixJDBC` component. Utilizing the ServiceMix Camel SE OSGi component enables loading of `CamelcdasmixJDBC` component packages during runtime (NFR_3).

CDASMix utilizes the existing ESB^{MT} registries in the Resources layer, described extensively in (Strauch et al., 2012b; Strauch et al., 2013b). A database schema modification in the Service Registry Database Cluster was necessary (Fig. 1), as it must be extended to persist tenant-related Cloud data access management information. We identify two required types of information when executing SQL statements on RDBMSs: connection related, and database related. The former contains the required information to connect to a database instance, e.g. access credentials, database URL, etc., while the latter includes information about the database schema, e.g. table identifiers. Therefore, we create a data structure which persists the tenant's backend database connection and structure information for enabling transparent access through one physical endpoint in CDASMix, to multiple backend data stores or data services located either on-premise or off-premise (FR_1, FR_3, FR_4).

3.2 Business Logic Layer

The Business Logic layer of ESB^{MT} enables role-based access control for administration and manage-

[10] Ehcache 2.6.0: http://ehcache.org

ment of the *JBI Container Instance Cluster* (Strauch et al., 2012b; Strauch et al., 2013b) (NFR_2). Enabling transparent Cloud data access support in CDASMix does not require architectural modifications on this layer, but demands implementation extensions for administrating access and persistence operations on the information related to tenant's backend data stores. Therefore, we extend the components highlighted in Fig. 1. The *Access Layer* is extended with operations executable at the tenant operator role level. The *Service Registry Manager* component is extended with operations for persisting and accessing the CDASMix-related data in the *Service Registry Database Cluster*.

3.3 Presentation Layer

The Presentation layer contains the *Web UI* and the *Web services API* components which allow the customization, administration, management, and interaction with the other layers (Strauch et al., 2012b; Strauch et al., 2013b). In CDASMix, no modifications on the Web UI were required, but a set of operations for configuring the transparent data access through CDASMix was included in the Web services API (FR_2).

4 EVALUATION

In this section we present the performance evaluation of our proposal. Therefore, we define the method and the experimental setup (Section 4.1), and discuss the experimental results (Section 4.2).

4.1 Evaluation Method & Experimental Setup

The evaluation of CDASMix in this work focuses on measuring the impact of introducing the ESB between an application and its migrated to the Cloud database. Three main scenarios are taken into consideration for this purpose, based on different Cloud infrastructures where the database can be deployed, as shown in Fig. 3:

1. in an RDBMS on the same VM as CDASMix (VM0 on Flexiscale[11]);
2. in an RDBMS hosted on a VM of a different Cloud provider (CDASMix on VM0 (Flexiscale) and MySQL hosted on Amazon EC2 [12] (VM1)); and
3. in a DBaaS solution (Amazon RDS [13]) with the CDASMix hosted on VM0.

For each of the scenarios, we compare the direct access to the database with a transparent access to the database through CDASMix. In order to simplify the discussion, we only use one tenant for database requests across all scenarios. Evaluating the multi-tenancy capabilities offered by CDASMix are the subject of ongoing future work.

For purposes of creating a data set and queries over it, we use the TPC-H benchmark[14] which emphasizes large volumes of data, queries with high degree of complexity, and answering critical business questions. The TPC-H benchmark provides a set of operations for generating different volumes of data, and a set of complex queries for a predefined database schema. The benchmark was used to create the database schema and populate it with a fixed data volume, which was consequently imported into the two MySQL deployments (on Flexiscale and Amazon) and on Amazon RDS. The generated data set has a total size of 1GB distributed among 8 tables, conforming to the TPC-H database schema.

A set of three complex (SQL) `SELECT` queries extracted from the benchmark was combined with two simpler `UPDATE` statements and used as the basis for an artificial work load. Figure 4 lists one of the select queries used as part of this process. The work load used consists of arbitrarily executed queries in a set of ten data requests in a loop of ten iterations, for a total of a hundred queries, where queries are distributed in a N updates to M selects ratio, with $(N,M) = (3,7)$. The ratio used was selected in order to force the cache of CDASMix to flush relatively often (roughly once every three queries).

Apache JMeter[15] is used for load driving purposes from a local computer, using the JDBC Connection Configuration and MySQL Connector/J (5.1.22) driver to submit the queries. In the case of direct connections to MySQL on Flexiant/Amazon EC2 and Amazon RDS, the loader has to be configured separately to address the various endpoints of the databases (connections D1, D2 and D3 in Fig. 3). When CDASMix is used, JMeter is configured only once with an ESB endpoint (connection E in Fig. 3) and the redirection of the queries to the various databases through connections E1 to E3 uses the configuration mechanisms offered by CDASMix, as explained in the previous section. JMeter is set up to measure the elapsed time for each request to be pro-

[11] Flexiant Flexiscale: http://www.flexiscale.com

[12] Amazon EC2: http://aws.amazon.com/ec2/

[13] Amazon RDS: http://aws.amazon.com/rds/

[14] Transaction Processing Performance Council Benchmark™ H http://www.tpc.org/tpch/

[15] Apache JMeter 2.9: http://jmeter.apache.org

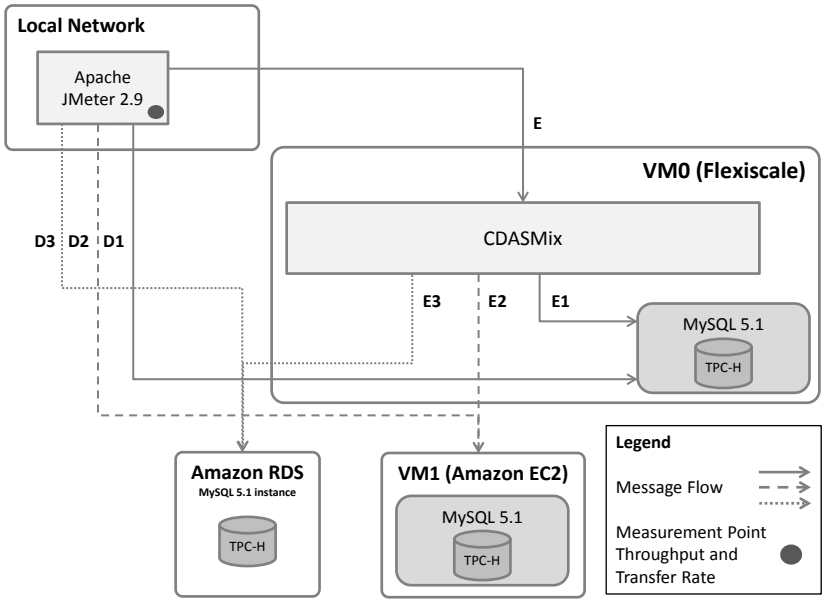


Figure 3: Overview of the Experimental Setup.

```
select s_acctbal,s_name,n_name,
    p_partkey,p_mfgr,s_address,s_phone
    ,s_comment from part,supplier,
    partsupp,nation,region
where p_partkey = ps_partkey and
    s_suppkey = ps_suppkey and p_size
    = 47 and p_type like '%BRASS' and
    s_nationkey = n_nationkey and
    n_regionkey = r_regionkey and
    r_name = 'MIDDLE EAST' and
    ps_supplycost = ( select min(
    ps_supplycost) from partsupp,
    supplier,nation,region where
    p_partkey = ps_partkey and
    s_suppkey = ps_suppkey and
    s_nationkey = n_nationkey and
    n_regionkey = r_regionkey and
    r_name = 'MIDDLE EAST')
order by s_acctbal desc, n_name,
    s_name, p_partkey limit 100;
```

Figure 4: Sample SQL Query Used.

cessed, out of which we calculate the average *throughput* of the load in executed transactions per second.

In terms of the experimental setup shown in Fig. 3, the virtual machine VM0 (8GB RAM, 4 CPUs AMD Opteron 2GHz with 512KB cache) hosted in the Flexiscale infrastructure runs Ubuntu 10.04 Linux OS. Deploying CDASMix and JBIMulti2 in VM0 also required the deployment of the JOnAS 5.2.2 application server (for JBIMulti2), and the PostgreSQL 9.1.1 database (for the registries). The maximum JVM memory heap size of the ServiceMix process was set to 1GB. In the same VM, a MySQL 5.1.67 database system with an internal cache size of 16MB was installed. In the Amazon AWS Cloud infrastructure, both the Amazon EC2 VM and the Amazon RDS database are hosted in the `eu-west-1a` availability zone. An `m1.xlarge` EC2 instance running the Ubuntu 12.04.2 LTS OS was created, and MySQL 5.1.67 database system was deployed on it, with an internal cache size of 16MB. In Amazon RDS, a `db.m1.xlarge` instance was created, running a MySQL 5.1.69 engine version, as the MySQL 5.1.67 engine version is not available.

4.2 Experimental Results

Figure 5 summarizes the measurements taken for the defined work load across the three database deployment scenarios. In addition, and for purposes of establishing a baseline, the average throughput with JMeter installed directly on VM0 (i.e. on the same VM) as the database was measured to be 0,15 transactions per second for the defined work load. When compared with the direct access to the same database from a local machine using the same load (leftmost column in Fig. 5) it can be concluded that network latency between the local machine and Flexiscale results in an average of 3,52% decrease in throughput. The effect of this latency has to be factored in for the remaining analysis whenever CDASMix is used.

In particular, the following conclusions can be drawn when examining Fig. 5:

- When comparing direct access to MySQL from

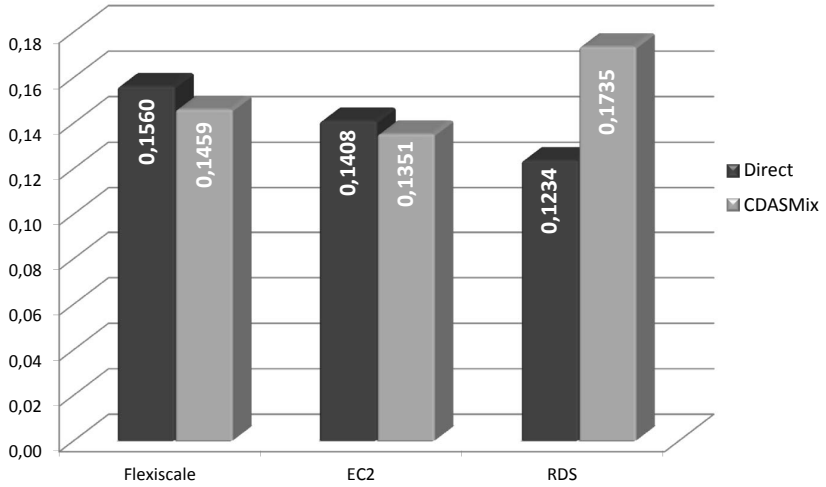


Figure 5: Average Throughput (Transactions per Second): Without (Direct) and With Using CDASMix

the (local) loader with indirect access through CDASMix, a 6,47% decrease in throughput is observed. Taking into account the 3,52% decrease due to network latency, we can conclude that the actual impact of introducing CDASMix amounts to 3%, which can be considered acceptable for many applications.

- A similarly low impact is also observed when comparing direct and indirect access to the same MySQL engine as before but on a different provider (EC2), with 4% decrease in throughput. The presence of caching on CDASMix is assumed to mitigate the effect of the additional network latency between the two providers (Flexiscale), resulting in better throughput when compared with the first scenario (i.e. Flexiscale only).
- The effect of caching is more eminent in the case of accessing RDS through CDASMix, where despite the additional network latency, using CDASMix actually improves throughput by 40,54%. However, in both cases, further evaluation of the effect of caching in relation to network latency is required.

In summary therefore, the effect of introducing CDASMix between an application and its database is acceptable for the majority of applications. Furthermore, any adversarial effects to performance, due to additional computations and dependence on network behavior, have to be measured against the benefits of the flexibility provided by the transparency of the data access layer.

5 RELATED WORK

To the extend of our knowledge, there is no existing approach aiming at using an ESB solution for transparent data access to databases in the Cloud. Most ESB solutions do provide support for storing data in relational database systems (RDBMS). However, all of them focus on providing access to databases through JDBC connections that are meant to be used for ESB-internal purposes. Fuse ESB, for example, provides JDBC data source support enabling users to connect to a database and make SQL based queries and data manipulations (FuseSource Corporation, 2013). However, it does not provide native support for incoming database messages, e.g. in the MySQL communication protocol, because data consumer endpoints supporting native database protocols are not deployable. The integration framework Apache Camel provides access to RDBMSs through the Camel-jdbc[16] component, but it also does not support interaction via native database communication protocol messages through consumer endpoints. Data sources in producer endpoints must be configured in both Apache Camel and existing ESBs prior their deployment, e.g. using Spring. This requirement forces the data sources' access configuration to be statically created, rather than dynamically created during runtime. JBoss Enterprise Data Service Platform[17] by Red Hat, Inc. provides data services with Service-Oriented Architecture (SOA) support and an ESB (Red Hat, Inc, 2011). Accessing the database

[16] Apache Camel JDBC: http://camel.apache.org/jdbc.html

[17] JBoss Data Services Platform: http://www.redhat.com/products/jbossenterprisemiddleware/data-services

layer using technologies implementing an architectural paradigm like SOA, however, might require adapting the application, and in particular the upper (presentation and business logic) architecture layers, in order to support these technologies. One of the main goals of our approach is to minimize the required adaptations of the upper application architecture layers when migrating the database layer partially or completely to the Cloud by introducing an ESB as DAL.

In terms of the performance evaluation of our approach, the different benchmarks and metrics developed in the domain of Cloud computing in the recent years focus on Cloud-related features such as elasticity (Brebner, 2012), performance isolation (Krebs et al., 2012), or on virtualization technology (Makhija et al., 2006). Given the absence of a commonly agreed approach for the evaluation of Cloud-enabled middleware components (like the ESB), in previous work (Strauch et al., 2013b) we extended an ESB-specific benchmark developed by AdroitLogic (AdroitLogic Private Ltd., 2011). This work however focused on evaluating the multi-tenancy awareness aspects of our proposed ESB solution, and it is not suitable for evaluating transparent data access. On the other hand, there are various established transaction processing and database benchmarks addressing performance evaluation of relational databases, e.g. the TPC-* benchmarks from the Transaction Processing Performance Council[18], and YCSB for Cloud database services (Cooper et al., 2010). Evaluating an ESB as transparent data access layer for accessing databases on-premise or in the Cloud is not covered by these existing benchmarks. For this purpose, and as discussed in the Section 4, we use an artificial work load taken from the TPC-H benchmark and Apache JMeter as the driver for this load.

6 CONCLUSIONS AND FUTURE WORK

In the previous sections we presented our proposal for using an ESB solution as the application Data Access Layer in order to enable transparent data access to multiple data sources, both on- and off-premise (i.e. in the Cloud). A set of requirements were identified for this purpose in terms of necessary functionalities, e.g. support of both data stores, i.e. databases hosted on an IaaS infrastructure, and Databases as a Service solutions, and of additional non-functional properties like security. Using the ESB as the DAL for multiple organizational domains (tenants) also incurs additional requirements to be considered.

The CDASMix solution discussed in this work addresses these requirements by building on a multi-tenant-aware ESB solution called ESB^{MT} that has been presented in (Strauch et al., 2013a). Adapting ESB^{MT} to allow applications to communicate with a database through the ESB required a number of modifications to it, including database-aware routing components, the implementation of a messaging proxy for native JDBC communications, and caching mechanisms for increased performance. The resulting solution, CDASMix, allows applications to communicate using JDBC with it, while transparently redirecting their (SQL) requests to databases either on-premise or in the Cloud. As shown during the evaluation section, the impact of introducing CDASMix as the application DAL, when compared to direct access to the database from the application, results in acceptable performance degradation for the used workload and database.

Our current efforts focus on expanding the evaluation of CDASMix to include the effect of multi-tenancy on performance when multiple applications use the same CDASMix instance, and improving the generality of our findings by introducing multiple workloads. Of particular interest to our work is our ongoing investigation on the interplay between the caching done on the level of CDASMix and that of the used database engine, in order to identify the optimal caching strategy for the ESB (Gómez Sáez et al., 2014). Furthermore, in terms of future work, we plan to support further database system communication protocols beyond MySQL, and enable the SQL statement transformations between different engines, e.g. MySQL to PostgreSQL, or between their versions. Communication and statement transformation support for non-relational database engines (NoSQL) offered as services in the Cloud like Amazon DynamoDB[19], Google Cloud Storage[20], and MongoHQ[21] is also under development.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from projects 4CaaSt (grant agreement no.

[18]Transaction Processing Performance Council: http://www.tpc.org

[19]Amazon DynamoDB: http://aws.amazon.com/dynamodb/

[20]Google Cloud Storage: http://cloud.google.com/storage

[21]MongoHQ: https://www.mongohq.com

258862) and Allow Ensembles (grant agreement no. 600792) part of the European Union's Seventh Framework Programme (FP7/2007-2013) and the German government funded BMBF project ECHO (01XZ13023G).

REFERENCES

AdroitLogic Private Ltd. (2011). Performance Framework and ESB Performance Benchmarking. http://www.esbperformance.org.

Andrikopoulos, V., Binz, T., Leymann, F., and Strauch, S. (2013). How to Adapt Applications for the Cloud Environment. *Computing*, 95(6):493–535.

Brebner, P. (2012). Is your Cloud Elastic Enough?: Performance Modelling the Elasticity of Infrastructure as a Service (IaaS) Cloud Applications. In *Proceedings of ICPE'12*, pages 263–266.

Chappell, D. A. (2004). *Enterprise Service Bus*. O'Reilly Media, Inc.

Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. *Commun. ACM*, 13(6):377–387.

Cooper, B. F. et al. (2010). Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of SOCC'10*, pages 143–154. ACM.

Fowler, M. et al. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional.

FuseSource Corporation (2013). Fuse ESB 4.4 – Configuring and Running Fuse ESB. http://fusesource.com/docs/esb/4.4/esb_runtime/.

Gómez Sáez, S. (2013). Extending an Open Source Enterprise Service Bus for Cloud Data Access Support. Diploma Thesis, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany.

Gómez Sáez, S., Andrikopoulos, V., Leymann, F., and Strauch, S. (2014). Evaluating Caching Strategies for Cloud Data Access Using an Enterprise Service Bus. In *Proceedings of IEEE IC2E'14*. IEEE Computer Society.

Hohpe, G. and Woolf, B. (2003). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional.

Josuttis, N. (2007). *SOA in Practice*. O'Reilly Media, Inc.

Krebs, R., Momm, C., and Kounev, S. (2012). Metrics and Techniques for Quantifying Performance Isolation in Cloud Environments. In *Proceedings of QoSA'12*, pages 91–100. ACM.

Makhija, V. et al. (2006). VMmark: A Scalable Benchmark for Virtualized Systems. Technical Report VMware-TR-2006-002, VMware, Inc.

Mell, P. and Grance, T. (2011). The NIST Definition of Cloud Computing.

Rademakers, T. and Dirksen, J. (2008). *Open-Source ESBs in Action*. Manning Publications Co., Greenwich, CT, USA.

Rao, F. Y. et al. (2006). Cache Mediation Pattern. In *Proceedings of EuroPloP'06*. Universitaetsverlag Konstanz.

Red Hat, Inc (2011). Gap Analysis: The Case for Data Services.

Sadalage, P. J. and Fowler, M. (2012). *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley Professional.

Strauch, S., Andrikopoulos, V., Gómez Sáez, S., and Leymann, F. (2013a). ESB^{MT}: A Multi-tenant Aware Enterprise Service Bus. *International Journal of Next-Generation Computing*, 4(3):230–249.

Strauch, S., Andrikopoulos, V., Gómez Sáez, S., and Leymann, F. (2013b). Implementation and Evaluation of a Multi-tenant Open-Source ESB. In *Proceedings of ESOCC'13*, volume 8135 of *Lecture Notes in Computer Science (LNCS)*, pages 79–93. Springer Berlin Heidelberg.

Strauch, S., Andrikopoulos, V., Gómez Sáez, S., Leymann, F., and Muhler, D. (2012a). Enabling Tenant-Aware Administration and Management for JBI Environments. In *Proceedings of SOCA'12*, pages 206–213. IEEE Computer Society Conference Publishing Services.

Strauch, S., Andrikopoulos, V., Leymann, F., and Muhler, D. (2012b). ESB^{MT}: Enabling Multi-Tenancy in Enterprise Service Buses. In *Proceedings of CloudCom'12*, pages 456–463. IEEE Computer Society Press.

All links were last followed on January 30, 2014.

Shadows on the Cloud: An Energy-aware, Profit Maximizing Resilience Framework for Cloud Computing

Xiaolong Cui, Bryan Mills, Taieb Znati and Rami Melhem
Department of Computer Science, University of Pittsburgh, Pittsburgh, U.S.A. *
{*mclarencui, bmills, znati, melhem*}*@cs.pitt.edu*

Keywords: Shadow Replication, Fault Tolerance, Scheduling, Resilience, Energy-aware.

Abstract: As the demand for cloud computing continues to increase, cloud service providers face the daunting challenge to meet the negotiated SLA agreement, in terms of reliability and timely performance, while achieving cost-effectiveness. This challenge is increasingly compounded by the increasing likelihood of failure in large-scale clouds and the rising cost of energy consumption. This paper proposes Shadow Replication, a novel profit-maximization resiliency model, which seamlessly addresses failure at scale, while minimizing energy consumption. The basic tenet of the model is to associate a suite of shadow processes to execute concurrently with the main process, but initially at a much reduced execution speed, to overcome failures as they occur. Two computationally-feasible schemes are proposed to achieve shadow replication. A performance evaluation framework is developed to analyze these schemes and compare their performance to traditional replication-based fault tolerance methods, focusing on the inherent tradeoff between fault tolerance, the specified SLA and profit maximization. The results show Shadow Replication leads to significant energy reduction, and is better suited for compute-intensive execution models, where up to 30% more profit increase can be achieved.

1 INTRODUCTION

Cloud Computing has emerged as an attractive platform for increasingly diverse compute- and data-intensive applications, as it allows for low-entry costs, on demand resource provisioning and allocation and reduced cost of maintaining internal IT infrastructure (Tchana et al., 2012). Cloud computing will continue to grow and attract attention from commercial and public market segments. Recent studies predict annual growth rate of 17.7 percent by 2016, making cloud computing the fastest growing segment in the software industry.

In its basic form, a cloud computing infrastructure is a large cluster of interconnected back-end servers hosted in a datacenter and provisioned to deliver on-demand, "pay-as-you-go" services and computing resources to customers through a front-end interface (Amazon, 2013). As the demand for cloud computing accelerates, cloud service providers (CSPs) will be faced with the need to expand their underlying infrastructure to ensure the expected levels of performance, reliability and cost-effectiveness, resulting in a multifold increase in the number of computing, storage and communication components in their datacenters. The direct implication of large datacenters is increased management complexity and propensity to failure. While the likelihood of a server failure is very small, the sheer number of computing, storage and communications components that can fail, however, is daunting. At such a large scale, failure becomes the norm rather than an exception (Schroeder and Gibson, 2010).

As the number of users delegating their computing tasks to CSPs increases, Service Level Agreements (SLAs) become a critical aspect for a sustainable cloud computing business model. In its basic form, an SLA is a contract between the CSPs and consumers that specifies the terms and conditions under which the service is to be provided, including expected response time and reliability. Failure to deliver the service as specified in the SLA subjects the CSP to pay a penalty, resulting in a loss of revenue.

In addition to penalties resulting from failure to meet the SLA requirement, CSPs face rising energy costs of their large-scale datacenters. It is reported

*This research is based in part upon work supported by the National Science Foundation under Grant Number CNS12-53218 and CNS12-52306. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

that energy costs alone could account for 23-50% of the expenses (Elnozahy et al., 2003) and this bill mounts up to $30 billion worldwide (Raghavendra et al., 2008). This raises the question of how fault tolerance might impact power consumption and ultimately the expected profit of the service providers.

Current fault tolerance approaches rely upon either time or hardware redundancy in order to tolerate failure. The first approach, which uses time redundancy, requires the re-execution of the failed task after the failure is detected. Although this can further be optimized by the use of checkpointing and rollback recovery, such an approach can result in a significant delay increase subjecting CSPs to penalties, when SLA terms are violated, and high energy costs due to re-execution of failing tasks.

The second approach exploits hardware redundancy and executes multiple instances of the same task in parallel to overcome failure and guarantee that at least one task reaches completion. This approach, which has been used extensively to deal with failure in critical applications, is currently used in cloud-computing to provide fault tolerance while hiding the delay of re-execution (Tsai et al., 2011; Ko et al., 2010). This solution, however, increases the energy consumption for a given service, which in turn might outweigh the profit gained by providing the service. The trade-off between profit and fault-tolerance calls for new frameworks to take both SLA requirements and energy awareness in dealing with failures.

In this paper, we address the above trade-off challenge and propose an energy-aware, SLA-based profit maximization framework, referred to as "Shadow Replication", for resilience in cloud computing. Similar to traditional replication, Shadow Replication ensures successful task completion by concurrently running multiple instances of the same task. Contrary to traditional replication, however, Shadow Replication executes the main instance of the task at the speed required to maximize profit and uses dynamic voltage and frequency scaling (DVFS) to slow down the execution of the replicas, thereby enabling a parameterized trade-off between response time, energy consumption and hardware redundancy. This allows CSPs to maximize the expected profit by accounting for income, potential penalties and energy cost.

The main challenge of Shadow Replication resides in determining jointly the execution speeds of all task instances, both before and after a failure occurs, with the objective to minimize energy and maximize profit. In this paper, we propose a reward-based analytical framework to achieve this objective. The main contributions of this paper are as follows:

- An energy-aware, SLA-based, profit maximization execution model, referred to as "Shadow Replication", for resilient cloud computing.
- A profit-based optimization model to explore the applicability of Shadow Replication to cloud computing, and to determine the optimal speeds of all task instances to maximize profit.
- In environments where either the specification or the detection of failure is hard to achieve, we propose a sub-optimal, yet practical resilience scheme, called profit-aware stretched replication.
- An evaluation framework to analyze profit and energy savings achievable by Shadow Replication, compared to existing resilience methods.

The analysis shows that in all cases, Shadow Replication outperforms existing fault tolerance methods. Furthermore, shadow replication would converge to traditional replication, when target response time is stringent, and to re-execution when target response time is relaxed or failure is unlikely, as expected.

The rest of the paper is organized as follows. We begin by describing a computing model typically used in cloud computing for compute- and data-intensive applications in Section 2. We then introduce the Shadow Replication framework in Section 3. Section 4, 5, and 6 present our analytical models and optimization problem formalization, followed by experiments and evaluation in section 7. Section 8 briefly surveys related work. Section 9 concludes this work.

2 CLOUD WORKLOAD CHARACTERIZATION

Cloud computing workload ranges from business applications and intelligence, to analytics and social networks mining and log analysis, to scientific applications in various fields of sciences and discovery. These applications exhibit different behaviors, in term of computation requirements and data access patterns. While some applications are compute-intensive, others involve the processing of increasingly large amounts of data. The scope and scale of these applications are such that an instance of a job running one of these applications requires the sequential execution of multiple computing phases; each phase consists of thousands, if not millions, of tasks scheduled to execute in parallel and involves the processing of a very large amount of data (Lin and Dyer, 2010; Ferdman and et. al., 2012). This model is directly reflective of the *MapReduce* computational model, which is predominately used in Cloud Com-

puting (Sangroya et al., 2012). An instance of this model, is depicted in Figure 1.

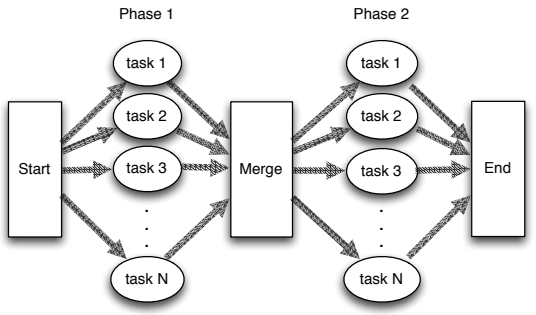


Figure 1: Cloud computing execution model with 2 phases.

Each job has a targeted response time defined by the terms of the SLA. Further, the SLA defines the amount to be paid for completing the job by the targeted response time as well as the penalty to be incurred if the targeted response time is not met.

Each task is mapped to one compute core and executes at a speed, σ. The partition of the job among tasks is such that each task processes a similar workload, W. Consequently, baring failures, tasks are expected to complete at about the same time. Therefore, the minimal response time of each task, when no failure occurs, is $t_{min} = \frac{W}{\sigma_{max}}$, where σ_{max} is the maximum speed. This is also the minimal response time of the entire phase.

As the number of tasks increases, however, the likelihood of a task failure during an execution of a given phase increases accordingly. This underscores the importance of an energy-efficient fault-tolerance model to mitigate the impact of a failing task on the overall delay of the execution phase. The following section describes Shadow Replication, a fault-tolerant, energy-aware computational model to achieve profit-maximizing, energy-efficient resiliency in cloud computing.

3 SHADOW REPLICATION

The basic tenet of Shadow Replication is to associate with each main process a suite of "shadows" whose size depends on the "criticality" of the application and its performance requirements, as defined by the SLA.

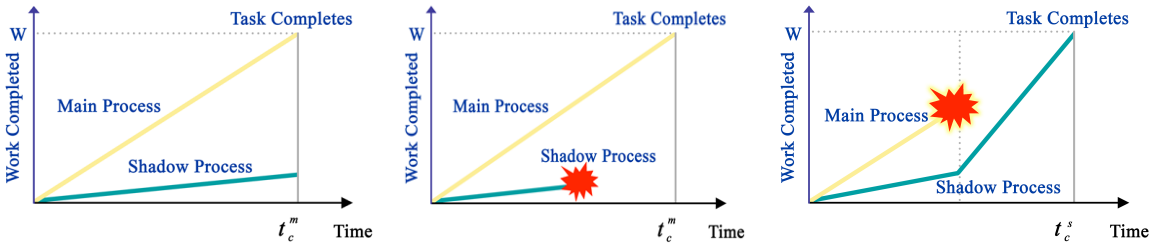
Formally, we define the Shadow Replication fault-tolerance model as follows:

- A main process, $P_m(W, \sigma_m)$, whose responsibility is to executes a task of size W at a speed of σ_m;
- A suite of shadow processes, $P_s(W, \sigma_b^s, \sigma_a^s)$ $(1 \leq s \leq \mathcal{S})$, where $\mathcal{S}$ is the size of the suite. The shadows execute on separate computing nodes. Each shadow process is associated with two execution speeds. All shadows start execution simultaneously with the main process at speed σ_b^s $(1 \leq s \leq \mathcal{S})$. Upon failure of the main process, all shadows switch their executions to σ_a^s, with one shadow being designated as the new main process. This process continues until completion of the task.

To illustrate the behavior of Shadow Replication, we limit the number of shadows to a single process and consider the scenarios depicted in Figure 2, assuming a single process failure. Figure 2(a) represents the case when neither the main nor the shadow fails. The main process, executing at a higher speed, completes the task at time t_c^m. At this time, the shadow process, progressing at a lower speed, stops execution immediately. Figure 2(b) represents the case when the shadow fails. This failure, however, has no impact on the progress of the main process, which still completes the task at t_c^m. Figure 2(c) depicts the case when the main process fails while the shadow is in progress. After detecting the failure of the main process, the shadow begins execution at a higher speed, completing the task at time t_c^s. When possible, the shadow execution speed upon failure must be set so that t_c^s does not exceed t_c^m. Given that the failure rate of an individual node is much lower than the aggregate system failure, it is very likely that the main process will always complete its execution successfully, thereby achieving fault tolerance at a significantly reduced cost of energy consumed by the shadow.

A closer look at the model reveals that shadow replication is a generalization of traditional fault tolerance techniques, namely re-execution and traditional replication. If the SLA specification allows for flexible completion time, shadow replication would take advantage of the delay laxity to trade time redundancy for energy savings. It is clear, therefore, that for a large response time, Shadow Replication converges to re-execution, as the shadow remains idle during the execution of the main process and only starts execution upon failure. If the target response time is stringent, however, Shadow Replication converges to pure replication, as the shadow must execute simultaneously with the main at the same speed. The flexibility of the Shadow Replication model provides the basis for the design of a fault tolerance strategy that strikes a balance between task completion time and energy saving, thereby maximizing profit.

Given that the probability of two individual nodes executing the same instances of a task fail at the same



(a) No Failure (b) Shadow Process Failure (c) Main Process Failure

Figure 2: Shadow replication for a single task and single replica.

time is low, we will focus on the study of Shadow Replication model with a single shadow. It is clear, however, that the model can be extended to support multiple processes, as required by the application's fault-tolerance requirement. Furthermore, we adopt the fail-stop fault model, where a processor stops execution once a fault occurs and failure can be detected by other processes(Gärtner, 1999; Cristian, 1991).

4 REWARD BASED OPTIMAL SHADOW REPLICATION

In this section, we describe a profit-based optimization framework for the cloud-computing execution model previous described. Using this framework we compute profit-optimized execution speeds by optimizing the following objective function:

$$\begin{aligned} &\max_{\sigma_m,\sigma_b,\sigma_a} E[profit] \\ &s.t. 0 \leq \sigma_m \leq \sigma_{max} \\ &0 \leq \sigma_b \leq \sigma_m \\ &0 \leq \sigma_a \leq \sigma_{max} \end{aligned} \tag{1}$$

We assume that processor speeds are continuous and use nonlinear optimization techniques to solve the above optimization problem.

In order to earn profit, service providers must either increase income or decrease expenditure. We take both factors into consideration for the purpose of maximizing profit while meeting customer's requirements. In our model, we set the expected profit to be expected income minus expected expense.

$$E[\text{profit}] = E[\text{income}] - E[\text{expense}] \tag{2}$$

4.1 Reward Model

The cloud computing SLA can be diverse and complex. To focus on the profit and reliability aspects of the SLA, we define the reward model based on job completion time. Platform as a Service (PaaS) companies will continue to become more popular causing an increase in SLAs using job completion time as their performance metric. We are already seeing this appear in web-based remote procedure calls and data analytic requests.

As depicted in Figure 3, customers expect that their job deployed on cloud finishes by a mean response time t_{R_1}. As a return, the provider earns a certain amount of reward, denoted by R, for satisfying customer's requirements. However, if the job cannot be completed by the expected response time, the provider loses a fraction of R proportional to the delay incurred. For large delay, the profit loss may translate into a penalty that the CSP must pay to the customer. In this model, the maximum penalty P is paid if the delay reaches or exceeds t_{R_2}. The four parameters, R, P, t_{R_1} and t_{R_2}, completely define the reward model.

There are two facts that the service provider must take into account when negotiating the terms of the SLA. The first is the response time of the main process assuming no failure (Figure 2(a) and Figure 2(b)). This results in the following completion time:

$$t_c^m = W/\sigma_m \tag{3}$$

If the main process fails (shown in Figure 2(c)), the task completion time by shadow process is the time of the failure, t_f, plus the time necessary to complete the remaining work.

$$t_c^s = t_f + \frac{W - t_f \times \sigma_b}{\sigma_a} \tag{4}$$

This reward model is flexible and extensible; it is not restricted to the form shown in Figure 3. In particular, the decrease may be linear, concave, or convex and the penalty can extend to infinity. This model can further be extended to take into consideration both the short-term income and long-term reputation of the service provider (Daw and Touretzky, 2002).

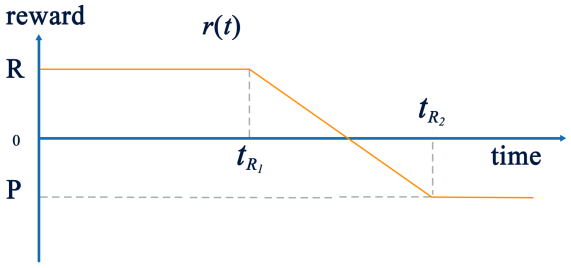


Figure 3: A reward function.

4.2 Failure Model

Failure can occur at any point during the execution of the main or shadow process. Our assumption is that at most one failure occurs, therefore if the main process fails it is implied that the shadow will complete the task without failure. We can make this assumption because we know the failure of any one node is rare thus the failure of any two specific nodes is very unlikely.

We assume that two probability density functions, $f_m(t_f)$ and $f_s(t_f)$, exist which express the probabilities of the main and shadow process failing at time t_f separately. The model does not assume a specific distribution. However, in the remainder of this paper we use an exponential probability density function, $f_m(t_f) = f_s(t_f) = \lambda e^{-\lambda t_f}$, of which the mean time between failure (MTBF) is $\frac{1}{\lambda}$.

4.3 Power and Energy Models

Dynamic voltage and frequency scaling (DVFS) has been widely exploited as a technique to reduce CPU dynamic power (Flautner et al., 2002; Pillai and Shin, 2001). It is well known that one can reduce the dynamic CPU power consumption at least quadratically by reducing the execution speed linearly. The dynamic CPU power consumption of a computing node executing at speed σ is given by the function $p_d(\sigma) = \sigma^n$ where $n \geq 2$.

In addition to the dynamic power, CPU leakage and other components (memory, disk, network etc.) all contribute to static power consumption, which is independent of the CPU speed. In this paper we define static power as a fixed fraction of the node power consumed when executing at maximum speed, referred to as ρ. Hence node power consumption is expressed as $p(\sigma) = \rho \times \sigma_{max}^n + (1-\rho) \times \sigma^n$. When the execution speed is zero the machine is in a sleep state, powered off or not assigned as a resource; therefore it will not be consuming any power, static or dynamic. Throughout this paper we assume that dynamic power is cubic in relation to speed (Rusu et al., 2003; Zhai et al., 2004), therefore the overall system power when executing at speed σ is defined as:

$$p(\sigma) = \begin{cases} \rho\sigma_{max}^3 + (1-\rho)\sigma^3 & \text{if } \sigma > 0 \\ 0 & \text{if } \sigma = 0 \end{cases} \quad (5)$$

Using the power model given by Equation 5, the energy consumed by a process executing at speed σ during an interval T is given by

$$E(\sigma, T) = p(\sigma) \times T \quad (6)$$

Corresponding to Figure 2, there are three failure cases to consider: main and shadow both succeed, shadow fails and main fails. As described earlier, the case of both the main and shadow failing is very rare and will be ignored. The expected energy consumption for a single task is then the weighted average of the expected energy consumption in the three cases.

First consider the case where no failure occurs and the main process successfully completes the task at time t_c^m, corresponding to Figure 2(a).

$$E_1 = (1 - \int_0^{t_c^m} f_m(t)dt) \times (1 - \int_0^{t_c^m} f_s(t)dt) \times (E(\sigma_m, t_c^m) + E(\sigma_b, t_c^m)) \quad (7)$$

The first line is the probability of fault-free execution of the main process and shadow process. Then we multiple this probablity by the energy consumed by the main and the shadow process during this fault free execution, ending at t_c^m.

Next, consider the case where the shadow process fails at some point before the main process successfully completes the task, corresponding to Figure 2(b).

$$E_2 = (1 - \int_0^{t_c^m} f_m(t)dt) \times \int_0^{t_c^m} (E(\sigma_m, t_c^m) + E(\sigma_b, t)) \times f_s(t)dt \quad (8)$$

The first factor is the probability that the main process does not fail, and the probability of shadow fails is included in the second factor which also contains the energy consumption since it depends on the shadow failure time. Energy consumption comes from the main process until the completion of the task, and the shadow process before its failure.

The one remaining case to consider is when the main process fails and the shadow process must continue to process until the task completes, corresponding to Figure 2(c).

$$E_3 = (1 - \int_0^{t_c^m} f_s(t)dt) \times \int_0^{t_c^m} (E(\sigma_m, t) + E(\sigma_b, t) + E(\sigma_a, t_c^s - t)) f_m(t)dt \quad (9)$$

Similarly, the first factor expresses the probability that the shadow process does not fail. In this case,

the shadow process executes from the beginning to t_c^s when it completes the task. However, under our "at most one failure" assumption, the period during which shadow process may fail ends at t_c^m, since the only reason why shadow process is still in execution after t_c^m is that main process has already failed. There are three parts of energy consumption, including that of main process before main's failure, that of shadow process before main's failure, and that of shadow process after main's failure, all of which depend on the failure occurrence time.

The three equations above describe the expected energy consumption by a pair of main and shadow processes for completing a task under different situations. However, under our system model it might be the case that those processes that finish early will wait idly and consume static power if failure delays one task. If it is the case that processes must wait for all tasks to complete, then this energy needs to be accounted for in our model. The probability of this is the probability that at least one main process fails, referred to as the system level failure probability.

$$P_f = 1 - (1 - \int_0^{t_c^m} f_m(t)dt)^N \tag{10}$$

Hence, we have the fourth equation corresponding to the energy consumed while waiting in idle.

$$\begin{aligned} E_4 =& (1 - \int_0^{t_c^m} f_m(t)dt) \times (1 - \int_0^{t_c^m} f_s(t)dt) \times \\ & 2P_f \times E(0, t_c^j - t_c^m) + \int_0^{t_c^m} f_s(t)dt \times \\ & (1 - \int_0^{t_c^m} f_m(t)dt) \times P_f \times E(0, t_c^j - t_c^m) \end{aligned} \tag{11}$$

Corresponding to the first case, neither main process nor shadow process fails, but both of them have to wait in idle from task completion time t_c^m to the last task's completion (by a shadow process) with probability P_f. Under the second case, only the main process has to wait if some other task is delayed since its shadow process has already failed. These two aspects are accounted in the first and last two lines in E_4 separately. We use the expected shadow completion time t_c^j as an approximation of the latest task completion time which is also the job completion time.

By summing these four parts and then multiplying it by N we will have the expected energy consumed by Shadow Replication for completing a job of N tasks.

$$E[\text{energy}] = N \times (E_1 + E_2 + E_3 + E_4) \tag{12}$$

4.4 Income and Expense Models

The income is the reward paid by customer for the cloud computing services that they utilize. It depends on the reward function $r(t)$, depicted in Figure 3, and the actual job completion time. Therefore, the income should be either $r(t_c^m)$, if all main processes can complete without failure, or $r^*(t_c^s)$ otherwise. It is worth noting that the reward in case of failure should be calculated based on the last completed task, which we approximate by calculating the expected time of completion allowing us to derive the expected reward, i.e. $r^*(t_c^s) = \frac{\int_0^{t_c^m} r(t_c^s) \times f_m(t)dt}{\int_0^{t_c^m} f_m(t)dt}$. Therefore the income is estimated by the following equation.

$$E[\text{income}] = (1 - P_f) \times r(t_c^m) + P_f \times r^*(t_c^s) \tag{13}$$

The first part is the reward earned by the main process times the probability that all main processes would complete tasks without failure. If at least one main process fails, that task would have to be completed by a shadow process. As a result, the second part is the reward earned by shadow process times the system level failure probability.

If C is the charge expressed as dollars per unit of energy consumption (e.g. kilowatt hour), then the expected expenditure would be C times the expected energy consumption for all N tasks:

$$E[\text{expense}] = C \times E[\text{energy}] \tag{14}$$

However, the expenditure of running the cloud computing service is more than just energy, and must includes hardware, maintenance, and human labor. These costs can be accounted for by amortizing these costs into the static power factor, ρ. Because previous studies have suggested (Elnozahy et al., 2003; Raghavendra et al., 2008) that energy will become a dominate factor we decided to focus on this challenge and leave other aspects to future work.

Table 1: Symbols used in our analytical model.

Symbols	Definition
W	Task size
N	Number of tasks
$r(t)$	Reward function
R, P	Maximum reward and penalty
t_{R_1}, t_{R_2}	Response time thresholds
C	Unit price of energy
ρ	Static power ratio
t_c^m, t_c^s, t_c^j	Completion time of main process, shadow process, and the whole job
$f_m()$, $f_s()$	Failure density function of main and shadow
λ	Failure rate
P_f	System level failure probability
σ_m, σ_b, σ_a	Speeds of main, shadow before and after failure (Optimization Outputs)

Based on the above formalization of the optimization problem, the MATLAB Optimization Toolbox (MathWorks, 2013) was used to solve the resulting nonlinear optimization problem. The parameters of this problem are listed in Table 1.

5 PROFIT-AWARE STRETCHED REPLICATION

We compare Shadow Replication to two other replication techniques, traditional replication and profit-aware stretched replication. Traditional replication requires that the two processes always execute at the same speed σ_{max}. Unlike traditional replication Shadow Replication is dependent upon failure detection, enabling the replica to increase its execution speed upon failure and maintain the targeted response time thus maximizing profit. While this is the case in many computing environments, there are cases where failure detection may not be possible. To address this limitation, we propose profit-aware stretched replication, whereby both the main process and the shadow execute independently at stretched speeds to meet the expected response time, without the need for the main processes failure detection. In profit-aware stretched replication both the main and shadow execute at speed σ_r, found by optimizing the profit model. For both traditional replication and stretched replication, the task completion time is independent of failure and can be directly calculated as:

$$t_c = \frac{W}{\sigma_{max}} \text{ or } t_c = \frac{W}{\sigma_r} \tag{15}$$

Since all tasks will have the same completion time, the job completion time would also be t_c. Further, the expected income, which depends on negotiated reward function and job completion time, is independent of failure:

$$E[income] = r(t_c) \tag{16}$$

Since both traditional replication and profit-aware stretched replication are special cases of our Shadow Replication paradigm where $\sigma_m = \sigma_b = \sigma_a = \sigma_{max}$ or $\sigma_m = \sigma_b = \sigma_a = \sigma_r$ respectively, we can easily derive the expected energy consumption using Equation 12 with E_4 fixed at 0 and then compute the expected expense using Equation 14.

6 RE-EXECUTION

Contrary to replication, re-execution initially assigns a single process for the execution of a task. If the original task fails, the process is re-executed. In the cloud computing execution framework this is equivalent to a checkpoint/restart, the checkpoint is implicitly taken at the end of each phase and because the tasks are loosely coupled they can restart independently.

Based on the one failure assumption, two cases must be considered to calculate the task completion time. If no failure occurs, the task completion time is:

$$t_c = \frac{W}{\sigma_{max}} \tag{17}$$

In case of failure, however, the completion time is equal to the sum of the time elapsed until failure and the time needed for re-execution. Again, we use the expected value $t_f^* = \frac{\int_0^{t_c} t \times f_m(t)dt}{\int_0^{t_c} f_m(t)dt}$ to approximate the time that successfully completed processes have to spend waiting for the last one.

Similar to Shadow Replication, the income for re-execution is the weighted average of the two cases:

$$E[\text{income}] = (1 - P_f) \times r(t_c) + P_f \times r(t_c + t_f^*) \tag{18}$$

For one task, if no failure occurs then the expected energy consumption can be calculated as

$$E_5 = (1 - \int_0^{t_c} f_m(t)dt) \times (E(\sigma_{max}, t_c) + P_f \times E(0, t_f^*)) \tag{19}$$

If failure occurs, however, the expected energy consumption can be calculated as

$$E_6 = \int_0^{t_c} (E(\sigma_{max}, t) + E(\sigma_{max}, t_c)) \times f_m(t)dt \tag{20}$$

Therefore, the expected energy consumption by re-execution for completing a job of N tasks is

$$E[energy] = N \times (E_5 + E_6) \tag{21}$$

7 EVALUATION

This section evaluates the expected profit of each of the fault tolerance methods discussed above under different system environment. We have identified 5 important parameters which affect the expected profit:

- Static power ratio ρ, which determines the portion of power that is unaffected by the execution speed.
- SLA - The amount of reward, penalty and the required response times.
- N - The total number of tasks.
- MTBF - The reliability of an individual node.
- Workload - The size, W, of each individual task.

Without loss of generality, we normalize σ_{max} to be 1, so that all the speeds can be expressed as a

fraction of maximum speed. Accordingly, the task workload W is also adjusted such that it is equal to the amount of time (in hours) required for a single task, preserving the ratios expressed in Equation 3 and Equation 4. The price of energy is assumed to be 1 unit. We assume that R in our reward model is linearly proportional to the number of tasks N and the maximal reward for one task is 3 units, so the total reward for a job is $3 \times N$ units. However, for the analysis we look at the average of expenditure and income on each task by dividing the total expenditure and income by N. In our basic configuration we assume that the static power ratio is 0.5, the task size is 1 hour, the node MTBF 5 is years, the number of tasks is 100000, and the response time thresholds for maximal and minimal rewards are 1.3 hours and 2.6 hours respectively. Since the maximal power consumption is 1 unit, the energy needed for the task with one process at maximal speed is also 1 unit.

7.1 Sensitivity to Static Power

With various architectures and organizations, servers deployed at different data centers will have different characteristics in terms of power consumption. The static power ratio is used to abstract the amount of static power consumed versus dynamic power.

Table 2: Speeds for different static power ratio. MTBF=5 years, N=100000, W=1 hour, t_{R_1}=1.3 hours, t_{R_2}=2.6 hours.

ρ	σ_m	σ_b	σ_a
0.0	0.77	0.65	1.00
0.1	0.78	0.66	1.00
0.2	0.83	0.66	1.00
0.3	0.84	0.68	1.00
0.4	0.85	0.70	1.00
0.5	0.86	0.72	1.00
0.6	0.87	0.73	1.00
0.7	0.91	0.81	1.00
0.8	1.00	1.00	1.00
0.9	1.00	1.00	1.00
1.0	1.00	1.00	1.00

Table 2 shows how the profit-optimized execution speeds of Shadow Replication will change as static power increases. The execution speeds increase to reduce the execution time as static power ration increases. Observe that σ_a is always equal to σ_{max}, which means that after sensing the failure of the main process, the shadow process should always shift to maximum speed. This is expected because the optimization will reduce the amount of work done by the shadow process before failure resulting in the maximum execution speed after failure, thus minimizing the amount of repeated work.

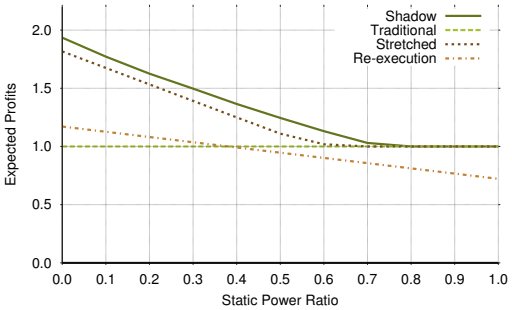


Figure 4: Profit for different static power ratio. MTBF=5 years, N=100000, W=1 hour, t_{R_1}=1.3 hours, t_{R_2}=2.6 hours.

The potential profit gains achievable by using profit-aware replication techniques decreases as static power increases, as is shown in Figure 4. The reason is that our profit-aware techniques rely upon the fact that one can reduce energy costs by adjusting the execution speeds. Modern systems have a static power between 40%-70% and it is reasonable to suspect that this will continue to be the case. Within this target range of static power, Shadow Replication can achieve, on average, 19.3% more profit than traditional replication, 8.9% more than profit-aware stretched replication, and 28.8% more than re-exeuction.

7.2 Sensitivity to Response Time

Response time is critical in the negotiation of SLA as customers always expect their tasks to complete as soon as possible. In this section we show a sensitivity study with respect to task response time. We vary the first threshold t_{R_1} from the minimal response time t_{min} to $1.9t_{min}$, and set the second threshold t_{R_2} to be always $2t_{R_1}$. We do not show results for varying the reward and penalty values of the SLA. The reason is that changing these values have no effect on the choice of fault tolerance methods because they are all affected in a similar way.

Table 3: Speeds for different response time threshold. ρ=0.5, MTBF=5 years, N=100000, W=1 hour.

t_{R_1}	σ_m	σ_b	σ_a
1.0	1.00	1.00	1.00
1.1	0.94	0.88	1.00
1.2	0.89	0.79	1.00
1.3	0.86	0.72	1.00
1.4	1.00	0.00	1.00
1.5	1.00	0.00	1.00
1.6	0.84	0.00	1.00
1.7	0.74	0.00	1.00
1.8	0.64	0.00	1.00
1.9	0.64	0.00	1.00

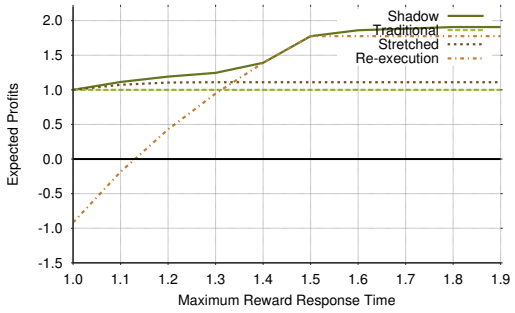


Figure 5: Profit for different response time threshold. ρ=0.5, MTBF=5 years, N=100000, W=1 hour.

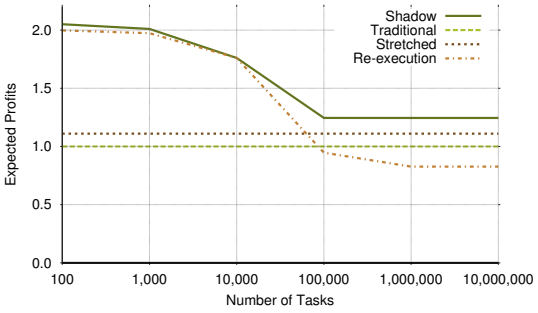


Figure 6: Profit for different number of tasks. ρ=0.5, MTBF=5 years, W=1 hour, t_{R_1}=1.3 hours, t_{R_2}=2.6 hours.

In Table 3 we see that Shadow Replication adapts the execution speeds to take advantage of the available laxity, reducing its speeds as laxity increases. It is clear that Shadow Replication has two different execution strategies separated by $t_{R_1} = 1.4$: when time is critical, it uses both a main and a shadow from the very beginning to guarantee that task can be completed on time; when time is not critical, it mimics re-execution and starts its shadow only after a failure. Also note that as t_{R_1} approaches t_{min}, the speeds of the main process and the shadow process converge, effectively causing Shadow Replication to mimic traditional replication when faced with time-critical jobs.

Figure 5 shows the effect that targeted response time has upon the profitability of each fault tolerance method. Compared to traditional replication, all the other methods increase their profit as the targeted response time increases, this is expected because each of the other techniques can make use of increased laxity in time to increase profit. Re-execution is the most sensitive to the target response time since it fully relies upon time redundancy, showing that it should only be used when the targeted response time is *not* stringent. Again, Shadow Replication always achieves more profit than traditional replication and profit-aware stretched replication, and the profit gains are 52.8% and 39.0% on average.

7.3 Sensitivity to Number of Tasks

Table 4: Speeds for different number of tasks. ρ=0.5, MTBF=5 years, W=1 hour, t_{R_1}=1.3 hours, t_{R_2}=2.6 hours.

N	σ_m	σ_b	σ_a
100	0.80	0.00	1
1000	0.84	0.00	1
10000	1.00	0.00	1
100000	0.86	0.72	1
1000000	0.86	0.72	1
10000000	0.86	0.72	1

The number of tasks has a direct influence upon the system level failure probability because as the number of tasks increase the probability that failure will occur to at least one task increases. Recall that even one failure can hurt the total income significantly, and keep the other processes waiting. Thus, shadow replication will adjust its execution speeds to reduce the waiting time.

Table 4 is similar to Table 3 in that there are also two execution strategies. When there are few parallel tasks, shadow replication chooses to execute the main processes at nearly full speed and keeps the shadow processes dormant. The reason is that it is very likely that all main processes can finish their tasks successfully, and the need for redundancy is thus less significant. The other case is when there is a huge number of tasks to execute, the shadow process would keep running at a slower speed than the main to protect the main as well as save energy. Since the system level failure probability is already 0.9 when N is 100000, the speeds stay the same when $N \geq 100000$.

Figure 6 confirms that for small number of tasks re-execution is more profitable than replication. However, re-execution is not scalable as its profit decreases rapidly after N reaches 10000. At the same time, traditional replication and profit-aware stretched replication are not affected by the number of tasks because neither are affected by the system level failure rate. On average, Shadow Replication achieves 43.5%, 59.3%, and 18.4% more profits than profit-aware stretched replication, traditional replication and re-exeuction, respectively.

7.4 Sensitivity to Failure Vulnerability

The ratio between task size and node MTBF represents the tasks vulnerability to failure, specifically it is an approximation of the probability that failure occurs during the execution of the task. In our analysis we found that increasing task size will have the same

effect as reducing node MTBF. Therefore, we analyze these together using the vulnerability to failure, allowing us to analyze a wider range of system parameters.

Table 5: Speeds for different task size over MTBF. ρ=0.5, N=100000, t_{R_1}=1.3 hours, t_{R_2}=2.6 hours.

$W/MTBF$	σ_m	σ_b	σ_a
2E-10	0.79	0.00	1.00
2E-09	0.79	0.00	1.00
2E-08	0.80	0.00	1.00
2E-07	0.84	0.00	1.00
2E-06	1.00	0.00	1.00
2E-05	0.86	0.72	1.00
2E-04	0.86	0.72	1.00
2E-03	0.86	0.72	1.00

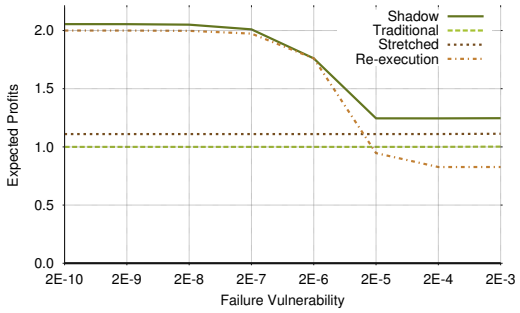


Figure 7: Profit for different task size over MTBF. ρ=0.5, N=100000, t_{R_1}=1.3 hours, t_{R_2}=2.6 hours.

As seen in Table 5 when the vulnerability to failure is low the execution speeds for the shadow process is such that no work is done before failure. However, as the vulnerability increases, the shadow process performs more work before failure. This is analogous to what we observed as we increased the number of tasks (Table 4). As expected re-execution is desired when the vulnerability to failure is low. As always, Shadow Replication can adjust its execution strategy to maximize the profits, as shown in Figure 7.

7.5 Application Comparison

To compare the potential benefit of "Shadow Replication" we evaluate the expected profit of each resilience technique using three different benchmark applications representing a wide range of application (Sangroya et al., 2012): Business Intelligence, Bioinformatics and Recommendation System. The business intelligence benchmark application is a decision support system for a wholesale supplier. It emphasizes executing business-oriented ad-hoc queries using Apache Hive. The bioinformatics application performs DNA sequencing, allowing genome analysis on a wide range of organisms. The recommendation system is similar to those typically found in e-commerce sites which, based upon browsing habits and history, recommends similar products.

Table 6: Cloud Applications (Sangroya et al., 2012).

Application	Processing Rate
Business Intelligence	3.3 (MB/s)
Bioinformatics	6.6 (MB/s)
Recommendation System	13.2 (MB/s)

Using the results of the experiments reported in (Sangroya et al., 2012), we derived the time required to process data for each application type (Table 6). We assume that these processing rates per task will not change when scaling the applications to future cloud environments. This is a reasonable assumption given that map-reduce tasks are loosely coupled and data are widely distributed, therefore data and task workload will scale linearly.

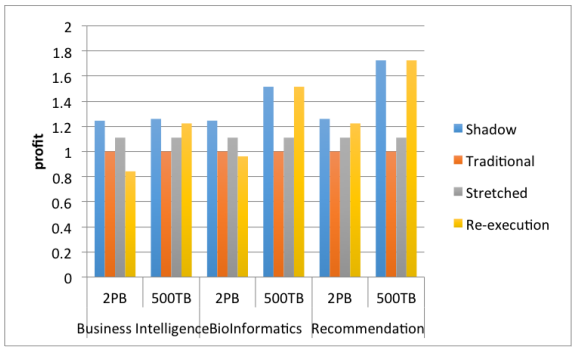


Figure 8: Application comparison. ρ=0.5, N=500000, t_{R_1}=1.3t_{min}, t_{R_2}=2.6t_{min}.

In Figure 8 we compare the expected profit for each application using each of the 4 resilience techniques. We consider two data sizes expected in future cloud computing environments, 500TB and 2PB. The figure shows that for business intelligence applications, Shadow Replication achieves significantly larger profits for both data sizes. This is because business intelligence applications tend to be IO intensive resulting in longer running tasks. Whereas recommendation systems tend to require little data IO resulting in shorter running tasks making re-execution as good as Shadow Replication. Bioinformatics tends to be in between these two applications resulting in shadow computing performing better when processing large datasets (2 PB) but not outstanding on smaller datasets (500 TB). The take away from this evaluation is that for the shown system parameters if phase execution is short, then re-execution performs as well as Shadow Replication. Alternatively, if a phase is long (20 minutes or greater), then Shadow

Replication can be as much as 47.9% more profitable than re-execution. The previous sensitivity analysis can be used to extrapolate expected profit for different system parameters.

8 RELATED WORK

The increase of failures in large-scale systems brought to the forefront the need for new fault-tolerance techniques. Coordinated checkpointing, with roll-back recovery, has been the dominant fault-tolerance method in high performance computing (HPC) (Agarwal et al., 2004; Alvisi et al., 1999; Daly, 2006; Helary et al., 1997). Based on this method, the execution state of each process is periodically saved to a stable storage. In case of failure, computation is rolled-back to the last error-free, consistent state recorded by all processes. As the scale of the system increases, the viability of coordinated checkpointing has become questionable. Despite numerous improvements of the basic coordinated checkpointing scheme, recent studies show that high failure rates, coupled with the checkpointing overhead, limit the feasibility of centralized, coordinated checkpointing (El Mehdi Diouri et al., 2012). Re-execution and state machine replication have emerged as viable schemes to deal with failures in large-scale systems.

Re-execution waits until a failure occurs and re-executes the failed process. The major shortcoming of this method stems from the large delay the completion of a job incurs. To avoid such a delay, state machine replication executes simultaneously one or more replicas of each process on different computing nodes. a (Ferreira and et. al., 2011; Sousa et al., 2005). Although it enhances fault tolerance without incurring excessive delay, state machine replication increases the computing resources needed to complete a job reliably.

Efforts have been devoted to increase the resiliency of cloud computing. Several of the proposed schemes aim at enhancing existing fault-tolerance approaches. In (Jhawar et al., 2013) the authors propose a high-level scheme that allows users to specify the desired level of resilience, while hiding implementation details. Similarly, (Zhao et al., 2010) take a middleware-based approach to fault-tolerance and propose a method that maintains strong replica consistency, while achieving transparency and low end-to-end latency. Authors in (Tchana et al., 2012) propose a collaborative solution to enhance fault-tolerance efficiency. In (Nicolae and Cappello, 2011), the authors leverage virtual disk image snapshots to minimize the storage space and checkpointing overhead. In (Zheng, 2010) investigates how redundant copies can be provisioned for tasks to improve MapReduce fault tolerance and reduce latency. Most of these schemes do not consider the impact of energy on the system.

As the significance of power and energy consumption in large datacenters increases, energy management becomes critical (Chen et al., 2012; Lin et al., 2011). Schemes are proposed to optimize power consumption by either shutting down servers, or using CPU DVFS. Our work takes a different approach to fault-tolerance, and proposes a new computational model, referred to as Shadow Replication, to achieve high-levels of resiliency, while minimizing energy consumption. In this work, we combine DVFS with traditional replication to achieve fault tolerance and maximize profit, while meeting users' SLA requirement.

9 CONCLUSION

The main motivation of this work stems from the observation that, as systems become larger and more complex, the rate of failures is highly-likely to increase significantly. Hence, understanding the interplay between fault-tolerance, energy consumption and profit maximization is critical for the viability of Cloud Computing to support future large-scale systems. To this end, we propose Shadow Replication as a novel energy-aware, reward-based computational model to achieve fault-tolerance and maximize the profit. What differentiates Shadow Replication from other methods is its ability to explore a parameterized tradeoff between hardware and time redundancy to achieve fault-tolerance, with minimum energy, while meeting SLA requirements.

To assess the performance of the proposed fault-tolerance computational model, an extensive performance evaluation study is carried out. In this study, system properties that affect the profitability of fault tolerance methods, namely failure rate, targeted response time and static power, are identified. The failure rate is affected by the number of tasks and vulnerability of the task to failure. The targeted response time represents the clients' desired job completion time, as expressed by the terms of the SLA. Our performance evaluation shows that in all cases, Shadow Replication outperforms existing fault tolerance methods. Furthermore, shadow replication will converge to traditional replication when target response time is stringent, and to re-execution when target response time is relaxed or failure is unlikely. Furthermore, the study reveals that the system static

power plays a critical role in the tradeoff between the desired level of fault-tolerance, profit maximization and energy consumption. This stems from the reliance of Shadow Replication upon DVFS to reduce energy costs. If the static power is high, slowing down process execution does not lead to a significant reduction in the total energy needed to complete the task.

REFERENCES

Agarwal, S., Garg, R., and Gupta, M. S. (2004). Adaptive incremental checkpointing for massively parallel systems. In *ICS: Proc. of the* 18th *annual Int. Conf. on Supercomputing*, pages 277–286. ACM Press.

Alvisi, L., Elnozahy, E., Rao, S., Husain, S., and de Mel, A. (1999). An analysis of communication induced checkpointing. In *Fault-Tolerant Computing.* 29th *Annual Int. Symp. on*, pages 242–249.

Amazon (2013). Amazon elastic compute cloud. http://aws.amazon.com/ec2/.

Chen, X., Liu, X., Wang, S., and Chang, X.-W. (2012). Tailcon: Power-minimizing tail percentile control of response time in server clusters. In *Reliable Distributed Systems (SRDS), IEEE* 31st *Symp. on*, pages 61–70.

Cristian, F. (1991). Understanding fault-tolerant distributed systems. *Commun. ACM*, 34(2):56–78.

Daly, J. (2006). A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Computer Systems*, 22(3):303 – 312.

Daw, N. D. and Touretzky, D. S. (2002). Long-term reward prediction in td models of the dopamine system. *Neural Comput.*, 14(11):2567–2583.

El Mehdi Diouri, M., Gluck, O., Lefevre, L., and Cappello, F. (2012). Energy considerations in checkpointing and fault tolerance protocols. In *Dependable Systems and Networks Workshops (DSN-W)*, pages 1–6.

Elnozahy, M., Kistler, M., and Rajamony, R. (2003). Energy conservation policies for web servers. In *Proc. of the* 4th *USENIX Symp. on Internet Tech. and Sys.*

Ferdman, M. and et. al. (2012). Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In *Proc. of the* 17th *Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, pages 37–48, New York, NY, USA. ACM.

Ferreira, K. and et. al. (2011). Evaluating the viability of process replication reliability for exascale systems. In *Proc. of Int. Conf. for HPC, Networking, Storage and Analysis*, SC, pages 44:1–44:12, New York.

Flautner, K., Reinhardt, S., and Mudge, T. (2002). Automatic performance setting for dynamic voltage scaling. *Wirel. Netw.*, 8(5):507–520.

Gärtner, F. C. (1999). Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Comput. Surv.*, 31(1):1–26.

Helary, J.-M., Mostefaoui, A., Netzer, R., and Raynal, M. (1997). Preventing useless checkpoints in distributed computations. In *Reliable Distributed Systems. Proc., The* 16th *Symp. on*, pages 183–190.

Jhawar, R., Piuri, V., and Santambrogio, M. (2013). Fault tolerance management in cloud computing: A system-level perspective. *Systems Journal*, 7(2):288–297.

Ko, S. Y., Hoque, I., Cho, B., and Gupta, I. (2010). Making cloud intermediate data fault-tolerant. In *Proc. of the* 1st *ACM Symp. on CC*, pages 181–192. ACM.

Lin, J. and Dyer, C. (2010). Data-intensive text processing with mapreduce. *Synthesis Lectures on Human Language Technologies*, 3(1):1–177.

Lin, M., Wierman, A., Andrew, L., and Thereska, E. (2011). Dynamic right-sizing for power-proportional data centers. In *INFOCOM, Proc.*, pages 1098–1106.

MathWorks (2013). Matlab: Optimization toolbox.

Nicolae, B. and Cappello, F. (2011). Blobcr: efficient checkpoint-restart for hpc applications on iaas clouds using virtual disk image snapshots. In *Proc. of Int. Conf. for HPC, Networking, Storage and Analysis*, pages 34:1–34:12.

Pillai, P. and Shin, K. G. (2001). Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proc. of the* 18th *ACM Symp. on Operating systems principles*, SOSP, pages 89–102.

Raghavendra, R., Ranganathan, P., Talwar, V., Wang, Z., and Zhu, X. (2008). No "power" struggles: coordinated multi-level power management for the data center. *SIGARCH Comput. Archit. News*, 36(1):48–59.

Rusu, C., Melhem, R., and Mossé, D. (2003). Maximizing rewards for real-time applications with energy constraints. *ACM Trans. Emb. Comp. Syst.*, 2(4):537–559.

Sangroya, A., Serrano, D., and Bouchenak, S. (2012). Benchmarking dependability of mapreduce systems. In *Reliable Distributed Systems (SRDS), IEEE 31st Symp. on*, pages 21–30.

Schroeder, B. and Gibson, G. (2010). A large-scale study of failures in high-performance computing systems. *Depend. and Sec. Comp., IEEE Tran. on*, 7(4):337–350.

Sousa, P., Neves, N., and Verissimo, P. (2005). Resilient state machine replication. In *Dependable Computing. Proc.* 11th *Pacific Rim Int. Symp. on*, pages 305–309.

Tchana, A., Broto, L., and Hagimont, D. (2012). Approaches to cloud computing fault tolerance. In *Comp., Infor. & Tele. Sys., Int. Conf. on*, pages 1–6.

Tsai, W.-T., Zhong, P., Elston, J., Bai, X., and Chen, Y. (2011). Service replication strategies with mapreduce in clouds. In *Autonomous Decentralized Systems,* 10th *Int. Symp. on*, pages 381–388.

Zhai, B., Blaauw, D., Sylvester, D., and Flautner, K. (2004). Theoretical and practical limits of dynamic voltage scaling. In *Design Automation Conf. Proc.* 41st, pages 868–873.

Zhao, W., Melliar-Smith, P., and Moser, L. (2010). Fault tolerance middleware for cloud computing. In *Cloud Computing, IEEE* 3rd *Int. Conf. on*, pages 67–74.

Zheng, Q. (2010). Improving mapreduce fault tolerance in the cloud. In *Parallel Distributed Processing, Workshops and Phd Forum, IEEE Int. Symp. on*, pages 1–6.

A SWRL Bridge to XACML for Clouds Privacy Compliant Policies

Hanene Boussi Rahmouni[1,3], Marco Casassa Mont[2], Kamran Munir[1] and Tony Solomonides[1,4]
[1]*Department of Computer Science and Creative Technologies, Faculty of Environment and Technology, University of the West of England, Coldharbour Lane, Bristol, BS16 1QY, U.K.*
[2]*Hewlett-Packard Labs, Cloud & Security Lab, Bristol, U.K.*
[3]*Higher School of Communication, University of Carthage, Tunis, Tunisia*
[4]*Center for Clinical and Research Informatics, North Shore University Health System, Illinois, U.S.A.*
{hanene2.rahmouni, kamran2.munir}@uwe.ac.uk, marco-cassa-mont@hp.com, asolomonides@northshore.org

Keywords: Privacy Policies, OWL, SWRL, XACML, Cloud.

Abstract: The management of privacy and personal information within multi-cultural domain such as clouds and other universal collaborative systems requires intrinsic compliance-checking and assurance modules in order to increase social trust and acceptance. Focusing mainly on medical domains, this issue is particularly important due to the sensitivity of health related data in international data protection law. The use of ontologies and semantic technologies can provide relatively easy interpretation of legislation at run time, and can allow the logging of data access events to serve for future audits. However, the enforcement of semantic web rules (SWRL rules) on complex and heterogeneous architectures is expensive and might present runtime overheads. We believe a mapping of our semantic web privacy policies to a standard access control language such as XACML would be a useful alternative. A translation to XACML, would allow the integration of these policies with existing security and privacy policies being adopted on clouds environments. This paper describes a mathematical formalism for mapping SWRL (Semantic Web Rule Language) privacy rules to XACML policies and also explains the underline implementation requirements of this formalism.

1 INTRODUCTION

The protection of patients' privacy in a pan-European cloud infrastructure is challenging and requires combined solutions from legislation, organisational and social frameworks. In this regards, a European public cloud infrastructure is still a challenging goal to attend (Brandic et al., 2010), but necessary for those nations wishing to collaborate for the advancement of medical research and public health. This challenge arises primarily due to the lack of harmonisation in legal frameworks governing privacy and data protection in Europe, not least the European Data Protection directive 95/46/EC (EC Directive, 1995), (McCullagh, 2006), (Beyleveld et al., 2004). For example, consent is not handled in the same way in Italy as in the UK. In Italy, consent could be provided for a broad purpose of data processing; whereas in the UK, obtaining a specific consent is a legal obligation (Iversen et al., 2006), (Italian Personal Data Protection Code, 2003). On top of this, there are significant conceptual and technical issues in-particular when expressing, interpreting and deriving operational consequences out of high-level policies. Finally, despite the attention that has been paid to security concerns for public and private clouds; such as infrastructure integrity and access control (typically authentication and authorization), this does not naturally extend to cover privacy concerns (often requiring context and purpose specification). Although they are newly emerging paradigms, clouds are very similar in many aspects to other distributed computing environments. Particularly, clouds are similar to large-scale systems that are based on virtualised technologies such as Grid systems (OCSI, 2010). These systems have high capabilities for sharing data and resources through the Internet. However they often fall short of providing measurable proof of compliance, which is required throughout the complete data sharing process. This is different than the case of traditional centralised systems, where the data security focus was directed only towards data access transactions. As such, it is an on-going challenge to search for ways to narrow the gaps between the various *legal*,

technical, *social* and *organizational aspects* of the problem.

The approach presented in this paper is an attempt to show that the use of Semantic Web technologies (Horrocks et al., 2004) can allow both the specification and enforcement of privacy requirements that traditional access control languages and mechanisms cannot achieve. We start from the high-level regulations that govern privacy and data protection in Europe and we progress towards the integration of privacy constraints interpreted from them within access controls specifications. For this matter, policies' decisions cannot be deduced from data identifiers and access control conditions that are evaluated against their attributes' values. Instead, the evaluation of privacy policies requires more information about resources; and hence, we face the need to record metadata about the protected resources in a computational infrastructure. We believe, the existing access control solutions for the cloud need to evolve in order to allow for such integration and in order to enable enforcement of the full range of *privacy* constraints.

Although semantic based languages can adequately capture and conceptually specify the *contexts*, *facts* and *rules* necessary for reasoning about data manipulation obligations, it is rather not suitable for implementation in a cloud context. This is due to the necessity for answering two major clouds requirements namely *performance* and *standardisation*. In order to enable better interoperability, while exchanging data in the cloud, it is important to use standard data management languages and services. This includes both standard access control and security languages (OCSI, 2006). Moreover, the use of semantic access control languages requires customised enforcement architectures that are different from the ones adopted on the cloud infrastructure and that are designed to enforce policies specified in a standard format. A similar change might be very expensive from the point of view of clouds services and infrastructure providers. In order to be easily enforced at the cloud's system-level, we suggest that the presented policies should eventually be specified in a way that conforms to a widely adopted policy language or standard. In particular, a standard that has proven efficiency in the enforcement of privacy policies. Our choice is the eXtensible Access Control Mark-up Language (XACML) (OASIS XACML, 2005). It is worth mentioning; and in order to eliminate confusion, that in the context of this work, we do not claim that XACML can handle privacy constraints in exactly the same way as it handles security constraints. The limitations of XACML, both as a policy language and as an enforcement mechanism, have been detailed in the literature (Casassa et al., 2007), (Sommer et al., 2008), (Casassa, 2010). Also additional limitations are presented in Section 3 of this paper. In this work, we seek to overcome some of these limitations. For a note to the readers, additional effort was also made in later version of XACML (OASIS XACML, 2013).

The remaining paper is organised as follows: Section 2 starts by clarifying the theory presented in this paper in comparison and continuation to the allied work that we have been doing previously in this domain. This Section also clarifies the major contributions presented in this paper. Section 3 presents a synopsis of the main technologies on which we have based our privacy specification and enforcement approach, which are presented in later Sections 4, 5 and 6. In particular, Section 4 discusses the SWRL-based privacy policies specification and Section 5 shows how they could be rewritten in a syntax conforming to the XACML standard. In Section 6, a formalism for mapping SWRL privacy rules into XACML access controls is presented. This is followed by the requirements and recommendations for implementing the projected formalism in Section 7; and finally, the conclusion and relevant future orientations are presented in Section 8.

2 PAPER CONTRIBUTION AND RELATION TO PREVIOUS WORK

In (Rahmouni et al., 2010) and (Rahmouni et al., 2011), we have described how Semantic Web technologies have been used to classify the resources that we would like to protect. At that stage the resources were specified using the metadata captured within an ontology. We have also shown in this existing work that how different scenarios of data/resource sharing have been modelled within the same ontology. In this paper, we describe extensions to the previous model (with necessary metadata added) and extend the data sharing scenarios to include privacy policy contexts. We then show how this allows the specification and editing of privacy and access control policies in terms of existing concepts within the ontology. There is research reported in the literature; such as (Muppavarapu and Chung, 2008), (Gowadia et al., 2008) and (Matteucci

et al., 2010), that has looked at the use of ontologies and Semantic Web technology in order to allow a better specification and enforcement of security and authorisation policies. Among these, only the "Consequence" project has looked at an approach that integrates requirements from high-level policies through the means of controlled natural language (Matteucci et al., 2010). This approach translates high-level policies extracted from data sharing agreements into a natural language-like formalism in order to allow enforceability. This work did not stop at the *control of access to data*, but has rather focussed on ways of controlling any type of data usage even after the data were shared with a party belonging to an external domain. This functionality is worth further consideration and is discussed in the future work section of this paper. In comparison, the actual status of our approach allows the disclosure of data handling policies to external parties receiving personal data, but does not enforce these policies within the receiver's domain. However, the work in (Matteucci et al., 2010) included little effort to integrate within access control policies, privacy requirements that are interpreted from primary legislation (text law). It made rather a focus only on traditional security services such as authorisation and the trust aspect of it. Hence this approach couldn't fit in a solution aiming for the big picture of regulatory compliance. This is because usually traditional security requirements covers only a very specific subset of jurisdictional requirements that are not general enough to cover any case of data sharing that might arise in the future.

3 SWRL AND XACML

In this section, an overview of the semantic web rule language (SWRL) and the extensible access control markup language XACML is presented. In this regard, an analysis of their expressiveness capabilities and utility for enforcing privacy policies in a cloud environment is also elaborated.

SWRL, the Semantic Web Rule Language (SWRL) (Horrocks et al., 2004) is based on a combination of the OWL-DL (Matteucci et al., 2010) and some sublanguages of the Rule Mark-up Language (RuleML) (Boley et al., 2010). SWRL includes a high-level abstract syntax for Horn-like rules in both the OWL-DL and OWL-Lite sublanguages of OWL (Bechhofer et al., 2004). The proposal extends the set of OWL axioms to include Horn-like rules. It thus enables the rules to be combined with an OWL knowledge base. Some model-theoretic semantics are given to provide the formal meaning for OWL ontologies, including rules written in an abstract syntax. With the combination of an XML syntax based on RuleML, the OWL XML Presentation Syntax and an RDF (Bechhofer et al., 2004) concrete syntax based on the OWL RDF/XML exchange syntax, SWRL presents an illustration of the extension of description logic into defeasible description logic (Wang et al., 2004). This makes it a promising technology for the modelling of regulations.

The proposed rules are of the form of an implication between an *antecedent (body)* and a *consequent (head)*. The intended meaning can be read as: *whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold.* Both the *antecedent (body)* and *consequent (head)* consist of *zero* or *more* atoms. An *empty antecedent* is treated as *trivially true* (i.e. satisfied by every interpretation), so the consequent must also be satisfied by every interpretation; an *empty consequent* is treated as *trivially false* (i.e., not satisfied by any interpretation), so the antecedent must also not be satisfied by any interpretation. Multiple *atoms* are treated as a *conjunction*. Note that rules with *conjunctive* consequents could easily be transformed into multiple rules, each with an *atomic consequent* (Gruber, 1995). Atoms in these rules can be of the form *C(x), P(x,y), sameAs(x,y)* or *differentFrom(x,y),* where *C* is an OWL description class, *P* is an OWL property, and *x,y* are either variables, OWL individuals or OWL data values. It is easy to see that OWL DL becomes undecidable when extended in this way as rules can be used to simulate role value maps (Gruber, 1995).

XACML (OASIS XACML, 2005) is an XML specification and syntax for expressing policies controlling the access to information through the Internet. It provides the enterprises with a flexible and structured way of managing access to resources. The specification language is based on a subject-target-action-condition policy syntax specified in an XML document. As specified in the Fig. 1 (OASIS XACML, 2005) a *Policy* is composed of a *Target*, which identifies the set of capabilities that the requestor must expose along with a set of rules varying from one to many. Every Rule contains the specific facts needed for the access control decision-making. It also has an evaluation Effect, which can be either *Permit* or *Deny*. At policy evaluation time a policy combining algorithm is used to deal with *(permit/deny)* conflicts which might arise in the rule decisions. A *Target* is composed of four sub-

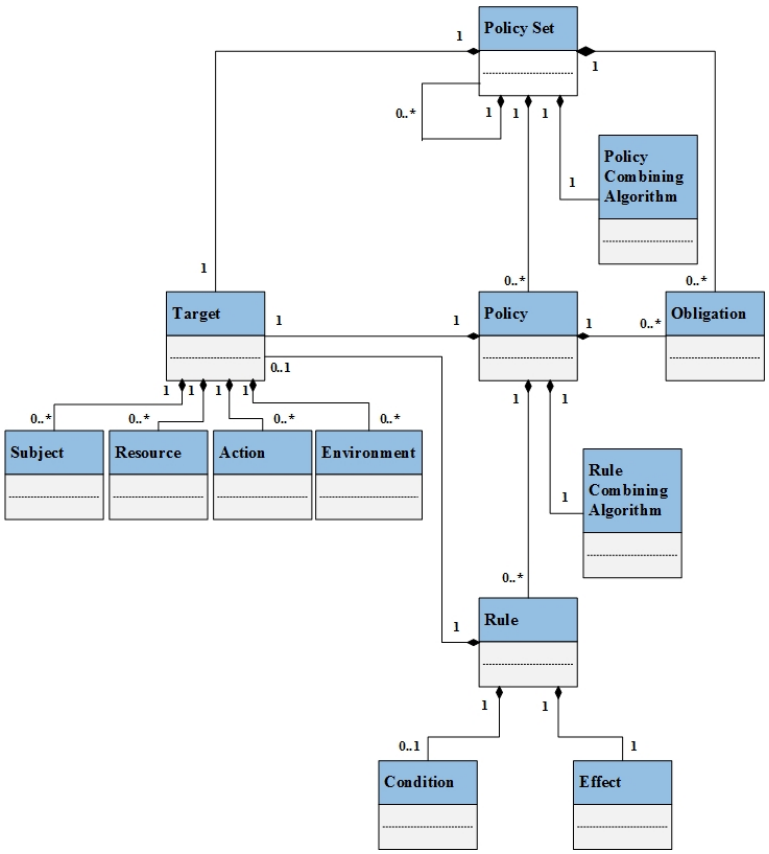


Figure 1: XACMLv2's data flow model.

elements: *Subjects*, *Actions*, *Resources*, and Environments. Beyond what is described in the Fig. 1, each target category is composed of a set of target elements, each of which contains an attribute identifier, a value and a matching function. Such information is used to check whether the policy is applicable to a given request. This could be specified in the condition section of a rule.

In most of the cases the language defines controls as a collection of attributes relevant to a principle. It includes both conditional authorisation policies and other policies to specify post conditions such as notifications to data subject. Like other policy languages that are based on XML, XACML lacks the required semantics to handle heterogeneity and permits interoperability, especially when managing data access within environments that involve multiple organisations. The different data access requests coming from users in different organisations might refer to the same data item with different naming. Additional semantics are needed in order to allow semantic alignment to the different terms used to describe the same data item (Muppavarapu and Chung, 2008). Moreover dealing with dynamic attributes such as the user's age or hierarchical attributes for example the user's role requires some additional semantics and integrated reasoning (Demchenko et al., 2008), (Priebe et al., 2006), (Damiani et al., 2004).

4 A SWRL-BASED PRIVACY POLICY SPECIFICATION

We have examined the legal privacy rules and obligation dictated in many jurisdictional texts and we have noted that the rules are specified according to a specific vocabulary describing many conceptual entities. These entities are usually associated together in different combinations in order to build generic rules that could be modelled in the form of *if-then* rule template. Following this assumption and similarly to the work done in (Powers et al., 2004) that simplifies policies dictated by the Ontario's Freedom of Information and Protection of Privacy Act (FIPAA) (Ontario, 2008), we have expressed the policies specified in European data protection text

law in a more simplified way using the different concepts that constitute its vocabulary. These concepts were captured in an OWL ontology that we have described in previous work (Rahmouni et al., 2010) and (Rahmouni et al., 2011). The policies were then matched to the rule template.

Privacy-Rule-Template:

If

[Context] and [Condition on User], [Condition on data], [Condition on Purpose], [Condition on Other] (including checking for privacy requirements)

Then

Allow [action] and Impose [Obligation]

The rule *privacy-rule-template* could be adapted and specialised, according to the context of application, in order to represent privacy requirements in a case based manner. On this basis, we have rewritten privacy policies interpreted from text law as SWRL rules using OWL classes and properties specified in our privacy ontology. The rule conforms syntactically to the SWRL human readable syntax:

Antecedent Clause *implies* Consequence Clause

Or, in a different notation:

`Antecedent → Consequent`

Adapting the rule to an access control policy format, it must conform to the following template:

```
Rule := Target ∧ Conditions → Effect ∧
        Obligations
```

Here we explain our SWRL privacy policies specification through a concrete rule example and a cloud data sharing scenario:

Example: Purpose Compatibility Rule

In order to clearly explain our approach we start by specifying an example of high-level policy extracted from European privacy legislation. The policy is further taken trough series of transformations towards an operational status in the format of XACML syntax. In this example, we show how we model the privacy policy stating (Iversen et al., 2006) that:

"A user may access a patient mammogram for a stated purpose provided that the patient has given informed consent for a specific processing purpose and the stated processing purpose is compatible with the purpose consented for".

We present below the application of the generic template of SWRL privacy rules to this rule example. For this we adopt a human readable SWRL syntax.

We denote by `(R, T, Con, E` and `Ob)` respectively the Rule elements `(Rule, Target, Conditions, Effect` and `Obligations)` described in the abstract syntax of privacy rules given above. The rule template is therefore rewritten as follows:

`R := T ∧ Con → E ∧ Ob`

In order to implement our rule example, we need to apply it to a concrete data sharing scenario. For this we present the example of data sharing in the cloud described in Fig. 2:

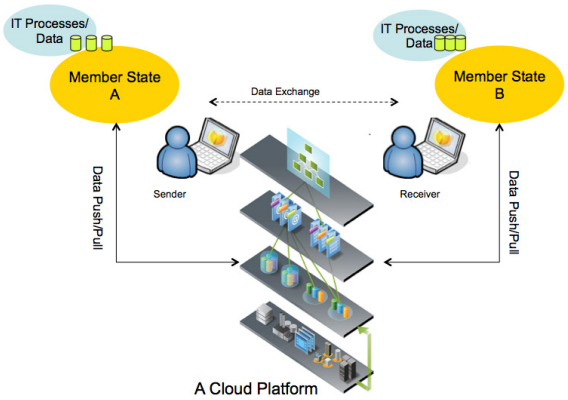


Figure 2: A cloud data sharing scenario.

The scenario we have chosen describes a case of data sharing in the *health* domain. We assume that two medical doctors belonging each to a different hospital in different European member states for example UK and Italy form the two data sharing parties. To be more precise, one of the medical doctors would like to get a second opinion on a patient's *Mammogram*.

The data will be exchanged on a cloud platform and it is required that the cloud security services could identify the right policy to apply in order to allow the sharing of the data, but in a lawful way. Since we are looking at a pan-European context, it wouldn't be always the case that the data processing law is interpreted and implemented in one member state in exactly the same way as in another. Stating as an example, when processing health data for the purpose of medical research, the patient consent must be a *specific consent* when referring to the law in the UK or France. However consent could be *broad* or *general consent* when referring to an Italian law.

For this more context information should be provided in the privacy rules specification for the cloud security processes in order to be able to make the right decision. It is therefore essential to indicate in the rule implementation the sender and receiver's locations and the member state from which the shared data comes. An instantiation of R in the context of the

rule example and the cloud data sharing scenario is then interpreted in the Fig. 3.

```
T:
          hasSender(?x, ?s)
       ∧  hasReceiver(?x, ?r)
       ∧  hasPurpose(?x, ?p)
Con:
       ∧  locatedIn(?s, UK)
       ∧  locatedIn(?r, Italy)
       ∧  concerning(?x, ?m)
       ∧  belongsTo(?m, UK)
       ∧  isForPatient(?m, ?pt)
       ∧  provided(?pt, InformedConsent)
       ∧  hasCollectionPurpose(?m,
          CollectionPurpose)
       ∧  compatibleWith(?p,
          CollectionPurpose)
E:    hasSharingDecision(?x, allow)
Ob:      hasObligation(?x,
         attachSecondaryUsePolicy)
```

Figure 3: Instantiation of the privacy rule template.

In this SWRL rule example OWL properties and classes were used to describe the different elements of the privacy rule target *T,* for example *hasSender* is an owl object property specifying the sender *s* involved in the data sharing ?*x*. Other properties are also used to declare *T* including *hasReceiver*, *hasPurpose* for specifying the receiver of the data and the purpose of sharing respectively. The OWL property *concerning* is used to capture the resource being shared. Since the scenario involves the sharing of patient *?pt* mammograms we have denoted the shared resource/object as *?m*.

The second part of the rule antecedent are the conditions section and it shows constraints the target elements should satisfy in order to infer the effect and obligations shown in the rule consequent section.

5 MAPPING AN ACCESS CONTROL SWRL RULE TO AN XACML CONFORMING SWRL RULE

For easy mapping to an XACML rule, the SWRL rule has to be specified in terms of attributes of only the generic entities that constitute an XACML Rule Target (see above) and other elements that are used to specify the general policy that the rule in question belongs to, e.g. the purpose of processing. In this regard, the OWL property:

`provided(Patient, Informed Consent)`

is a property of the patient whose data is to be shared and indicates that the patient has provided informed consent. The patient or the data subject is not one of the XACML "Rule Target" components; therefore, we express the same condition in terms of property of the class "O" (the resource or object. In our case it is the data the subject is requesting access to). The result is presented in Table 1:

Note: we do not need to translate provided (?pt, InformedConsent) as we are not keeping constraints about patients in the XACML version of the rule. For example, in the XACML conforming SWRL Rule, the consent is an attribute of the object and not of the patient any more.

The rule described above is an extension or privacy aware version of traditional access control rules that pays no significant attention to privacy constraints and obligations. If specified in SWRL syntax, an

Table 1: Mapping from SWRL rule to XACML conforming SWRL rule.

Initial SWRL Privacy-Aware access control rule	XACML Conforming SWRL Privacy-Aware access control rule
dataSharing(?x)	hasRuleContext(?r, ?rc)
∧ hasAction(?x, ?ac)	∧ hasContextAction(?rc, ?ac)
∧ hasSender(?x, ?s)	∧ hasContextSubject(?rc, ?s)
∧ hasReceiver(?x, ?r)	∧ hasReceiver(?ac, ?rec)
∧ concerning(?x, ?m)	∧ hasContextObject(?rc, ?o)
∧ hasPurpose(?x, ?p)	∧ hasContextPurpose(?rc, ?p)
∧ action(?ac, send)	∧ action(?ac, send)
∧ locatedIn(?s, UK)	∧ locatedIn(?s, UK)
∧ belongsTo(?m, UK)	∧ consent(?o, true)
	∧ hasConsentType(?o, InformedConsent)
∧ compatibleWith(?p,CollectionPurpose)	∧ compatibleWith(?p, ?cp)
Rule Implication	
hasSharingDecision(?x, allow)	hasRuleEffect(?r, allow)
∧ hasObligation(?x, attachSecondaryUsePolicy)	∧ hasObligation(?r, attachSecondaryUsePolicy)

example of this traditional access control rule would look as shown in Fig. 4.

```
hasRuleContext(?r, ?rc)
∧     hasContextSubject(?rc, ?s)
∧     hasContextObject(?rc, ?o)
∧     hasContextAction(?rc, send)
∧     hasContextPurpose(?rc, ?p)
∧     hasRole(?s,doctor)
∧     isForPatient(?o,?pt)
∧     isDoctorOf(?s,?pt)
→     hasRuleEffect(?r, allow)
```

Figure 4: Instantiation of the privacy rule template.

The only constraints the rule above tests for before allowing the disclosure of the data is the role of the *subject* or *requestor*. In this case, the role of the subject must be a medical doctor of the patient whose data is to be disclosed.

XACML was designed to notate access control policies and to provide a reference framework for their enforcement. Its major focus is on security policies, although privacy is mentioned in the specification of version 2.0 (OASIS XACML, 2005). It is verbose and complex and still lacks expressiveness. The XACML version 3.0 however, seems to provide a better privacy specification profile (OASIS XACML, 2013). We have also noticed that some examples included in the XACML privacy profile, which were supposed to specify a policy compliant with the "specific and compatible purpose" privacy principle, in fact test for equality or matching of purposes rather than compatibility of purposes. We believe this is due to the language's lack of semantics and reasoning ability with regards to privacy constraints on protected data. If this lack is not addressed, a straightforward mapping from SWRL policies to XACML will not be possible. From the examples of the SWRL access control and privacy rules presented earlier in this paper, we conclude that privacy obligations should be specified for each rule as they are matched according to the data sharing context that we declare to be unique for each rule. This is different from the way obligations are specified in XACML. Obligations in XACML are related to a policy and not to the individual rules that a policy is made up of. We have resolved this problem by allowing each policy to include only one rule and its applicable obligations. Indeed, this decision was already implicit at the time we designed our SWRL privacy aware access control policies. For it, we decided to include one rule per policy. In fact, dealing with more than one privacy obligation at once might require a large amount of contextual information. Therefore, the equivalent SWRL rule would become too long and less readable.

6 MAPPING AN XACML CONFORMING SWRL RULE TO AN XACML POLICY

In this section, we present an attempt to formalise a mapping of a SWRL rule to an XACML policy.

There is some existing work that has looked at formalisms of XACML with many purposes in mind such as in (Kolovski et al., 2006), (Kolosvki et al., 2008), (Kolovski and Hendler, 2008), (Masi et al., 2012) and (Jeremy et al., 2007). In particular, the work presented in (Kolovski et al., 2006) and (Kolosvki et al., 2008) has started from a BNF representation of an XACML rule and has produced a DL formalism that allows the mapping of an XACML rule to DL syntax. Our approach takes into consideration the syntactic difference between DL and SWRL. SWRL is an extension of OWL-DL that can be mapped to DL syntax. It has inherited *Horn-like* propositional logic syntax from RuleML and this characteristic would influence the deviation from a DL formalism provided in (Kolovski et al., 2006) and (Kolosvki et al., 2008). The mapping process has already started from the previous section when we have transformed our SWRL access control rule into an XACML conforming representation. This was done by translating all the properties occurring in the antecedent and consequent to properties applied only on concepts that could be identified in the set of entities that occur in the XACML language model. After the transformation, we suggest that our SWRL access control rules can be generalised under the following formalism.

Formalism1:

```
Rule  := Target ∧ Conditions
         → Effect ∧ Obligations
R := Tgt ∧ Con → Eft ∧ Ob
```

We denote by:

- **R:** an OWL concept representing an access control rule.
- **Tgt:** the target of a rule R that usually constitutes of the elements Subject, Object, Action and Purpose.
- **Con:** the constraints to be imposed on the different elements of a target and that should be satisfied in order for the decisions specified in the consequence clause to be satisfied.
- **Eft:** the effect of a rule R that could be a Permit

or Deny

- **Ob:** the set of obligations that could be associated with the rule R

Formalism 1 may be mapped to Formalism 2 as described below:

Formalism 2:

$$\text{Rule}\left(\text{R}\right)\wedge[\overset{3}{\underset{i=0}{\wedge}}\text{Pd}_i\left(\text{R},\ \text{pd}_i\left(\text{R}\right)\right)]\wedge[\underset{i=<3,k=<n}{\overset{m}{\wedge}}\text{Pc}_k\left(\text{C}_i,pc_k\left(\text{C}_i\right)\right)]\rightarrow \text{Eft}\left(\text{R},\text{e}\right)\wedge[\overset{m}{\underset{j=0}{\wedge}}\text{Ob}_j\left(\text{R},ob_j\left(R\right)\right)]$$

Where i, j and k are natural numbers ranging, respectively, over the number of rule target elements (0..3), the number of properties in our ontology (0..n), and the number of obligations that would be associated with the rule R (0..m), and where:

- $\wedge$ with limits is the symbol for multiple conjunctions;
- **Pd** denotes an OWL property used for declaration of the target elements of a rule R;
- **Pc** denotes an OWL property used for specifying constraints on the elements constituting a target of a rule;
- **C** represents a given class of our ontology with C0, C1, C2 and C3 representing respectively the entities constituting the elements of a target of a rule in the order: Subject, Object, Action and Purpose;
- **Eft** is an OWL property specifying the effect of a rule R, its value is a literal e where e belongs to {permit, deny}.

Table 2 provides a one to one mapping of the entities

Table 2: Logical formalism of SWRL access control rules.

Entity	SWRL formalism
SWRL Rule: SR	$\text{Rule(R)}\wedge[\overset{3}{\underset{i=0}{\wedge}}\text{Pd}_i\left(\text{R},\ \text{pd}_i\left(\text{R}\right)\right)]\wedge[\underset{i=<3,j=<n}{\overset{m}{\wedge}}\text{Pc}_k\left(\text{C}_i,pc_k\left(\text{C}_i\right)\right)]\rightarrow\text{Eft}\left(\text{R},\text{e}\right)\wedge[\overset{m}{\underset{j=0}{\wedge}}\text{Ob}_j\left(\text{R},ob_j\left(R\right)\right)]$
Effect	`Eft(R, e) e ::= Permit | Deny`
Target	`Subject(R,Sub)∧ Object(R,Obj)∧ Action(R,Act)∧ Purpose(R,Pur)` $::=\overset{3}{\underset{i=0}{\wedge}}\text{Pd}_i\left(\text{R},\ \text{pd}_i\left(\text{R}\right)\right)$
Conditions	`Conditions(Sub) ∧ Conditions(Act) ∧ Conditions(Res)∧ Conditions(Pur)::=` $\underset{i=<3,j=<n}{\overset{m}{\wedge}}\text{Pc}_k\left(\text{C}_i,pc_k\left(\text{C}_i\right)\right)$
Obligations	$\pi\text{Ob}::=\overset{m}{\underset{j=0}{\wedge}}\text{Ob}_j\left(\text{R},ob_j\left(R\right)\right)$

Table 3: SWRL to XACML mappings.

Entity	SWRL formalism	XACML formalism
Rule	`Formalism 2`	`R ::= (Rule Tgt Eft)`
Effect	`Eft(R, e) e ::= Permit | Deny`	`Eft ::= Permit | Deny`
Target	`Subject(R,Sub)∧ Object(R,Obj) ∧ Action(R, Act) ∧ Purpose (R, Purpose) ::=` $\overset{3}{\underset{i=0}{\wedge}}\text{Pd}_i\left(\text{R},\ \text{pd}_i\left(\text{R}\right)\right)$	`Tgt ::= ((Sub) (Act)(Res) (Pur))`
Condition Clause	`πConditionsπ Sub) ∧ Conditionsπ(Act) ∧ Conditions∧π(Res) ∧ Conditions∧π(Pur) ::=` $\underset{i=<3,j=<n}{\overset{m}{\wedge}}\text{Pc}_k\left(\text{C}_i,pc_k\left(\text{C}_i\right)\right)$	`Each Pc(C, Pc(C)) := Sub |Act | Res| Pur ::= Any | Fn Fn::= AV | Fn \ Fn | Fn [Fn | ¬ Fn AV::=(attr-id attr-val) attr-id attr-value`
Obligations	$\pi\text{Ob}::=\overset{m}{\underset{j=0}{\wedge}}\text{Ob}_j\left(\text{R},ob_j\left(R\right)\right)$	`Each Ob(R, ob(R))::= AV`

constituting an XACML rule and Formalism 2.

In order to be able to translate our SWRL rules to XACML rules we suggest allowing a one to one mapping between our SWRL Formalism 2 and the BNF formalism of an XACML rule provided in (Kolovski et al., 2006) and (Kolosvki et al., 2008). To achieve this, we have extended the XACML BNF notation with the purpose element of a rule target and the rule obligations clause. The mapping is described in Table 3.

Based on the above *one to one* mapping, we mapped the purpose compatibility SWRL rule produced earlier to the XACML Rule presented in Fig. 5. In this rule, we have chosen to name the *sender* and *receiver* specified previously in the cloud scenario (Section 4) as *Dr_House* and *Dr_Casa* respectively. Other context variables describing the objet to be shared and the sharing purpose were also replaced with concrete values. We have chosen *M1* to indicate the mammogram being sent by *Dr_House* and the sending purpose were specified as *SecondOpinionOnTreatment.*

```
<Rule RuleId = "1" Effect="Permit">
<Target>
<Subjects>< Attribute AttributeId="Subject-Id" DataType= "String">
<AttributeValue> Dr_House </attributeValue>
< Attribute AttributeId = "Location" DataType= "String">
<AttributeValue> UK </attributeValue>
< Attribute AttributeId = "Receiver-Id" DataType= "String">
<AttributeValue> Dr_Casa</attributeValue>
< Attribute AttributeId = "Receiver-Location" DataType= "String">
<AttributeValue> Italy </attributeValue>
< Attribute AttributeId = "Role" DataType= "String">
<AttributeValue> Doctor </attributeValue>
</Subjects>
<Resources>< Attribute AttributeId = "ResourceId" DataType= "String">
<AttributeValue> M1</AttributeValue>
< Attribute AttributeId = "Consent" DataType= "Boolean">
<AttributeValue> true</AttributeValue>
</Resources>
<Action> >< Attribute AttributeId = "Action-Id" DataType= "String">
<AttributeValue>send</AttributeValue></Action>
<Purpose>
<Attribute AttributeId= "purpose-id" DataType="String">
<AttributeValue> SecondOpinionOnTreatment</AttributeValue>
</Attribute>
<Attribute AttributeId= "compatibleWith" DataType= "bag">
</Attribute>
</Purpose>
</Target>
<Condition
<Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>
 <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-is-in">
 -- Consent-Purpose is the purpose for which the data subject has consented or the purpose for which the data (Resource) is
legally stored on the grid database we have specified this purpose as an attribute of the resource in question--
<ResourceAttributeDesignator attributeId= Consent Purpose DataType = "string"/> BreastCancerDiagnosisAndTreatment
</ResourceAttributeDesignator>
</Apply>
<Apply>
<PurposeAttributeDesignator attributeId= "CompatibleWith"  DataType = "bag"/>
</Apply>
</Condition>
</Rule>
```

Figure 5: Privacy aware XACML rule.

7 SEMI-AUTOMATED MAPPING OF SWRL RULES TO XACML RULES

In order to further automate the mapping of SWRL rules to XACML rules, we rely on mapping templates where we can specify for each OWL property an equivalent XACML attribute ID. Furthermore, we need a detailed *one to one* mapping between the OWL axioms specifying conditions/constraints on the different elements of a rule target and the standard XACML functions that could be used as alternatives to these axioms once applied on XACML attribute-ids.

In most of the cases, an XACML equality function/predicate would be the relevant function to allow the translation of an OWL property constraint. The two operands of the equality are first, the *attribute_id* that should hold the name of the OWL property and second the attribute value that should be the same as the OWL property value. XACML distinguishes between several equality checking functions depending on the *data types* of the *operands*. The equality functions in XACML include *string-equal*, *Boolean-equal*, *Integer-equal* and other types. Deciding on which one we need to select is based on a mapping between the OWL *data type* of the property *value* and XACML *data types*. If the property value is determined by an *object property* then an XACML attribute matching function of type *string* should be used. If the property value is determined by a *data type property*, then the XACML attribute matching function should have the same type as the *data type property value*. The work presented in (Kolovski et al., 2006) and (Kolosvki et al., 2008) provides a detailed mapping of XACML data types to OWL data types. A reverse mapping is needed in our case, since we are interested in mapping OWL axioms to XACML conditions instead.

8 CONCLUSIONS AND FUTURE WORK

In this paper, we have modelled high level policies interpreted from European and national data protection law as privacy aware access control policies. The use of Semantic Web technologies such as OWL and SWRL allowed the integration of privacy requirements highlighted in text law such as requirements of consent and other safeguards of patient rights as policy constraints. Towards an easy enforcement in the security architecture of highly distributed infrastructures such as clouds, we have used mapping templates to transform the Semantic Web access control policies to a de facto and highly portable standard of access control notably XACML. The work is validated through the use of examples and scenarios of data processing and one of them is selected and presented in this paper i.e. *"Medical Images Exchange"*. This permitted to conclude that the use of ontologies and semantic technologies could provide relatively easy interpretation of legislation at an operational level. Few challenges were faced when conducting this work that we have overcome by mapping the SWRL privacy policies to XACML policies. An interesting future work in this area for us is to produce an extended XACML enforcement architecture that is able to adequate the added semantic layer for the SWRL to XACML mapping task. This will require both an implementation of the mapping formalism and testing it on the extended enforcement architecture.

REFERENCES

Brandic, I. and Dustdar, S. and Anstett, T. and Schumm, D. and Leymann, F. and Konrad, R., 2010. Compliant Cloud Computing (C3): Architecture and Language Support for User-Driven Compliance Management in Clouds, *IEEE 3rd International Conference on Cloud Computing.*

Bechhofer, S. et al., 2004. OWL Web Ontology Language Reference, [Online] *W3C* Available at: http:// www.w3.org/ TR/owl-ref/ [Accessed 2013].

Beyleveld D, Townend D., Rouillé-Mirza S., Wr ight J., 2004. Implementation of the Data Protection Directive in relation to medical research in Europe, *Ashgate Publishing Limited,* ISBN-10: 0754623696.

Boley, H. et al., 2010. Schema Specification of RuleML 1.0, [Online] Available at: http://ruleml.org/1.0/ [Accessed 2012].

Casassa Mont, M., Crosta, S., Kriegelstein, T. & Sommer, D., 2007. PRIME Architecture V2, *Deliverable D14.2.c.* [Online] Available at: https://www. primeproject.eu/prime_products/reports/arch/pub_del_ D14.2 .c_ec_WP14.2_v1_Final.pdf [Accessed 2013].

Casassa Mont, M., Shen, Y., Kounga, G. & Pearson, S., 2010. *EnCoRe Project Deliverable D2.1. Technical Architecture for the first realized Case Study.* [Online] (1.0) Available at: http://www.encoreproject.info [Accessed June 2013].

Damiani, E., di Vimercati, S. D. C., Fugazza, C. & Samarati, P., 2004. Extending Policy Languages to the Semantic Web, *in Proceedings of the 4th International Conference of Web Engineering*, Springer.

Demchenko, Y., Koeroo, O., de Laat, C. & Sagehaug, H., 2008. Extending XACML authorisation model to support policy obligations handling in distributed applications. *In Proceedings of the 6th International Workshop on Middleware for Grid Computing*, ACM.

EC.Directive 95/46/ECofthe European Parliament and of the Council.,1995 (cited 2010). *Available online from:* http://ec.europa.eu/justice/policies/privacy/law/index_en.htm#directive.

Gowadia, V., Scalavino, E., Lupu, E. & Aziz, B., 2008. The Consequence Project, *Deliverable D3.1: Models and framework for Meta-data generation and policy infrastructure.* [Online] Available at: http://www.consequenceproject.eu/ Deliverables_Y1/D3.1.pdf.

Gruber, T. R., 1995. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43(4-5),pp. 907--928.

Horrocks, I. et al., 2004. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. [Online] *W3C Available at:* http://www.w3.org/Submission /SWRL/ [Accessed 2013].

Italian Personal Data Protection Code, 2003. Legislative Decreeno. 196 of 30 June 2003. Online; 2003 (cited 2012). Available from: http://www.privacy.it/ privacycode-en.html.

Iversen A., Liddell K., Fear N., Hotopf M., Wessely S. Consent, 2006. Confidentiality and the Data Protection Act, *British Medical Journal (Clinical Research Ed)*, 332 (7534):165–169.

Jeremy W. Bryans, John S. Fitzgerald, 2007. Formal engineering of XACML access control policies in VDM++, *Proceedings of the formal engineering methods 9th international conference on Formal methods and software engineering*, November 14-15, Boca Raton, FL, USA.

Kolosvki, V. 2008. Logic-based Framework For Web Access Control Policies, *PhD Thesis, Digital Repository at the University of Maryland*, College Park, Md.

Kolovski, V. & Hendler, J., 2008. XACML policy analysis using description logics, [Online] *Available at:* http://www.mindswap.org/□kolovski/KolovskiXACMLAnalysis- JCSSubmission.pd [Accessed 2012].

Kolovski, V., 2006. Formalizing XACML Using Defeasible Description Logics. *Technical Report TR-233-11,* University of Maryland, College Park.

Masi, M., Pugliese, R., Tiezzi, F., 2012. Formalisation and Implementation of the XACML Access Control Mechanism, *In ESSoS. LNCS* 7159, 60-74, Springer.

Matteucci, I., Petrocchi, M. & Sbodio, M.L., 2010. CNL4DSA – a Controlled Natural Language for Data Sharing Agreements, *In Proceedings of the 2010 ACM Symposium on Applied Computing,* Sierre, Switzerland, ACM.

McCullagh, K., 2006. Study of data protection: harmonization or confusion? *In Proceeding of the 21st BILETA Conference: Globalisationand Harmonisation in Technology Law*, Malta.

Muppavarapu, V. & Chung, S.M., 2008. Semantic-Based Access Control for Grid Data Resources in Open Grid Services Architecture - Data Access and Integration (OGSA-DAI), *in 20th IEEE International Conference on Tools with Artificia lIntelligence (ICTAI 2008),* Dayton, Ohio, USA, 2008. IEEE Computer Society.

OASIS XACML, 2005. eXtensible Access Control Markup Language (XACML), Version 2.0 (2005), *Available online at* http://docs.oasisopen.org/xacml/2.0/XACML-2.0-OS NORMATIVE.zip.

OASIS XACML, 2013. eXtensible Access Control Markup Language (XACML), Version 3.0, http://docs.oasis-open.org /xacml/3.0/xacml-3.0-core-spec-os-en.pdf [Accessed 2013].

OCSI: The Open Cloud Standards Incubator, 2010. Architecture for Managing Clouds, *White Paper from the Open Cloud Standards Incubator* 1.0, DMTF DSP-IS0102, [Online] Available at: http://www.dmtf.org/ standards/published_ documents/DSP-IS0101_1.0.pdf.

Ontario, 2008. Freedom of Information and Protection of Privacy Act, [Online] Available at: http://www.elaws.gov.on.ca/html/statutes/english/elaws_statutes_90f31_e.htm [Accessed 2013].

Powers, C., Adler, S. & Wishart, B., 2004. EPAL Translation of the Freedom of Information and Protection of Privacy Act, *White Paper IBM Tivoli and Information and Privacy Commissioner,* Ontario.

Priebe et al., 2006. Mitigate Content-Related Risks With Enterprise Rights Management. *Trends. Forrester Research.*

Rahmouni H. B. , Solomonides T., Casassa Mont M, Shiu S., 2010. Privacy compliance and enforcement on European Healthgrids: an approach through ontology. *Philosophical Transactions of the RoyalSociety*, 368: pp 4057–4072.

Rahmouni H. B., Solomonides T., Mont M. Casassa, Shiu S, Rahmouni M. A., 2011, Modeldriven Privacy Compliance Decision Support for Medical Data Sharing in Europe. *Methods Inf Med.* 2011 Aug 15;50(4):326–36.

Rahmouni, H. B., Solomonides, T., Casassa Mont, M. & Shiu, S., 2011. Ontology Based Privacy Compliance for Health Data Disclosure in Europe. *PhD Thesis, University of the West of England*, Bristol, UK.

Sommer, D., Casassa Mont, M. & Pearson, S., 2008. PRIME Architecture V3. *Deliverable 14.2.d*. [Online] Available at: https://www.primeproject.eu/prime_products/reports/arch/pub_del_D14.2.d_ec_WP14.2_v3_Final.pdf [Accessed 2013]

Wang, K., Billington, D., Blee, J. & Antoniou, G., 2004. Combining Description Logic and Defeasible Logic for the Semantic Web. *In Rules and Rule Markup Languages for the Semantic Web: Third International Workshop*, RuleML. Hiroshima, Japan, Springer.

Procurement Auctions to Maximize Players' Utility in Cloud Markets

Paolo Bonacquisto, Giuseppe Di Modica, Giuseppe Petralia and Orazio Tomarchio
Department of Electrical, Electronic and Computer Engineering, University of Catania, Catania, Italy
{*firstname.lastname*}*@dieei.unict.it*

Keywords: Cloud Market, Procurement Auction, Bidding Strategy, Cloud Simulations.

Abstract: Cloud computing technology has definitely reached a high level of maturity. This is witnessed not just by the interest of the academic community around it, but also by the wide adoption in commercial scenarios. Today many big IT players are making huge profits from leasing their computing resources "on-demand", i.e., letting customers ask for the amount of resources they need, and charging them a fixed price for each hour of consumption. Recently, studies authored by economists have criticized the fixed-price applied to cloud resources, claiming that a better pricing model can be devised which may increase profit for both the vendor and the consumer. In this paper we investigate how to apply the mechanism of procurement auction to the problem, faced by providers, of allocating unused resources. In particular, we focus on the strategies providers may adopt in the context of procurement auctions to maximize the utilization of their data centers. We devised a strategy, which dynamically adapts to changes in the auction context, and which providers may adequately tune to accommodate their business needs. Further, overbooking of resources is also considered as an optional strategy providers may decide to adopt to increase their revenue. Simulations conducted on a testbed showed that the proposed approach is viable.

1 INTRODUCTION

Cloud computing aims to provide computing resources to customers like public utilities such as water and electricity (Buyya et al., 2008). In an Infrastructure-as-a-Service (IaaS) cloud environment, physical resources are packaged into distinct types of virtual machines (VMs) and offered to customers. A cloud customer, on the other hand, will purchase VMs to run his applications, by looking for specific resource requirements in terms of CPU, memory and disk. Given the finite capacity for each type of resources in each data center, a fundamental problem faced by IaaS provider is how to select the price and allocate resources for each type of VM services in order to best match the interests of the customers while maximizing his revenue. This issue is further complicated by the fact that, differently from traditional utility markets, cloud demand is strongly time varying and often burstly.

The resource allocation and trading mechanisms used by the current cloud computing systems are inefficient and inflexible due to the flat rate pricing model adopted. We argue that a fixed price-based resource allocation currently in use in cloud computing systems do not provide an efficient allocation of resources and do not maximize the revenue of the cloud providers. In a previous work (Di Modica et al., 2013), we already showed that a better alternative would be to use auction-based resource allocation mechanisms. In this paper we address issues related to the bidding strategies adopted by providers of computing resources in the context of procurement auctions. We try to analyze all the factors that mainly impact the strategic choices of providers in the acquisition of the goods allocated through auctions. The purpose of this work is not to devise an optimal bidding strategy, but rather, to prove that any strategy will have its objective guaranteed by the procurement mechanism. We also devised a tentative provider's strategy which adapts its aggressiveness to the earlier mentioned factors. In the addressed market scenario, we stress that our attention is devoted to the optimization of the utilization rate of providers' data centers and the utility of providers.

The remainder of the paper is structured as follows. Section 2 makes a review of the literature and gives some rationale of the work. Section 3 introduces the proposed idea and delves into technical details about procurement auctions. In Section 4 simulation results are presented and discussed. Finally, the work is concluded in Section 5.

2 RELATED WORK

All main commercial IaaS providers "comply" with the On-Demand approach, and offer to charge customers for the actual time frame during which the resource is actually utilized [1]. They ask users pay a fixed price for computing capacity by the hour. The only chance for the customer to get discounts on the price-per-hour is to opt on the *reservation* or the *flat rate* charging models. According to these options, customers get a discount on the price but are committed to longer periods of lease (from a month to a year). Among commercial providers, the only one that successfully proposed an approach alternative to the fixed-price is Amazon with its *Spot Instance* model[2]. This model enables the customer to bid for what they call unused computing capacity. Virtual machines are charged the *Spot Price*, which is set by Amazon and fluctuates periodically depending on the supply/demand rate for computing capacity. According to this model, on the one hand the provider gives the customer the possibility to acquire computing resources at a lower price then the standard; on the other one, whenever the resources' demand increases, the provider reserves the right to preempt those resources and give them to better bidders. This model represents the very first attempt to build up a virtual market of computing resources regulated by market prices, i.e., prices which dynamically fluctuate according to offer and demand. In spite of this, the model is still unclear (the formula of price fluctuation is not known) and is not proved to be resistant to potential malicious behaviors of customers (dishonest customers can abuse the system and obtain short-term advantages by bidding large maximum price bid while being charged only at the lower spot price (Wang et al., 2012)). Furthermore in (Agmon Ben-Yehuda et al., 2011) authors prove that the Amazon's Spot Price is not market driven, rather is typically generated as a random value near to the hidden reserve price within a tight price interval. The consideration stemming from this observation is that a provider, being an interested party, may not be a guarantee for the correctness of the price determination. Instead, a third party broker should be in charge of calling out prices in auction-based contexts. A quick review of the recent literature proves that researchers are very much concerned with the application of auction mechanisms to the problem of the allocation (read "sale") of computing resources. In (Risch et al., 2009) authors propose a marketplace of computing resources where prices are determined using an exchange market. In (Chard and Bubendorfer,) authors discuss several strategies that cloud providers should adopt in order to reach high performance and to overcome most of criticisms of auctions like high overheads and high latency using techniques like overbooking and Flexible Advanced Reservations. They propose several bidding functions but each one takes into account only one parameter among those monitored by a cloud provider. Our work is different as we try to guarantee the provider's utility maximization through an adaptive strategy based on several parameters referring to the actual condition of resources allocation, intentionally weighted by the provider in order to address a specific target.

For the majority of researchers, combinatorial auctions are the most appropriate sale mechanism for allocating virtual machines in the cloud. In combinatorial auctions the participants bid for bundles of items rather than individual items (Cramton et al., 2005). This mechanism seems to perfectly fit the Cloud context, as customers usually need to acquire not just one resource but a bunch of resources (e.g., one for hosting the database server, one for the application server and one for the web server). In (Wang et al., 2012) authors propose a suite of computationally efficient and truthful auction-style pricing mechanisms, which enable customers to fairly compete for resources and cloud providers to increase their overall revenue. (Zaman and Grosu, 2013) proposes a combinatorial auction-based protocol for resource allocation in grids. They considered a model where different grid providers can provide different types of computing resources. Buyya et al. (Buyya et al., 2010) propose an infrastructure for auction-based resource allocation across multiple cloud systems. In (Vinu Prasad et al., 2012) authors address the scenario of multiple resource procurement in the realm of cloud computing. In the observed context, they preprocess the user requests, analyze the auction and declare a set of vendors bidding for the auction as winners based on the Combinatorial Auction Branch on Bids (CABOB) model.

The discussed works mainly focus on solving the problem of optimal sale of resources in combinatorial auctions, which is known to be NP-hard. The work we propose, instead of defining yet another suboptimal allocation algorithm, takes a different direction. From a strictly technical point of view, one of IaaS providers' main issue is to adopt an efficient allocation scheme which allows them to map customers' requests to virtual machines (or virtual resources, in general) in an efficient way. Often, when a new re-

[1] http://aws.amazon.com/ec2/
http://www.microsoft.com/windowsazure/
http://www.rackspace.com/

[2] http://aws.amazon.com/ec2/spot-instances/

quest must be served there are many management operations that need to be carried out along. Let us suppose that an IaaS provider has a number of hosts, and a new request R demanding some computing power has arrived. According to both the actual hosts' occupancy rate and the adopted allocation scheme many different actions may be taken. For instance, computing capacity may be allocated in the host where the computing availability best (or worst) fits the computing request. Or, computing capacity may be allocated in an unloaded host running on a stand-by state. Or again, a running virtual machine allocated in host H_1 may be migrated to host H_2 in order to make room for R (whose size fits better in H_1 than in H_2). Those cited here are only a few of the many examples of management issues that providers must face with.

The profit of a provider strongly depends on its capability of keeping the hosts' average occupancy rate as high as possible. For their nature, computing resources can be regarded as perishable goods that need to be sold within a certain time frame otherwise they get wasted. Not selling a virtual machine in a given slot time means a profit loss for the provider, who anyway is spending money to keep the physical machines up and running. We then look at the trade of computing resources from a new perspective, in which providers, in the aim of maximizing their data center's occupancy rate, may be willing to attract customers by lowering the offer price. On their turn, customers may get what they need, at the time they need it, at a price which is lower than the standard price at which they usually buy. We advocate that the market model best fitting this perspective is the one which guarantees the sale of computing resources through **procurement auctions**. Procurement auctions (Klemperer, 1999) (also called reverse auctions) reverse the roles of sellers and buyers, in the sense that the bidders are those who have interest in selling a good (the providers), and therefore the competition for acquiring the right-to-sell the good is run among providers.

The sale of computing resources through procurement auctions will work as follows. The market gathers computing demands from any customers. For each computing request a procurement auction is publicly called out. Providers can search the market in order to identify the specific request(s) which best fits their need in that particular moment (e.g., a request for a virtual machine of a specific size that would "fill" a given host's capacity) and participate in the respective call(s). A call will be won by the provider offering to serve the demand at the lowest price. On the provider's side, this mechanism is profitable since the huge number of computing demands gathered by the market increases the chances of the provider to find the one(s) satisfying their needs. In their turn, many customers will be attracted by the possibility to get what they ask at a price that is lower than the standard, therefore will be stimulated to push their requests to the procurement-based market.

3 PROCUREMENT AUCTION MARKET

The purpose of this work is not to convince providers to abandon the direct-sell mechanism in favor of the procurement-based market. Providers have their regular customers, who issue requests which most of the times have a well known timing. For this kind of requests the most appropriate model is the *direct-sell/fixed-price*, in that it provides guarantees for both the provider and the customer. What we propose is the adoption of an alternative, dynamic pricing model for selling what is usually referred to as the *unused capacity*, i.e., the residual capacity that, on average, the provider is not able to sell through direct-sell.

Let us define the utilization rate $U(t)$ as the fraction of the overall unused capacity committed to serve customers' requests at time t . The lower U, the higher the profit loss for the provider. In the aim of maximizing the utilization rate (minimizing the residual capacity) providers need to adopt new selling strategies. The simplest strategy could just be lowering the price per computing unit. Amazon currently leases its unused capacity by adopting an auction-inspired price strategy that let the customers acquire resources for a price which is lower than the standard. We argue that providers, to avoid "wasting" computing capacity, are willing to give up a portion of profit per computing unit (same as it happens for sale of perishable goods). As far as we know, the Amazon's is the only example of dynamic price strategy that is alternative to the fixed-price. If on the one hand it is true that customers benefit from low prices, on the other one the proprietary mechanism by which the virtual machines' price fluctuates has never been disclosed. The computing capacity's actual supply/demand rate is not shared to customers, nor the price policy has ever been released.

In this paper we propose the design of a market of computing capacity (Figure 1), to which any provider is admitted, and where computing resources can be sold through auction-based allocation schemes. The perspective is that of procurement auctions, where an initial price is called out and bidders iteratively have to call lower prices to win. The market mechanism is the following. Customers communicate their com-

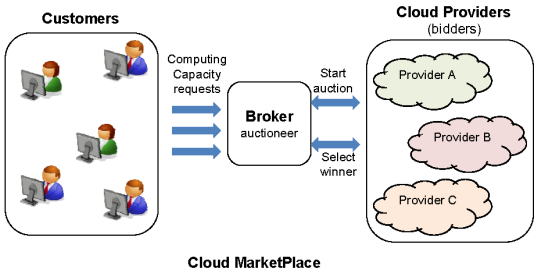


Figure 1: The Cloud Marketplace scenario.

puting demand to the market. A *broker* will take care of demands. For each specific demand, the broker (auctioneer) will run a public auction to which any provider (bidder) can participate and compete for "acquiring" the demand. The winning provider (who offered the lowest price) will eventually have to serve the demand. Being the auctions open to the participation of multiple providers, the competition is granted. Providers will have to fight to gain the right-to-serve the demand. Bidding strategies enforced by providers can range from the most conservative to the most aggressive. The determination of the final price is driven only by the evaluation that each provider has on the goods to acquire (i.e., the customer's demand to be served). Customers will get their demand served at the lowest price. Further, they will have no more the burden to search for providers, as providers are gathered in the market. As for the broker, its income will be evaluated in a way proportional to both the number of transactions (auctions) it will be able to carry out and the prices formed in the transactions.

3.1 Customer's Demand

Customers need to acquire cloud resources to satisfy their computing needs. Following the cloud paradigm, the computing power of a physical machine can be adequately shaped to satisfy the most demanding customer need, in terms of CPU cores, CPU cycles, Mips and main memory size. In practice, all cloud providers offer a restricted number of virtual machines (VM) configurations (instance types), which go from the very minimalist to the most powerful one. In this paper we will refer to a classification of VM instance types which recalls that of Amazon. Though discrepancies exist between the classification being presented and those adopted by other providers, schemes that map the respective items can easily be derived (but are out of the scope). In the rest of the paper, we will then assume that customers' demand will address the following VM instance types:

- **General Purpose.**
  - M1.small - 32/64-bit architecture, 1 vCPU, 1 ECU, 1.7GB RAM, 160GB Storage, Low Bandwidth;
  - M1.medium - 32/64-bit architecture, 1 vCPU, 2 ECU, 3.75GB RAM, 410GB Storage, Moderate Bandwidth;
  - M1.large - 64-bit architecture, 2 vCPU, 4 ECU, 7.5GB RAM, 820GB Storage, Moderate Bandwidth;
  - M1.xlarge - 64-bit architecture, 4 vCPU, 8 ECU, 15GB RAM, 1.6TB Storage, High Bandwidth;
  - M3.xlarge - 64-bit architecture, 4 vCPU, 13 ECU, 15GB RAM, 0 Storage, Moderate Bandwidth;
- **Compute Optimized.**
  - C1.medium - 32/64-bit architecture, 2 vCPU, 5 ECU, 1.7GB RAM, 350GB Storage, Moderate Bandwidth;
  - C1.xlarge - 64-bit architecture, 8 vCPU, 20 ECU, 7GB RAM, 1.6TB Storage, High Bandwidth;
- **Memory Optimized.**
  - M2.xlarge - 64-bit architecture, 2 vCPU, 6.5 ECU, 17.1GB RAM, 420GB Storage, Moderate Bandwidth.

We also assume that customers may demand for combinations of the above listed resources. For instance, a customer may issue a request for three M2.xlarge instances and four M1.large instances. Along with the (combination of) resources needed, the customer must communicate the time at which the resource(s) need to be accessed (*start-provision time*) and the time at which the resource provision must terminate (*stop-provision time*).

3.2 Auction Types

Auctions are the tool that the broker will make use of in order to allocate providers' computing capacity to customers' demand. An auction can be run by the broker as soon as a new demand has arrived. For a given demand, the broker can choose from some auction types to run. The choice of the best suiting auction type is driven by several factors, which are object of observation in this paper. Also, it is up to the broker the setting of the initial price of the auction.

As outlined before, the typology of auction that best fits the depicted context is the *procurement auction*. In a procurement auction the buyer (customer, in our case) advertises a need (demand for computing capacity) and sellers (providers) compete to provide the service (virtual machines to satisfy the demand).

This leads to decreasing prices being offered during the period of the procurement auction.

In this paper we will focus on three different types of procurement auctions. The common part of the three auction mechanisms is the auction preparation, which provides that upon the arrival of a demand, the broker issues a public "call for proposal" (CFP) to invite providers. The CFP shall specify a minimum set of auction parameters including the start-provision time, the stop-provision time, the initial price (from which discount bids are expected), the bidding rules (who can bid and when, restrictions on bids) and the clearing policy (when to "terminate" the auction, who gets what, at what price). After having collected the willingness of providers to participate in the auction, the preparation phase ends up and the bargain starts according to what is specified in the bidding rules and the clearing policy. What characterizes one auction mechanism from another one is the information specified in the bidding rules and the clearing policy. For our purpose we will address the following mechanisms:

- **English Reverse (ER)**(Parsons et al., 2011). The ER is a multi-round auction. The CFP specifies the initial price from which discounting bids (offers) are expected. The participating bidders can post their offers. Discounting offers are called out, so that every bidders is always aware of the reference price for which further discounts are to be proposed. If no offer arrives within a time-frame (publicly set in the CFP), the good will be assigned to the last best (i.e., the lowest priced) offer. This type of auction allows bidders gather information of each other's evaluation of the good.
- **First Price Sealed Bid (FPSB)**(Parsons et al., 2011). The FPSB is a single-round auction, i.e., all bidders have the chance to bid just once, before the auction being cleared. When bidders receive the CFP, they check the initial price and decide to either bid or not to bid. After all participants have posted their bid, the broker clears the auction and allocates the "demand" to the bidder who has valued it the most (in the case of a reverse auction, the least). The peculiarity of this auction is that bidders are not aware of each other's offer, as only the winning bid will be broadcast at the end of the auction.
- **Second Price Sealed Bid (SPSB)**. Like the FPSB, it is a single round auction. But with the difference that the one who wins the auction (i.e., who offers the highest price) will pay the second highest bid price. This mechanism, applied to a reverse auction, aims at improving the provider's *utility* (refer to the next section for the definition of the provider's utility). In fact, according to this mechanism each provider may keep its good's evaluation secret, and if they will win the good they will be acknowledged a price which is higher than their own bid, thus increasing the overall utility.

3.3 Provider's Strategy

One of the objective of this work is the study of an adaptive (i.e., dynamic) strategy for the providers that participate in procurement auctions. By strategy we mean a set of rules producing the decisions a provider must take to maximize their own business objective. Basically, a strategy shall drive the provider in choosing the right actions to be undertaken when competing for the acquisition of a good (e.g., whether to participate in a given auction, to bid in a given round, not to bid, which price to offer). In the strategy design, the first step was to outline the main factors that may impact such choices. Secondly, we tried to devise a dynamic strategy which accounts for the just mentioned factors and smoothly adapt their fluctuations. Finally, we set up and configured a test environment to analyze the results produced by the strategy.

According to the literature, the behavior of an auction's participant is mainly driven by the information the participant has on the value of the good being sold (Klemperer, 1999). In respect to this information, two basic auction models are possible: 1) the *private-value* model, where each bidder has an estimate of the good for sale, and that estimate is private and unaffected by others' estimates, and 2) the *pure common-value* model, where the actual value of the object is the same for everyone, but bidders have different private information on how much that value actually is. Combined models can also be derived from the cited ones.

If we better analyze the context of cloud auctions, a computing resource can be seen as a good whose actual value (price) is common to all providers. In fact, though for computing resources we can not yet speak of conventional "market prices", all providers in their regular sales adopt well known, leveled prices. We can then conclude the actual values of such kind resources are somewhat common to providers. In the context of a procurement auction of computing resources, the estimate E_{pi} of the i-th provider for a given good may differ from the the estimate E_{pj} of the j-th provider according to the diverse needs each provider may have in pursuing their own business objective.

Primary objective of a provider is to maximize what is referred to as *Utility*. Given a resource to

be allocated through an auction sale, the provider's Utility for that request is defined as the difference between the winning bid price and the evaluation that the provider gives to the resource. (Wang et al., 2012). Of course, the provider aims at maximizing the average utility for the resources they compete for. Recalling the considerations made earlier, in the context of an auction sale of spare resources this objective can be pursued: a) by keeping the data center's utilization as high as possible; b) by bidding prices higher than the personal evaluation (which we will refer to as *lower bound*) and c) by choosing the most profitable combination of customers' request to serve.

We identified a non-exaustive list of factors which may strongly influence the strategy of a provider in a procurement auction.

- The duration of the customer task (demand) to be served (L). The longer the task, the higher the profitability for the provider, since the required capacity will be committed for a longer time. A provider, then, might prefer to participate in auctions where long tasks are traded.
- The type of VM instance required to serve the customer task (T). Of course, the profitability of a task is directly proportional to the task's requirements in terms of amount of computing capacity per hour, so providers may be motivated in pointing on auctions calling for a higher capacity/hour. But depending on the actual utilization level of both each single host and the whole data center, it might not be possible to serve further tasks requiring high capacity VMs.
- The gap between the potential revenue obtainable from serving the task the standard way (i.e., through the fixed-price market) and that obtainable by serving the task at the price called by the auctioneer (G_r). The revenue for serving a task is given by the L times the price (P) of the resource that will serve the task. This factor strongly depends on the provider's enforced revenue policy. A provider pointing on auctions to sell their unused capacity might accept a much lower revenue (bidding a lower price) in the case that expenses are already covered. Conversely, the provider might not be willing to excessively lower the price in the case that expenses are not yet covered.
- The utilization of the particular physical machine that is going to serve the customer's request. The marginal revenue, in fact, is affected by the utilization level of a host: if a host is already running and serving other tasks, adding more tasks to that host "costs" less than activating a new host.

Finally, some considerations need to be made about the *lower bound*. Each strategy must envision an "exit condition", which represents the condition that, when verified, forces the exit of the provider from the auction. When the provider decides to participate in an auction, they will have to set the lower bound price, which represents the maximum discount that the provider is willing to offer for the good being traded in that auction. Of course, this parameter only makes sense in multi-rounds auctions, as in single round auctions actually exit is imposed by the mechanism itself at the end of the first round. The lower bound parameter actually represents the evaluation of the provider for a given good (customer's request). It incorporates all provider's consideration regarding the costs for executing a VM, managing a VM's life cycle and supporting the customer.

The objective of a strategy is to suggest the provider the price to call for the next bid. In calling a price, a strategy may be more or less "aggressive", i.e., may propose higher or lower discounts. We discuss two different strategies. One is driven by randomness (aggressiveness is randomly chosen auction by auction, round by round). The other is adaptive, in the sense that is able to adapt the aggressiveness according to the above listed factors. For this kind of strategy, the aggressiveness can be tuned by adequately weighting the factors.

Recalling a formula presented in (McAfee, 1987), the adaptive strategy will suggest the next bid as:

$$bid = \frac{n-1}{n-(1-\alpha)} * lastWinningBid \qquad (1)$$

where n is the number of bidders participating in the auction and *lastWinningBid* is the price of the bid that won the last round. In case of single-round auctions and in the case of first rounds of multi-round auctions, *lastWinningBid* will be the auction's starting price. The parameter α is calculated as follows:

$$\alpha = w_1 * U(t) + w_2 * \frac{L}{L_{max}} + w_3 * \frac{P_a}{P_f} + w_4 * \frac{T_{vm}}{T_{max}} + w_5 * H(t) \qquad (2)$$

As we can see in the equation 2, α depends on:

- $U(t)$, the current utilization rate of the pool of spare resources; the less $U(t)$, the higher α, so the evaluated bid price will decrease (in a reverse auction, lowering the bid price means pointing to gain the good). As expected, the aggressiveness of a strategy increases with the reduction of the utilization rate.
- $\frac{L}{L_{max}}$, the ratio between the time period for which the computing resource is requested and the maximum time period for which a resource can be

requested[3]. The ratio will increase for requests with longer execution time. The provider will be more aggressive in auctions where longer customer tasks are negotiated, as those ensure a higher utilization of the data center and, as a consequence, higher revenues.

- $\frac{P_a}{P_f}$ the ratio between the resource's starting price in the auction and the corresponding price in the standard fixed-price market. The provider's aggressiveness will be higher when the price at the start of a round is closer to the reference price (price at which resources are traded in regular markets, or, direct-sell price). The more the round price decreases, the lesser the provider's aggressiveness.
- $\frac{T_{vm}}{T_{max}}$ the ratio between the computing power of the resource being traded in the auction and the computing power of the highest resource. This factor increases the provider's aggressiveness in the case of customer tasks demanding high computing power. The higher the requested computing power, the higher the task's initial price. Further, a highly demanding task requires a bigger capacity on the data center, thus increasing the overall utilization rate.
- $H(t)$, the current utilization of the host on which the customer task to serve will be scheduled. This factor increases the α parameter and, therefore, increases the provider's aggressiveness. Recalling a previous consideration, the provider is more conservative in their strategy if for serving a task a new physical machine has to be activated.

Each parameter is weighted by a factor (w_1,w_2,w_3,w_4,w_5), for which the following constraint applies:

$$\sum_{i=1}^{5} w_i = 1 \tag{3}$$

Different combinations of weights lead to different strategies. Finally, in the adaptive strategy the lower bound price will depend on α according to the following equation:

$$Lb = P_f * (1 - discount) \tag{4}$$

where *discount* is

$$discount = (0.5 * \alpha) \pm rand * 0.03 \tag{5}$$

and P_f is the price of the resource advertised in the standard fixed-price market. The maximum *discount* on the fixed price is evaluated as the 50% of α; the higher alpha, the lower the bound. A variability of 3% was also introduced to model a differentiation among providers, which reflects their respective personal evaluations.

[3]In real situations the time period for which a resource can be requested has no bound; in our simulation we will take into account tasks lasting no longer than 24 hours

3.4 Resource Overbooking

The auction mechanism causes a waste of computing resources at the provider's end. A provider may participate in many auctions (say m) at the same time. For each auction, no matter they win or lose, the provider will have to reserve a pool of resources to accommodate the customer's request for which they are competing. The number of auctions every provider will participate depends on the instant capacity of their free computing resources. In general, provider will win n auctions, being $n <= m$, thus, for the duration of all m auctions there may be a waste of resources proportional to the number of lost auctions $(m-n)$. To overcome this limitation, the provider may decide to participate in more auctions and compete for customers'requests which they are not potentially able to meet. This mechanism, also known as resource *overbooking*, contributes to decrease the resource waste on the one hand, but on the other may bring to situations where the provider runs out of computing resources and may not honor one or more contracts signed at the time they won the auctions. In this cases, the *risk compensation* principle is applied (Phillips, 2005), and the provider will incur penalties which are proportional to their actual bid.

In order to implement such mechanism in our market, we let the provider count on an amount of virtual computing capacity (namely, *overbooking capacity*) which is set to 20% of their real capacity. The provider is then able to participate in $m+o$ auctions, where o is proportional to the overbooking capacity. This way the number of won auctions will increase, and the provider's utilization rate will get closer to 1. In the case the provider won more auctions than those they are actually able to serve, a penalty is due. When an auction appoints as winner a provider who is not eventually able to honor a request, the second best bidding provider is chosen. If, again, the latter is not able to serve the request, the third best is chosen, and so on. In this chain, all providers are subjected to penalties. The penalty is a monetary cost calculated as:

$$penalty_{it} = \frac{P_i - bid_{it}}{P_i - winnerBid_i} * P_i * duration_i \tag{6}$$

where $penalty_{it}$ is the penalty for the i-th CFP due by provider t, P_i is the auction's starting price, bid_{it} is

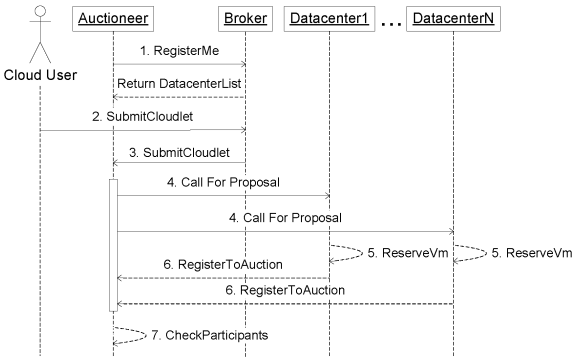


Figure 2: Auction preparation phase.

the bid called by provider t, $winnerBid_i$ is the winner's bid price and $duration_i$ is the time frame for which the computing resource is required by the customer. This law aims at penalizing the providers proportionally to their risk attitude. The auction winner who is eventually unable to meet the request will pay a penalty of $P_i * duration_i$. The following best bidders (2nd, 3rd, ect..) who on their turn are not able to serve the request will pay a lower penalty as their bid is higher than the winner's. If all participating providers happen to fail the provision due to the overbooking, the auction will be closed and a new auction will be called up.

4 EXPERIMENTAL RESULTS

In order to prove the viability of our proposal a simulative approach was followed. A prototype was implemented on the well-known Cloudsim simulator (Calheiros et al., 2011). We developed a new component (the **Auctioneer**) and modified the behavior of other existing components (**Datacenter**, **Broker** and **Cloudlet**). Cloudlet is the component of Cloudsim representing the task submitted by the customer, while Datacenter is representative of the provider that will compete for acquiring the task.

Figure 2 shows the basic steps of the auction preparation phase. The Auctioneer registers to the Broker (step 1) and receives the list of available Datacenters. When a Cloudlet is submitted to the Broker (step 2) it is passed on to the Auctioneer (step 3). The Auctioneer broadcasts a Call For Proposals (CFP) which specifies the task and the auction requirements (step 4). Each interested Datacenter checks its own capacity, reserves a VM and responds to the call (step 5-6). If the Datacenter has run out of physical resources, but is adopting the overbooking mechanism, it will check out whether the overbooking limit has been overcome. If not, it will answer the call. If at least two Datacenters answers the CFP (step 7), the auction is ready to start.

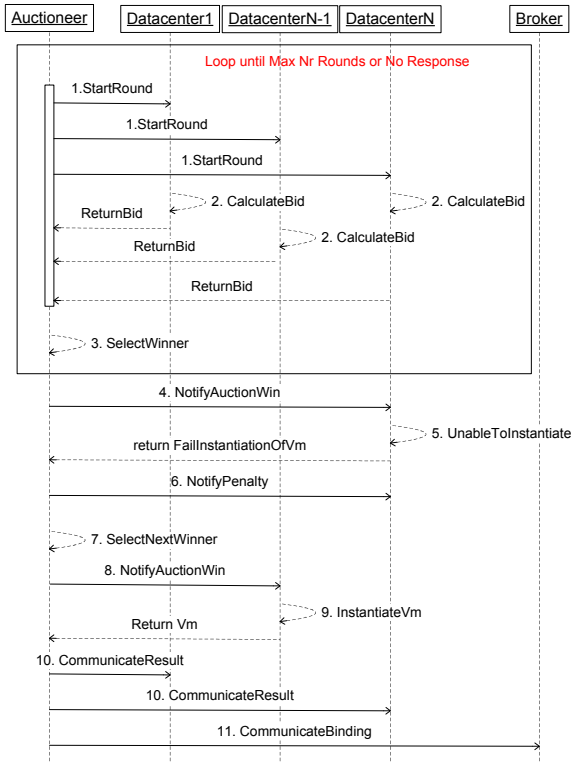


Figure 3: Auction running phase.

The auction running phase is depicted in Figure 3. The Auctioneer notifies the participant that the round is open (step 1). At the beginning of each round the starting price is communicated to the participants. In the very first round the starting price is set equal to a standard on-demand fixed-price. In the following rounds the initial price is the price of the winning bid of the previous round (this applies only to English auctions). Each Datacenter calculates the bid according to its strategy (step 2) and submits it to the Auctioneer, who will appoint the winner (step 3). If the auction type is multi-round, the bidding procedure is repeated until the exit condition holds true. For instance, if the auction is of English type it terminates when no bid has been received within a round. When the auction is closed, the Auctioneer sends the result to the winner (step 4) and wait for the Ack. If the winning Datacenter is not able to serve the request (step5), it will return a negative Ack. The Auctioneer will then contact the second best bidder. If again, the latter has run out of resources, the process will go further until a positive Ack is returned (step 10) or no Datacenter is able to honor the contract. In the first case the auctioneer notifies the losers (step 11) and the broker that will bind the VM to the winning Dat-

Table 1: Weight Setting for the Datacenters' strategies.

Datacenter ID	Strategy	w_1	w_2	w_3	w_4	w_5
DC1	Adaptive	0.6	0.1	0.1	0.1	0.1
DC2	Adaptive	0.1	0.6	0.1	0.1	0.1
DC3	Adaptive	0.1	0.1	0.6	0.1	0.1
DC4	Adaptive	0.1	0.1	0.1	0.6	0.1
DC5	Adaptive	0.1	0.1	0.1	0.1	0.6
DC6	Adaptive	0.2	0.2	0.2	0.2	0.2
DC7	Adaptive	0.2	0.2	0.2	0.2	0.2
DC8	Adaptive	0.2	0.2	0.2	0.2	0.2
DC9	Adaptive	0.2	0.2	0.2	0.2	0.2
DC10	Adaptive	0.2	0.2	0.2	0.2	0.2
DC11	Random					

acenter (step 12). In the second case the auction is considered unresolved, and a new auction is called.

Typical Datacenters' VM allocation policies are:

- *First fit*: a VM is allocated in the first available host which is capable of running it. According to this policy, a single physical machine is used to host VMs until there is available computing capacity. When that machine capacity is saturated, a new unused physical machine will be activated.
- *Worst fit* or *Balanced allocation*: a VM is allocated in the less saturated host among those capable of running it. This policy ensures a balancing of the load among the hosts, but causes high fragmentation of hosts' computing capacity.
- *Best fit*: a VM is allocated on the host having the least amount of unused capacity among those capable of running it. This policy optimize the resource utilization. This is the policy we used for our tests.

To test the adaptive strategy, we created a set of 11 Datacenters, ten of which adopt the proposed adaptive strategy, while one adopts a *Random strategy*: the latter makes its bids using the same equation of the adaptive strategy (Eq. 1), where the α parameter is assigned random values in the [0,1] range, without any specific objective to pursue. The weights characterizing the α parameter ((Eq. 2) are shown in Table 1. As the reader may notice, strategies were expressly split in *unbalanced*, for which Datacenters point on just one factor, and *balanced*, for which all the weights are assigned the same value.

The objective of the simulation is to show that strategies actually guide Datacenters in the choice of tasks to compete for. Every Datacenter counts 70 hosts, each characterized by the following features:

- number of cores uniformly chosen between [64,128,256,512];
- RAM: 320 GB;
- Storage: 10 TB.

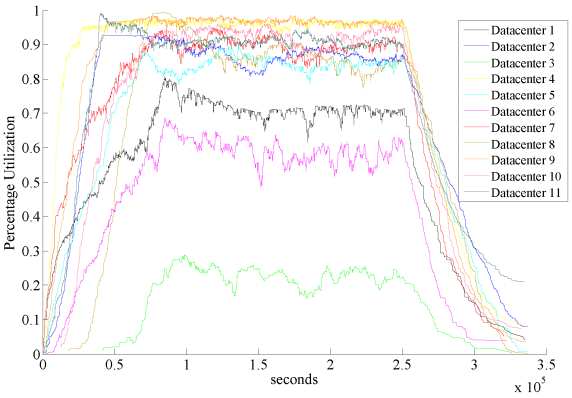


Figure 4: Datacenters Utilization: ER Auction.

A core is modeled in CloudSim with a capacity of 2400 Mips. in the first battery of simulation we submitted 25000 cloudlets having a length uniformly distributed in the range [1,24] hours and requiring a VM type uniformly distributed in the range of the eight VM types described in Section 3.1. The interarrival times of the cloudlets are distributed accordingly to a Poisson distribution, with $\lambda = 0.1$ (10 secs is the maximum time that lapses between the arrivals of two consecutive cloudlets). From early results, we noticed that the adaptive strategy of each Datacenter guarantees the achievement of the objective, regardless of the specific auction type. For this reason (but also for space reasons) we are going to show the results obtained from testing the English Reverse Auction.

The main parameter we measured was the utilization rate of the Datacenters, which is depicted in Figure 4. We can observe that all the Datacenters (DC) pursuing one (or a combination) of these two objectives 1) to acquire VMs that require high capacity in terms of computing resources and 2) to obtain longer lasting tasks, actually reach an high level of utilization rate (in Table 1 DC2 and DC4 respectively).

DC11, that adopts the Random strategy, reaches an high level of utilization rate too, because it can easily win auctions for tasks that do not meet the objectives of other Datacenters (i.e. low performing VMs, short tasks).

DC1's objective is to optimize the utilization rate; in the graph it can be noticed that after reaching an utilization rate between 60% and 80%, it is not able to further increase it, as its objective has almost been reached: the strategy aggressiveness decreases in a way that no more auctions are won.

The Datacenter that obtains the lowest utilization rate (around 20%) is DC3; it exhibits low aggressiveness in the auctions as its objective is to obtain cloudlet with a price not too far from a standard on-demand fixed-price. However, as it can be seen in Figure 5 where the revenue loss percentage of the Data-

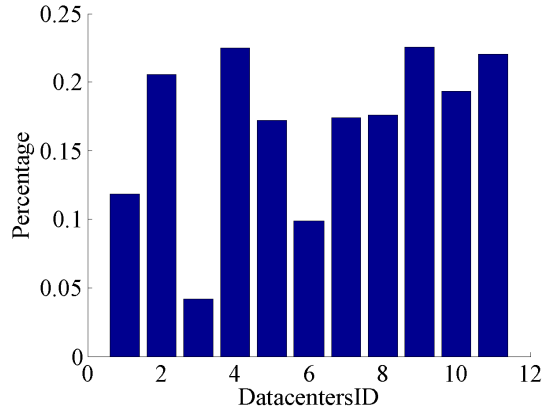


Figure 5: Revenue Loss Percentage per Datacenter.

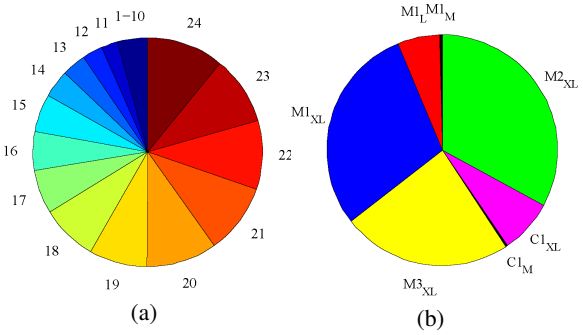


Figure 6: Auctions won by DC2 and DC4, grouped by the length of the Cloudlets the type of VMs respectively.

centers is shown, the objective of DC3 guarantees the lowest revenue loss. Datacenters with balanced strategy may also avoid revenue losses while, at the same time, reaching a better utilization rate than DC3.

Finally, we report some graphs showing the cloudlet characteristics of auctions won by two specific Datacenters. Figure 6(a) shows the rate of auctions won by DC2, grouped by the length of the cloudlets expressed in hours. DC2 mainly won cloudlets with a length of more than eleven hours (the reader can check in Table 1 that the weight of parameter related to the length of the cloudlets is the highest). Figure 6(b) depicts the auctions won by DC4. It mainly wins auctions requiring high performing VMs, as it strategy is set to point on that type of VMs.

In order to check the overbooking mechanism a new simulation battery was run. We created a set of 8 Datacenters enforcing an adaptive strategy with balanced weights, as shown in Table 2. First, the simulator was fed with 40.000 cloudlets having a length uniformly distributed in the range [1,12] hours and poissonian arrivals with $\lambda = 0.01$. In Figure 7 we may notice that all datacenters reach an utilization very close to 100%. This is due to the fact that cloudlets are long (in terms of time required by the task) and

Table 2: Weight Setting for the Datacenters' strategies.

Datacenter ID	Strategy	w_1	w_2	w_3	w_4	w_5
DC1	Adaptive	0.2	0.2	0.2	0.2	0.2
DC2	Adaptive	0.2	0.2	0.2	0.2	0.2
DC3	Adaptive	0.2	0.2	0.2	0.2	0.2
DC4	Adaptive	0.2	0.2	0.2	0.2	0.2
DC5	Adaptive	0.2	0.2	0.2	0.2	0.2
DC6	Adaptive	0.2	0.2	0.2	0.2	0.2
DC7	Adaptive	0.2	0.2	0.2	0.2	0.2
DC8	Adaptive	0.2	0.2	0.2	0.2	0.2

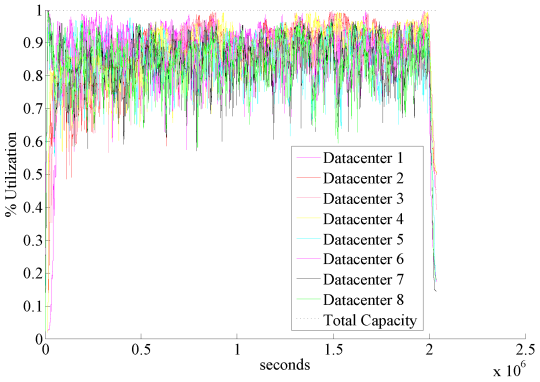


Figure 7: Datacenters Utilization without overbooking - 40000 Cloudlets, max length = 12h, $\lambda = 0.01$.

distant enough (100 secs is the cloudlets' interarrival time). We then repeated the simulation with the same number of cloudlets but with a length uniformly distributed in [1,6] and poissonian arrivals with $\lambda = 0.06$ (17 seconds between consecutive cloudlets). In this case the average utilization decreases to 70-80%, as shown in Figure 8.

What happens is that, being the interarrival time shorter, datacenters simultaneously engage in many auctions. Each datacenter will only win a subset of these auctions, which are also very short, thus it will not be able to saturate its capacity. In this specific case

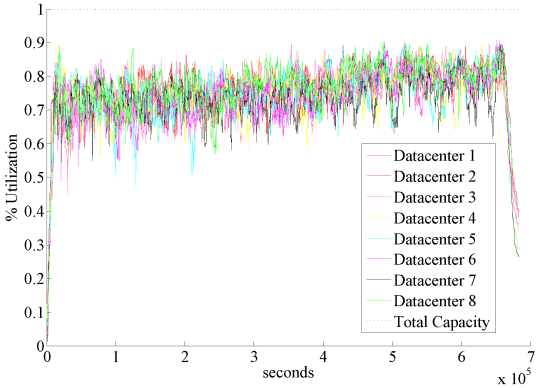


Figure 8: Datacenters Utilization without overbooking - 40000 Cloudlets, max length = 6h, $\lambda = 0.06$.

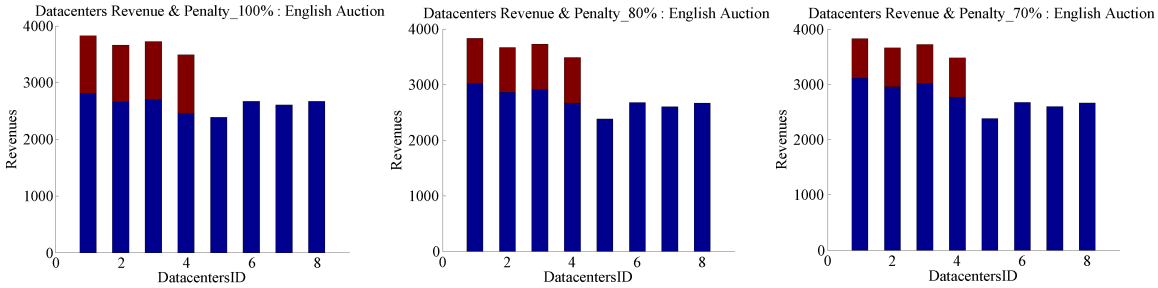


Figure 10: Overbooking Revenues and Penalties.

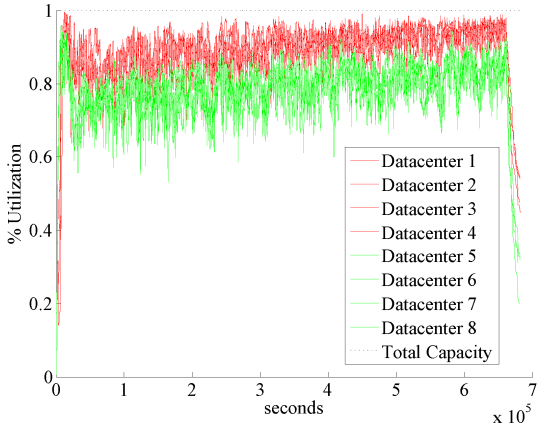


Figure 9: Datacenters Utilization with overbooking - 40000 Cloudlets, max length = 6h, $\lambda = 0.06$.

it may be of help to opt on the overbooking. We then ran a new simulation, with the same parameters, but configuring four of the eight datacenters to enforce a 20% of overbooking (DC1,DC2,DC3,DC4). Figure 9 shows that the the overbooking datacenters (depicted in red) reach a very high utilization (close to 100%). The collateral effect is of course that they incur penalties, which have been evaluated with the formula in 6. The revenue of datacenters enforcing the overbooking drops below the revenue of datacenters which do not use overbooking, as depicted in 10(a). Put in this way, there is no point in opting for the overbooking. We considered new formulas for the evaluation of penalties which consider a reduction of 20% and 30% with respect to the penalty evaluated in 6. Figures 10(b) and 10(c) shows that with the new formulas datacenters enforcing the overbooking have an acceptable revenue. This is to say that overbooking is an opportunity which must be carefully evaluated by datacenters, and must be suitably tuned against the penalty policy adopted by the market. Last consideration is on single round auctions. Both First and Second Price Sealed Bid did not provide encouraging results for the overbooking. This is because the auctions are resolved in a very short time and the utilization of resources is not as dynamic as it is in a multi-round auction. We have then compared the performance of the First and the Second Price Sealed Bid auctions focusing on the average utility of the provider. As depicted in Figure 11, the second price auction guarantees, on average, a better utility. This kind of auction, in fact, let the datacenter bid its real evaluation of the cloudlet preventing the utility from excessively decreasing.

5 CONCLUSION

Cloud computing has stimulated a great interest both in the academic community and in business contexts. More and more IT players look at this technology as a great opportunity of increasing their profit. Though several studies report the cloud services' market revenue is rocketing, economists say the business potential of cloud computing is not yet fully exploited. There is not yet an open market of cloud resources where providers and consumers can meet to satisfy their needs. In this paper we propose a market of resources where demand and offer of resources can be matched in auction-based sales. Specifically, we looked at this market from the perspective of the provider, who needs a strategy to allocate at best their unused computing capacity. We proposed an adaptive strategy that, suitably tailored to the provider's business objective, will help them to maximize the revenue in the context of procurement auctions. Also, the resource overbooking mechanism has been investigated as an optional strategy providers may adopt in order to increase their revenue. Simulations run to test the proposed approach gave encouraging results, by showing that each provider is able to reach their objectives by finely tuning the weights associated to their strategy. In the future, more factors will be taken into account in the definition of the provider's strategy. Further, the business models of the broker of resources (auctioneer) will also be investigated, in order to prove that a market model based on procurement

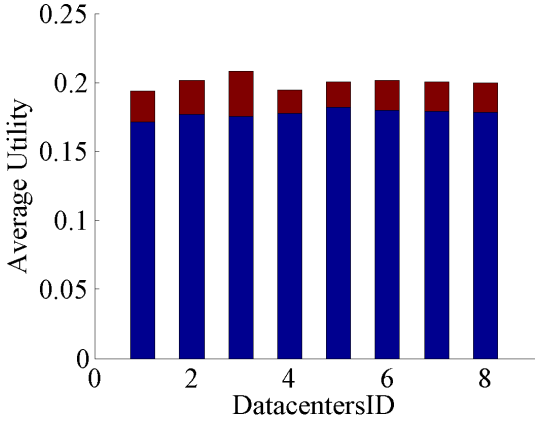


Figure 11: Utility Comparison between FPSB and SPSB.

auctions can yield profit for all market actors.

REFERENCES

Agmon Ben-Yehuda, O., Ben-Yehuda, M., Schuster, A., and Tsafrir, D. (2011). Deconstructing amazon ec2 spot instance pricing. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 304–311.

Buyya, R., Ranjan, R., and Calheiros, R. N. (2010). Intercloud: utility-oriented federation of cloud computing environments for scaling of application services. In *Proceedings of the 10th international conference on Algorithms and Architectures for Parallel Processing - Volume Part I*, ICA3PP'10, pages 13–31.

Buyya, R., Yeo, C. S., and Venugopal, S. (2008). Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *High Performance Computing and Communications, 2008. HPCC '08. 10th IEEE International Conference on*, pages 5 –13.

Calheiros, R., Ranjan, R., Beloglazov, A., De Rose, C. A. F., and Buyya, R. (2011). Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. In *Software: Practice and Experience*.

Chard, K. and Bubendorfer, K. High performance resource allocation strategies for computational economies. In *IEEE Transactions on Parallel and Distributed Systems*, volume VOL. 24, NO. 1.

Cramton, P., Shoham, Y., and Steinberg, R. (2005). Combinatorial auctions. The MIT Press.

Di Modica, G., Petralia, G., and Tomarchio, O. (2013). Procurement auctions to trade computing capacity in the Cloud. In *Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC 2013)*, Compiegne (France).

Klemperer, P. (1999). Auction theory: A guide to the literature. *Journal Of Economic Surveys*, 13(3).

McAfee, R. P. e McMillan, J. (1987). Auctions and bidding. *Journal of Economic Literature*, 15:699–738.

Parsons, S., Rodriguez-Aguilar, J. A., and Klein, M. (2011). Auctions and bidding: A guide for computer scientists. *ACM Computing Surveys*, 43(2).

Phillips, R. (2005). Pricing and revenue optimization. Stanford University Press.

Risch, M., Altmann, J., Guo, L., Fleming, A., and Courcoubetis, C. (2009). The gridecon platform: A business scenario testbed for commercial cloud services. In *Grid Economics and Business Models*, volume 5745, pages 46–59. Springer Berlin Heidelberg.

Vinu Prasad, G., Rao, S., and Prasad, A. (2012). A combinatorial auction mechanism for multiple resource procurement in cloud computing. In *Intelligent Systems Design and Applications (ISDA), 2012 12th International Conference on*, pages 337–344.

Wang, Q., Ren, K., and Meng, X. (2012). When cloud meets ebay: Towards effective pricing for cloud computing. In *INFOCOM, 2012 Proceedings IEEE*, pages 936–944.

Zaman, S. and Grosu, D. (2013). Combinatorial auction-based allocation of virtual machine instances in clouds. *J. Parallel Distrib. Comput.*, 73(4):495–508.

SHORT PAPERS

Portable Green Cloud Services

Stephan Ulbricht[1], Wolfram Amme[1], Thomas Heinze[1], Simon Moser[2] and Hans-Dieter Wehle[2]
[1]*Friedrich-Schiller-University Jena, Jena, Germany*
[2]*IBM Germany Research & Development GmbH, Böblingen, Germany*
{*stephan.ulbricht, wolfram.amme, t.heinze*}*@uni-jena.de,* {*smoser, hdwehle*}*@de.ibm.com*

Keywords: Cloud Computing, TOSCA, Green IT, Policy, Energy Efficiency, Standardized Cloud Services.

Abstract: The areas of cloud computing and green IT are amongst the fastest growing markets in the IT industry, until now there are very few opportunities to combine the potential of both. In this paper, we present a method to combine the advantages of both to create standardized and energy efficient cloud services. For their description, we will use the cloud computing standard TOSCA(OASIS, 2013). Thereby, it is possible to create standardized and model-based cloud applications which can be deployed in many cloud environments. We will further show how it is feasible to combine policies with TOSCA to realize energy-efficient management of cloud services. To accomplish this, we will provide ideas on how to extend the TOSCA language as well as the cloud operating environment to achieve the goal of portable, energy-efficient cloud services. The core of this work is the identification and integration of the underlying system architecture for a common solution concept. For this, the architectures and necessary adjustments are explained.

1 INTRODUCTION

In times of climate change and rising energy costs, even IT companies (who are traditionally not in the spotlight when it comes to naming the biggest polluters) start to think about improving their energy efficiency. While moving a traditional IT application from an data center to a modern, energy-efficient virtualized cloud implementation already has a positive impact on the energy bill, such a step also allows to improve the automated energy-management of the application. In practice, there are two concerns: a.) for the consumer, moving an application to the cloud practically means committing to one cloud provider, as there is no standardized description format that helps to avoid a provider lock-in, and b.) in terms of energy-efficiency, there is no way for a cloud service provider to formally describe the energy guidelines that he would like to have followed. So neither the consumer nor the provider of a cloud service have the possibility to influence the environmental friendliness of an IT service - they are dependent on what the data center or cloud provider offers, both in terms of the infrastructure and the energy sources that get used as well as during the actual operation of the service.

This a huge opportunity to give both cloud service-consumers and -providers a choice: For a cloud service consumer to state that he wants to use "green" cloud services, and for cloud service providers to model the service with a "green" mindset as well as to influence or define the operation of an actual service instance. The first thing that is needed to address this challenge is an accurate model of the service. With model-based cloud services, the dependency on a specific data center or cloud provider can be reduced (since they are portable), and if these cloud services contain energy guidelines, such guidelines could even be reflected when the service is operated. Chapter 3 introduces TOSCA(OASIS, 2013), a standard for describing such model-based cloud services.

2 RELATED WORK

The efforts regarding energy-efficient and environmental sustainable IT solutions have increased significantly in recent years and many different approaches and techniques have been developed. There are three categories where energy improvement actions can be undertaken: Hardware, Software and IT environment.

On the *hardware* side the simplest method for a reduction of energy consumption is the shutdown of unused devices. Such techniques are shown in (Hwang and Wu, 2000, Sueur and Heiser, 2010, Dargie, 2012). In the *software* category it is possible for multiple users to work simultaneously on the same hardware,

through virtualization of computing resources. Research showed different approaches how environmental sustainability can be achieved by process scheduling - c.f. (Duy et al., 2010, Wang et al., 2010). The *IT-Environment* includes a number of components and characteristics, ranging from the power supply via the used cooling system to the layout of the components. Various papers have dealt with the subject of reducing fossil energy sources (Zhang et al., 2011, Liu et al., 2011). These approaches propose to allocate IT requests depending on available energy sources (wind, solar, etc.) or the overall energy costs.

To achieve a sustainable energy improvement of data centers, a holistic solution spanning hardware, software and IT-environment is required. Analogous to our solution, there is other research in this field, presenting frameworks and architectures to realize green enhancements in cloud computing environments - c.f. (Younge et al., 2010, Buyya et al., 2010, Beloglazov et al., 2012, Berl et al., 2010). To create a powerful and widely used solution concept, it is necessary to provide a standardized approach. This is the prerequisite for the acceptance and dissemination in the economy. In the remainder of this paper we'll show how this can be achieved with the help of cloud standards and policy languages.

3 CLOUD STANDARD: TOSCA

Chapter 1 already outlined the need for standardized, model-based cloud services. To ensure the interoperability and portability of cloud services, the standards body OASIS has started the standardization initiative TOSCA (Topology and Orchestration Specification for Cloud Applications) (OASIS, 2013). TOSCA is a meta-language for the model-based description of cloud services. This language not only allows the definition of the topology of an application, but also of its management behavior by requiring business processes which are operating directly on the application topology and thus define the operational semantics. The aim of TOSCA is to create standardized cloud services and simplify exchangeability. These standardized cloud services then get executed by a runtime entity called a "TOSCA container", which e.g. interacts with a hypervisor to install the topology and interprets the management plans to manage an actual service instance. As Figure 1 shows, a TOSCA-based cloud service is described by a *service template*, which is made up of a *topology template* and a (set of) *plan(s)*. The *topology template* defines the structure of the service, i.e., the base components and their relationships. A component can be a VM, an OS, a

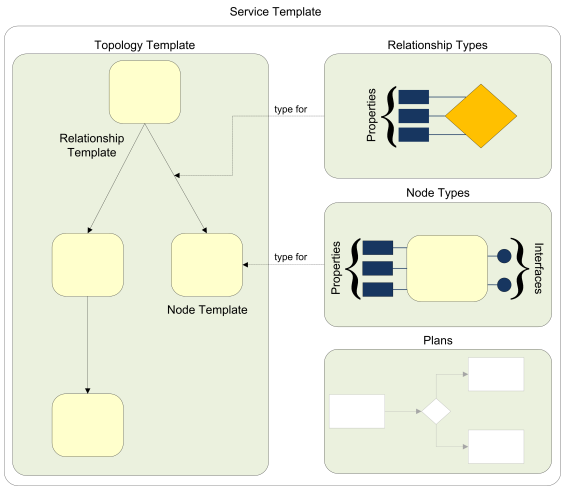


Figure 1: TOSCA service template.

network connection, etc.

Build- and Management Plans define the business process models which control the operational management behavior of the service, operating on the topology. If modeled correctly, the operational management behavior spans the complete life cycle of a service: A build plan for setting up the service, plans for any operation during its lifetime (e.g., trigger backups, install patches, elastic scaling, ...), as well as a termination plan when the service gets deprovisioned. The scope and the implementation of the plans are specific to the service and determined by the service modeler. When finished, the service modeler creates a *Cloud Service Archive* (CSAR), a package that contains the topology template, the plans and all other required artifacts (e.g., image files, scripts, etc.). These CSAR files can then be exchanged between different IT environments. More precisely, a topology template can be represented by a graph, where the nodes are the base components (e.g. a VM) and the edges are the relationships between the components. Both nodes and relationships are first defined outside of a concrete service, as they usually represent reusable building blocks. They are called *Node Types* and *Relationship Types*. A *Node Type* is a node of the type "VM" when used outside of a topology, but once added to a template (potentially with some properties attached), it becomes a *node template*. The same is true for the edges of the graph: *Relationship Types* define abstract relationships (like "hosted on" or "connected to"), but when used in a topology template they become *relationship templates*, describing the relationship between the node templates and also defining the potential states of a service instance.

The structure and functionality of service templates will be shown by example of an LAMP-infrastructure. The corresponding topology template

has the elements ”Linux OS”, ”Apache Web Server”, ”MySQL Database” and ”PHP Module”. The web server and the database are connected via the relationship ”hosted on” with the operating system, and so is the PHP module with the web server. The underlying relationship template ”hosted on type” is characterized by the source and destination element. The second type of relationship (connects to) exists between the PHP module and the database. The description of a node type, e.g. the Web Server, defines properties such as the IP address of an instance.

Another feature of TOSCA is the fact that it allows policies for certain elements, e.g. for *Service Templates* and *Node Templates*. Policies enable the dynamic control of the behavior, i.e. rules can be defined for access control (authorization policy) or to specify requirements on a system (obligations policies). Such a mechanism is useful for the combination with the topic of Green IT. Policies have advantages like reusability, higher efficiency, expandability and better conflict management (Tonti et al., 2003) when it comes to managing large services or systems. Policies consist of rules that control the behavior of computer systems, derived from overarching goals such as company policy, service level agreements or safety requirements or for describing and enforcing e.g. energy efficiency requirements.

4 GREEN CLOUD SERVICES: SYSTEM AND ARCHITECTURE

To create a holistic solution that brings together model-based cloud services with Green IT policies end-to-end, we need to examine three things: service lifecycle, architectures of Cloud Management Platform and Policy Management Systems and their descriptor formats. The lifecycle of a model-based cloud service can be divided into four phases: modeling, deployment, instantiation and runtime. In the modeling phase, the TOSCA-modeler creates the topology template, plans, scripts, images and all other required artifacts required for the execution of the service and packages them into an archive (CSAR). In the next phase, the service definition and all related artifacts are stored in a database component, where they are made accessible by the public through e.g. a service catalog. Once published, the cloud service consumer can select a service from the service catalog - this triggers the instantiation of the service. The core component during the instantiation phase is the TOSCA-container, as it orchestrates the generation of the service, e.g. generating the virtual machines, installing the needed software, setting up the

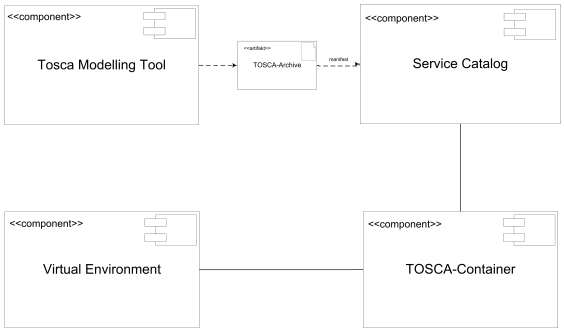


Figure 2: Overview of the components involved in a TOSCA Cloud Management Platform.

network connections etc. by running the associated plans which operate directly on the topology. Figure 2 gives an overview of the involved components.

The next step to create a holistic solution is to map the components from the Cloud Management Platform to key architectural components of a policy management system (IETF, 2001, OASIS, 2005). For illustration, we use the architecture model provided by the Internet Engineering Task Force (IETF, 2001). In this model, the basic components of a policy system are described - Figure 3 gives an overview.

The responsibility of the *Policy Management Tool* is the creation, modification and deployment of policies, the transformation from abstract to concrete policies as well as error handling. The *Policy Repository* provides a storage for the policies, as well as provides the policies when they should be provisioned. Finally, the *Policy Decision Point* interprets policies, receives events and determines the appropriate actions. These components match nicely to the pieces of a TOSCA cloud management platform. To illustrate and motivate the combination into a holistic system, we examine each lifecycle phase: In the *Modeling phase*, the TOSCA specification allows for the injection of policies at different points within a topology template. The policy is part of the topology template description and is stored within the CSAR. The modeler can either directly specify the policy code inside the topology template or refer to it by a reference (c.f.

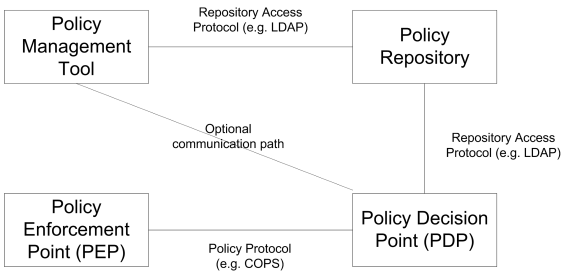


Figure 3: Overview of the components involved in a Policy System, according to (IETF, 2001).

section 5). The reference allows for the integration of externally defined policies. Thus, the TOSCA modeling supports creating and changing policies. During the modeling phase, all attributes of the policy tag have to be defined. The actual content of a policy is described within the tag. In the *Deployment phase*, the policy management tool component of the IETF-architecture is also responsible for the distribution of policies. In TOSCA, all artifacts including policies are grouped into an archive and stored in a service catalog, which is a implementation of the policy repository. When a service has been requested by a user, we enter the *Instantiation phase*. The CSAR archive with the artifacts is loaded into the TOSCA-container. In the receiving environment, the package is parsed and the individual artifacts are analyzed by a topology engine, a part of the TOSCA Container, which also realizes the order of the elements. The engine implements the topology template with the help of provisioning mechanisms into the production system.

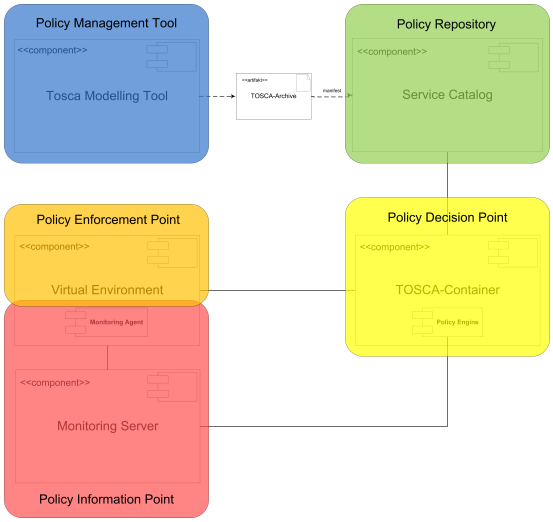


Figure 4: TOSCA- and Policy System Architectures combined.

To create an instance of an application, a *build plan* is executed. For the interpretation and execution of the plan, the TOSCA container uses a process engine which communicates with the hypervisor and triggers e.g., the creation of virtual machines. Additional software is installed on the virtual machines by starting installation scripts. The TOSCA Container picks the policy implementation based on the URI of the policy definition, but for the processing of the incoming events and to determine the actions to be performed, an additional policy engine is needed. It has to be able to afford all functions of the policy decision point. To respond to events, the policy engine includes an event handler component. The task of the event handler is to relate the events to the corresponding policies. Afterwards, the policies are interpreted, i.e., conditions are evaluated and the resulting instructions are determined. In the *Runtime phase*, the management and control of virtual machines is done by the hypervisor. The instructions, determined by the policy engine, will also be implemented by the hypervisor (policy enforcement point). Since virtual environments are very complex and include many properties, special monitoring agents are used to ensure a flexible and comprehensive monitoring. They are installed on the servers to monitor all relevant metrics. The considerations in this section have shown that it is possible to map the IETF- to the TOSCA- model. In Figure 4, the TOSCA-components and their equivalents are shown.

5 GREEN CLOUD SERVICES: METADATA FORMAT

After the combination of a TOSCA cloud management platform with a policy system has been outlined,

```
1  <ServiceTemplate ... >
2   <Import ... />
3   <TopologyTemplate>
4   <NodeTemplate id="VmApache" name="VM for
        Apache" type="ns1:VirtualMachine">
5   <Properties> ... </Properties>
6   <Policies>
7   <Policy
8   name="LowWorkloadPolicy"
9   policyRef="lamp:LowWorkloadPolicy-VmApache"
10  policyType="ns1:LowWorkloadPolicy"/>
11  </Policies>
12  </NodeTemplate>
13  <NodeTemplate id="VmMySQL" name="VM for
        MySQL" type="ns1:VirtualMachine">
14  <Properties> ... </Properties>
15  </NodeTemplate>
16  </TopologyTemplate>
17  <PolicyTypes>
18  <PolicyType
19  name="LowWorkloadPolicy"
20  targetNamespace="...">
21  <PropertiesDefinition element="ns1:
        LowWorkloadPolicy"/>
22  <AppliesTo>
23  <NodeTypeReference typeRef="ns1:
        VirtualMachine"/>
24  </AppliesTo>
25  </PolicyType>
26  </PolicyTypes>
27 </ServiceTemplate>
```

Figure 5: Excerpt of a service template describing an energy-efficient LAMP service.

let's illustrate input to such a holistic system based on the LAMP service template.

In TOSCA, a policy also follows the type / template pattern previously introduced: The *Policy Type* defines the properties of policy (e.g. the data needed to influence a behaviour), whereas *Policy Templates* provide actual values for the properties in a concrete scenario.

Figure 5 shows an excerpt from a Service Template for the LAMP service: In lines 4 and 13, two node templates have been defined, representing a WebServer and a Database. To emphasize our point, think of them as a Web Server Cluster and a Database Cluster. In line 19, a *LowWorkloadShutdownPolicy* has been attached to the Web Server Cluster. In Line 22, the payload of this policy is referenced.

The payload just defines the properties that an event is required to carry, in order for the policy definition point to evaluate the policy and trigger an appropriate action. In our example of shutting down a cluster member when the workload becomes too low, we need only two properties: the old and the new workload value. So far we did not define an action that needs to be taken, nor did we define a concrete policy language. Policies in TOSCA can be defined independently of a policy language. While this supports portability, it requires adaption to a specific language for every cloud provider. But TOSCA allows us also to provide a policy in a specific language:

```
1  <PolicyType  name="LowWorkloadPolicy"
2    policyLanguage="xs:http://www.ponder2.net/"
3    targetNamespace="...">

5   <PropertiesDefinition element="ns1:
        LowWorkloadPolicy"/>
6   <AppliesTo>
7    <NodeTypeReference typeRef="ns1:
         VirtualMachine"/>
8   </AppliesTo>
9  </PolicyType>
```

Figure 6: PolicyType from Figure 5 with a Policy Type defined in the language Ponder2.

In our prototype, we have chosen a policy language called *Ponder2* (PONDER, 2013). *Ponder2* is a declarative, object-oriented language that allows for the definition of both, obligation and authorization policies. *Ponder2* can be used in a wide range of different devices ranging from ad-hoc networks to distributed systems. A Ponder2 system provides the ability to manage a variety of objects such as sensors, alarm systems, routers, etc. These basic elements are called managed objects. They are created in Java and their provided functionalities can be adapted to the requirements of an environment and expanded accordingly. The standard distribution of Ponder2 includes three predefined object types: domains, events and policies. Additional functionality can be added to a system by means of new managed objects. To allow a dynamic interaction between heterogeneous IT-Environments, Ponder2 utilizes so called Self Managed Cells (SMC). SMCs represent independent systems, i.e. a set of hardware and software components that are able to manage themselves. The communication between the different components occurs through an content-based event bus. For the message exchange, the proprietary language PonderTalk is used. The Java objects implement an interface (Managed Object) that enables the mapping from PonderTalk to Java methods. For the communication between different systems, the Ponder2-Framework offers the possibility to access external Managed Objects. Thus, objects can be exported and imported between different SMCs.

To make the holistic system work, leveraging the *Ponder2* technology, several steps are necessary: a.) An event emitter has to be installed on the virtual machines - in our example, both on the mySQL VM and on the WebServer VM. Conceptually, it is an agent that monitors relevant metrics from the VMs (e.g., the CPU load) and triggers respective events. This design is very much in line with industrial cloud management systems, where a set of agents (for various purposes) are installed on top of the provisioned VMs. b.) The produced event has to be send to the TOSCA-container where an obligation policy interpreter receives the event and looks for applicable policies. This requires c.) correlating the event to a running instance and then identifying the model where this instance was derived from to get the policy. The policy then evaluate their conditions and invoke the corresponding actions (Keoh et al., 2007). Figure 7 illustrates how the concrete Ponder2 code looks like:

```
1  event:= root/factory/event create: #("name" "
       new" "old").
2  root/event at: "greenEvent" put: event.
3  policy:= root/factory/ecapolicy create.
4  policy event: root/event/greenEvent.
5  policy condition: [ :name :oldValue :newValue
       |
6  name == "Workload" & (new < 30) & (old >= 30)
       ].
7  policy action: [
8  // Execute shutdownClusterMember Plan to
       shutdown a cluster member].
9  root/policy at: "workloadLow" put: policy.
10 policy active: true.
```

Figure 7: Concrete LowWorkloadPolicy in PONDER2.

In Figure 7, a management plan called *shutdownClusterMember* is triggered by the policy, following the event-condition-action (ECA) rule used in policy systems. Figure 8 illustrates the plan: First the plan has to check whether the cluster manager is available - obviously, this is a safety measure. If successful, the plan issues a request against the cluster member to shut down. This step might require more detail, e.g., if it is a database we might want to stop the datasource first, to avoid data inconsistencies, but conceptually this is the step that needs to be taken. Once stopped, the cluster member can be removed from the cluster manager and thus will no longer consume energy.

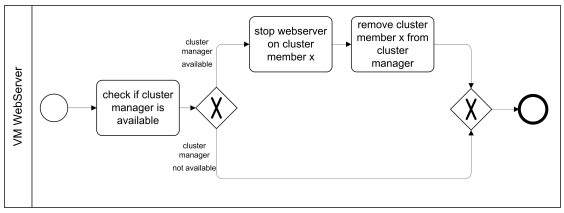


Figure 8: The *shutdownClusterMember* Plan.

A policy-based mechanism to shut down a cluster member to save energy is just one example how policies can be used to realize energy-efficient cloud services. Other use cases can be imaged: If clouds would have an energy certification, i.e. the data center is operated on renewable energy, a cloud service could declare a policy requiring such a certification, and if not fulfilled the service model is not allowed to be instantiated. Furthermore, policy-based scheduling or workload planning or other energy-saving, cloud service specific measures can be imagined.

6 CONCLUSION

In this paper, we have shown how to design, develop and run standardized, energy-efficient cloud services. We extended a TOSCA cloud management platform with policy handling parts. We showed how to combine TOSCA models with policies (in the policy language *Ponder2*). Thereby we realized description and execution of portable green cloud services. Ponder2 has been chosen as our policy language because of its dynamicity and versatility. We believe this work to be the foundation to combine two of the most important future markets of the IT industry. To our knowledge there is no work conducted to combine cloud services with Green IT aspects so far, so this work to combine cloud services with ecological aspects promises extraordinary potential for cloud service providers. With cloud computing in general becoming more and more a commodity, the offering of energy efficient services will become a competitive advantage as well as it will serve the environment.

This work was partially funded by the BMWi project Migrate! (01ME11055).

REFERENCES

Beloglazov, A., Abawajy, J., and Buyya, R. (2012). Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing.

Berl, A., Gelenbe, E., Girolamo, M. D., Giuliani, G., Meer, H. D., Dang, M. Q., and Pentikousis, K. (2010). Energy-efficient cloud computing.

Buyya, R., Beloglazov, A., and Abawajy, J. (2010). Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges. In *Int. Conf. on Parallel and Distributed Processing Techniques and Applications*.

Dargie, W. (2012). Analysis of the power consumption of a multimedia server under different dvfs policies. In *CLOUD*. IEEE.

Duy, T. V. T., Sato, Y., and Inoguchi, Y. (2010). Performance evaluation of a green scheduling algorithm for energy savings in cloud computing. In *Parallel & Distributed Processing, Workshops and Phd Forum*.

Hwang, C.-H. and Wu, A. (2000). A predictive system shutdown method for energy saving of event-driven computation.

IETF (2001). Ietf policy model. http://tools.ietf.org/html/rfc3060.

Keoh, S. L., Twidle, K., Pryce, N., Schaeffer-Filho, A. E., Lupu, E., Dulay, N., Sloman, M., Heeps, S., Strowes, S., Sventek, J., and Katsiri, E. (2007). Policy-based management for body-sensor networks. In *4th Int. Workshop on Wearable and Implantable Body Sensor Networks*. Springer Berlin Heidelberg.

Liu, Z., Lin, M., Wierman, A., Low, S., and Andrew, L. (2011). Geographical load balancing with renewables.

OASIS (2005). Xacml policy model extensible access control markup language (xacml) version 2.0, pp. 16-18.

OASIS (2013). Tosca - topology and orchestration specification for cloud application.

PONDER (2013). http://www.ponder2.net.

Sueur, E. L. and Heiser, G. (2010). Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proc. of the 2010 int. conf. on Power aware computing and systems*. USENIX Association.

Tonti, G., Bradshaw, J. M., Jeffers, R., Montanari, R., Suri, N., and Uszok, A. (2003). Semantic web languages for policy representation and reasoning: A comparison of kaos, rei, and ponder. In *The Semantic Web-ISWC 2003*. Springer Berlin Heidelberg.

Wang, L., von Laszewski, G., Dayal, J., and Wang, F. (2010). Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with dvfs. In *CCGrid*. IEEE/ACM.

Younge, A. J., von Laszewski, G., Wang, L., Lopez-Alarcon, S., and Carithers, W. (2010). Efficient resource management for cloud computing environments. In *Green Computing Conf.* IEEE.

Zhang, Y., Wang, Y., and Wang, X. (2011). Greenware: Greening cloud-scale data centers to maximize the use of renewable energy. In *Middleware 2011*.

A Template Description Framework for Services as a Utility for Cloud Brokerage

Li Zhang[1], Frank Fowley[2] and Claus Pahl[2]
[1]*Northeastern University, Software College, Shenyang, China*
[2]*IC4, Dublin City University, Dublin, Ireland*
zhangl@swc.neu.edu.cn, {cpahl, ffowley}@computing.dcu.ie

Keywords: Cloud Broker, Service Brokerage, Service Description, Reference Architecture.

Abstract: Integration and mediation are two core functions that a cloud service broker needs to perform. The description of services involved plays a central role in this endeavour to enable services to be considered as commoditised utilities. We propose a conceptual framework for a cloud service broker based on two parts: a reference architecture for cloud brokers and a service description template that describes the mediated and integrated cloud services. Structural aspects of that template will be identified, formalised in an ontology and mapped onto a set of sublanguages that can be aligned to the cloud development and deployment process.

1 INTRODUCTION

Organisations like Gartner and NIST (Gartner, 2012; NIST, 2013) have identified cloud service brokerage as a key concern for future cloud technology development and research. Cloud brokerage refers to the customisation, aggregation and integration of cloud services, possibly by third parties. Here, we investigate some conceptual problems towards the implementation of a cloud brokerage solution. Integration and mediation are two core functions that a cloud broker needs to perform. We propose a conceptual framework for a cloud broker based on two parts: a reference architecture for cloud brokers and a service description template that describes mediated, i.e. customised, aggregated or integrated cloud services.

Our conceptual reference architecture will extend the NIST cloud architecture specification, taking on board the NIST and Gartner categorisations. The NIST architecture is not sufficient as a common reference framework, as it does not detail how integration between services – horizontally between different providers and vertically between the different cloud layers – can be achieved. Equally do current service modelling and specification approaches (Pahl 2005) not cover adequately the vertical integration across the cloud layers infrastructure (IaaS), platform (PaaS) and software (SaaS) (Konstantinou 2009; Mietzner 2008; Rodero-Merino 2010). We define here a conceptual model for a service description template that would allow the two dimensions to be covered. Structural aspects of that template will be identified, formalised in an ontology and mapped onto a set of sublanguages that can be aligned to the cloud development and deployment process.

The paper is organised as follows. The next Section 2 defines the reference architecture. The following Section 3 introduces and defines the service template. As this concept plays a crucial role, it will be investigated in more depth. Furthermore, we outline a solution architecture for an implementation based on open-source cloud platforms in Section 4 and also discuss some trends, before ending with related work (Section 5) and some conclusions (Section 6).

2 ARCHITECTURE FRAMEWORK

As already explained, we propose a conceptual framework for an automated architecture for a cloud service broker based on two parts: a reference architecture for cloud brokers compliant with the NIST proposal and a service description template model that describes mediated cloud services. While descriptions for Web services exist in the format of WSDL, or any semantic extension like SAWSDL or WSMO, some specific requirements for the cloud context need to be addressed. Enabling cross-layer description of concerns and facilitating cloud provisioning and configuration is required (Fehling 2011). Service description templates

(and their instances) are the core artefacts that are used to describe, select, deploy and manage the different cloud resources, which are made available as services in an independent way.

2.1 Architecture and Broker Types

Figure 1 describes the high-level architectural setting. We follow Gartner here in the separation of integration, customisation and aggregation functions of a broker. The broker sits between the consumer and a range of different, independent providers. The role of the service templates is indicated:

- forming the basis of requests by consumers,
- collected from providers and made available for customisation, integration, aggregation by broker.

The detailed structure of the templates (operation, quality, resources, policies as the main description categories) will be explained on later. Note that while the structure of the reference architecture might be easier to agree upon as it brings expected features into a common framework, requirements for the service templates are more difficult to establish as they not only have to describe 'what' is in the architecture, but also 'how' the architectural framework can be converted into an effective solution. Cross-layer support in a federated context (i.e. vertical and horizontal integration support) is the critical challenge (Barrett et al., 2006; Buyya 2010). The reference architecture thus aims to provide integration in two dimensions:

- Horizontal integration across federated clouds
- Vertical integration across the cloud stack layers (IaaS, PaaS, SaaS)

We follow Gartner in their terminology. The Gartner categorisation is illustrated in Fig. 1. NIST follows a similar three-pronged classification. They define a cloud broker as an entity that manages the use, performance and delivery of cloud services and negotiates relationships between cloud providers and cloud consumers. According to NIST, a cloud broker can provide services in three categories:

- Service Intermediation: A cloud broker enhances a given service by improving some specific capability and providing value-added services to cloud consumers. The improvement can be managing access to cloud services, identity management, performance reporting, enhanced security, etc.
- Service Aggregation: A broker combines and integrates multiple services into one or more new services (Benslimane 2008). The broker provides data integration and ensures secure data transfer between consumer and multiple providers.
- Service Arbitrage: Service arbitrage is similar to service aggregation except that the services being aggregated are not fixed. Arbitrage means a broker has the flexibility to choose services from multiple agencies (Benson 2010). The cloud broker, for example, can use a credit-scoring service to measure and select an agency with the best score.

2.2 Cloud Broker Interfaces and Access

We envisage the cloud consumer to interact with the cloud broker in two ways. Firstly, request submission and cloud service construction: this is pre-deployment and provisioning. Secondly, service monitoring and management: this is post-deploying during the provisioning of the service. Fig. 2 illustrates the two layers:

- An Integration Layer allows the construction (aggregation) of a complex or customised cloud service based on individual, independent services. Data and information that is communicated needs to be integrated through appropriate mediation and transformation techniques. In addition, the integration of wider processes is often necessary, similar to work on document-based service process integration in frameworks such as ebXML.
- A Management Layer allows the monitoring and analysis of broker-provided services. The broker is often the direct contract partner, resulting in the need to deal with service-level agreements (SLAs) and billing/payment issues at that level. Monitoring provides the quality and usage data for both quality and payments concerns.

2.3 Broker Features and Requirements

In order to clarify the requirements for a service description notation for cloud service brokerage, we detail the functions we expect to be performed by the broker across the cloud layers IaaS, PaaS and SaaS – see Figure 3. For the pre-deployment, the following aspects are relevant. Integration, interoperability and portability are the key concerns to support the consumer in creating multi-vendor applications and to move or migrate between vendors (Bernstein 2009; Pahl 2013a; Pahl 2013b).

- Portability can be supported and automated at lower layers by adopting OVF or TOSCA – the latter a standard which specifically supports portability between public and private clouds.
- Expected workload is a concern that should be considered at design/built-time, as service selection and later provisioning need to take this into

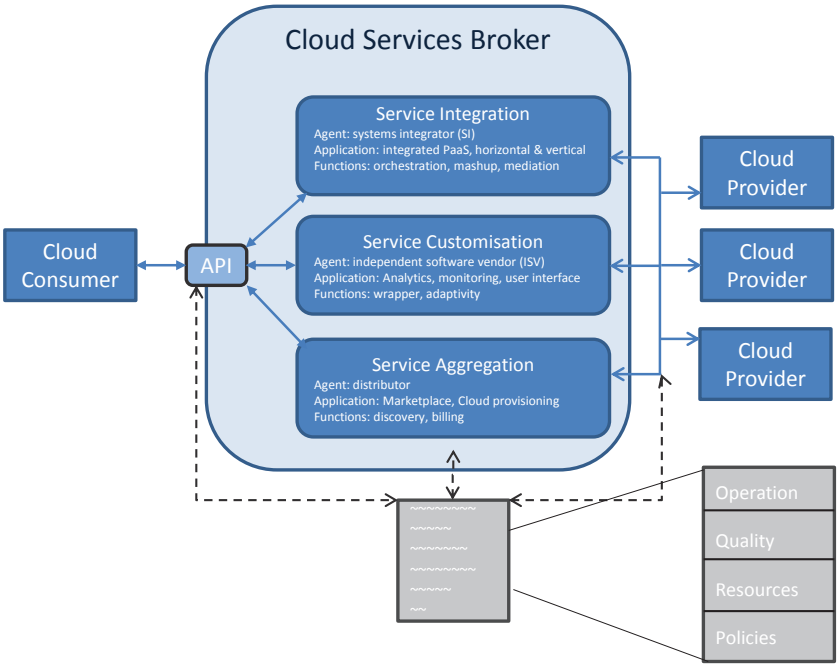


Figure 1: High-level Architectural Cloud Broker Setting.

account. This requires workload to be made explicit at all layers.

- Horizontal interoperability refers to the integration of services within a cloud layer, requiring common interfaces. Standards compatibility is a solution. OCCI is a lifecycle management standard describing APIs for service management.
- Vertical interoperability requires the integration of infrastructure, platform and application needs. Key performance indicators (KPIs) differ across the layers – while at lower infrastructure layers network bandwidth and latency are concerns, further up it is response time or availability per service. However, higher-level ones need to be mapped down to lower layers, and vice versa. Thus, formalised mappings need to be defined.

We see the vertical cloud stack as an implementation of layering as a SOA principle, resulting in a reference architecture with applications (SaaS in cloud), middleware (PaaS in cloud) and infrastructure services (IaaS in cloud). Critical is in vertical cloud stack is the integration of non-functional properties (QoS), i.e., mappings between the layers. For SaaS and PaaS, a relevant metric is for instance the service response time. For IaaS, metrics are based on compute (CPU load), storage (data size) and network (throughput or latency). While not covered here in detail, mappings between layers can be statically fixed. As input, we can used our work on predicting and deriving e.g. PaaS-response time from IaaS-metrics like CPU utilisation or network bandwidth (Zhang, 2013).

For post-deployment management, life cycle support is crucial, covering the configuration and provisioning as well as on-going monitoring and analytics.

- Resource need to be configured, based usually on quality requirements such that the resources match the needs of the consumer. Selection, matching and configuration support is necessary.
- The user requirements and the resulting abstract configuration have to be correlated with the available resources provided at the time of provisioning the services. In addition to the configuration concerns, this requires lower-level input from resource managers and workload balancers.
- Monitoring is an essential part of a provisioning platform. Monitoring solutions are available – commercial and open-source. However, due to the requirement of supporting federated clouds, the need to collect and integrate monitored data from different sources arises.
- Like monitoring, the analysis of data gathered requires interoperability between monitoring sources and the analysis tool. The requirements are similar with respect to standardisation and compatibility, e.g. SLA-related data.

This discussion provides a first requirements list for the service template, discussed in the next section.

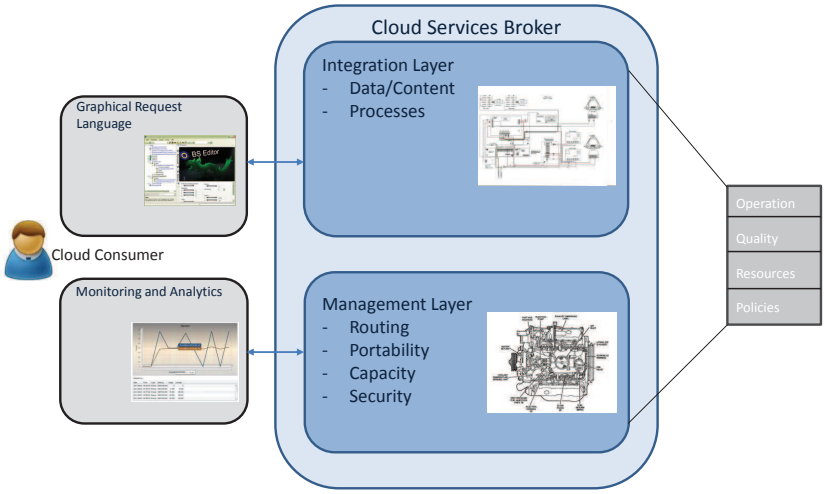


Figure 2: Cloud Broker Interfaces and Access to Functions.

3 SERVICE DESCRIPTION TEMPLATES – STRUCTURE AND FORMALISATION

A service template is an abstract description of a service's capabilities, in terms of functional and non-functional aspects. Different types of templates to describe the capabilities of cloud resources have been proposed, often under a different name such as Recipes (Chef, Cloudify) (Cloudify, 2012), Blueprints (4CaaSt) (4Caast, 2013) or Manifests (OVF, CompatibleOne) (Compatible One, 2013). In Section 5, see Fig. 7 below, we summarise the language definition challenges and requirements presented in (Pahl et al, 2013). We categorise the service description languages, such as (CloudFoundry, 2013; DeltaCloud, 2013; Jclouds, 2013; Libcloud, 2013; Mosaic, 2013; OpenShift, 2012; Optimis, 2013; simpleAPI, 2013), in terms of the cloud layer support and specific features they provide. We categorise the solutions along the following concerns, which form requirements for a template solution:

- System Type: Multi Cloud API, IaaS Fabric Controller, Open PaaS Solution, Open PaaS Provider.
- Distribution Model: Open Source (for all solutions considered)
- Core Capabilities: Multi-IaaS Support, Multi Language/Framework Support, Multi Stack
- Core Features/Components (development and deployment time): Service Description Language, Native Data Store, Native Message Queue, Programming Model, Elasticity & Scalability, QoS/SLA Monitoring.
- Advanced Features/Components: Service Discovery/Composition, Broker, Marketplace – towards broker and marketplace features.

We can see a trend towards the upper broker and marketplace solutions (Mietzner 2008; Rodero-Merino 2010). These demonstrate to some extent a multi-faceted description with functional and quality aspects. However, a comprehensive coverage is not available. A richer description notation is needed, resulting in additional complexity due to the required multi-layer support, leaving both vertical integration and comprehensive to be addressed.

3.1 Template Structure

We propose a 4-part structure for the service description template: operation, quality, resources and policies. The first two cover service-internal aspects – the functional and non-functional aspects of the service. The remaining two cover more the service-external perspective with the resources required and also the security and compliance requirements. We identify a 4-part hierarchy of concepts, which we will later formalise as the core taxonomy of a service description ontology (Pahl 2007; Papazoglou 2011b).

- Operation – the functional service interface. Its subconcepts follow WSDL in its structure:
  - Service type/interface defines the interface as an abstract data type
  - Operations that provide service functionality
  - Messages and Types to communicate data to and from the service for operation invocations
- Quality – the non-functional properties. Its subconcepts are performance-related concerns:

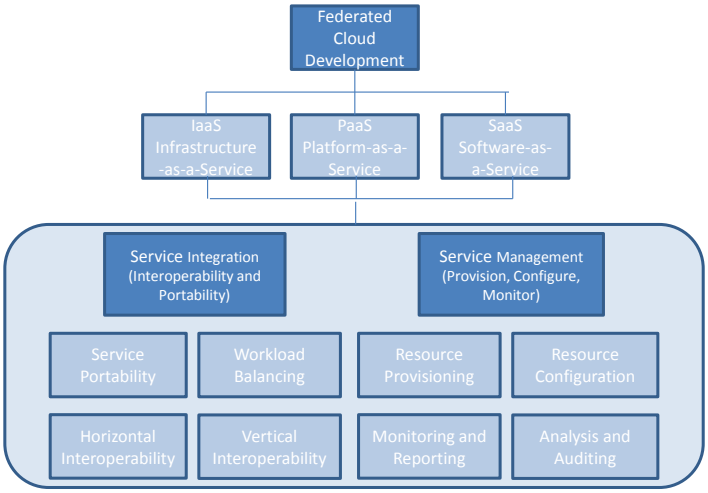


Figure 3: Cloud Broker Features and Capabilities.

  - Availability of the service in question
  - Latency/Response Time of service/operations
  - Bandwidth as another quality concern
- Resources – deployment requirements. Its subconcepts are aspects specific to the provisioning of a service in a PaaS or IaaS environment:
  - Average/peak workload requirements is a typical example for a configuration requirement
- Policies – the business agreements and constraints. Its subconcepts are:
  - Security and Privacy concerns in terms of confidentiality, integrity or authenticity
  - Compliance and Governance as further rules to describe to correct alignment of a service to its execution environment

3.2 Formalisation

We propose to formalise the service template as an ontology. An ontology is a conceptual model formulated in a formal description language. We utilised in this context the core of the Semantic Web technology, the Web Ontology Language OWL. This language is based on description logics, i.e. has a formal definition and is supported by various tools including reasoners that can even be used at runtime. Using semantic technologies brings us closer to interoperability of meaningful descriptions in an Internet context.

An ontology is a representation of concepts (or classes) that are defined in terms of their properties, i.e. relationships to other concepts. Concepts can be instantiated into concrete values. Properties can be object or data valued, depending on whether they refer to another complex object (concept – an example is the relationship of a service to another service through invocation or inheritance) or data (an example is the performance of a service as a numeric value). Concepts can be arranged into a hierarchy using a sub/superconcept relationship. This forms the taxonomy of the ontology. Our ontology has a root concept, which is 'ServiceTemplate'. Using an 'isPartOf' relationship, the aspects Operation, Quality, Resources and Policies are connected to ServiceTemplate. Individual concerns, such as ServiceType or Message for Operation or Availability for Quality, are the subconcepts of the respective four core aspects. These properties define a basic hierarchy. However, we need a richer ontology that addresses the requirements outlined earlier, see Figure 4.

- In terms of interfaces and operations, we need to distinguish the different layers – as ontology subconcepts of the Operation concept. For IaaS-Operation, we define a service API with the corresponding operations that is OCCI-compliant. While VM image management is relatively uniform (thus, OCCI is possible as a standard), at the PaaS layer, we need to provide for standard tools like databases or application server. For instance a Tomcat server could be modelled as an instance of a Web application server concept. Standardised solutions in terms of APIs/operations would then only be specific to a tool category.
- Quality concerns (KPIs for the different layers) need to be differentiated. We assume each quality criterion to be qualified by an attribute (data-valued property) that indicates the relative cloud layer to which it applies.

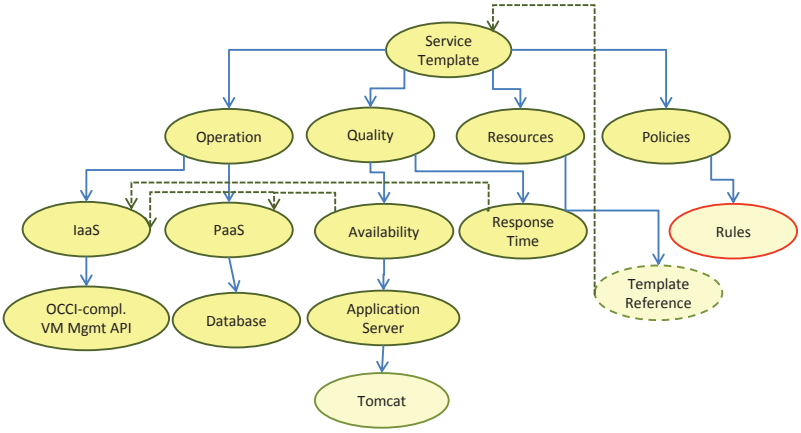


Figure 4: Service Template Ontology.

- Required resources can be explicitly or abstractly specified by referring to other templates that capture the required resource. This is facilitated by a special resources relationship. The need for formality becomes clear as circular resource relationships cannot be tolerated and need to be detected.
- The Policies part needs to link to a special kind of entities: rules. We propose to utilise the SWRL language (Semantic Web Rule Language). The specified rules can be evaluated at build time and dynamically verified.

The ontology is meant to be extensible for particular circumstances. For concrete resources, instances are attached. This clarifies the definition of a template – a template is concept-level in terms of the ontology and can be instantiated by providing instance-level values.

Templates can be generated from other resources and provide input to further processing. Based on SLAs or other legal agreements, some operational, quality, and policy requirements can be derived that would define a partial template. Service integration plans need to be generated in many cases where a single service will not satisfy the need. A service process needs to be orchestrated in a classical Web service manner, associated to an integration process connecting abstract resources.

3.3 Sublanguages

While we have proposed a comprehensive ontology capturing all conceptual aspects of the service template language framework, in order to tailor this to the needs of different actors and components of the reference architecture, a facilitation through different languages would be beneficial. As core technical languages (Figure 5) we can identify:

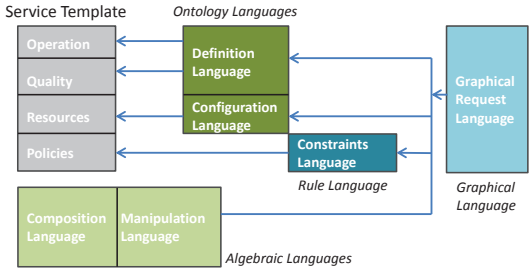


Figure 5: Service Template Languages Framework.

- Service Template Definition Language (Operation and Quality) addresses service internal aspects
- Service Template Configuration/Instantiation Language (Resources) addresses first type of external aspects
- Service Template Constraints Language (Policies) also addresses external aspects, but, as explained, a rule language rather than an ontology language needs to be facilitated here.

These core languages can be complemented by two template manipulation operators – core composition and non-compositional manipulation (Benslimane 2008), both possibly supported by a graphical request language:

- Service Template Composition Language (Integration Processes) is singled out to deal with service aggregation by supporting a range of standard composition operators to allow services to be orchestrated into a process across different cloud providers (Pahl 2007).
- Service Template Manipulation Language provides non-compositional template manipulation. This could include the refinement/specialisation of a template, or the merger or restriction.

- Graphical Request Specification Language. Requests for a specific service are expressed by end-users. To facilitate access for different users with different degrees of technical knowledge, we suggest a graphical request language visualising the template structure.

The implementation shall follow the ontology operator calculus for architecture manipulation from (Pahl 2007) on ontology-based architecture and pattern languages that combine composition and abstraction mechanisms. The conceptual core of the languages, particularly for the composition and manipulation language is an operator calculus, including operations such as match/select, merge, restrict, refine, behaviourally compose, abstract/instantiate, to support the following core manipulation and composition activities:

- Refinement, i.e. the derivation of a related service description that preserve functional properties (the scope of the service can be tailored through restrict and union operations).
- Instantiation, i.e. providing concrete values for an abstract, concept-level template in order to denote a concrete service.
- Composition, i.e. the use of aggregation operators such as sequence, selection and iteration to orchestrate a selection of services to form a composite service process serving a more complex goal.

4 ARCHITECTURE AND VALIDATION

An implementation architecture for the service templates is the brokerage test-bed currently developed at IC4 (a cloud research centre at our university), where a combination of OpenStack as the IaaS solution and CompatibleOne as the core PaaS broker solution is being developed. This serves us as an implementation to investigate the vertical integration of IaaS, core PaaS and broker layers. While CompatibleOne is a broker, the support implemented does not satisfy the requirements of our needs here – see Sections 2.1 for a CompatibleOne review and 2.3 for a requirements discussion. The following architecture (Fig. 6) describes the architectural setup of OpenStack and Compatible One that we propose to be used as the experimentation environment.

CompatibleOne (Compatible One, 2013) is an academic and industry consortium that set out to provide a framework to allow for portability and interoperability of cloud software. The project focusses on producing a broker platform following the Gartner

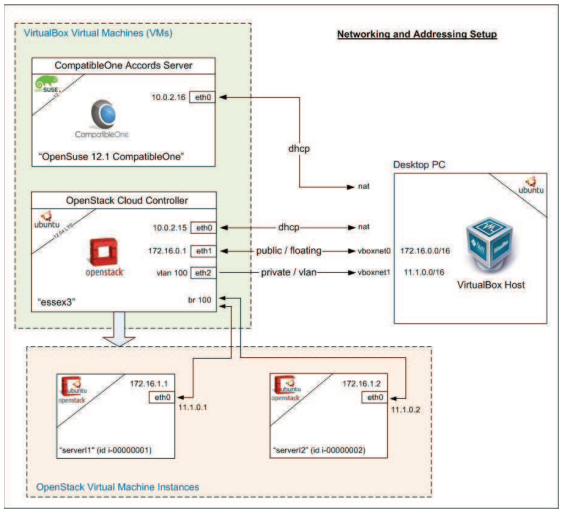


Figure 6: OpenStack and CompatibleOne Architecture

definition of a cloud broker to provide Intermediation, Aggregation and Arbitration of cloud services. Such a broker could allow for cloud service consumers requirements to be mapped to cloud vendors' resources. The project is committed to the OCCI standard. To date, the project has produced the CORDS information model and the ACCORDS software platform.

- The CORDS model is as an open, object-oriented, OCCI-compliant representation of cloud computing entities and can be used to build complex cloud middleware.
- The ACCORDS platform is a suite of cloud middleware programs that provide broker services. Using ACCORDS, a customer can describe service requirements in a CORDS-compliant format. These can be matched to available provider offerings and then ultimately provisioned at run-time by interacting with the actual vendor resources.

The model includes representations for service level agreements, pricing structures and transaction management for usage billing.

OpenStack (OpenStack, 2012) is a cloud enabling solution that transforms a physical hardware infrastructure into an Infrastructure as a service (IaaS). The platform is a Cloud Platform or IaaS Cloud Fabric controller. The platform can be used to run cloud compute and storage infrastructure. The OpenStack open source siolution makes its services available through Amazon EC2/S3 compatible APIs and hence client tools written for AWS can be used with OpenStack clouds. Five main services are exposed by OpenStack: Compute (Nova), Storage (Swift), Imaging (Glance), Identity (Keystone) and Network management (Quantum).

The adequacy of the proposed service template

model can be validated by discussing some scenarios as implementations in the above testbed architecture:

- Customisation, Aggregation, Integration: these are the three broker types, which need to be supported by adequate operators.

  We have proposed an ontology-based operator calculus that provides suitable support. Refinement allows customisation. The composition operators that we propose allow aggregation. And integration is facilitated by allowing semantic integration through a rich ontology and a rules-based approach to control the integration into specific environments. Thus, specifically the manipulation and composition operators are necessary to fully support service brokerage. An ontology-based solution helps with integrating multi-facetted aspects into a coherent manipulation and reasoning framework.

- CORDS and ACCORDS integration: the CORDS model with its OCCI-based description layer provides a standards-based platform to enable utilisation in a concrete OCCI-compliant IaaS solution.

  The CORDS model relating to the ACCORDS platform features can be fully represented in terms of the service template notation. ACCORDS services are instantiated service templates. The OCCI-compliant OpenStack is the execution platform. With CompatibleOne, we have used one of the most advanced broker platforms available (we discuss 4CaaSt, another advanced solution, in the next section) based on our comparison in Fig. 7 earlier. Thus, we can be confident that the vertical layering problem can be addressed for a rich, multi-facetted description template.

While we have not fully implemented the proposed service template model as an extension of the CORDS matching, our experiments with CompatibleOne support the feasibility of a full integration of the two cloud solutions. The experiments validate the scope that a service description template has to cover for vertical integration as proposed in Section 4.

5 RELATED WORK

We select 13 specification notations here for a more in-depth description including (CloudFoundry, 2013; DeltaCloud, 2013; Jclouds, 2013; Libcloud, 2013; Mosaic, 2013; **?**; OpenShift, 2012; Optimis, 2013; simpleAPI, 2013). Fig. 7 summarises a range of solutions ordered from bottom to top based on the focus in the cloud technology stack (from basic IaaS to advanced SaaS marketplace support). We compare these in terms of their capabilities and components as comparison categories as discussed in Section 3. A more detailed, general discussion can be found in (Grozev, 2012), which also covers systems like mOSAIC or jclouds. Our work here is not specific to any of the intercloud architecture types identified there. Our discussion is also more specific to languages and service description, whereas (Grozev, 2012) covers wider concerns like location-awareness or pricing.

An ontology-based foundation enables reasoning support. It also acts as an integration, e.g. via ODM framework with MOF/UML-style notations and via the predicate logic link to other logic-based languages. In this respect, our template approach is a step ahead compared with e.g. the Blueprint approach of 4CaaSt or the recipes techniques in Cloudify. Ongoing work in the EU Broker@Cloud project uses an extension of the RDF-based version of USDL (Kourtesis, 2013), which demonstrates the relevance of semantic technologies for integration and reasoning across platforms, but here our full OWL goes beyond the RDF-based USDL formalisation.

The two most advanced ones in terms of brokerage – 4CaaSt and CompatibleOne – shall be discussed. The 4CaaSt Blueprint approach (Nguyen 2011; Papazoglou 2011b) supports through its different languages a range of cloud brokerage activities. Blueprints are initially abstract specifications, which can be resolved, i.e. mapped to lower cloud stack layers until the specifications are finally deployed using a standard configuration and deployment manager. Resources and Services are described in a Blueprint (BP), which is an abstract description of what needs to be resolved into infrastructure entities. BPs are stored and managed in a BP repository via a REST API. A BP is resolved when all requirements are fulfilled by another BP, via the Resolution Engine. This is a service orchestration feature. A number of XML blueprint languages are supported: BDL Description Language (for cloud operations and capabilities), BCL Constraint Language (for SLA parameters definition), BML Manipulation Language (for operations, e.g. service match, merge, compose, delete), BRL Request Language (for user and developer request definitions). Blueprints are suitable to support simple marketplace offering single, isolated services as a product, but lack the range of brokerage activities in their toolkit. This entails a further complication – compositional brokerage needs to be mapped down the layer, which is non-trivial as particularly quality aspects are non-compositional in nature.

The CompatibleOne manifests suffer from similar limitations. The units of a CompatibleOne Service Manifest are Image and Infrastructure. An Im-

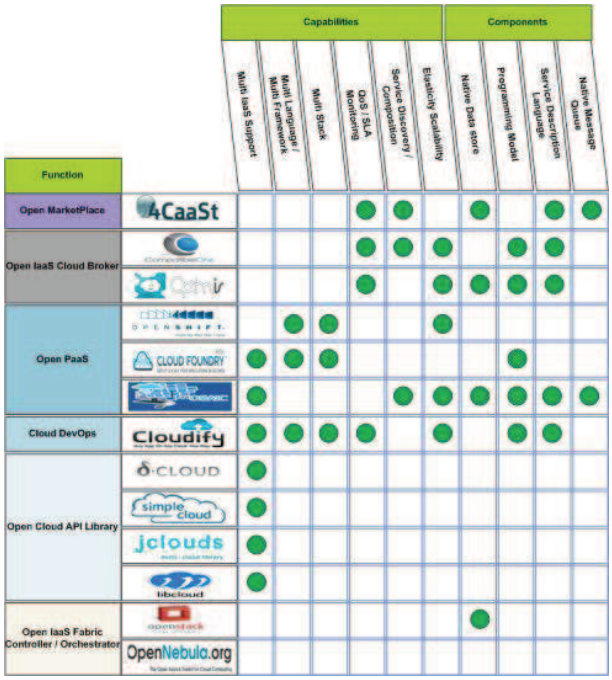


Figure 7: Summary Service Description Comparison.

age consists on a System (the base OS) and a Package (reflecting the software stack configuration). Infrastructure covers the classical IaaS concerns Storage, Compute and Network. An Image is a description of a manual build of an application, used by development and configuraton tools (devops, e.g. puppet or chef). An Image has an agent that is embedded in a VM and runs on startup. Agent is a script to run required configurations, set up monitoring probes, or download required components. Cordscript is proprietary CompatibleOne scripting language for expressing configuration actions. These support metadata handling within VMs, managing required linkages between VMs, and setting up monitoring probes in VMs. Cordscript is a statement-oriented language. A feature under development in CompatibleOne is a cloud-implementable graph of OCCI category instances. Particularly well solved is the standards-compliancy at the lower layer (making it a good testbed), but the manifests only cover core concerns.

The consumer expresses provisioning requirements by creating a CORDS manifest. This contains attributes like ID and name. Key element is a provisioning plan involving nodes and regions affected. The node configuration topology includes image and infrastructure requirements (compute, storage, network). Actions for configurations and releases can be specified. Other information concerns a) account, pricing and invoicing and b) security settings. CORDS is configuration, provisioning and deployment oriented, where our template adds quality beyond security and governance through policies. The templates can be seen as higher-level (abstract) specifications on top of e.g. a CORDS manifest.

6 CONCLUSIONS

Standards and compatibility are effectively the only way to make a truly interoperable cloud service broker work. We can note a trend towards cloud marketplaces utilising brokerage facilities. The proposed solution in this paper is a first step in this direction that needs to facilitate language support in terms of request specification languages and also template manipulation operators that will support central marketplace functions (like matching and selection) – based on a the reference architecture is the other solution that defines how architectural components interact and how the service templates are processed in this context. Abstract descriptions of cloud services are the key to a service brokerage solution allowing to consider services as commoditised utilities that can be brokered. We presented a conceptual model for the service templates. We have defined the core properties as an ontological model. We have suggested a range of languages to support this feature.

The proposed OpenStack and CompatibleOne ar-

chitecture should serve as an adequate basis to develop a solution based on the languages for service templates as broker implementation, which remains future work. On the theoretical side, the full specification of the operator calculus needs to be completed, but can follow (Pahl 2007) here.

ACKNOWLEDGEMENTS

This research has been supported by the Irish Centre for Cloud Computing and Commerce, an Irish national Technology Centre funded by Enterprise Ireland and the Irish Industrial Development Authority.

REFERENCES

4Caast. 4CaaSt PaaS Cloud Platform. http://4caast.morfeo-project.org/. 2013.

R. Barrett, L. M. Patcas, C. Pahl and J. Murphy. Model Driven Distribution Pattern Design for Dynamic Web Service Compositions. International Conference on Web Engineering ICWE'06. ACM Press. 2006.

T. Benson, et al. Peeking into the Cloud: Toward User-Driven Cloud Management. Clouds 2010, 2010.

D. Benslimane, S. Dustdar, A. Sheth. Services Mashups: The New Generation of Web Applications. Internet Computing, vol.12, no.5, pp.13-15, 2008.

D. Bernstein et al. Blueprint for the Inter-cloud: Protocols and Formats for Cloud Computing Interoperability. Intl Conf Internet and Web Appl and Services. 2009.

R. Buyya, R. Ranjan, R.N. Calheiros. Intercloud: Utility-Oriented Federation of Cloud Computing Environments For Scaling of Application Services. Intl Conf on Alg and Arch for Parallel Processing. 2010.

Cloud Foundry. Open Source PaaS Cloud Provider Interface. http://www.cloudfoundry.org/. 2013.

Cloudify. Cloudify Open PaaS Stack. http://www.cloudifysource.org/. 2012.

CompatibleOne. Open Source Cloud Broker. http://www.compatibleone.org/. 2013.

DeltaCloud. Deltacloud REST cloud abstraction API. http://deltacloud.apache.org/. 2013.

C. Fehling, R. Mietzner. Composite as a Service: Cloud Application Structures, Provisioning, and Management. IT - Information Technology: Vol. 53, No. 4, pp. 188-194. 2011.

F. Fowley, C. Pahl and L. Zhang. Cloud Brokerage Architecture – State-of-the-Art and Challenges. Workshop on Cloud Service Brokerage CSB2013. 2013.

Gartner - Cloud Services Brokerage. Gartner Research. http://www.gartner.com/technology/research/cloud-computing/cloud-services-brokerage.jsp. 2012.

N. Grozev and R. Buyya. Inter-Cloud architectures and application brokering: taxonomy and survey. Softw: Pract. Exper.. doi: 10.1002/spe.2168. 2012.

Jclouds. jclouds Java and Clojure Cloud API. http://www.jclouds.org/. 2013.

A. V. Konstantinou et. al. An Architecture for Virtual Solution Composition and Deployment in Infrastructure Clouds. 3rd International Workshop on Virtualization Technologies in Distributed Computing. June 2009.

D. Kourtesis, K. Bratanis, A. Friesen, Y. Verginadis, A.J.H. Simons, A. Rossini, A. Schwichtenberg, P. Gouva. Brokerage for Quality Assurance and Optimisation of Cloud Services: an Analysis of Key Requirements. ICSOC Cloud Service Brokerage Workshop CSB2013. Springer. 2013.

Libcloud. Apache Libcloud Python library. http://libcloud.apache.org/. 2013.

R. Mietzner, F. Leymann, M. Papazoglou. Defining Composite Configurable SaaS Application Packages Using SCA, Variability Descriptors and Multi-tenancy Patterns. 3rd Internet and Web Appl and Services. 2008.

Mosaic. mOSAIC Multiple Cloud API. http://www.mosaic-cloud.eu/. 2013.

NIST Cloud Computing Reference Architecture. NIST. http://www.nist.gov/customcf/get_pdf.cfm?pub_id=909505. Sept. 2011.

D.K. Nguyen, et al. Blueprint Template Support for Cloud-Based Service Engineering. In Proceedings of the Service-Wave'11, Poznan, Poland, October 2011.

3 OpenNebula. OpenNebula - Open Source Data Center Virtualization. http://opennebula.org/. 2012.

OpenShift. Cloud computing platform as a service. https://openshift.redhat.com/. 2012.

OpenStack. OpenStack Open Source Cloud Computing Software. http://www.openstack.org/. 2012.

Optimis. Optimis - Optimized Infrastructure Services. http://www.optimis-project.eu/. 2013.

P. Jamshidi, A. Ahmad, C. Pahl. Cloud Migration Research: A Systematic Review IEEE Transactions on Cloud Computing. 2014.

C. Pahl, S. Giesecke and W. Hasselbring. Ontology-based Modelling of Architectural Styles. Information and Software Technology (IST). 1(12): 1739-1749. 2009.

C. Pahl, H. Xiong. Migration to PaaS Clouds - Migration Process and Architectural Concerns. IEEE 7th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems MESOCA 2103. IEEE. 2013

C. Pahl, H. Xiong, R. Walshe. A Comparison of On-premise to Cloud Migration Approaches. European Conference on Service-Oriented and Cloud Computing ESOCC 2013. Springer LNCS. 2013

M. P. Papazoglou, W.J. van den Heuvel. Blueprinting the Cloud. IEEE Internet Computing, November 2011.

L. Rodero-Merino et al. From Infrastructure Delivery to Service Management in Clouds. Future Generation Computer Systems, vol. 26, pp. 226-240. Oct. 2010.

simpleAPI. Simple API for XML. http://en.wikipedia.org/wiki/Simple_API_for_XML. 2013.

L. Zhang, B. Zhang, C. Pahl, L. Xu, Z. Zhu. Personalized Quality Prediction for Dynamic Service Management based on Invocation Patterns. Intl Conference on Service Oriented Computing ICSOC 2013. 2013

Performance Prediction for Unseen Virtual Machines

John O'Loughlin and Lee Gillam
Department of Computing, University of Surrey, Guildford, GU2 7XH Surrey, U.K.
{john.oloughlin, l.gillam}@surrey.ac.uk

Keywords: Cloud Computing, Performance Prediction, Virtualisation, Scheduling.

Abstract: Various papers have reported on the differential performance of virtual machine instances of the same type, and same supposed performance rating, in Public Infrastructure Clouds. It has been established that instance performance is determined in large part by the underlying hardware, and performance variation is due to the heterogeneous nature of large and growing Clouds. Currently, customers have limited ability to request performance levels, and can only identify the physical CPU backing an instance, and so associate CPU models with expected performance levels, once resources have been obtained. Little progress has been made to predict likely performance for instances on such Public Clouds. In this paper, we demonstrate how such performance predictions could be provided for, predicated on knowledge derived empirically from one common Public Infrastructure Cloud.

1 INTRODUCTION

Infrastructure as a Service (IaaS) Clouds offer resources such as object storage, attachable storage, configurable networks and virtual machines (VMs) to customers over the internet, and are available to anybody with an internet connection and a credit card. Resources are provided on-demand, meaning they are provisioned at the time of the request, and can be 'terminated' when no longer needed. Together with an 'illusion of infinite capacity' (Armbrust et al., 2009) publically available computing environments allow customers to grow and shrink their usage as required and when required, and customers can be charged, typically, on a per-hour basis for resources used. The provider's ability to support this is referred to as elasticity; it is this that differentiates Clouds from dedicated hosting environments where servers can take up to a day to provision, are charged on a per-month basis, and require customers to enter into contracts for a pre-specified period of time irrespective of usage.

Infrastructure Clouds are the first successful attempt at public delivery of compute resources, and naturally there are concerns about migrating on-premise systems to them. Such concerns include: security, availability, vendor lock-in and performance; this latter concern of performance, in particular compute performance, is the focus of this paper.

As the largest, and arguably the de facto standard, of Infrastructure Clouds, performance of instances on EC2 has been widely studied, although mainly for its suitability for HPC and scientific computing. Consequently, differential performance of instances of the same type has been reported on by a number of authors, (Phillips, Engen, and Papay, 2011, and Ou et al., 2012) and it is now established that performance variation is due to the heterogeneous nature of large and growing Clouds. Although EC2 acknowledges heterogeneity, claiming that instances of the same type should deliver a 'consistent and predictable' level of compute (Amazon EC2 FAQS, no date) – despite the published evidence to the contrary– is certainly not ideal.

As users scale their infrastructure, knowing how many instances they should start in order to meet their needs requires an understanding of the deliverable performance. At present users are, potentially, in the unsatisfactory position of either (1) having to guess how many instances to start in order to obtain the desired number of satisfactory ones, with obvious cost implications, or (2) obtaining a number of instances that might not deliver the required performance. EC2 customers who are building services for their own customers on top of these instances may run the risk of failing any SLAs they provide unless they undertake rigorous performance evaluations themselves.

In this paper, we consider the problem of determining the expected performance of a set of instances before they are provisioned. The rest of this paper is structured as follows: Section 2 provides background information on Cloud structure, and in section 3 we discuss CPU model distribution in a number of AZs as is relevant to the discussions in this paper. In section 4 we describe our AZ model, and Sections 5-7 discusses simulations of VM offerings. In section 8 we compare our simulated data with real data from Amazon's "us-west-1c" zone. In section 9 we discuss assumptions made in our simulations and how this relates to other scheduling algorithms one could use. In section 10 we review related work, and section 11 concludes and discusses future work.

2 CLOUD INFRASTRUCTURE

EC2, Google Compute Engine (GCE), HP Public Cloud and Rackspace OpenCloud (AWS Global Infrastructure, no date, Google Cloud Platform, no date, HP Public Cloud, no date and Rackspace Global Infrastructure, no date) structure their Clouds into Regions and Zones; and all define these terms in essentially the same manner: Regions are dispersed into geographic areas consisting of Zones, and each Zone is presented as an isolated location. Zones have separate power and network connections and are tied together with high speed interconnects. Consequently, failure of one Zone should not, in theory, disrupt operations in another. EC2 refer to Zones as AZs and we will use these terms interchangeably.

There exist some operational differences between Regions/Zones in different Clouds. For example, EC2 will not automatically replicate any resources placed in one Region into another, whilst GCE state that they make 'no guarantee that project data at rest is kept only in that Region'. However, for the purposes of our present discussion these operational differences are not important.

Clouds offer VMs in a range of sizes known as instance types, with similar types grouped into instance classes. Instances types within the same class typically have the same ratio of quantity of CPU cores to RAM.

Providers give their instance types a performance rating, which expresses the compute power an instance's vCPUs should deliver, with a vCPU rating multiplied by the number of vCPUs offered to provide an overall rating for the type. On Amazon's Elastic Compute Cloud (EC2), this is called the EC2 Compute Unit (ECU), whilst Google Compute Engine has the Google Compute Engine Unit (GCEU). Such ratings should guarantee, if not identical, then certainly very similar, levels of performance of instances of the same type.

However, we have previously shown (Author1 and Author2, 2013) that m1.small instances on EC2 backed by a Xeon E5430 will run a bzip2 compression benchmark on average in 445s, whilst instances backed by a Xeon E5507 will take on average 621s to run the same benchmark. This is a 39% increase in time taken for instances backed by Xeon E5507 compared to the Xeon E5430. These instances are 'rated' the same and the user is charged the same price for each, but a performance-based pricing would clearly be expected to distinguish these.

By a heterogeneous Cloud we mean one where instances of the same type may run a variety of underlying hardware (which we refer to as hosts). Of the providers considered, EC2 is the only one which acknowledges a heterogeneous infrastructure, stating: 'EC2 is built on commodity hardware, over time there may be several different types of physical hardware underlying EC2 instances' (Amazon EC2 FAQs, no date). In such an environment performance variation is to be expected. As Amazon adds new Regions into EC2, and new Availability Zones (AZ) into existing Regions performs hardware refreshes, and as CPU models are retired by manufacturers, heterogeneity is seemingly inevitable. One would assume that, over time, any successful Cloud provider is will end up with a heterogeneous environment.

We have however found a stable association between hardware, as identified by CPU model, and instance classes. That is, an instance of a given class only runs on particular set of hardware. For example, First Generation Standard (FGS) instances run on hardware with Intel Xeon E5430, E5-2650, E5645 and E5507 CPUs. Over the 6 month period, in which we have been running various performance experiments on EC2 (April to September 2013), these sets and associations have not changed.

We would of course expect change over time, for the reasons discussed above. Indeed, in Ou, et al. (2012), the authors find Intel Xeon E5430, E5645, E5507 and AMD Opteron 270 backing FGS instances. They note that the AMD model is present less often in 2012 than compared to results they obtain in 2011. We also find that the hardware associated to different instance classes is distinct from each other. We would assume that hardware associated to different classes has either different

performance characteristics, or a different ratio of CPU cores to RAM. This allows instances from different classes to be differentiated along these lines. Thus, in addition to FGS instance class we have High CPU, High Memory and Second Generation Standard Instance classes. We may logically view AZs as being partitioned into hardware platforms associated with instance classes that are available there.

Amazon is currently the only provider to publish specific CPU model information associated with certain instance types. Generally these are specialised high-performance instance types such as Cluster Compute with 10Gb networking, or GPU instances. These types are only available, at the time of writing, in a limited number of Regions. The recently introduced High Memory class (m3.xlarge and m3.2xlarge) is the only general purpose class of instances globally available on EC2 whereby the CPU model is provided as part of the instance description. Whether EC2 will continue this trend; and specify CPU model for all new, and possibly existing, types, or find that heterogeneity is also inevitable for these new instance types and remove specific information, remains to be seen.

Dividing a Region into AZs, and encouraging their use for load balancing and high availability, tends to suggest that the resources available in the various AZs are likely to be broadly similar. However, we have experimentally determined that the proportions of CPU models backing instances are quite different in different AZs. Indeed, with heterogeneous hardware in the same Zone, even Zone-specific load balancing, and related scalability matters, may necessitate further thought.

In the next section, we present some experimental results showing the distribution of CPU models found in a number of AZs.

3 CPU PERFORMANCE AND DISTRIBUTION IN EC2

In April 2013 (Author1 and Author2 2013), we benchmarked the compute capacity of close to 1300 instances on EC2, across 14 AZs in 6 (out of the 8) Regions. We included 11 (non-specialised) instance types across 4 instance classes: First Generation Standard, High CPU, High Memory and Second Generation Standard. The image id backing the instances was ami-a73264ce which is a para-virtualised image. As such the cpuid instruction (used by the Linux kernel to populate /proc/cpuinfo) does not trap to the hypervisor. Through cpuid we identify the CPU model underlying an instance. And so can determine CPU model to instance class associations, and also the proportions of CPU models found in a given AZ.

We measure an instance's compute capacity by timing the compression of an Ubuntu 10.04 desktop image ISO with bzip2, a compression tool found on Linux systems. Bzip2 is a compute bound application and forms part of the SPECint 2006 CPU benchmark suite. In Table 1 below we record the benchmark times of 540 m1.small instance, broken down by CPU model backing them.

Table 1: Bzip2 compression times on m1.small instances.

Statistic	E5430	E5-2650	E5645	E5507
Mean(s)	445	470	510	621
Sd(s)	14	13	11	28
Min(s)	425	443	488	578
Max(s)	482	519	543	716

From Table 1 we see that, for this particular task, it is clearly preferable to obtain an E5430 than a E5507. These results concur with related work (reviewed in Section 9), which shows that an instance's (compute) performance is primarily determined by the CPU model backing it, and that variation between instances backed by the same CPU is much smaller than the variation between instances backed by different CPUs. As such, determining the expected performance of a set of instances depends, in part, on being able to *predict* which CPU models will back them. Since, in this paper, we seek to elaborate a capability to undertake such predictions, we consider the following problem: *In a request for a number of instances of a given type (from a given AZ), what is the expected number of each CPU model backing the instances?*

We begin with the observation that it will depend (at a minimum) upon (1) the proportions of CPU models found in a given AZ, and (2) the scheduling algorithm being used to place instances onto hosts.

In September 2013 an updated survey was conducted focusing on the AZs us-east-1a and us-east-1c in the Region US East. We ran 150 instances of types m1.small, m1.medium and m1.large in both us-east-1a and us-east-1c, that is 900 instances in total. Table 2 shows the percentages of each model found backing our instances in the 2 AZs. We abbreviate us-east-1a and us-east-1c by 1a and 1c.

Table 2: Proportion of CPU models for FGS instances is us-east-1a and us-east-1c.

Type	AZ	E5430	E5-2650	E5645	E5507
m1.small	1a	0	0	98%	2%
m1.medium	1a	0	85%	15%	0
m1.large	1a	0	67%	33%	0
m1.small	1c	25%	0	26%	49%
m1.medium	1c	24%	1%	39%	36%
m1.large	1c	14%	0	7%	79%

We *do not* interpret the above percentages as an estimate for the actual proportions of hosts with a given CPU type in the Zone, rather as an estimate of the probability of obtaining a particular model. Clearly, one would assume the percentages found to be reflective of actual host proportions in the AZ. From Table 2 we see that in us-east-1a we have very high probability of an m1.small obtaining an E5645, whilst the m1.medium has a much smaller chance. However, we do expect probabilities to change over time, and perhaps even frequently. We discuss this further in section 7.

We also note that without much further consideration a customer (given knowledge of Table 2) may well choose us-east-1a over us-east-1c for certain instance types. However, we also note that us-east-1c may offer better price/performance compared to us-east-1a due to it having a larger proportion of the E5430 model.

4 MODELLING AN AZ

In this section we develop a simple model of an AZ that will allow us to simulate requests and responses. Public Clouds are generally opaque - few architectural details are made available. And so we are guided by knowledge derived from the OpenStack reference architectures (OpenStack, no date), experiments on EC2, and our experiences of running both Eucalyptus and OpenStack Private Clouds at our institution.

To simplify matters we model requests only for FGS instances, the m1.small, m1.medium, m1.large and m1.xlarge instances. As discussed earlier, this class is backed by Intel Xeon E5430, E5-2650, E5645 and E5507 CPUs.

By VM density we refer to the ratio of vCPUs to physical cores that a host presents. For example, a host with 8 cores and VM density = 1 presents 8 vCPU. In this case the host could run, for example, 1 m1.xlarge (requiring 4 vCPU) and 4 m1.small instances (1 vCPU each). However, if the VM density is increased to 1.5 we now have 12 vCPUs presented, and so the host may now run (for example) an additional 2 m1.large instances. Of course, running a Cloud at a higher VM Density will lead to greater resource contention amongst the VMs, resulting in larger performance variation. We set the VM Density = 1.

The scheduler in our model (in common with the OpenStack scheduler) is capable of filtering instances types onto associated hardware. We therefore avoid considering hardware not associated with FGS instance class.

In our model, a host for a given class can run a mix of its instance types. For example, a FGS host with sufficient vCPUs could have an m1.small, m1.medium, m1.large and an m1.xlarge running on it at the same time. There is no obvious reason why this should not also be true on EC2.

We have chosen to use the chance scheduling algorithm. This chooses hosts at random and, if there are available resources, will allocate the VM to the host.

Hence our AZ consists of:

- A number of host machines running a hypervisor in order to host VM instances.
- Heterogeneous hosts (hosts may have different hardware).
- Each host presents a number of vCPUs which are allocated to instances running on it.
- One scheduler, responsible for placement of VM instances onto hosts.
  - The scheduler can be configured to implement a number of scheduling algorithms. We use chance scheduling.
  - The scheduler also filters instance types onto hosts of a given type.
- VM Density = 1.

5 SIMULATING USER REQUESTS

We restate the problem of interest: *In a request for a number of instances of a given type, what is the expected number of each CPU model backing the instances?*

By state of the Zone, we mean the total number of available vCPUs and their arrangement over the host machines. Clearly, the state of the Zone will change over time, potentially for a number of different reasons including demand fluctuation, capacity management, short-term unplanned outages, planned maintenance and hardware replacement and refreshes. Some of these are merely transient, whilst other will change the composition of the Zone.

To generate a state, we need to make an assumption about data centre loading, which we consider to be the number of *in use* vCPUs. Whilst we have no reliable data for actual VM loadings in data centres, and providers do not release such information, it is generally claimed to be high. Indeed, one of the positive arguments for Cloud adoption is high utilisation. Further, providers claim they can maintain high rates and that whilst an individual customer's demand fluctuates; aggregate demand (over all customers) remains fairly constant. This has been referred to as one of the 'Laws' of Cloudonomics (Weinman, 2008). From the discussion above, we believe that scenario 2 would be a reasonable assumption on data centre loading.

Our algorithm for generating a Zone state depends on size of the Zone, the load and the instances we wish to run in it. Assuming 1000 hosts each with 8 cores (without load), so 8000 vCPUs in total, able to run m1.medium, m1.large and m1.xlarge instance, we do as follows:

```
LOADING = randomint(0.7*8000,0.9*8000)
while ( LOADING > 0 ) do
    k = random-select(1, 2, 4)
    schedule(20 instances of k vCPUs)
    LOADING = LOADING - 20*k
```

With all hosts, and so all vCPUs, free, we generate a random loading of 70% to 90% of the vCPU count. Next, we randomly select (random numbers drawn from the uniform distribution) a vCPU size (1, 2, 4 to reflect the vCPU requirement of m1.medium, m1.large and m1.large, respectively) and schedule 20 instances of this type into the Zone. This is repeated until we have the required load. This gives a mix of types running on hosts. In our code the schedule function ensures that a VM is only placed on a host which has a sufficient number of free vCPUs. The schedule function updates the host and VM state after allocation.

We choose a static (Monte Carlo) simulation of user requests, proceeding as follows: We first generate a Zone State as described above, we then simulate a request (over this state) to the scheduler, which allocates the required number of instances onto hosts. We then count the number of CPU models in the hosts backing our instances. An example output of a request for 20 m1.small instances, in a Zone containing CPU models E5430, E5645 and E5507 would be: (5 E5430, 4 E5645, 11 E5507).

For each iteration in the simulation, the requested instance type is constant, the request size is 20 (the maximum on EC2), and Zone state is generated anew so if we are interested in m1.small types then the simulation asks for 20 m1.small instances 50,000 times (each time over a new Zone state). By repeatedly sampling over Zone states a sufficiently large number of times (50,000 in this case) we generate a distribution for the CPU models from which we calculate estimates of the expected number of CPU models per request. In practice there will be some request failures, however for simplicity we assume that all instances are successfully placed in each request.

We investigate two scenarios:
Scenario 1: Zero loading in the data centre.
Scenario 2: 70% to 90% of the cores in the data centre already in use.

To run a simulation, we need to specify the proportions of the CPUs in the Zone. Whilst we have data from our survey of us-east-1c, we are faced with 2 immediate problems. Firstly, as already discussed, the percentages are estimates of the probability of obtaining a model and not estimates of the actual host proportions. Clearly though they are likely to be indicative of them and as there is no obvious way to more accurately determine host proportions we take the percentages as estimates for our model. Next, instance types share common platforms, and yet from Table 2 are not being allocated to them in the same fashion. And so, if we wish to model m1.small requests should we use 49% as the percentage? Or should we average over 49%, 36% and 79%, which would give 55%? When modelling a given type we take the percentage found for the type and not the average.

6 SCENARIO 1

We are requesting 20 FGS instances from the pool of all available hosts. We start by considering the m1.medium instance type which has 1 vCPUs, and so requires 1 physical core by our assumptions. We are using a chance scheduler, which chooses hosts randomly and allocates a VM to it, if it has sufficient resources (if not choose a new host). Thus we are sampling over the set of cores without replacement. In this case the joint probability density function is described by the multivariate hyper-geometric distribution. The marginal distributions are also hyper-geometric, however when the sample size is small compared to the population, as in this case, the hyper-geometric tends to binomial distribution. And, so we have, for example E5430 ~ B(n,p) where n = 20 and p is proportion of E5430 cores expressed as a decimal for m1.medium instance. We also note that, whilst this request is for m1.medium instances, the

argument above, with slight modifications, applies to all FGS instance sizes.

7 SCENARIO 2

The AZ is assumed to have a loading such that 70% to 90% of the cores are already in use. That means the total available core count is between 800 and 2400. The first simulation is with m1.small requests in us-east-1c with CPUs in the following proportions: 25% E5430, 26% E5645 and 49% E5507. We then simulate m1.medium and m1.large using data from Table 2.

We calculate the marginal distributions for our random variables by simply counting the number of times a particular output occurs. To calculate the marginal probability distributions we simply divide the number of occurrences by the 50,000, from this we calculate the expected number of each CPU. In Table 3 below we present the calculated expectations for all CPU models in each simulation, and in Figure 1 we present the frequency distribution for the discrete random variable defined by the number of E5430 models returned in response to a request for 20 m1.large instances.

Table 3: Expected Value of CPU Returns.

	E5430	E5645	E5507
m1.small	5.00	5.197	9.795
m1.medium	5.0114	5.195	9.793
m1.large	5.008	5.192	9.799

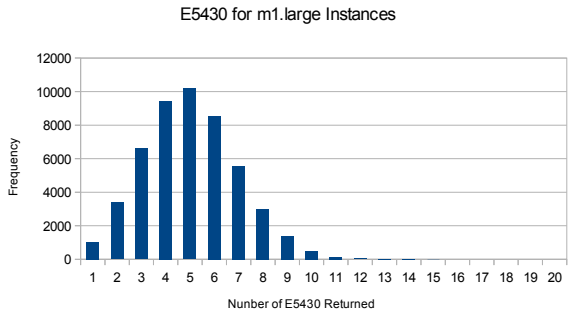


Figure 1: E5430 for m1.large Instances.

We suspect that the distributions are binomial. For the distribution depicted in Figure 1 we perform a Chi-Squared Goodness of Fit Test, where the null hypothesis is E5430 ~ B(20,0.25), and E5430 denotes the discrete random variable as defined in the paragraph above. For the test we generate expected output from B(20,0.25) and then calculate the chi squared statistic as 17.99. With 20 degrees of freedom, at a 5% significance level, we fail to reject the null hypothesis. Similarly, our simulations for m1.small and m1.medium instance types also produce marginal distributions which are binomial.

8 COMPARISONS WITH POST SIMULATION DATA FROM US-EAST-1C

From the above simulation E5430 ~ B(20,0.25). By a direct calculation P(E5430 <=7) – P(E5430 <=2) = 0.673. And so in 2 out of 3 samples of 20 instances we would expect the number of E5430 models obtained to be between 3 and 7.

In a follow on experiment to the one described in section 3, paragraph 6, and with results in Table 2; we requested 3 samples of 20 m1.large instances, with a 10 minute interval between requests and no resources were released until we had concluded the experiment. In Table 4 below we record the results.

Table 4: CPU models returned in request in us-east-1c for 20 m1.large.

Sample	E5430	E5645	E5507
1	3	0	17
2	5	1	14
3	2	0	18

In a final experiment, we launched 100 m1.small instances. Assuming a binomial distribution, and using our data in Table 2 as an estimate for p, the expected numbers (100*p) of each CPU model would be (25,24,49). We obtained (21,29,50).

It is not yet clear to us why instance types within the same class obtained the same CPUs in different proportions. It may well be the case that instances types within the same class do not run on the same host at the same time. That is, a host may run 2 m1.xlarge machines, and if one is terminated will only host another m1.xlarge. However, if both m1.xlarge instances are terminated then the host is available again to all instance types in the class. Over time such a scheme may lead to the distributions we have seen. We intend to pursue this idea further.

9 SCHEDULERS AND AZ STATE

We chose the chance scheduler as our staring point as it is was previously the default scheduling algorithm in OpenStack. There are a number of algorithms that schedule more intelligently than

chance, and reflect the priorities of a site. As an example, minimise the number of hosts in use; whilst keeping the hosts not in use in a low energy state. Others algorithms attempt to make efficient use of resources; and these are generally variations on the bin packing problem, as described in Wilcox, et al. (2010), with the scheduler employing a heuristic to solve the problem. The AZ states and the CPU distributions generated by this scheduler are potentially quite different from those that a chance scheduler would generate. Similarly, schedulers that seek to take advantage of co-locating VMs, for either deduplication of base images or page sharing amongst VMs may also generate AZ states different from both the chance scheduler and bin packing.

10 RELATED WORK

In Armbrust, et al. (2009), the authors describe EC2 Cloud performance as unpredictable. Similarly, Osterman, et al. (2010), describe performance in the Cloud as unreliable. The Magellan Report on Cloud Computing for Science (Yelick, et al., 2011) found that 'applications in the Cloud…experience significant performance variations' and noted that this is in part caused by the heterogeneous nature of EC2. However, EC2 was the only Public Cloud they considered. Similarly, in Phillips, Engen, and Papay, (2011), also discovered differential performance in instances of the same type on EC2 when attempting to predict application performance.

Iosup, et al. (2011), demonstrate that performance variation exists in a range of AWS services, including S3, SDB and SQS, and so performance variation of applications using these services is not solely dependent upon instance compute capability.

Schad, et al. (2010), show that the compute performance of instances sampled from US East N. California and the EU West Dublin Region falls into two distinct performance bands. Upon further investigation they detect two different CPU models backing their instances and speculate that previous results are explained by differences in CPU model. They also found a difference in the amount performance variation in two out of the four AZs used (us-east-1c and us-east-1d).

Ou, et al. (2012), suggest that the heterogeneous nature of Clouds can be exploited by estimating the probability of obtaining a particular CPU model backing an instance. Their method assumes that instances will be randomly sampled from all AZ available in US East N. California, which, as noted by the authors in Schad, et al. (2010) is not necessarily the case. To fully exploit heterogeneity, requests must be made per AZ, and this requires an analysis of the CPU distributions per AZ in the Region of interest. Additionally, they assume that the CPU model obtained in a particular request is *independent* of the one previously obtained. There is no guarantee of this; and VM placement algorithms need to be considered.

There is a large literature base for VM scheduling; for example Bazarbayev, et al. (no date), consider scheduling of VMs with identical or similar base images onto the same hosts. Reig, Alonso and Guitart, (2010), add machine learning capabilities to the scheduler in order for it to be able to predict resources required for a given execution time. However, as far as we aware, there are no papers looking specifically at how scheduling may affect the probability of obtaining particular resources.

11 CONCLUSIONS AND FUTURE WORK

The compute performance of an instance is primarily determined by the CPU model backing it. Currently customers cannot request either a desired level of performance, or, failing that, a particular CPU model. Being able to predict the performance of a set of instances before provisioning them, is therefore an important issue for Cloud users. Being able predict the range of CPU models that may be obtained in response to a request will go someway towards answering this question.

In this paper we therefore address the following problem: *In a request for a number of instances of a given type, what is the expected number of each CPU model backing the instances?*

By create a simple AZ model we showed that in the case of an empty (but sufficiently large) AZ then the number of models of a given type in a request is binomially distributed B(20,p), where p is the proportion of CPU models in the zone.

We then modelled an AZ and under a load of 70% to 90% capacity used, this describes the number of vCPUs in use. We used the uniform distribution to generate the AZ state, and then used a chance scheduling algorithm to allocate FGS instances into our AZ. The simulation samples over the set of possible states, so we estimate a joint probability distribution for our random variables E5430, E5645 and E5507 (denoting the CPU models

found in AZ respectively). From this joint distribution we calculated the marginal distributions and using a chi-squared goodness of fit test we showed that these are drawn from a Binomial distribution.

Therefore, in both scenarios the answer to our original question is this: the expected number of CPUs of a given type is the request size multiplied by the probability of obtaining a CPU (in a request for one instance). Comparing with a (small) set of samples drawn from us-east-1c we find that n*p appears to be reasonably good estimate of the number of models we may obtain.

We intend to investigate commonly found scheduling algorithms and incorporate them into future models, and compare the distributions our models generate with instance sampled from providers.

We recognise that the work involved in implementing such a model, and estimating possibly frequently changing probabilities, may make it impractical for certain users. However, we believe that this is feasible for Infrastructure Cloud brokers; and indeed they are ideally placed to do so.

More generally, we believe this will enable brokers to make performance related pricing offers to customers of the following form: Offer 1: We are 95% sure that at least 15 of the 20 VMs will run the workload in less than or equal to T seconds. Our future work is aimed at advancing this.

REFERENCES

Phillips, S., Engen, V., & Papay, J., 2011. Snow white clouds and the seven dwarfs, in *Proc. of the IEEE International Conference and Workshops on Cloud Computing Technology and Science*, pp738-745, Nov. 2011.

Ou, Z., Zhuang, H., Nurminem, J.K., Yla-Jaaski, A., & Hui, P., 2012. Exploiting Hardware Heterogeneity within the same instance type of Amazon EC2, presented at *4th USENIX Workshop on Hot Topics in Cloud Computing*, Boston, MA. Jun. 2012.

Reig, G., Alonso, J., & Guitart, J., 2010. Prediction of Job Resource Requirement for Deadline Schedulers to Manage High-Level SLAs on the Cloud, in *2012 Ninth IEEE International Symposium on Networking Computing and Applications,* pp 162-167, July 2010.

Amazon EC2 FAQs, no date. Aws.amazon.com. [Online]. Available at: < http://aws.amazon.com/ec2/faqs> [Accessed: 30 September 2013].

Google Cloud Platform, no date. cloud.google.com. [Online]. Available at:https://cloud.google.com [Accessed: 30th September 2013].

HP Public Cloud, no date. www.hpcloud.com. [Online]. Available at: <https://www.hpcloud.com/> [Accessed: 2nd July 2013].

Rackspace Global Infrastructure, no date. www.rackspace.com [Online]. Available at: http:// www.rackspace.com/information/aboutus/datacenters/ [Accessed: 30th September 2013].

AWS Global Infrastructure, no date. aws.amazon.com. [Online]. Available:<http://aws.amazon.com/about-aws/globalinfrastructure/> [Accessed: 30th September 2013].

Armbrust, M., et al, 2009. Above the clouds: a Berkely view of cloud computing, Technical Report EECS-2008-28, EECS Department, University of California, Berkeley.

OpenStack Scheduling, no date. Docs.openstack.org [Online]. Available at: < http://docs.openstack.org/ grizzly/openstack-compute/admin/content/ch_schedu ling.html> [Accessed: 30th September]

Weinman, J., 07/09/2008, 10 laws of Cloudonmics. Giga.com. [Online]. Available at http://gigaom.com /2008/09/07/the-10-laws-of-cloudonomics/ [Accessed: 30th September

Bazarbayev et al., no date. Content based scheduling of Virtual Machines in the Cloud. [Online]. Available at https://www.perform.csl.illinois.edu/Papers/USAN_pa pers/12BAZ01.pdf > [Accessed: 30th September]

Wilcox, D., McNabb, A., Seppi, K., & Flanagan, K., 2010. Probabilistic Virtual Machine Assignment, Cloud Computing 2010: First International Conference on Cloud Computing, Grids and Virtualisation, Lisbon, November 21-16, 2010.

S. Osterman et al, A performance analysis of EC2 cloud computing services for scientific computing, Cloud Computing, Lecture Notes of the Institute for Computer Sciences, *Social-Informatics and Telecommunications Engineering*, vol 34, 2010, pp 115-131.

Iosup, A., Nezih, Y., and Dick, E., 2011. On the performance variability of production cloud services. In Cluster, Cloud and Grid Computing (CCGrid), 2011

Yelick, K., et al, 2011. The Magellan Report on Cloud Computing for Science. [Online]. Available at: http://www/alcf.anl.gov/magellan [Accessed at: 2nd January 2014].

OpenStack, no date. www.openstack.org [Online]. Available at: http://www.openstack.org [Accessed: 1st January 2014].

Cloud Computing Adoption Factors and Processes for Enterprises
A Systematic Literature Review

Rania Fahim El-Gazzar
Department of Information Systems, University of Agder, Gimlemoen 25A, Kristiansand, Norway
rania.f.el-gazzar@uia.no

Keywords: Cloud Computing, Adoption, Implementation, Integration, Evaluation, IT Governance, Decision, Cost, Security, Regulations, Enterprise.

Abstract: Cloud computing (CC) has received an increasing interest from enterprises since its inception. With its innovative Information Technology (IT) services delivery model, CC could add technical and strategic business values to enterprises. However, it poses highly concerning, internal and external, issues. This paper presents a systematic literature review to explore cloud computing adoption processes in the context of enterprise users and the factors that affect these processes. This is achieved by reviewing 37 articles published about CC adoption. Using the grounded theory approach, articles are classified into eight main categories: internal, external, evaluation, proof of concept, adoption decision, implementation and integration, IT governance, and confirmation. This is concluded in two abstract categories: CC adoption factors and CC adoption processes whereas the former affect the latter. The results of this review indicate that there are serious issues need to be tackled before enterprises decide to adopt CC. Based on the findings of this review, the paper provides future Information Systems (IS) research directions toward the previously under-investigated areas regarding the phenomenon. This involved the call for further theoretical and in-depth empirical contributions to the area of CC adoption by enterprises.

1 INTRODUCTION

CC is receiving a notable interest from enterprise users nowadays. It could revolutionize the way enterprises acquire and manage their needs from computing resources efficiently and cost-effectively. Thus, enterprises are able to focus on their core business activities; consequently, productivity is increased (Garrison et al. 2012). The adoption of CC is growing rapidly due to the scalability, flexibility, agility, and simplicity it offers to enterprises (Parakala and Udhas, 2011); (Herhalt and Cochrane, 2012); (Venters and Whitley, 2012). A recent survey by (RightScale, 2013) on the adoption rates of CC by enterprises reported that 77% of large enterprises are adopting CC while 73% of small and medium enterprises (SMEs) are adopting CC. CC is defined by the National Institute of Standards and Technology (NIST) as "a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" (Hogan et al. 2011, p.14). Cloud service providers offer three different service models depending on the enterprise's requirements (i.e., Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and/or Infrastructure-as-a-Service (IaaS)). CC service models share five common essential characteristics of CC (i.e., on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service). CC services can be deployed in one or combination of the four deployment models (i.e., public, private, community, or hybrid) (Mell and Grance, 2011). A recent survey reported that 61% of enterprises are currently using public clouds while 38% are using private clouds and 29% are using hybrid clouds (RightScale, 2013).

In spite of its appealing benefits for enterprises, CC arouses serious technical, economic, ethical, legal, and managerial issues (Marston et al., 2011); (Venters and Whitley, 2012). This review is important for two reasons. First, it complements the previous literature that focused more on technical issues of CC with less attention to business issues regarding the adoption of CC (Yang and Tate, 2012). Second, it systematically offers detailed

perspectives on CC adoption factors and processes as CC has unique attributes that involve not only technical and managerial aspects, but its use also needs to comply with legal issues (Heiser and Nicolett, 2008). Further, there is a lack of in-depth studies about issues related to CC adoption process in the context of enterprise users (Yang and Tate 2012); (Timmermans et al., 2010).

The purpose of this paper is to systematically review the extant literature in IS research with an attempt to answer these two questions: what are the processes of adopting CC by enterprises? And what are the factors that affect these processes? Therewith, the under-investigated issues are identified and future IS research pathways are proposed accordingly. This review is important for two reasons.

The remainder of this paper is organized as follows: Section 2 presents the methodology followed for conducting the systematic literature search process and classification schemes adopted. Section 3 presents the findings from the review. Implications for future IS research are discussed in Section 4. Conclusions of this review are presented in Section 5.

2 RESEARCH METHOD

This paper aims at conducting a systematic literature review method to represent the current state of IS research regarding CC adoption issues in the context of enterprise users. This literature review process is following the fundamental guidelines for conducting an effective literature review by (Webster and Watson, 2002); (Levy and Ellis, 2006); (vom Brocke et al., 2009). This literature review process is done within boundaries (Webster and Watson 2002). The contextual boundary is within the scope of CC adoption by enterprise users, as there are significant issues need to be addressed before enterprises start using clouds (Marston et al., 2011). The temporal boundary of this review covers the published articles in all previous years until August 2013.

2.1 Literature Search Process

Literature search process of this review involved querying seven quality scholarly literature databases (AISeL, IEEE Xplore, ScienceDirect, EBSCOhost, ProQuest, Wiley online library, and ACM digital library). These databases provide access to IS leading journals and high quality peer-reviewed IS conferences (Levy and Ellis, 2006). The search criterion was limited to articles' titles to ensure the relevance of the articles. The phrases used for searching all the seven databases are 'cloud computing' and 'adopt*'. This initially resulted in 69 articles.

The practical screen involved, manually, reading the abstract of the articles to decide their relevance to the research question raised by this review. Further, the filtering criteria involved the exclusion of recurring articles, research-in-progress articles, articles that are not written in English, periodical articles published by news websites, and trade journals and magazines. These exclusion criteria are defined to delimit the sample of articles so that the literature review becomes practically manageable (Okoli and Schabram, 2010). Eventually, this resulted in 37 articles ready for the classification.

2.2 Classification Scheme

The reviewed articles are classified according to the research methods to identify how adequately the adoption of CC is researched. Research methods used in the reviewed articles are lab experiment, field inquiry, Delphi study, interviews, descriptive research, case study, and survey. Some articles were labelled under 'conceptual paper', as they did not have a methodology section and providing some concepts or adopting theories without an empirical testing.

This review sought to develop a classification scheme to better gain insights from the preceding academic contributions to the area of CC adoption. Classification of the 37 articles according to cloud adoption issues involved using bottom-up Grounded Theory (GT) approach (Glaser and Strauss, 1967). GT approach is said to be valuable for conducting a rigorous literature review (Wolfswinkel et al., 2011), "instead of force-fitting the data to an a priori theory" (Rich 2012, p.3).

The classification process involved a close reading of the articles. Then, the open coding is utilized to and resulted in labelling 28 concepts. Next, the axial coding is applied and resulted in grouping the 28 subcategories into eight corresponding categories (i.e., internal, external, evaluation, proof of concept, adoption decision, implementation and integration, IT governance, and confirmation). Then, selective coding was applied to integrate the eight main categories and resulted in two abstract categories: cloud adoption factors (i.e., internal and external) and cloud adoption processes (i.e., evaluation, proof of concept, adoption decision, implementation and integration, IT governance, and

confirmation) whereas the former influence the latter (see Figure 1). To avoid a coarse analysis, these factors and processes are elaborated in more details in this review.

3 FINDINGS

The mapping of articles to the grounded classification scheme and research methods is demonstrated in Table 1.

CC Adoption Factors. Comprise internal and external factors that have impact on CC adoption processes.

External Factors. This comprises factors related to the outside social environment in which the enterprise operates, and by which its CC adoption is influenced. The adoption of CC is, arguably, surrounded by different levels of environmental and inter-organizational uncertainties (Cegielski et al., 2012). There is a wide emphasize on the importance of government regulations at the national and international levels (Espadanal and Oliveira, 2012); (Nuseibeh, 2011); (Nkhoma et al., 2013). In spite of their role to facilitate the CC adoption securely, the lack of government regulations can hinder enterprises from adopting the cloud (Nkhoma et al., 2013); (Luoma and Nyberg, 2011). Some regulations, such as Sarbanes-Oxley Act (SOX) for corporate accounting data and Health Insurance Portability and Accountability Act (HIPAA), have been enacted before CC being increasingly adopted and they might not be sufficient to facilitate its adoption (Kim et al., 2009). However, some countries started to enact laws specific to CC such as cloud first policy in the USA and cloud computing strategy by the Australian government (Bhat, 2013). The inconsistency of international government regulations is a further concern. Some countries tend to restrict enterprises to store their data in cloud infrastructure only if it is within the borders (Jensen et al., 2011). For instance, the EU's privacy laws prohibit the exchange of personal information outside the users' jurisdiction (Mcgeogh and Donnellan 2013). This because of cloud providers running outside their home country must comply with the host country's regulations and government surveillance which is difficult for enterprises to cope (Abokhodair et al., 2012); (Bhat, 2013). In line with this, standards are also cited to be an important factor for dealing with enterprises' concerns about security and interoperability of cloud solutions (Nuseibeh, 2011); (Mcgeogh and Donnellan, 2013). Although they are still missing in the puzzle of CC picture for enterprises (Bhat, 2013); (Lin and Chen, 2012); (Nkhoma et al., 2013), some efforts to develop security standards took place. For instance, Cloud Security Alliance (CSA)'s document Security Guidance for Critical Areas of Focus in Cloud Computing for managing CC risks (Mcgeogh and Donnellan, 2013). Eventually, cloud providers would have to comply with government regulations and industrial standards to gain liability, reputation, and trustworthiness among their potential and present enterprise users (Bhat, 2013).

Cloud providers are required to be transparent in explaining information to enterprises about possible benefits and risks from adopting CC (Dargha, 2012). Cloud providers build their reputation by their experience in managing enterprises' needs and concerns in a responsible manner (Jensen et al., 2011); (Malladi and Krishnan, 2012); (Nuseibeh, 2011). This is enabled by delivering the promised benefits from CC to enterprises and protecting their data from potential threats including fate sharing in case the cloud provider went out of business (Iyer et al., 2013); (Nuseibeh, 2011). Further, cloud providers' demonstration of successful business cases and models is likely to enable CC adoption decision (Lin and Chen, 2012); (Alshamaila and Papagiannidis, 2013). Cloud providers might trigger a major concern to if they outsource some of their services to another service provider (Nuseibeh, 2011). Observing business partners perceiving benefits from using CC is proved to be an important motive towards its adoption (Low et al., 2011); (Lin and Chen, 2012). Business partners' request, with power, the enterprise to adopt CC to keep on collaboration with them (Cegielski et al., 2012). Additionally, competitors play an effective role in incentivizing enterprises to adopt CC for gaining market visibility, operation efficiency, and new business opportunities (Espadanal and Oliveira, 2012); (Borgman et al., 2013), especially when the enterprise operates in a high-tech rapidly changing industry (Low et al., 2011); (Alshamaila and Papagiannidis, 2013). However, competitors pressure may not be relevant to SMEs as they are concerned about other CC adoption issues (Alshamaila and Papagiannidis, 2013).

Internal Factors. This comprises the enterprise's internal characteristics and capabilities that affect its CC adoption process. The enterprise's willingness to invest in and use CC both financially and organizationally is claimed to be an important indicator for the adoption of CC (Espadanal and Oliveira, 2012); (Rath et al., 2012). This depends on

the role of the top management's IT knowledge and capability of providing the suitable organizational climate for adopting CC (Bharadwaj and Lal, 2012); (Borgman et al., 2013; Alshamaila and Papagiannidis, 2013). Top management activities involve: (1) understanding of CC and its architecture, service models, and strategic values (Luoma and Nyberg, 2011); (Espadanal and Oliveira, 2012); (Misra and Mondal, 2011). (2) identifying enterprise's business needs and aligning IT decisions with business strategies (Mcgeogh and Donnellan, 2013); (Subramanian, 2012). (3) evaluating the readiness of the existing IT infrastructure, IT knowledge and skills of the human resources, available resources, and culture (Luoma and Nyberg, 2011); (Borgman et al., 2013); (Espadanal and Oliveira, 2012). (4) holding the steering wheel towards CC adoption (i.e., decide adoption strategy, govern integration and implementation, and evaluate cloud services after use) with the guidance of external regulatory and professional bodies (Morgan and Conboy, 2013); (Mcgeogh and Donnellan, 2013). Additionally, firm size is an important factor to CC adoption (Espadanal and Oliveira, 2012); (Borgman et al., 2013); (Low et al., 2011). A survey study conducted in Taiwan indicated that large enterprises are likely to adopt CC (Low et al. 2011). Contrarily, a survey study in India indicated that SMEs can benefit the most (Rath et al., 2012). Large enterprises are claimed to have sufficient resources to afford on-premise solutions (Borgman et al., 2013). Enterprises need to further ensure compliance of CC solutions with the internal (i.e., corporate policies) and external (i.e., regulations and standards) constraints (Morgan and Conboy, 2013); (Mcgeogh and Donnellan, 2013); (Borgman et al., 2013).

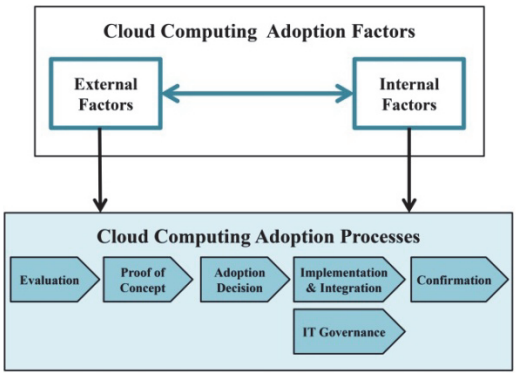


Figure 1: Cloud Computing Adoption Factors and Processes.

Further, the cloud/IT skills of non-IT employees is said to be a crucial factor affecting CC adoption; as they may be involved in strategic IT decisions, their understanding of CC is very important (Borgman et al., 2013); (Low et al., 2011); (Espadanal and Oliveira, 2012). Likewise, IT employees would have to adjust their skills and be aware of CC benefits and risks and how to deal with the risks instead of being overprotective (Morgan and Conboy, 2013). Another factor affecting CC adoption decision is the enterprise's prior experience and familiarity with similar technologies such as virtualization (Alshamaila and Papagiannidis, 2013).

CC Adoption Processes. Are processes that enterprises normally follow to adopt CC along with responsibilities and challenges faced in each process.

Evaluation. Prior to CC adoption, the top management is responsible for evaluating the enterprise's suitability for adopting CC as well as the suitability of CC for the enterprise (Misra and Mondal, 2011); (Lin and Chen, 2012). This includes: (1) evaluating costs and benefits associated with CC such as profitability, comparing the revenue generated from the firm's IT resources with the revenue expected from CC, ROI, cost of migration and integration, and cost of implementation (Misra and Mondal, 2011; (Celine) Hoe et al., 2012). (2) evaluating the impact of CC on people and business processes (Mcgeogh and Donnellan 2013). (3) evaluating the internal readiness of the enterprise, existing IT infrastructure and IT human resources, for adopting CC (Low et al., 2011). (4) selecting the cloud service provider based on cloud provider's capability to provide a robust security controls, enterprise's understanding of issues related to the control over the data, the type of service model needed, and the perceived cloud provider's honesty, reputation, and sustainability (Onwudebelu and Chukuka, 2012); (Bharadwaj and Lal, 2012).

Proof of Concept. Trying cloud services prior to the actual adoption to evaluate its applicability for the enterprise is likely to convince the enterprise to adopt CC (Lin and Chen, 2012). Trialability proved to have a positive impact on the adoption of CC (Alshamaila and Papagiannidis, 2013). There is a wide agreement on the significant influence of perceived benefits and risks on the adoption of cloud services (Alshamaila and Papagiannidis 2013); (Iyer et al., 2013). A survey study reported that the management's perception of security, cost-effectiveness, and IT compliance are likely to have a significant impact on the decision to adopt CC (Opala and Rahman, 2013). CC brings plenty of benefits that are relatively convincing for enterprises

to adopt it (i.e., cost savings, agility, flexibility, ease of use, scalability, facilitating collaboration between business partners, less operational effort on Chief Information Officers (CIOs), and increased productivity) (Borgman et al., 2013); (Malladi and Krishnan, 2012); (Nkhoma et al., 2013); (Lin and Chen, 2012). With all these appealing benefits, yet some SMEs are still negative to adopt CC services (Alshamaila and Papagiannidis, 2013). They are concerned about various types of risks (Jensen et al., 2011); (Nkhoma et al., 2013); (Lin and Chen, 2012); (Iyer et al., 2013): (a) policy and organizational risks: this covers the risk of vendor and data lock-in. (b) technical risks: this includes loss of data, downtime, and cyber-attacks. (c) Legal risks: this includes data protection regulations and licensing issues. (d) Non-technical risks: this refers to the misuse of cloud services and natural disasters.

According to a survey study in India on four sectors (SMEs, education, banking, hospital) showed that vendor risk was dominating among SMEs and education sectors, while security risk was dominating among banking and hospital sectors (Iyer et al., 2013). Regarding performance risks, moving a huge amount of data to cloud servers takes a long time, and when moving further in the adoption, this will require increasing bandwidth and connectivity which is costly (Morgan and Conboy, 2013).

Adoption Decision. When deciding to adopt CC services, the top management is involved in: (1) evaluating core business requirements and competencies (Subramanian, 2012). (2) determining what data and applications should move to the cloud (i.e., critical vs. non-critical data and applications) (Kim et al., 2009); (Misra and Mondal, 2011); (Kantarcioglu et al., 2011). (3) evaluating the strategic value of CC such as agility by delivering strong coordination IT capabilities, process management maturity, and reduced operational burden on CIOs so they can focus on strategic activities (Malladi and Krishnan, 2012). (4) planning for implementation of CC systems; the managerial time required to plan and implement cloud solutions was not problematic (Rath et al., 2012), while the problem was the planning for implementation of specific deployment models that suite the current applications (Dargha, 2012). (5) selecting the right service model based on the enterprise's needs (Bamiah et al., 2012); (Onwudebelu and Chukuka, 2012). (6) selecting deployment models based on the sensitivity of the applications; if the applications are determined to be core, then they should be deployed on a private cloud, and if the applications are determined to be non-core, then they should be deployed on a public cloud (Dargha, 2012). Another study suggests the core applications should not be deployed on the cloud at all (Kim et al. 2009). (7) negotiating cloud service contract and SLA with the cloud provider based on the sensitivity of the data (Misra and Mondal, 2011), and reaching an agreement on data ownership, data confidentiality, risk management, contract terms and termination effect, location of the data, and guarantee of service availability (Tsai et al., 2010); (Onwudebelu and Chukuka, 2012).

Implementation and Integration. Compared to on-premise approach, cloud systems can be implemented and running in 24 hours instead of six months (Morgan and Conboy, 2013). Thus, implementation of cloud systems is not problematic for enterprises, while integrating cloud systems with the enterprise's existing IT infrastructure can negatively impact their adoption of CC (Morgan and Conboy, 2013). Further, the use of cloud system by IT staff is straightforward, while it is challenging for the non-IT staff. This can be attributed to the degree of complexity of cloud systems in terms of the ease of understanding, use, and implementation or integration of cloud services. Although CC proved to be easy to understand and use, it arouses integration complexity issues (Borgman et al., 2013). Complexity is claimed to be a barrier to the adoption of CC (Lin and Chen, 2012); (Espadanal and Oliveira, 2012); (Low et al., 2011); (Borgman et al., 2013). Integration problems emerge from the less standardized interoperability between cloud systems and the existing IT infrastructure which triggers integration cost (Borgman et al., 2013). Incompatibility of cloud solutions with existing IT infrastructure can be a barrier to the adoption of CC (Low et al., 2011).

IT Governance. The loss of IT governance within the enterprise appeared to be a major concern because the data and applications are under the control of the cloud provider (Onwudebelu and Chukuka, 2012); (Mcgeogh and Donnellan, 2013). Enterprises are advised to conduct audit trail meetings with the cloud provider to ensure risk free implementation of cloud solutions that complies with regulations, standards, and enterprise policies (Morgan and Conboy, 2013). This is enabled by the top management through IT governance structures (i.e., federal or centralized) and processes (i.e., request, prioritize, fund, monitor, enforce, and realign) (Borgman et al., 2013). Contrarily, IT governance processes in highly regulated industries will decelerate the adoption of CC (Borgman et al.,

2013). Likewise, IT governance processes might hinder the adoption if the integration of cloud solutions with the existing infrastructure appeared to be difficult (Borgman et al., 2013).

Despite the massive advancements in securing the cloud, yet these security solutions are not tested extensively (Misra and Mondal, 2011). This matter could be dissolved by IT governance initiatives to ensure that enterprise policies, security standards, and legal requirements are met (Borgman et al., 2013; Morgan and Conboy, 2013). IT governance is attained by identifying responsibilities, for instance, "the cloud provider is responsible for the security of the hypervisor and below, while the customer's responsibility begins at the operating system level and above" (Mcgeogh and Donnellan, 2013, p.4). While in some cases, data security at the level of PaaS and IaaS service models is a shared responsibility between the cloud service provider and the adopting enterprise (Bamiah et al., 2012). Furthermore, pursuing traceability and auditability are cited to have impact on CC adoption (Morgan and Conboy, 2013). Cloud providers are required to provide traceable access controls to govern (i.e., who can access what object under which conditions) that can be validated, by top management, for its conformity to internal and external constraints (Morgan and Conboy, 2013). These controls are used to ensure data integrity and confidentiality (Onwudebelu and Chukuka, 2012); (Morgan and Conboy, 2013).

In regard with risk management during the planning for CC implementation, a study suggests that the enterprise should consider evaluating the risk of storage damage, data loss, and network security (Tsai et al., 2010). For instance, the enterprise would maintain an on-site backup of the data moved to the cloud (Kim et al., 2009). Eventually, securing enterprise's information from potential risks is more than processes, technical solutions, and people; it is an enterprise wide security strategy to orchestrate these various elements (Opala and Rahman, 2013).

Confirmation. Only one study proposed a model for implementation and confirmation stages in adopting CC (Dargha, 2012). This study suggests the evaluation of cloud services based on the perceived attributes from using cloud services (i.e., relative advantage, complexity, compatibility, and trialability) to decide whether to continue on using cloud services or not.

4 DISCUSSION AND FUTURE RESEARCH AVENUES

This article sought to review the current state of IS research regarding CC adoption in the context of enterprise users. The review involved classifying the extant literature into CC adoption factors and processes that emerged from the reviewed articles along with research methods employed. This review shows that no IS journal articles found yet contributing to this research area. While few IS conference articles appeared to contribute to the understanding of CC adoption. On the other hand, articles from other disciplinary journal and conference outlets were dominant in investigating the phenomenon.

Table 1 indicates a lack of in-depth field and case studies regarding CC adoption processes compared to the CC adoption factors, while the quantitative (i.e., surveys) studies and conceptual articles are dominating. However, the majority of these theoretical attempts have not been tested yet. For instance, a study proposed, theoretically, a CC adoption assessment model that considers criteria for selecting the cloud service provider but this model has not been tested empirically yet (Nasir and Niazi, 2011). Hence, further qualitative research needs to be undertaken to explore further issues and test empirically the previous theoretical developments regarding this area. Consequently, this triggers questions on the IS research round table: Why enterprises adopt CC in spite of its potential risks? Or reversely, why enterprises do not adopt CC in spite of its potential benefits? These questions need to be investigated thoroughly using multiple qualitative case studies in different contexts (i.e., countries and industries) to better understand CC adoption factors and processes. Noteworthy, all the reviewed articles study cloud adoption issues in a list-like broad perspective. Therefore, there is a need for interpretive case studies to investigate each of the issues found from this review (Walsham, 1995).

Obviously, Table 1 shows that CC adoption processes received less attention from IS researchers in terms of exploring the challenges faced in each process and how enterprises cope with these challenges for risk free adoption of cloud services. Further, there is a conflict regarding the relationship between the firm's size and the likelihood to adopt CC which needs more in-depth studies to address this conflict. Further, CC is recognized as a cost reduction solution, however, this cost reduction may not be significant particularly in SMEs sector as reported by a survey study conducted in India (Iyer

Table 1: Mapping of articles to classification scheme and research methods.

Classification Scheme						Research Methods							
Cloud Computing Adoption Processes					Cloud Computing Adoption Factors	Lab Experiment	Field Study	Case Study	Delphi Study	Survey	Interviews	Conceptual Paper	Literature Review
Confirmation	IT Governance	Implementation & Integration	Adoption Decision	Proof of Concept	Evaluation	Internal	External							
	(Tsai et al. 2010)		(Tsai et al. 2010)	(Tsai et al. 2010)									
					(Abokhodair et al. 2012)		(Abokhodair et al. 2012)	(Abshamaila & Papagiannidis 2013), (Bharadwaj & Lal 2012), (Mcgeogh & Donnellan 2013), (Morgan & Conboy 2013), (Saedi & Iahad 2013)					
	(Mcgeogh & Donnellan 2013), (Morgan & Conboy 2013)	(Morgan & Conboy 2013)		(Abshamaila & Papagiannidis 2013), (Bharadwaj & Lal 2012), (Morgan & Conboy 2013), (Saedi & Iahad 2013)	(Mcgeogh & Donnellan 2013), (Bharadwaj & Lal 2012)		(Mcgeogh & Donnellan 2013), (Abshamaila & Papagiannidis 2013), (Saedi & Iahad 2013)						
						(Luoma & Nyberg 2011)	(Cegielski et al. 2012), (Luoma & Nyberg 2011)		(Cegielski et al. 2012), (Rawal 2011), (Nkhoma et al. 2013), (Lin & Chen 2012), (Malladi & Krishnan 2012), (Low et al. 2011), (Iyer et al. 2013)				
(Opala & Rahman 2013)		(Lin & Chen 2012), (Low et al. 2011)	(Malladi & Krishnan 2012), (Rath et al. 2012), (Luo 2013)	(Lin & Chen 2012), (Dyer et al. 2013), (Opala & Rahman 2013), (Malladi & Krishnan 2012), (Nkhoma et al. 2013), (Lin & Chen 2012), (Gupta et al. 2013), (Rawal 2011), (Luo 2013), (Stankov et al. 2012)	(Lin & Chen 2012), (Low et al. 2011)	(Rath et al. 2012), (Rawal 2011), (Low et al. 2011), (Yang et al. 2012)					
(Borgman et al. 2013)	(Borgman et al. 2013)			(Borgman et al. 2013)		(Borgman et al. 2013), (Yang et al. 2012)	(Cegielski et al. 2012), (Borgman et al. 2013)						
(Dargha 2012)	(Onwudebelu & Chukuka 2012), (Misra & Mondal 2011), (Kim et al. 2009)	(Espadanal & Oliveira 2012)	(Subramanian 2012), (Kim et al. 2009), (Misra & Mondal 2011), (Kantarcioglu et al. 2011), (Dargha 2012), (Onwudebelu & Chukuka 2012), (Jensen et al. 2011)	(Subramanian 2012), (Nuseibeh 2011), (Onwudebelu & Chukuka 2012), (Jensen et al. 2011), (Chang et al. 2013)	(Misra & Mondal 2011), ((Celine) Hoe et al. 2012), (Onwudebelu & Chukuka 2012), (Dargha 2012), (Kantarcioglu et al. 2011), (Nasir & Niazi 2011)	(Espadanal & Oliveira 2012), (Nasir & Niazi 2011), (Misra & Mondal 2011), (Subramanian 2012), (Chang et al. 2013)		(Espadanal & Oliveira 2012), (Nuseibeh 2011), (Kim et al. 2009), (Bhat 2013), (Jensen et al. 2011), (Dargha 2012), (Chang et al. 2013)	
	(Bamiah et al. 2012)		(Bamiah et al. 2012)	(Ewuzie & Usoro 2012)									(Ewuzie & Usoro 2012)

et al., 2013). Likewise, when the enterprise maintains an on-premise backup, this adds to the cost as well (Kim et al., 2009). Moreover, there is a need for further exploring the impact of IT

governance processes throughout the implementation process. There still plentiful legal, ethical and inter-organizational or institutional issues need to be investigated regarding improvements of laws and standards. It would be insightful to investigate internal readiness and selection of cloud provider issues in the context of SMEs and/or large enterprises.

Further, if CC helps enterprises reduce IT-related costs, then how significant will be this cost reduction? Further studies should explore the impact of vendor and cloud solutions compatibility, trialability, and observability on the adoption of cloud services. Trust issues are not addressed extensively in the reviewed articles although it is claimed to be important for a successful adoption of CC (Garrison et al., 2012). In spite of its appealing merits it brings to enterprises (i.e., cost savings, flexibility, efficiency, agility and so forth), CC puts enterprises in a decisive choice between on-premise and on-demand approaches. In this regard, the Delphi method would provide insights for IT managers about what are the most important issues, and their priorities, that shall be considered when deciding to adopt cloud solutions (Dalkey, 1972). Another issue to consider, if CC releases enterprises from managing the IT infrastructure so they can focus on their core business activities, then what data and applications should be moved to the cloud and what should remain in-house? Delphi studies can be appropriate for providing recommendations for practice regarding internal preparation, service model selection, and contract negotiation issues.
Finally, this review advocates the need for conducting longitudinal studies to assess the impact of CC implementation on both technical and managerial capabilities of the enterprise (i.e., integration with existing IT infrastructure, planning, risk management, and IT governance) as well as the impact of the confirmation process on the organizational innovation.

5 CONCLUSION

This paper sought to conduct a systematic review of the extant literature regarding issues related to CC adoption process in the context of enterprise users. This involved exploring the under-investigated issues and the contributions of IS research regarding the phenomenon. The classification of reviewed articles, findings, and implications for future IS research is achieved according to research methods and CC adoption factors and processes that emerged from the utilized grounded theory approach. Yet, plentiful legal, ethical, and technical, and managerial issues waiting for IS researchers to explore. Thus, the paper suggested a future IS research agenda based on the discussed findings. This article is not free of limitations; it sought to review only academic articles from seven literature databases leaving white papers, magazine articles, other scholarly literature databases, and articles from forward and backward search which would help capturing more issues about CC adoption by enterprises. The search criteria might be limited as some articles discuss CC adoption entitled with different words (i.e., utility computing or application service provision) that may not have been included in the search results of this review.

REFERENCES

(Celine) Hoe, S., Kantarcioglu, M. and Bensoussan, A., 2012. Studying Dynamic Equilibrium of Cloud Computing Adoption with Application of Mean Field Games. In *Proceedings of the 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton). IEEE*, pp. 220–224.

Abokhodair, N. et al., 2012. Heading for the Clouds? Implications for Cloud Computing Adopters. In *Proceedings of the 18th Americas Conference on Information Systems (AMCIS 2012)*.

Alshamaila, Y. and Papagiannidis, S., 2013. Cloud computing adoption by SMEs in the north east of England: A multi-perspective framework. *Journal of Enterprise Information Management*, 26(3), pp.250–275.

Bamiah, M. et al., 2012. A Study on Significance of Adopting Cloud Computing Paradigm in Healthcare Sector. In *Proceedings of the International Conference on Cloud Computing Technologies, Applications and Management (ICCCTAM 2012). IEEE*, pp. 65–68.

Bharadwaj, S.S. and Lal, P., 2012. Exploring the Impact of Cloud Computing Adoption on Organizational Flexibility: A Client Perspective. In *Proceedings of the International Conference on Cloud Computing Technologies, Applications and Management (ICCCTAM 2012). IEEE*, pp. 121–131.

Bhat, J.M., 2013. Adoption of Cloud Computing by SMEs in India : A study of the Institutional Factors. In *Proceedings of the 19th Americas Conference on Information Systems*. pp. 1–8.

Borgman, H.P. et al., 2013. Cloudrise: Exploring Cloud Computing Adoption and Governance with the TOE Framework. In *Proceedings of the 46th Hawaii International Conference on System Sciences*. IEEE, pp. 4425–4435.

Vom Brocke, J. et al., 2009. Reconstructing the Giant: On the Importance of Rigour in Documenting the Literature Search Process. In *the 17th European*

Conference on Information Systems (ECIS 2009). Verona, Italy, pp. 2206–2217.

Cegielski, C.G. et al., 2012. Adoption of Cloud Computing Technologies in Supply Chains: An Organizational Information Processing Theory Approach. *International Journal of Logistics Management, The*, 23(2), pp.184–211.

Chang, B.-Y. et al., 2013. The Determinant of Adoption in Cloud Computing in Vietnam. In *Proceedings of the International Conference on Computing, Management and Telecommunications (ComManTel 2013)*. IEEE, pp. 407–409.

Dalkey, N.C., 1972. The Delphi method: An experimental study of group opinion. In N. C. Dalkey, ed. *Studies in the quality of life: Delphi and decision-making*. Lexington Books, p. 161.

Dargha, R., 2012. Cloud Computing : From Hype to Reality . Fast Tracking Cloud Adoption. *In Proceedins of the Intermational Conference on Advances in Computing, Communications, and Informatics*. pp. 440–445.

Espadanal, M. and Oliveira, T., 2012. Cloud Computing Adoption by Firms. In *Proceedings of the Mediterranean Conference on Information Systems (MCIS 2012)*.

Ewuzie, I. and Usoro, A., 2012. Exploration of Cloud Computing Adoption for E-Learning in Higher Education. In *Proceedings of the Second Symposium on Network Cloud Computing and Applications*. IEEE, pp. 151–154.

Garrison, G., Kim, S. and Wakefield, R.L., 2012. Success Factors for Deploying Cloud Computing. *Communications of the ACM*, 55(9), pp.62–68.

Glaser, B.G. and Strauss, A.L., 1967. The Discovery of Grounded Theory: Strategies for Qualitative Research, *Chicago: Aldine Publishing Company*.

Gupta, P., Seetharaman, a. and Raj, J.R., 2013. The usage and adoption of cloud computing by small and medium businesses. *International Journal of Information Management*, 33(5), pp.861–874.

Heiser, J. and Nicolett, M., 2008. Assessing the Security Risks of Cloud Computing Available at: http://www.globalcloudbusiness.com/SharedFiles/Download.aspx?pageid=138andmid=220andfileid=12

Herhalt, J. and Cochrane, K., 2012. Exploring the Cloud: A Global Study of Governments' Adoption of Cloud,

Hogan, M. et al., 2011. NIST Cloud Computing Standards Roadmap, Available at: http://www.nist.gov/customcf/get_pdf.cfm?pub_id=909024.

Iyer, E.K. et al., 2013. Analysis of Dissatisfiers that Inhibit Cloud Computing Adoption across Multiple Customer Segments. In *Proceedings of the European Conference on Information Management and Evaluation*. pp. 145–152.

Jensen, M. et al., 2011. Security Prospects through Cloud Computing by Adopting Multiple Clouds. In *Proceedings of the 4th International Conference on Cloud Computing (CLOUD)*. IEEE, pp. 565–572.

Kantarcioglu, M., Bensoussan, A. and (Celine) Hoe, S.R., 2011. Impact of Security Risks on Cloud Computing Adoption. In *Proceedings of the 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, pp. 670–674.

Kim, W. et al., 2009. Adoption Issues for Cloud Computing. In *Proceedings of the 11th International Conference on Information Integration and Web-based Applications and Services - iiWAS'09*. New York, New York, USA: ACM Press, p. 2.

Levy, Y. and Ellis, T.J., 2006. A Systems Approach to Conduct an Effective Literature Review in Support of Information Systems Research. *Informing Science Journal*, 9, pp.181–212.

Lin, A. and Chen, N.-C., 2012. Cloud Computing as an Innovation: Percepetion, Attitude, and Adoption. *International Journal of Information Management*, 32(6), pp.533–540.

Low, C., Chen, Y. and Wu, M., 2011. Understanding the Determinants of Cloud Computing Adoption. *Industrial Management and Data Systems*, 111(7), pp.1006–1023.

Luo, L., 2013. Reference Librarians' Adoption of Cloud Computing Technologies: An Exploratory Study. *Internet Reference Services Quarterly*, 17(3-4), pp.147–166.

Luoma, E. and Nyberg, T., 2011. Four Scenarios for Adoption of Cloud Computing in China. In *Proceedings of the European Conference on Information Systems (ECIS 2011)*.

Malladi, S. and Krishnan, M.S., 2012. Cloud Computing Adoption and its Implications for CIO Strategic Focus – An Empirical Analysis. In *Proceedings of 33rd International Conference on Information Systems (ICIS 2012)*. pp. 1–19.

Marston, S. et al., 2011. Cloud Computing – The Business Perspective. In *Proceedings of the 44th Hawaii International Conference on System Sciences (HICSS 2011)*. pp. 1–11.

Mcgeogh, B.T. and Donnellan, B., 2013. Factors That Affect The Adoption Of Cloud Computing For An Enterprise : A Case Study Of Cloud Adoption Within Intel Corporation. In *Proceedings of the 21st European Conference on Information Systems*. pp. 1–13.

Mell, P. and Grance, T., 2011. The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology, Available at: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

Misra, S.C. and Mondal, A., 2011. Identification of a Company's Suitability for the Adoption of Cloud Computing and Modelling its Corresponding Return on Investment. *Mathematical and Computer Modelling*, 53(3-4), pp.504–521.

Morgan, L. and Conboy, K., 2013. Factors Affecting The Adoption Of Cloud Computing : An Exploratory Study. In *Proceedings of the 21st European Conference on Information Systems*. pp. 1–13.

Nasir, U. and Niazi, M., 2011. Cloud Computing Adoption Assessment Model (CAAM). In *Proceedings of the 12th International Conference on Product Focused*

Software Development and Process Improvement. pp. 34–37.

Nkhoma, M.Z., Dang, D.P.T. and Souza-daw, A. De, 2013. Contributing Factors of Cloud Computing Adoption : A Technology- Organisation-Environment Framework Approach. In *Proceedings of the European Conference on Information Management and Evaluation*. pp. 180–189.

Nuseibeh, H., 2011. Adoption of Cloud Computing in Organizations. In *Proceedings of the 17th Americas Conference on Information Systems (AMCIS 2011)*.

Okoli, C. and Schabram, K., 2010. Working Papers on Information Systems A Guide to Conducting a Systematic Literature Review of Information Systems Research. *Sprouts: Working Papers on Information Systems,* 10(26).

Onwudebelu, U. and Chukuka, B., 2012. Will Adoption of Cloud Computing Put the Enterprise at Risk? In *Proceedings of the 4th International Conference on Adaptive Science and Technology (ICAST 2012)*. IEEE, pp. 82–85.

Opala, O.J. and Rahman, S. (Shawon) M., 2013. An Exploratory Analysis of the Influence of Information Security on the Adoption of Cloud Computing. In *Proceedings of the 2013 8th International Conference on System of Systems Engineering*. pp. 165–170.

Parakala, K. and Udhas, P., 2011. The Cloud Changing the Business Ecosystem, KPMG.

Rath, A. et al., 2012. Decision Points for Adoption Cloud Computing in Small , Medium Enterprises (SMEs). In *Proceedings of the 7th International Conference for Internet Technology and Secured Transactions (ICITST 2012)*. pp. 688–691.

Rawal, A., 2011. Adoption of Cloud Computing in India. *Journal of Technology Management for Growing Economies,* 2(2), pp.65–78.

Rich, P., 2012. Inside the Black Box : Revealing the Process in Applying a Grounded Theory Analysis. The Qualitative Report, 17, pp.1–23.

RightScale, 2013. RightScale State of The Cloud Report, Available at: http://www.rightscale.com/pdf/rightscale-state-of-the-cloud-report-2013.pdf.

Saedi, A. and Iahad, N.A., 2013. An Integrated Theoretical Framework for Cloud Computing Adoption by Small and Medium-Sized Enterprises. In *PACIS 2013 Proceedings*. pp. 1–12.

Stankov, I., Miroshnychenko, Y. and Kurbel, K., 2012. Cloud Computing Adoption in German Internet Start-up Companies. In *Proceedings of BLED Conference (BLED 2012)*.

Subramanian, B., 2012. The Disruptive Influence of Cloud Computing and its Implications for Adoption in the Pharmaceutical and Life Sciences Industry. *Journal of Medical Marketing: Device, Diagnostic and Pharmaceutical Marketing*, 12(3), pp.192–203.

Swanson, E.B. and Ramiller, N.C., 2004. Innonvating Mindfully with Information Technology. *MIS Quarterly*, 28(4), pp.553–583.

Timmermans, J. et al., 2010. The Ethics of Cloud Computing: A Conceptual Review. In *Proceedings of the Second International Conference on Cloud Computing Technology and Science*. IEEE, pp. 614–620.

Tsai, C. et al., 2010. Information Security Issue of Enterprises Adopting the Application of Cloud Computing. In *Proceedings of the 6th International Conference on Networked Computing and Advanced Information Management (NCM 2010)*. pp. 645–649.

Venters, W. and Whitley, E. a, 2012. A Critical Review of Cloud Computing: Researching Desires and Realities. *Journal of Information Technology*, 27(3), pp.179–197.

Walsham, G., 1995. Interpretive Case Studies in IS Research : Nature and Method. *European Journal of Information Systems*, 4, pp.74–81.

Webster, J. and Watson, R., 2002. Analysing the Past to Prepare for the Future: Writing a Literature Review. *MIS Quarterly*, 26(2), pp.xiii–xxiii.

Wolfswinkel, J.F., Furtmueller, E. and Wilderom, C.P.M., 2011. Using grounded theory as a method for rigorously reviewing literature. *European Journal of Information Systems*, 22(1), pp.45–55.

Yang, C., Hwang, B. and Yuan, B.J.C., 2012. Key Consideration Factors of Adopting Cloud Computing for Science. In *Proceedings of the 4th International Conference on Cloud Computing Technology and Science*. pp. 597–600.

Yang, H. and Tate, M., 2012. A Descriptive Literature Review and Classification of Cloud Computing Research. *Communications of the Association for Information Systems*, 31(1).

Essential Elements of an SME-specific Search of Trusted Cloud Services

Andrea Horch[1], Constantin Christmann[2], Holger Kett[2], Jürgen Falkner[2] and Anette Weisbecker[2]
[1]*Institute of Human Factors and Technology Management IAT, University of Stuttgart, Nobelstrasse 12, Stuttgart, Germany*
[2]*Fraunhofer Institute for Industrial Engineering IAO, Nobelstrasse 12, Stuttgart, Germany*
{*andrea.horch, constantin.christmann*}*@iao.fraunhofer.de*

Keywords: Cloud Computing, Cloud Services, Trust, Semantic Search, Small and Medium-sized Enterprises, SMEs.

Abstract: Cloud computing holds tremendous potential for small and medium-sized enterprises (SMEs) since it offers technologies which can improve their strategic, technical and economic situation. However, along with the benefits there are also risks which concern legal, technical and economic issues associated with cloud computing, which cause concerns to the SMEs. These concerns can be reduced by improving the transparency of cloud services including the understanding of the technology and knowledge about the service providers, already when searching for the appropriate services. To overcome these concerns and increase the transparency cloud service search systems must assist the user during the search process and provide more details about the cloud services, in particular about the attributes which influence the trust of an SME into a service and its provider. The paper introduces a solution for searching appropriate cloud services by focusing on the essential elements. A unique element in this approach is the automated monitoring and evaluation of the provided attributes of the cloud services.

1 INTRODUCTION

Cloud computing is an information technology model where resources, in terms of infrastructure, software applications or data are deployed via the internet by one or several service providers (Zabalza et al., 2012). Following the service model of (Mell and Grance, 2009) cloud computing products can be categorized into Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS). Cloud computing holds great potential for small and medium-sized enterprises (SMEs) since it can improve the strategic, technical and economic situation of an SME (Zabalza et al., 2012).

This paper focusses on the SaaS service model, which covers the provisioning of software typically in form of web applications (Mell and Grance, 2009). In order to make such cloud services accessible for customers the providers of the services have to register in a service registry or a cloud service marketplace. Whereas service repositories only provide an access and collection point for publishing and finding services, marketplaces offer a completely managed environment controlling business terms and conditions (Menychtas et al., 2011).

Along with the benefits there are also risks and legal issues associated with cloud computing causing concerns to the SMEs. The study presented by (Sahandi et al., 2012) shows the main concerns, which are a cause for the slow adoption of cloud computing by SMEs. These concerns are about security and data privacy. When outsourcing IT data, which may contain critical data like business data or customer data, security and data privacy are the key risks for an SME. The SMEs do not have control over the data which they have transferred into the cloud. As cloud services can be accessed by a web interface SMEs fear that this interface could be attacked to get unauthorized access to the application data. Another concern of the SMEs is due to the multiple data centers of the cloud service providers, which are located at different places in the world. Customers do not know at which location their data is stored and so they cannot consider the legal and regulatory issues associated with the physical location of their data.

In this paper we present the essential elements for a trusted service search for SMEs which addresses the user concerns described above. For this purpose the essential elements include in particular (1) a user query transformation and (2) an automated monitoring and evaluation of trust criteria. This paper only focuses the second point of trust monitoring and criteria.

This paper is organized as follows: Section 2

presents related work, then section 3 describes the methodology we did apply for designing a trusted service search for SMEs. In Section 4 an overview of the essential elements of the trusted search system is given and these elements are described in detail. Finally section 6 concludes with a discussion and an outlook.

2 RELATED WORK

An example for a cloud services search is Asperado[1], which offers a marketplace for cloud services including a service search. Asperado marketplace recommends cloud services based on user ratings and the Asperado seal of quality.

The Cloudberia website[2] acts as an open source hub for general information about cloud computing and related technologies. The website runs a service repository where everybody can register own cloud services. The repository offers an API, which acts like a proxy for the clients of the provided services.

A further registry for public cloud services is service-repository.com[3]. It contains a built-in SOAP client to browse registered services and to call their operations. For quality assurance of the registered services service-repository.com offers user ratings and a continuous availability check. Services are registered by providing the WSDL description of the service.

(Kang and Sim, 2010) present Cloudle, a multi-criteria cloud service search engine, which offers a search interface, which assists the user when searching for cloud services. The search interface proposes search criteria and search values provided by a cloud ontology.

AppDirect[4] is a cloud service marketplace and a management platform as it connects application developers with small businesses through a network of white-label marketplaces. It provides a suite of delivery solutions that enable partners to become a gateway for businesses to find, buy, and use cloud services.

Table 1 compares the related work concerning the features for improving cloud service search systems which we have proposed in section 1. The table shows that Asperado and Cloudberia do not implement requirements transformation or automated monitoring of trust criteria. Cloudle uses a cloud ontology for transforming user queries and for offering search options to assist the user during the search process. Nevertheless, the search options offered by the cloud ontology do not provide a flexible search input to the user and the user is not supported in an automated way during the specification of the functional requirements. Service-repository.com implements a monitoring of the service availability and some user ratings but other criteria relevant for the trust of users are not covered. The same holds for AppDirect as it also offers user ratings and comments leaving out other relevant trust criteria.

[1] http://marktplatz.asperado.de/
[2] http://www.cloudberia.com/
[3] http://www.service-repository.com/
[4] http://www.appdirect.com/

3 METHODOLOGY

Figure 1 gives an overview over the individual steps that lead to the description of essential elements for a trusted service search for SMEs as presented in this paper. As a first step we did evaluate the literature and different studies in order to get a feeling for hurdles of SMEs when adopting cloud computing. This evaluation led to the motivation (as summarized in section 1) to address the problems of cloud service adoption with a trusted service finder for SMEs. In a preliminary design step we did sketch the outline of such trusted search system. In order to get to a more detailed design we had to find solutions for the individual elements of this preliminary design. The research in this step focused two crucial features of such a search system: On the one hand, the solution for gathering user requirements in natural language and for transforming them into a formal language for service search and, on the other hand, the identification of promising approaches for an automatic monitoring of trust criteria regarding services and providers. In this paper we focus on topic 2 (Identification of approaches to monitor service criteria), topic 1 will become the key issue of further work.

4 ESSENTIAL ELEMENTS OF A TRUSTED SEARCH SYSTEM

The essential elements of a trusted search system we have identified are:

- Requirements collection
- User query transformation
- Trust monitoring
- Service repository
- Result presenting

Table 1: Comparison of the related work.

	service registry	service marketplace	requirements transformation	automated monitoring of trust criteria
Asperado	●	●	○	○
Cloudberia	●	○	○	○
service-repository.com	●	○	○	◐
Cloudle	●	○	◐	○
AppDirect	●	●	○	◐

Legend: ○ not implemented, ◐ partially implemented, ● implemented

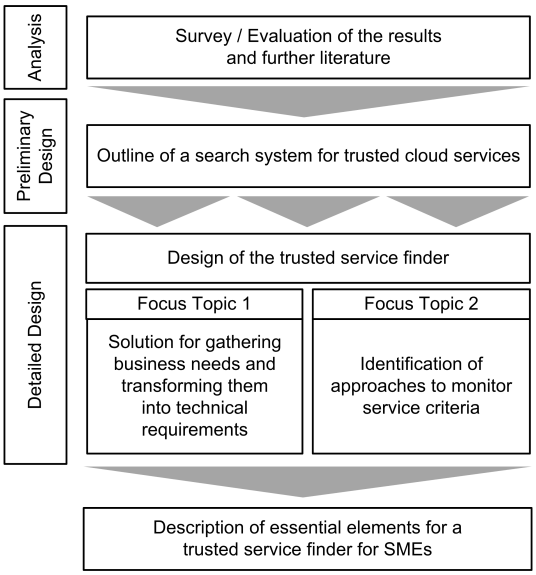


Figure 1: The applied methodology for designing a trusted search system.

In order to conduct a search the user requirements need to be gathered by the system. However, as the user may not have a high technical understanding this has to be done in a language the user is able to understand. In a second step a transformation must be applied where (based on the user input) a technical search profile is deduced, which can then be used by the system to find appropriate services. After performing the service search the result of the search must be presented to the user.

A repository is needed which stores the required attributes to describe a service. Certain attributes of a service and its provider influence the level of trust of an SME (Kett et al., 2012). It is important to increase the quality of the information of the services' attributes. Thus, techniques for trust monitoring and evaluation must be applied, which consecutively check and update service attributes automatically.

The following section describes the system element in detail, which is essential for improving trust.

5 TRUST MONITORING

This section describes our results regarding the identification of promising approaches for automatically monitoring trust criteria of cloud services and providers.

5.1 Categories of Data Sources

As a first step we tried to find adequate categories for possible data sources which can be used to gather information regarding a cloud service or its provider:

- Website of the service/provider: A website covering a cloud service or its provider offers general pieces of information. Typically only positive characteristics of a service are mentioned and eventually the information is even sugarcoated.
- Public registries (i.e. CrunchBase[5]): Such registries offer various information regarding a service or a provider in structured kind of way (i.e. year of the establishment of a company, number of employees, product overview ...). A registry is typically independent form a service/provider and the information in it is relatively trustworthy and up-to-date. Some registries also offer an Application Programming Interface (API) which makes accessing this information easy.
- Official registries (i.e. Electronical Federal Bulletin of Germany[6]): Such registries offer economic figures like the annual sales of a company. Due to the official nature of such registry this information is trustworthy - though it can be outdated. However, it depends on legal regulations if and how a service provider might be represented within such registry and which information is presented. Also automated access to the registry might not be desired.

[5]http://www.crunchbase.com/

[6]https://www.bundesanzeiger.de

- Measurements: Specific operational figures of a service can be directly measured. One example is to measure the uptime of a service by accessing a certain service URL on a regular basis.
- Review by users or experts: On the one hand the search system itself can offer an interface for user feedback regarding cloud services. On the other hand there are social networks, blogs or communities, which might offer valuable information regarding cloud services. Depending on the type of user (random user, employee of the service provider, auditor, certified accountant) the granularity varies by which such evaluation is possible and how trustworthy these results are.

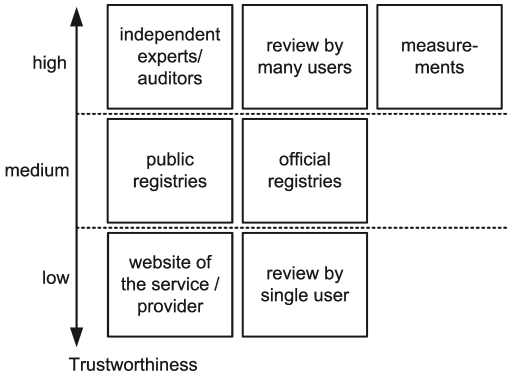


Figure 2: Weighting of the trustworthiness of different data sources for service/provider information.

In order to quantify the trustworthiness of these data sources we used a simple model where the trustworthiness of a data source is directly linked to the trustworthiness of the person/institution offering the information. Furthermore, in our model the trustworthiness is improved if a data source considers the opinions of many independent persons. Figure 2 depicts the trustworthiness of the described data sources based on this model.

5.2 Identification of Monitoring Approaches

For each of these data sources we have looked at the SME-specific trust factors described in (Kett et al., 2012) and tried to find automatic approaches to monitor these. The following list describes these approaches:

- Crawling of service and provider websites: The website of the cloud service and of the service provider can be crawled and relevant information can be extracted.
- Accessing public registries: Some public registries can be accessed through APIs, which offer some information regarding a service/provider.
- Monitoring of social networks and forums: Social networks and forums can be monitored by aggregating and evaluating the individual opinions regarding a service or a provider.
- Evaluation of user interface: The user interface of services can be evaluated by displaying screenshots of random services contained in the repository to users of the search services. The users can then decide if they think that the user interface is visually appealing (similar to the rating website Hot-or-Not[7]).
- Rating of trust aspects: The users of the cloud services can rate and comment the individual service descriptions. Furthermore, the assessment of such ratings and comments (as described in (Noy et al., 2005)) is now established in the internet and can also be adopted by the search system.
- Continuous benchmark: For each service a benchmark can be established (i.e. with a scripting language) which covers typical operations of this service. This benchmark can be run on a regular basis, which allows insight regarding the performance and availability. Furthermore, by checking the protocols used for communication the security of the data transfer can be monitored.
- Measuring popularity: The popularity of a service can be measured by utilizing user profiles and by keeping track of the number of users of a certain user group (i.e. the group of users working in the same business segment) signaled interested in this service. An indicator for interest could be i.e. if a user follows a link provided on the results page.
- Model-based evaluation of quality metrics: By offering the service provider the possibility to provide additional information regarding the internal processes within the service in the form of a Service Effect Specification (SEFF), this SEFF can be used by the search system to perform an evaluation of the performance (and possibly other quality metrics) using i.e. the Palladio Component Model (Rathfelder and Klatt, 2011).

5.3 Evaluation of Monitoring Approaches

In order to make sure that the monitored information is trustworthy on the one hand and on the other hand covers aspects of the cloud service/provider that are relevant to the users, each monitoring approach was

[7] http://www.hotornot.de/

Table 2: Evaluation of monitoring approaches regarding trustworthiness and relevance for users. The percentage value behind the trust aspects is taken from (Kasper et al., 2012) and indicates the percentage of questioned people which did assign a high relevance to this aspect. The trust level behind each data source is based on the model for the trustworthiness of the individual data sources as visualized in figure 2.

	Monitoring approaches							
	Crawling of service and provider websites	Accessing public registries	Monitoring of social networks and forums	Evaluation of user interface	Continuous benchmark	Rating of trust aspects	Measuring popularity	Model-based evaluation of quality metrics
Most relevant trust aspects based on (Kasper et al., 2012)								
Security of data transfer (93 %)	○	○	○	○	●	○	○	○
Security of data storage (92 %)	○	○	○	○	○	○	○	○
Security of data access (91 %)	○	○	○	○	○	○	○	○
Availability / resilience (90 %)	○	○	○	○	●	●	○	●
Functionality meets requirements (83 %)	○	○	○	●	○	●	○	○
Ease of use (83 %)	○	○	○	●	○	●	○	○
Clear presentation of services range (81 %)	○	○	○	○	○	●	○	○
Performance and speed (81 %)	○	○	○	○	●	●	○	●
Backup and security concept (79 %)	●	○	○	○	○	○	○	○
Easy to start with (77 %)	○	○	○	●	○	●	○	○
Coverage of aspects / relevance	**10%**	**0%**	**0%**	**30%**	**30%**	**60%**	**0%**	**20%**
Data sources								
Independent expert/auditor (h)	○	○	○	○	○	○	○	○
Review by many users (h)	○	○	●	●	○	●	○	○
Measurements (h)	○	○	○	○	●	○	●	●
Public registries (m)	○	●	○	○	○	○	○	○
Official registries (m)	○	●	○	○	○	○	○	○
Website of the service/provider (l)	●	○	○	○	○	○	○	○
Review by single user (l)	○	○	○	○	○	○	○	○
Trustworthiness due to data source	**l**	**m**	**h**	**h**	**h**	**h**	**h**	**h**

Legend: ○ no assignment, ● assignment, l = low, m = medium, h = high

evaluated by determining the following two characteristics:

- Trustworthiness: The trustworthiness was derived from the trustworthiness of the involved data source (high, medium or low).
- Relevance: The relevance of the monitored information was determined based on the survey results of (Kasper et al., 2012). This study offers the percentage of questioned people which did assign a high relevance to various aspects of cloud services and providers[8]. By selecting the aspects with the highest percentage values we filtered out the 10 most relevant aspects. In order to quantify the relevance of a monitoring approach we counted how many of these 10 aspects are touched by the monitoring.

[8]The survey in (Kasper et al., 2012) was conducted by questioning employees in german craft businesses. Typically such craft businesses can be categorized as SMEs, so this survey offers insight into the weighting of trust aspects from the persective of SMEs.

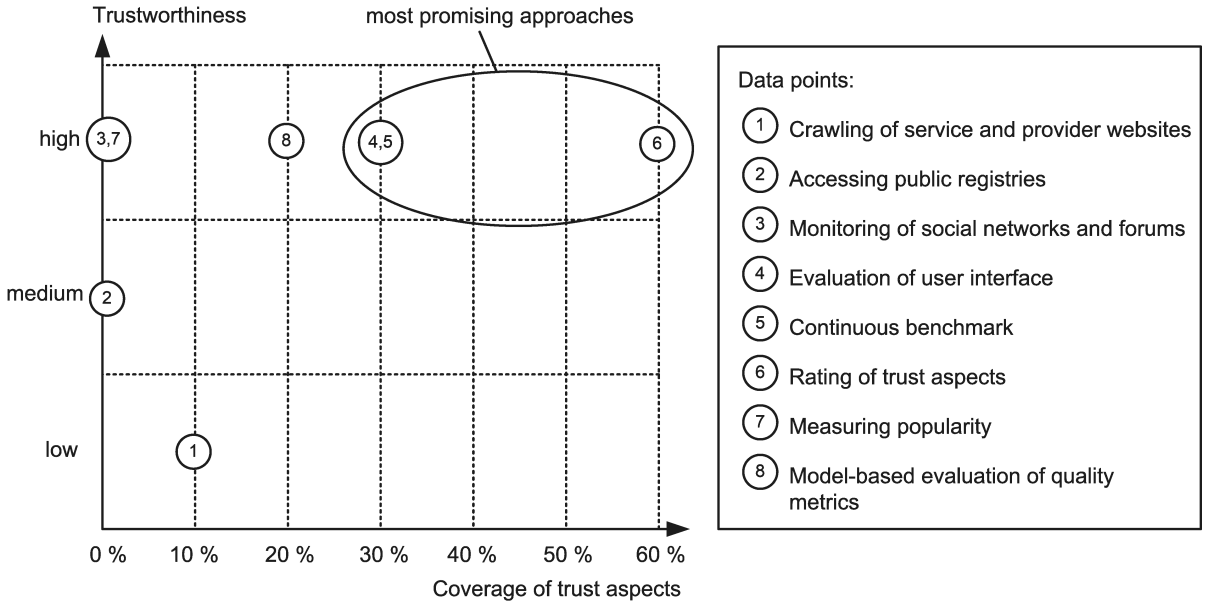


Figure 3: Distribution of monitoring approaches based on the associated trustworthiness and their coverage of trust aspects.

In table 2 the result of this evaluation is shown and figure 3 visualizes the distribution of all monitoring approaches based on this evaluation. As described in section 4 the trust monitoring is an essential element of a trusted search system and due to this distribution the rating of trust aspects by users, the evaluation of user interface and the continuous benchmark are promising approaches for this trust monitoring, as they cover relatively many important trust aspects and their data source has a high level of trust.

The most trustworthy and efficient monitoring approaches are:

1. Rating of trust aspects: For this approach the data source are the ratings of many users. There is a need for a structured approach, where every trust aspect is rated. Many user ratings produce high quality results.
2. Evaluation of user interface: This approach reflects the subjective opion of a user. It is an unstructured proceeding to get the average rating of all users regarding to functionality and usability of a service.
3. Continuous benchmark: Proceeding to get objective and reliable conclusions about trust aspects of a service.
4. Monitoring of social networks and forums: This approach holds significant potential to collect values for many trust aspects. Currently there are to less communities where data could be collected, but since there will be built up more communities in future, the data which could be collected would be a very good data source.

6 OUTLOOK & CONCLUSION

In this paper we have presented the essential elements for a search system for cloud services with a focus on business users from SMEs. The focus of our contribution lies on the following aspect:

- Identification of promising approaches for an automated monitoring of specific service/provider attributes which are relevant to the trust of users.

It is the intention of the authors to actually build such a search system using the described elements as guidance for its design. This development will be carried out in the context of the research project CLOUDwerker[9] and here the application domain for the implementation will be craft business. Besides the implementation of the search system we are going to examine the process of service selection of SMEs in order to better support this process by offering adequate information and guidance to potential customers of cloud services.

[9] http://www.cloudwerker.de/

ACKNOWLEDGEMENT

This work was funded by means of the German Federal Ministry of Economy and Technology under the promotional reference “01MD11041”. The authors take the responsibility for the contents.

REFERENCES

Kang, J. and Sim, K. M. (2010). Cloudle: A multi-criteria cloud service search engine. In *Proceedings of the 2010 IEEE Asia-Pacific Services Computing Conference*, pages 339–346. IEEE Computer Society.

Kasper, H., Kett, H., and Weisbecker, A. (2012). *Potenziale von Cloud Computing im Handwerk*. Fraunhofer-Verlag.

Kett, H., Kasper, H., Falkner, J., and Weisbecker, A. (2012). Trust factors for the usage of cloud computing in small and medium sized craft enterprises. In Vanmechelen, K., Altmann, J., and Rana, O. F., editors, *Economics of Grids, Clouds, Systems, and Services*, volume 7714 of *Lecture Notes in Computer Science*, pages 169–181. Springer-Verlag.

Mell, P. and Grance, T. (2009). The nist definition of cloud computing. Technical report, National Institute of Standards and Technology (NIST).

Menychtas, A., Gatzioura, A., and Varvarigou, T. (2011). A business resolution engine for cloud marketplaces. In *Third International Conference on Cloud Computing Technology and Science 2011 (CloudCom)*, pages 462–469. IEEE Computer Society.

Noy, N. F., Guha, R., and Musen, M. A. (2005). User ratings of ontologies: Who will rate the raters. In *Proceedings of the AAAI Spring Symposium on Knowledge Collection from Volunteer Contributors*, pages 56–63.

Rathfelder, C. and Klatt, B. (2011). Palladio workbench: A quality-prediction tool for component-based architectures. In *9th Working IEEE/IFIP Conference on Software Architecture 2011 (WICSA 2011)*, pages 347–350. IEEE Computer Society.

Sahandi, R., Alkhalil, A., and Opara-Martins, J. (2012). Smes' perception of cloud computing: Potential and security. In Camarinha-Matos, L. M., Xu, L., and Afsarmanesh, H., editors, *Collaborative Networks in the Internet of Services*, volume 380 of *IFIP Advances in Information and Communication Technology*, pages 186–195. Springer-Verlag.

Zabalza, J., Rio-Belver, R., Cilleruelo, E., Garechana, G., and Gavilanes, J. (2012). Benefits related to cloud computing in the smes. In *6th International Conference on Industrial Engineering and Industrial Management*, pages 637–644.

Automatic Software Development as a Service (ASDaaS)

Hind Benfenatki[1], Catarina Ferreira Da Silva[1], Nabila Benharkat[2],
and Parisa Ghodous[1]
[1]*Université Lyon 1, LIRIS, UMR5205, F-69622, Lyon, France*
[2]*INSA Lyon, LIRIS, UMR5205, F-69621, Lyon, France*
{*hind.benfenatki, catarina.ferreira-da-silva*}*@liris.cnrs.fr, nabila.benharkat@insa-lyon.fr, ghodous@liris.cnrs.fr*

Keywords: Cloud Computing, Business Applications, Requirement Expression, Linked-Data.

Abstract: Cloud-based services have become a norm for business application development. With Cloud Computing and the convergence toward Everything as a Service (XaaS), we no longer consider the classical context of application development, where IT teams or integrators are solicited. Current approaches in Cloud environments are usually designed for a specific Cloud platform; moreover, they are only designed for technical users. To overcome the lack of generic and complete methodology for business application development, we propose a methodology for Automatic Software Development as a Service (ASDaaS), which is designed for non-technical users and promotes services reuse. In this paper, we focus on the phase of business software requirement gathering of our methodology. We define the requirement vocabulary based on linked data principles, and extend the Linked-USDL language to describe business stakeholder requirements as service functions, business constraints, user preferences and QoS parameters. Our approach is illustrated with an e-commerce example.

1 INTRODUCTION

Web and Cloud services are a popular medium for application development and deployment on the Cloud. Modern enterprises are moving towards Cloud service-oriented architectures to promote reuse and interoperability of services; and to benefit from the Cloud Computing advantages, such as small initial investment, no license acquisition, accessibility from everywhere and every time, high availability and so on.

With Cloud Computing and the convergence toward Everything as a Service, we no longer consider the classical context of application development, where IT teams or integrators are solicited to perform software development. We propose a business application development process that minimizes business stakeholder intervention and meets Cloud characteristics.

Let us consider an e-commerce scenario in which a business stakeholder in a European company "A" would automate the IT hardware acquisition process. The business stakeholder wants to make available to its staff a Cloud hosted business application that allows the following functions: (i) to select products with the edition of purchase orders, (ii) to proceed to the online purchase, (iii) and to make delivery of the product. The business stakeholder has to describe functional and non-functional requirements. Functional requirements include application business functions and business constraints. One of the business constraints is the budget, and for this "A" imposes a maximum purchase order value equal to 1000 euros. Non-functional requirements include user preferences and QoS Parameters. The business stakeholder prefers to use "PayPal Service" for the online purchase. For the deployment of the business application, the company "A" relies on previous good experience using "Amazon EC2" (http://aws.amazon.com/fr/ec2/) and prefers to deploy its business application on this same platform. Company "A" prefers to pay deployment costs with the European currency "euro". This scenario requires great caution with respect to data privacy and data integrity due to the online payments service. For maintaining the confidentiality of their data, "A" prefers invoked services be located in European region.

Based on our scenario, we identify the following three challenges, which are addressed in the paper: (i) how to develop a Cloud business application that meets the business stakeholder requirements with minimal intervention and that leverages the Cloud

benefits. Nowadays, there is a lack of a standard approach for Cloud applications development, as this research aspect is still in progress, (ii) how to specify and model the functional and non-functional business requirements. There is a need for a model that provides explicit description of requirements to ensure then the correct selection of services, and (iii) how to adapt or extend service description languages to facilitate services selection that meet user's requirements.

In this paper, we propose a methodology for business application development for Cloud environments allowing business stakeholders to perform the automatic development of those applications. This methodology is called Automatic Software Development as a Service (ASDaaS) and promotes the discovery and the composition of Cloud services that match functional and non-functional business application requirements. Functional requirements describe service features and business constraints. Non-functional requirements describe user preferences and Quality of Service (QoS) parameters.

The originality of our work presented in this paper lies in the business application requirement description phase of the methodology. We define the requirement vocabulary based on the linked data principles (http://www.linkeddata.org/), and extend the Linked USDL language (http://www.linked-usdl.org/) so that it can describe business stakeholder's requirements as service functions, business constraints, user preferences and QoS parameters.

The rest of this paper is organised as follows. Section 2 describes the works related to existing Cloud software development methodologies. Section 3 presents the proposed ASDaaS methodology. Section 4 introduces ASDaaS's architecture. We describe the implementation and evaluate our work in section 5. Section 6 draws final conclusions and describes our future work.

2 RELATED WORKS

In Cloud Computing paradigm, there is a lack of complete application development methodologies. However, several partial approaches for the development of applications exist in literature. In (Sledziewski et al., 2010), the authors propose an approach based on Domain Specific Languages (DSL) within the development process. The main inconvenient with this approach is the huge time that consumes the DSL development in early phase of their approach.

Giove and colleagues (Giove et al., 2013) propose a library called CPIM (Cloud Provider Independent Model) abstracting from the details that are specific of the underlying PaaS provider; and allowing an application developer to implement his application in a PaaS independent way. At deployment time, the developer specifies the PaaS to be used. At runtime, CPIM library acts as a mediator between the application code and the services offered by the PaaS. Our work will reuse and integrate this interesting approach for developing undiscovered services.

In (Ardagna et al., 2012), the authors propose the MODACLOUDS system, a European project (http://www.modaClouds.eu/.) that uses the principle of MDD (Model Driven Development) for the development of applications on the Cloud. Applications are designed at a high level of abstraction of the target Cloud, making them capable of operating on multiple Cloud platforms. The main lack in this work is that the PaaS Cloud services selection is only taking into account QoS parameters to the detriment of the platform's Application Programming Interfaces (APIs).

The work proposed by (Kommalapati et al., 2011) describes a SaaS Development Life Cycle (SaaSDLC). The authors present an approach that promotes evaluation of the Cloud provider based on capabilities of a platform. The SaaSDLC does not consider reuse of Cloud services. It promotes the development to a specific platform, making application portability more difficult.

In (Guha et al., 2010), the authors advocate the intervention of Cloud provider in the Agile eXtreme Programming software development process, especially in planning, designing, building, testing and deployment phases to mitigate the challenges associated with Cloud software development, and make it more advantageous. These authors integrate the notion of roles for the various stakeholders in the agile development process for Cloud applications, but do not consider the other characteristics of Cloud applications that can influence the development process.

In (Sun et al., 2010), the authors describe Service-Oriented Software Development Cloud (SOSDC), a Cloud platform for developing service-oriented software and a dynamic hosting environment. The SOSDC adopts an architecture covering the three levels of Cloud services. The IaaS level is primarily responsible for providing infrastructure resources. The PaaS level provides App Engine for testing, implementing and monitoring the deployed application without having to consider the technical details. SaaS level aims to provide "Online Service-Oriented Software Development Environment". This approach aims to supply a dynamic development environment by providing on demand appliance for developers, it is dedicated to a specific platform and does not exploit public Cloud platforms.

In summary, the state of the art analysis shows there is a need for a complete methodology that guides Cloud-based developers to perform the development of Cloud service-oriented business applications, considering the constraints and benefits of Cloud environments.

The methodology we propose in the next section, (i) obeys the SOA principles and techniques that promote the reusability, the loose coupling and the composability of the underlying XaaSs; (ii) meets the requirements of the distributed nature of Cloud; (iii) aims to make software development more accessible for non IT-professionals; and (iv) is independent of a particular platform.

3 AUTOMATIC SOFTWARE DEVELOPMENT AS A SERVICE METHODOLOGY (ASDaaS)

Our methodology for Automatic Software Development as a Service (ASDaaS) is aimed at business stakeholders who need a business application to automate a frequent task. In this section, we describe the ASDaaS methodology through the described scenario.

The originality of our approach lies in the following: (i) it allows selection of a development platform (PaaS) according to user preferences and restrictions of available Cloud services, such as suitable APIs, and (ii) it allows selection of IaaS for deployment that meets the predefined user preferences and QoS parameters.

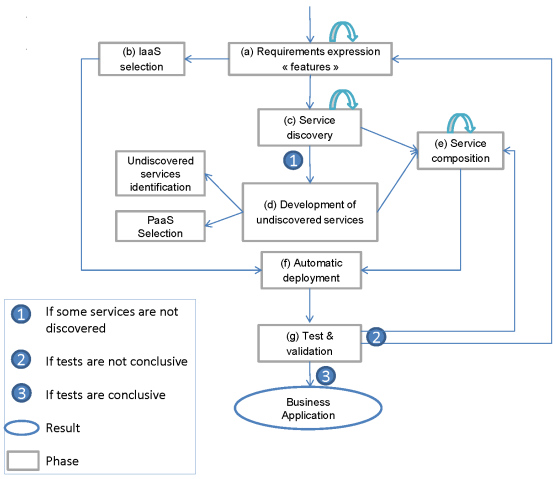


Figure 1: Automatic Software Development as a Service methodology (ASDaaS).

Figure 1 illustrates the different phases of the methodology, which are the following:

a) Requirements expression;

b) IaaS selection for application deployment;

c) Service discovery;

d) Development of undiscovered services;

e) Service composition;

f) Automatic deployment of business application;

g) Tests and Validation of the deployed business application.

These phases are explained in the next sections. In this paper we focus on business application requirement expression, i.e. the initial phase of the ASDaaS methodology.

3.1 Requirements Expression

Services description languages such as WSDL (Web Service Description Language) focus on the description of functional aspects of software component interfaces (Coulouris et al, 2011) to the detriment of non-functional aspects.

The USDL (Unified Service Description Language) was created to address WSDL shortcomings by describing business, operational and technical aspects in services description. Linked USDL consists on reuse of existing vocabularies and ontologies for services description. The rationale behind the use of Linked Data is the requirement that USDL descriptions should be shared between interested parties and linked to other descriptions, standards and formats (Cardoso et al, 2013).

We extend Linked USDL with a new module enabling to describe business application requirements. This way it will be easier to match business stakeholder's requirements with Cloud services described in Linked USDL.

The business stakeholder describes his/her business application requirements (functional and non-functional). Functional requirements are the application features and the business constraints imposed by the stakeholder's organization. Non-functional requirements are the QoS parameters and stakeholder preferences. The latter describes, among others, the location of services an payment means. The QoS parameters describe the stakeholder's business application requirements in terms of quality of service of Cloud platforms and infrastructures.

Following the predefined scenario, the stakeholder defines his/her business application requirements as follows:

- Application functions:
  - Laptop review service,

- Computer purchase service,
- Delivery service.

- Business constraints:
  - Purchase cost max value = 1000 Euros.
- QoS parameters: in order to describe its priorities regarding QoS parameters, he/she should assign coefficients to QoS parameters so that the sum of all the coefficients does not exceed 10. As business application integrates online payment, the stakeholder prefers to value data privacy and integrity more than service response time with the following coefficients:
  - Data privacy: 3,
  - Data integrity: 3,
  - Availability: 3,
  - Response time: 1.
- Stakeholder preferences:
  - Services location: Europe,
  - Currency: Euro,
  - Cost max value: 500,
  - Preferred deployment provider: ”Amazon”,
  - Preferred provider for the purchase service: ”Paypal”.

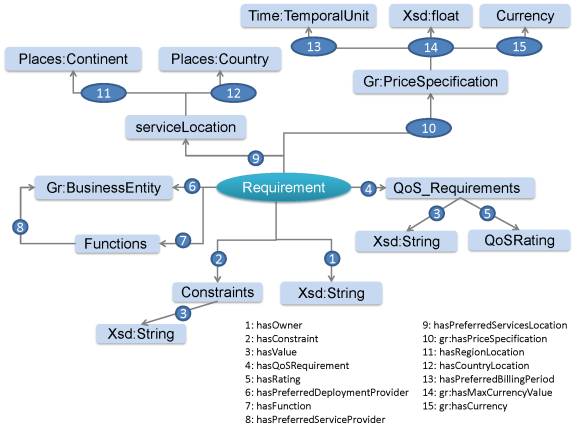


Figure 2: USDL-requirement vocabulary.

The existing USDL modules (https://github.com/linked-usdl.) are not sufficient to describe non-functional aspects of stakeholder business requirements, such as the preference for invoking a business provider rather than another. To achieve the goal of providing a unified language to describe business application requirements, we propose to extend the Linked USDL language by defining a ”USDL-requirement” vocabulary, which will allow matching the business application requirements with the Linked USDL Cloud service descriptions. Multiple vocabularies exist (http://lov.okfn.org/dataset/lov/); we will reuse some of their classes and properties following linked data principles. The USDL-requirement vocabulary describes business application requirements in terms of stakeholder preferences, QoS parameters, service functions, and business constraints. Figure 2 illustrates classes and properties that we define in the USDL-requirement vocabulary. Rectangles represent classes, and the circles represent the properties. For example, the property 7 illustrates that a requirement comprises functions, and the property 8 shows that the stakeholder can define his preferred provider for a given function. The gr:BusinessEntity is a Good Relation’s (Hepp 2011) class that describes, in our context, a service provider.

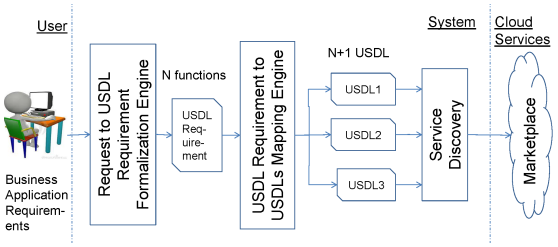


Figure 3: Requirements to USDL translation.

Figure 3 illustrates the matching process from Linked USDL-requirement business application description to Linked USDL Cloud services. Once the stakeholder expresses its business application requirements, these are formalized according to the template of our ”USDL-requirement” vocabulary. This process generates a file that is further divided into several USDL files, each one describing a desirable Cloud service. These USDL files provide input for the Cloud service discovery process. If the USDL-requirement file describes ”N” functions, the USDL-requirement to USDL mapping engine generates ”N+1” USDL files describing abstract services. ”N” USDL files correspond to the ”N” functions, plus one USDL file describing the deployment platform.

Listing 1 illustrates the functions expression using USDL-requirement vocabulary in our scenario. Line 3 presents the hasFunctions property for describing several functions needed by the stakeholder (from line 4 to 6). Lines 8 to 10 show that the stakeholder prefers invoking the paypal service for the computing purchase function. From this listing file, four USDL service files are generated, corresponding to the requirements description of abstract services, which are the products review service, the computer purchase service, the delivery service and the IaaS deployment platform. The service discovery process receives these USDL files as input.

```
1: @prefix rq: <usdl-requirements vocabulary>.
2:
3: rq: hasfunctions
4:  :laptop review service,
5:  :computer purchase service,
6:  :delivery service.
7:
8: :computer_purchase_service a rq:function
9:   rq:hasPreferedProvider
10:  [a gr:BusinessEntity :Paypal]
```

Listing 1: Functions expression with USDL Requirement vocabulary.

3.2 Service Discovery

The process of service discovery starts from the USDL files describing abstract services for outputting a business service or a composite business service.

In this phase, through a business services search engine, we have to discover business services corresponding to several required functions and meeting location, currency and preferred provider criteria. For each service, several services can be discovered and have to be ranked. The advantage of this phase is to reduce the workload and improve reusability of business services. If no business services are discovered for some abstract services, we move on to the phase of service automatic development.

We have to discover services in a Cloud marketplace by matching the USDL describing abstract services and the marketplace's USDL Cloud services. Table 1 illustrates discovered services for the requirements described in our scenario with their QoS indicators and cost. Where Q1 represents data privacy, Q2 is data integrity, Q3 data loss, Q4 access control, Q5 availability, Q6 response time. Values of Q1 to Q5 represent satisfaction percentage of each indicator. We assume that these values are retrieved from a history of invoking these services.

Table 1: Scenario's discovered services.

Needed functions	Discovered services	Q1 (%)	Q2 (%)	Q3 (%)	Q4 (%)	Q5 (%)	Q6 (ms)	Cost (€/month)
Products review	S1	90	85	0	100	90	10	20
	S2	100	95	2	98	80	20	40
Computer purchase	Paypal	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	32
Delivery	S3	60	50	5	70	60	30	100
	S4	85	90	3	90	50	5	59
IaaS	Amazon	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	26

Currently, for the discovery of existing services, we apply a syntactical matching. The next section describes how the services selection and their composition are done.

3.3 Service Selection and Composition

The service selection phase translates as a multi-criteria decision making problem (Figueira, 2005). Based on stakeholder's preferences and QoS parameters, we weight the discovered services to select a service that meets most requirements and preferences of the user. The service provider with the highest rank will be selected. The rank is calculated based on the coefficients associated to QoS attributes. Two scenarios are available: Let S_i be a service and Q_i an indicator.

$$R(S_i, Q_j) = \begin{cases} R_{upper} \\ R_{lower} \end{cases} \quad (1)$$

Case 1: the higher the value of the attribute is, better is the service, for instance, the service availability. In this case the rank associated with this attribute for a given provider is calculated as follows:

$$R_{upper} = \frac{Value}{Max} * Coefficient \quad (2)$$

Where: Value is the value of the attribute for a given provider, Max is the maximum value of the attribute among all providers, Coefficient is the coefficient previously assigned to the attribute by the stakeholder. The sum of several coefficients for all attributes does not exceed 10.

Case 2: the smaller the value of the attribute is, for instance, the response time, better is the service. In this case the rank associated with this attribute for a given provider is calculated as follows:

$$R_{lower} = (1 - \frac{Valeur}{Max}) * Coefficient \quad (3)$$

Let $R(S_i)$ be the global ranking regarding the whole indicators for a service S_i. The IaaS provider with the highest rank is selected.

$$R(S_i) = \sum_{j=1}^{n} R(S_i, Q_j) \quad (4)$$

In our scenario, the service discovery phase generates a list of services corresponding to different functions described by the user. For each function, the service with the highest rank is selected. The cost is considered as a global criterion. Table 2 illustrates cost and rank calculation for each selected service in our running scenario. Service S1 is selected for the laptop review service requirement. Service S4 is selected for the delivery service requirement. In our scenario, the stakeholder has chosen "Amazon" as an

IaaS provider. The final composition includes the services S1, S4, Paypal and Amazon.

Table 2: Service costs and ranks.

Services	Services rank	Service cost (€/month)
S1	R(S1)=(90/100)*3+(85/95)*3 +(90/90)*3+(1-(10/20))*1 =8.88	20
S2	R(S2)=(100/100)*3 +(95/95)*3+(80/90)*3 +(1-(20/20))*1=8,66	40
Paypal	/	32
S3	R(S3)=(60/85)*3+(50/90)*3 +(60/60)*3+(1-(30/30))*1 =6.78	100
S4	R(S4)=(85/85)*3+(90/90)*3 +(50/60)*3+(1-(5/30))*1 =9.33	59
Amazon	/	26

The service composition research domain has been subject of several surveys explaining methodologies, approaches and composition languages (Milanovic, 2004), (Srivastava, 2003).

In our methodology the service composition phase combines the selected and the developed services. The composition of services is a continuous process, which does not stop even after the deployment of the business application. A VCS (Version Control System) will manage different versions of compositions for each business application in order to allow backtracking, if necessary. If a new service S1' is discovered, with the same functionalities as S1 and better QoS, S1' replace S1 and a new composition with S1' instead of S1 is made, then a new version of the business application is generated.

3.4 IaaS Selection for Application Deployment

A Cloud infrastructure is either chosen by business stakeholder or automatically selected for the deployment of the business application, according to performance indicators required by the user and the QoS parameters characterizing the Cloud Infrastructure. IaaS selection is performed by ranking different IaaS, according to QoS indicators (response time, availability, accessibility, security, etc.) and following the ranking and selection processes described in section 3.3.

3.5 Development of Undiscovered Services

In traditional systems of service discovery, the discovery process will output the discovered business services, and gives no output if no services are discovered. In our methodology, when the service discovery algorithm does not return a result, we propose a development of the service. Undiscovered services are developed from UML (Unified Modelling Language) models. This phase identifies and develops the features described in the stakeholder's business application requirements that have not been discovered as business services. This involves three key steps:

1. *Undiscovered Services Modeling*. Services are modeled in UML without imposing a particular platform, in order to allow automatic generation of their code for a targeted Cloud platform with full knowledge of the tools it offers. Our work will reuse and integrate the CPIM library proposed by Giove and colleagues (Giove et al, 2013) for developing the undiscovered services.
2. *Discovery and Selection of Development Platforms (PaaS) for each service*. This phase allows to select the platform based on the service to be developed and the technologies allowed by the PaaS, as well as on the QoS requirements. This creates a distributed multi-platform development. The selection of platforms is the same as for the selection of infrastructure, with the only difference that we consider the APIs offered by different providers. The benefits of choosing a PaaS for each service development are: (i) the development is custom made to each service, (ii) dependencies of a particular PaaS are reduced, because a specific PaaS deployment is based on a MDD approach, (iii) addressing the distributed nature of the Cloud, thus increasing the availability of the final application.
3. *Automatic Deployment and Publication of New Developed Services*. After the development of new undiscovered services, these are deployed and published in a marketplace of services.

3.6 Automatic Deployment

The automatic deployment phase consists in automatically deploying the resulted business application on the IaaS preferred by the business stakeholder or on the one automatically selected by our system according to stakeholder's preferences and QoS parameters.

The deployment occurs until achieving stakeholder satisfaction. If after the tests and validation

phase, tests are not conclusive and/or the stakeholder does not validate results of the composition, IaaS resources are freed, and another composition is selected and deployed until stakeholder's satisfaction is achieved. Or the stakeholder can proceed to the refinement of its requirements.

3.7 Tests and Validation

The acceptance tests phase occurs after the business application deployment, where the stakeholder tests the deployed business application and verifies application features and the respect of business constraints, with the aim of validating or bringing modifications to the requirements expression.

If tests are conclusive, the stakeholder validates the deployed application. Else, other services are selected from the service marketplace, composed and deployed, or, the stakeholder introduces changes in his/her requirements expression for the business application, after, he/she has to test and validate the new deployed business application.

In the next section, we describe the architecture for ASDaaS. This architecture applies the previously explained methodology.

4 ARCHITECTURE FOR AUTOMATIC SOFTWARE DEVELOPMENT AS A SERVICE

The figure 4 illustrates the architecture for business application development as a service, which consists of three levels: client level, system level and service level. The client level allows to describe the application requirements (business functions, business constraints, QoS parameters and stakeholder preferences) via a web form; and to access to the deployed business application via internet.

The system level describes the core of the architecture and is the focus of the remainder of this section. It consists of the following services: project management, discovery as a service, PaaS discovery as a service, IaaS discovery as a service, undiscovered services development as a service, composition as a service and automatic deployment as a service.

The project management service is in charge of managing several projects. The discovery as a service is responsible for the service discovery operation based on service functions, QoS parameters and stakeholder preferences. The undiscovered services development as a service provides development of undiscovered services from models, on the Cloud

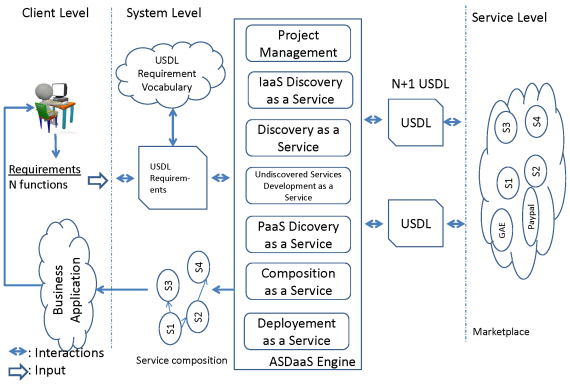


Figure 4: Automatic Software Development as a Service Architecture.

platform selected by the service PaaS discovery as service. This module generates the code exploiting the PaaS APIs and respecting the Cloud provider architecture. The developed services are then deployed. The "PaaS Discovery as a Service" module enables to find PaaS platform where each new developed service will be deployed. The IaaS discovery as a service enables the choice of an IaaS infrastructure for the deployment of the business application. This phase needs to match the QoS characteristics of different IaaS with those desired by the stakeholder. The deployment as a service consists of automatically deploying the generated business application on a selected infrastructure.

The service level includes the marketplace in which our system selects services for composition.

5 IMPLEMENTATION & VALIDATION

In order to validate and evaluate our proposal, we are implementing a prototype for ASDaaS using Java language. The phases of requirements expression, services discovery and IaaS selection are implemented. We use the Apache Jena Library for modeling and manipulating the USDL files and the SPARQL Query Language for querying them. We have implemented an IT history service that is responsible for returning the QoS parameters values for each discovered service.

Due to the early stage of implementation, few tests have been done. We have evaluated the service discovery execution time variation with the number of services in the marketplace. Figure 5 depicts these results obtained for the service discovery phase for our running scenario.

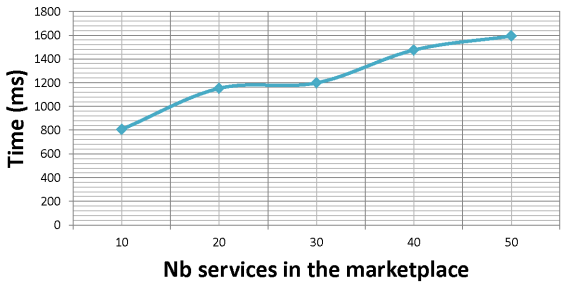


Figure 5: Variation of the service discovery execution time versus the marketplace's number of services.

6 CONCLUSIONS

With Cloud computing, the application development is changing, providing environments enabling to empower business stakeholders and to automatize the software development activities. In this paper, we have described a methodology for Cloud-based collaborative software development, and then presented the ASDaaS architecture that addresses the challenges of interoperability, independency to a specific platform, and respects the distributed nature of the Cloud environment. ASDaaS generates a business application based on discovered and developed on-the-fly business Cloud services.

Currently we are implementing the business Cloud service composition module. In the future, we will deploy the ASDaaS architecture in the Cloud Platform of the Lyon 1 University for evaluating and testing this work with a large set of cloud services.

REFERENCES

Amazon. (2013). Available: http://aws.amazon.com/fr/ec2/

Ardagna, Danilo., Di Nitto, Elisabetta., Casale, Giuliano., Petcu, Dana., Mohagheghi, Parastoo., Mosser, Sbastien., Matthews, Peter., Gericke, Anke., Ballagny, Cyril., DAndria, Francesco., Nechifor, Cosmin-Septimiu., and Sheridan, Craig. (2012). MODACLOUDS: A Model-Driven Approach for the Design and Execution of Applications on Multiple Clouds. *ICSE Workshop on Modeling in Software Engineering* (MISE), 2012. pp 50-56.

Cardoso, J. (2013). A Unified Language For Service Description: A Brief Overview. Available: http://www.issip.org/2013/04/26/a-unified-language-for-service-description-a-brief-overview/

Cardoso, J. B. (2010). Towards a unified service description language for the internet of services: Requirements and first developments. *IEEE International Conference on Services Computing, Florida*, USA (2010).pp 602 - 609.

Cardoso, J., Pedrinaci, C., Leidig, T., & Rupino, P. a. (2012). Open semantic service networks. International Symposium on Services Science 2012 (ISSS 2012). pp 141-154.

Coulouris, G. D., Dollimore, J., Kindberg, T., and Blair, G., (2011). Distributed Systems: Concepts and Design. Fifth Edition, published by Addison Wesley, May 2011.

Figueira, J.; Greco, S. and Ehrgott M.,. (2005). Multiple Criteria Decision Analysis: State of the Art Surveys. *Springer, 2005*.

Giove, Filippo., Longoni, Davide., Shokrolahi Yancheshmeh, Majid., Ardagna, Danilo. and Di Nitto, Elisabetta. (2013). An Approach for the Development of Portable Applications on PaaS Clouds. *CLOSER 2013 - 3rd International Conference on Cloud Computing and Services Science*, 2013. pp 591- 601.

Hepp M. (2011). GoodRelations Language Reference. Available: http://www. hepp-netz.de/ontologies/goodrelations/v1.html

Guha, Radha. and Al-Dabass, David. (2010). Impact of Web 2.0 and Cloud Computing Platform on Software Engineering. *IEEE, International Symposium on Electronic System Design,* 2010. pp 213-218.

Kommalapati, Hanu., and Zack, William H. (2011). The SaaS Development Lifecycle. 2011. Available: http://www.infoq.com/articles/SaaS-Lifecycle.

Linked Data - Connect Distributed Data across the Web. (2013). Available: http://www.linkeddata.org/

Linked Open Vocabularies (LOV). (2013). Available: http://lov.okfn.org/dataset/lov/

Linked USDL. (2013). Available: http://www.linked-usdl.org/

Milanovic, N. a. (2004). Current Solutions for Web Service Composition. *IEEE Internet Computing.* Vol 8. pp 51-59.

Linked USDL modules. (2013). Available: https://github.com/linked-usdl.

MODACLOUDS. Available: http://www.modaClouds.eu/.

Sledziewski, Krzysztof., Bordbar, Behzad., and Anane, Rachid. (2010). A DSL-based Approach to Software Development and Deployment on Cloud. *24th IEEE International Conference on Advanced Information Networking and Applications,* 2010. pp 414-421.

Srivastava, B. a., Koehler, J., (2003). Web service composition-current solutions and open problems. *Workshop on Planning for Web Services*. page 28-35.

Sun, Hailong., Wang, Xu., Zhou, Chao., Huang, Zicheng., Liu, Xudong.,. (2010). Early Experience of Building a Cloud Platform for Service Oriented Software Development. *2010 IEEE International Conference on Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS)*. pp 1-4.

Iris: An Inter-cloud Resource Integration System for Elastic Cloud Data Centers

Ryousei Takano, Atsuko Takefusa, Hidemoto Nakada, Seiya Yanagita and Tomohiro Kudoh
Information Technology Research Institute, National Institute of Advanced Industrial Science and Technology (AIST), Central 2, 1-1-1 Umezono, Tsukuba, Ibaraki 305-8568, Japan
{*takano-ryousei, atsuko.takefusa, hide-nakada, s.yanagita, t.kudoh*}*@aist.go.jp*

Keywords: Inter-cloud, Virtual Infrastructure, Hardware as a Service, Nested Virtualization, Software Defined Network.

Abstract: This paper proposes a new cloud computing service model, Hardware as a Service (HaaS), that is based on the idea of implementing "elastic data centers" that provide a data center administrator with resources located at different data centers as demand requires. To demonstrate the feasibility of the proposed model, we have developed what we call an Inter-cloud Resource Integration System (Iris) by using nested virtualization and OpenFlow technologies. Iris dynamically configures and provides a virtual infrastructure over inter-cloud resources, on which an IaaS cloud can run. Using Iris, we have confirmed an IaaS cloud can seamlessly extend and manage resources over multiple data centers. The experimental results on an emulated inter-cloud environment show that the overheads of the HaaS layer are acceptable when the network latency is less than 10 ms. In addition, we reveal the large overhead from nested virtualization and show positive prospect for this problem. We believe these results provide new insight to help establish inter-cloud computing.

1 INTRODUCTION

The inter-cloud paradigm provides a new perspective on computing services by connecting distributed data centers through high-speed networks. It is also an attractive infrastructure for big data processing, disaster recovery, and highly available applications. However, an optimal approach to how cloud providers federate distribute resources to construct a virtual infrastructure over distributed inter-cloud resources is not well understood.

This paper focuses on inter-cloud federation among Infrastructure as a Service (IaaS) clouds. IaaS provides isolated sets of resources, including computers, storage, and networks, with multiple users. Typical IaaS clouds are managed by an IaaS platform middleware, including OpenStack and CloudStack. IaaS may require large amounts of resources as necessary, however the capacity of physical resources in a single data center is limited. The inter-cloud federation is a most promising solution for this problem. Some researcher has proposed inter-cloud architectures to facilitate cloud federation and interoperability (Buyya et al., 2010; Demchenko et al., 2012). Some academic and industrial groups have been standardizing a service model, a protocol, and an interface such as the Open Cloud Computing Interface (Nyren et al., 2011), the Cloud Data Management Interface (SNIA, 2012), and the Open Virtualization Format (DMTF, 2013). However, a single standard has not yet found wide-spread acceptance because standardization takes a long time to implement on most of IaaS platforms.

This paper proposes a novel inter-cloud service model called *Hardware as a Service (HaaS)*, which enables us to implement "elastic data centers." The term HaaS was originally introduced in 2006 (Carr, 2006), but now it has been superseded by IaaS. We have redefined the term HaaS as "on-demand provisioning of resources for IaaS providers" in this paper. In contrast to an IaaS model, our HaaS model targets on building a virtual infrastructure among multiple data centers. This means that a HaaS provider is a resource broker for data center administrators, i.e., IaaS providers. While an existing federation service of IaaS clouds introduces a new administrative interface, a HaaS service allows an IaaS provider to seamlessly manage resources over an inter-cloud environment with an IaaS administrative interface.

To confirm the feasibility, we have developed Iris, an inter-cloud resource integration system. Iris constructs a set of virtual resources over physical resources by using nested virtualization and OpenFlow technologies, and connections between the assembled

resources and the IaaS data center. Using Iris, we have demonstrated CloudStack can seamlessly manage resources over multiple data centers on an emulated inter-cloud environment. The experimental results show that the overheads of the HaaS layer are acceptable when the network latency is less than 10 ms. Nested virtualization remains a costly solution today; nevertheless it is fascinating because it allows flexible operation of inter-cloud resources. Both hardware and software improvements for nested virtualization will lead to resolve this problem in the near future as the overhad of virtualization has been dramatically reduced by full virualization.

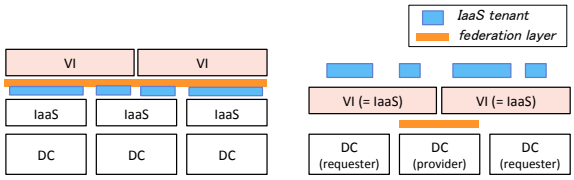
The rest of the paper is organized as follows. Section 2 describes the background of inter-cloud resource management and proposes our HaaS model. The design of a HaaS system is presented in Section 3, followed by the implementation in Section 4. Section 5 shows the experimental results, and we discuss the feasibility of our HaaS model in Section 6. Finally, Section 7 summarizes the paper.

2 INTER-CLOUD RESOURCE MANAGEMENT MODELS

Inter-cloud resource management is intended to construct *a Virtual Infrastructure (VI)* in such a way that providers can efficiently utilize resources over distributed data centers. A VI is an isolated set of resources, including computers, storage, and networks. Each data center is operated by utilizing a resource management middleware suite, i.e., *a Cloud OS*, such as CloudStack or OpenStack. Each cloud OS, however, provides users with services in a variety of different manners, and there is no commonly agreed standard interface.

From the viewpoint of constructing a VI over inter-cloud resources, we define two resource management models: an overlay model and an extension model, as shown in Figure 1. In other words, the overlay model is *a virtual Infrastructure as a Service (vIaaS)* model; and the extension model is *a Hardware as a Service (HaaS)* model. We categorize existing inter-cloud solutions as a vIaaS model, and we propose a HaaS model in this paper.

The key to both is the federation layer and the interface. By introducing a federation layer, abstraction of underlying resources is achieved. vIaaS bundles several resource sets provided by IaaS clouds into a single VI. For instance, RightScale[1] offers federation of IaaS clouds in such a way that it serves as a glue layer among independently developed cloud interfaces. Moreover, some cloud OSs provides a Virtual Private Network (VPN) feature to connect an on-premise data center with a public cloud. This is a popular solution to implement a hybrid cloud, and it is one form of vIaaS model. On the other hand, HaaS provides IaaS clouds with isolated resources, and an IaaS cloud can run on each VI distributed over data centers. Each VI can transparently scale in and out without concern for the boundary of data centers. HaaS does not depend on a specific IaaS, and HaaS can provide services with multiple and heterogeneous IaaS clouds. While a vIaaS model has to introduce a new administrative interface for the operation of a VI, a HaaS model allows an IaaS provider to seamlessly manage resources over an inter-cloud environment with a Cloud OS interface. A Cloud OS, which each requester uses in their own data center, can manage the provided remote resources in a provider data center as if they were located in the requester's data center. We focus on the HaaS model in the rest of the paper.



(a) vIaaS: overlay model (b) HaaS: extension model

Figure 1: Inter-cloud resource management models: the relationship between Virtual Infrastructures, IaaS systems, and data centers.

3 DESIGN

This section introduces Iris, a system for implementing our HaaS model.

3.1 Overview

Figure 2 shows the overview of a HaaS system, which enables IaaS clouds to extend over the HaaS layer in a HaaS data center. A Cloud OS runs on each VI over inter-cloud resources. This system works as follows: 1) an IaaS administrator detects an indication of service degradation due to excessive access from IaaS users; 2) the IaaS administrator requests additional resources from a HaaS system; 3) to construct a VI, the HaaS system allocates and isolates resources in the HaaS data center, and it extends the layer 2 network from the IaaS data center; 4) the size of the VI can be elastically changed as needed.

[1] http://www.rightscale.com/

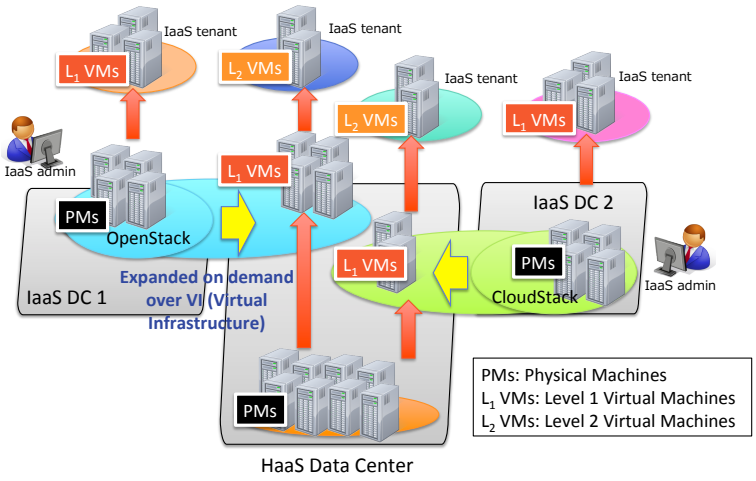


Figure 2: Elastic data center based on a HaaS system.

3.2 Requirements

The main requirements for implementing a HaaS system are described as follows:

Ease of Use. To make IaaS providers widely accept this service, the system must be easy to install on existing IaaS systems. Any Cloud OS must be able to handle HaaS resources as well as IaaS resources, and then the Cloud OS must be able to run on a VI without any modifications.

Secure Isolation. The system should establish secure isolation between HaaS and IaaS systems to avoid security incidents in a HaaS data center. For instance, a compute node has multiple network interfaces, one for the data plane; another for the control plane. The system should prohibit access to the control plane network, which is used for internal communications to control a HaaS data center, from the IaaS system.

Multi-tenancy. The system should provide multiple IaaS providers with the service. Each IaaS provider can use the overlapped range of IP addresses and VLAN IDs. To do that, virtualization of the data center network is required.

3.3 Resource Virtualization

To meet the above requirements, we have developed our HaaS system and called it Iris, which stands for an Inter-cloud Resource Integration System. Iris consists of compute and network resource management modules. The key mechanisms underlying Iris are multi-level virtualization of compute and network resources. Iris allows IaaS clouds to introduce a HaaS service with minimum effort. The following sub sections introduce two key technologies: nested virtualization and OpenFlow, and show the benefits for a HaaS system.

3.3.1 Nested Virtualization

The system provides IaaS providers with an abstract machine instead of a physical machine. Nested virtualization, which enables multiple guest hypervisors to run on a host hypervisor, is a quite useful technology that provides a way to achieve the requirement. Since Intel architectures only support single level virtualization, a hypervisor needs to trap and emulate VMX instructions, which is an extension instruction set for hardware-assisted virtualization. Popular open source hypervisors, KVM and Xen, already support nested virtualization (Ben-Yehuda et al., 2010; Williams et al., 2012). Virtage (Ueno et al., 2010) is yet another proprietary hypervisor on the Intel architecture. Virtage allows to provide a logical partition, commonly called an LPAR, which is a subset of hardware resources, on the Intel architecture. This LPAR technology is essentially the same as a virtual machine (VM) technology. Virtage also supports nested virtualization, and it is called "KVM on LPAR."

The physical machine, single-level VM, and second-level VM are shortened to PM (L_0 VM), L_1 VM, and L_2 VM, respectively. As shown in Figure 2, each data center is comprised of PMs. IaaS provides their users with a set of L_1 VMs. On the other hand, HaaS provides IaaS with L_1 VMs, which belong to an isolated layer 2 network, and unifies the networks for an IaaS data center. IaaS can handle these L_1 VMs in the same way as their own PMs. Therefore, a Cloud OS works on a VI without modification. Notice that,

in contrast to existing IaaS clouds, some IaaS users are offered L_2 VMs that physically run in a HaaS data center. Such a VI dynamically expands and shrinks as an IaaS administrator requires.

3.3.2 Network Virtualization

The aims of network virtualization are an extension of a layer 2 network between data centers and support of multi-tenancy. In other words, each VI belongs a isolated and flat layer 2 network. To achieve these goals, first, a HaaS system divides its network into several isolated sub-networks for each IaaS by network virtualization. Second a HaaS system connects the sub-network with an IaaS data center network by layer 2 network extension. In addition, a HaaS system is required to support VM migration inside a VI over data centers.

An IaaS data center requires only a gateway node to connect a HaaS data center. Iris builds a full-mesh overlay network for isolating each VI from the others, using OpenFlow (McKeown et al., 2008) and layer 2 tunneling. We can use several layer 2 tunneling protocols, such as Generic Routing Encapsulation (GRE) and VXLAN (Mahalingam et al., 2013). Moreover, to extend a layer 2 network between data centers, there are two options available, layer 2 tunneling and dynamic network circuit provisioning, including OSCARS (Guok et al., 2008) and OGF Network Service Interface (NSI) (Belter et al., 2013). Currently Iris only supports the GRE protocol.

4 Implementation

We have been developing Iris on top of GridARS (Takefusa et al., 2011), as shown in Figure 3. An IaaS administrator manages their data center using a Cloud OS like CloudStack. Iris dynamically extends and shrinks the scale of the data center as demand requires.

GridARS is an Inter-cloud resource management system, which provides performance-assured resources over inter-cloud environments; and appropriate monitoring information for each user via a Web services-based interface. GridARS consists of a Resource Management System (RMS) and a Distributed Monitoring System (DMS), and this hierarchical architecture allows worldwide scaling. RMS co-allocates various resources, such as computers and storage in clouds and network connection between the clouds by using common interfaces such as OGF NSI. DMS automatically gathers distributed monitoring information from the allocated resources for each user in cooperation with RMS.

Iris works as a data center RMS and interacts with a GridARS resource coordinator via a Web services-based interface. It manages the resources of a HaaS data center and asociated gateway nodes. Iris is comprised of three main components: a compute controller, an OpenFlow switch, and an OpenFlow controller. Open vSwitch (Pfaff et al., 2009) software switches run on all compute nodes and gateway nodes. An OpenFlow controller, *Iris controller*, is implemented based on the Floodlight controller[2]. Iris controller keeps track of the membership and the network topology of a VI. Such information is stored in a management registry. The management registry includes a mapping table of MAC addresses and IP addresses like an ARP table. It is initialized from a given configuration file when launching a VI.

The management registry should update appropriately when a Cloud OS executes VM migration. Iris controller detects VM migration and updates the management registry without explicit interaction with a Cloud OS. An ARP request is sent to Iris controller by a PacketIn mechanism. After that, it works as follows. If the source MAC address of the ARP request is inside a HaaS data center, the management registry is updated and then an ARP response packet is returned. Otherwise, the ARP request is forwarded to an IaaS data center via the gateway node. Moreover, a guest OS sends RARP packets after the user VM migrates. Iris controller receives an RARP packet, followed by updating flow tables of Open vSwitches.

5 EXPERIMENTS

To demonstrate the feasibility of the proposed HaaS system, we have conducted experiments using Iris on an emulated inter-cloud environment. We have compared the performance with a generally configured IaaS system from the following three points of view: the deployment time of a User VM (UVM); the migration time of a UVM; and the computing and I/O processing performance on a UVM.

5.1 Experimental Settings

We used two experimental settings: AGC and HCC. The former is an emulated inter-cloud environment, which consists of two data centers. The latter is an experimental testbed for evaluating nested virtualization technologies.

[2]http://www.projectfloodlight.org/floodlight/

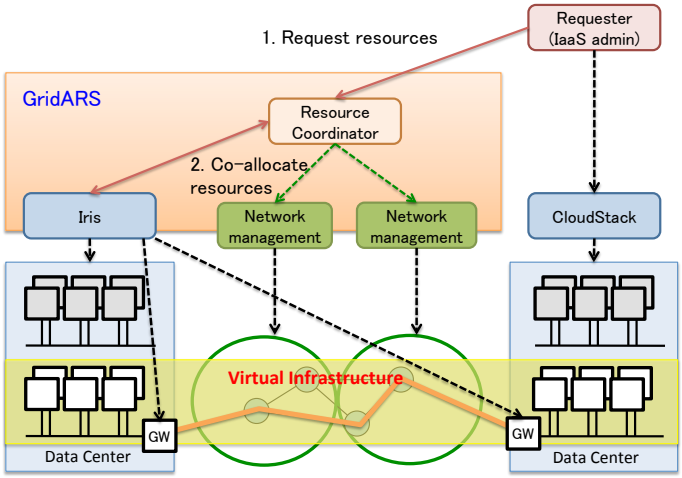


Figure 3: Integration of Iris into the GridARS framework.

5.1.1 Setting 1: AGC

We used a 13 node-cluster, which is a part of the AGC cluster. The cluster consists of Dell PowerEdge M610 blade servers, and is comprised of 2 quad-core Intel Xeon E5540/2.53GHz CPUs, 48 GB of memory, a 300 GB RAID1 SAS disk array, and a Broadcom NetXtreme II 10 Gigabit Ethernet card. The Dell M1000e blade enclosure holds 16 blade servers and a 24 port 10 Gigabit Ethernet switch. Hyper Threading was disabled.

Figure 4 shows our emulated inter-cloud environment, where the above cluster is divided into two sub-clusters, and both are connected through a network emulator. The left hand side shows an IaaS data center, which consists of seven compute nodes and one gateway node. The Apache CloudStack version 4.0.2 was installed on these compute nodes. To isolate the private network, a VLAN ID was assigned to each tenant, namely an IaaS user, from the range between 100 and 200. The right hand side shows a HaaS data center, which consists of four compute nodes and one gateway node. The two data centers communicate with each other only via gateway nodes. Since all nodes had a single network interface card (NIC), the network configuration was set in such a way that only the inter-data center traffic was forwarded to the network emulator by using VLAN routing. To emulate a wide area network environment, we employed GtrcNET-1 (Kodama et al., 2004), a hardware network testbed that can control network latency up to 100 ms at wire rate. The MTU sizes of compute nodes are set to 1500 bytes; the MTU sizes of the two switches are set to 9000 bytes.

On the IaaS data center, the host OS was Ubuntu 12.04. On the HaaS data center, the host OS and L_1 guest OS were Ubuntu 12.10 and Ubuntu 12.04, respectively. On both data centers, CentOS 5.5 was running on both IaaS and HaaS UVMs. The VM images were created using the qcow2 format. Live migration is required for the shared storage among the source and destination nodes. In this experiment, an NFS server ran on the IaaS data center. We used Open vSwitch version 1.4.3 and Floodlight version 0.90.

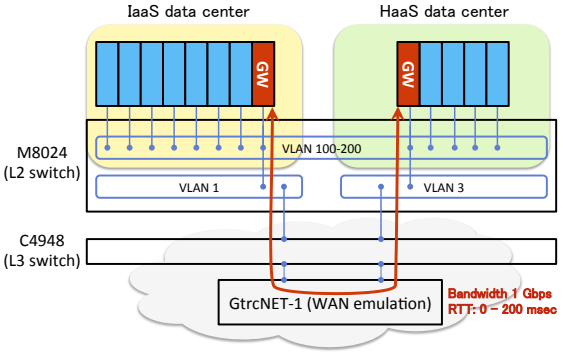


Figure 4: An emulated inter-cloud environment on AGC.

Table 1 summarizes the specification of a UVM environment. L_1 VM corresponds to an IaaS user VM and a HaaS host VM; L_2 VM corresponds to a HaaS user VM. In this experiment, a single L_1 VM ran on a single L_0 physical machine, and a single L_2 VM also ran on a single L_1 VM. Note that multiple VMs may run on a single machine because CloudStack implicitly launches System VMs (SVMs), including a virtual router and a console proxy.

5.1.2 Setting 2: HCC

We used two Hitachi BladeSymphony 2000 blade servers, which consist of 2 octal-core Intel Xeon E5-

Table 1: Specification of a user VM on AGC.

	CPU	mem	disk	network
L_2	1	512 MB	5 GB	virtio_net
L_1	4	16 GB	32 GB	virtio_net
L_0	8	48 GB	300 GB	bnx2x

2690/2.9 GHz CPUs, 64 GB of memory, Gigabit Ethernet NICs, and 8 Gbps Fibre Channel HCAs. Hyper Threading was enabled.

On this environment, we compared the performance of two nested virtualization technologies: KVM on Virtage and nested KVM. Virtage was installed to one server; Fedora 18 was installed on another server. Virtage produces two LPARs, which consist of 8 CPUs and 16 GB of memory, and directly assigns a Gigabit Ethernet NIC and FC storage. Red Hat Enterprise Linux 6.2 was running on the LPARs or L_1 VM, and L_2 VM. Note that this environment is only used in Section 5.4.1.

5.2 User VM Deployment

We have measured the elapsed time that it takes to deploy a VM to IaaS or HaaS data centers. In this experiment, a CloudStack user created a single availability zone over two data centers. All VM images are located on the primary storage, that is the NFS server in an IaaS data center. Table 2a shows the deployment times of a UVM, where the network latency varies from 0 ms to 100 ms. The round trip time is double the latency. Note that the inter-data center network is not used for deployment inside an IaaS data center. These numbers are obtained from CloudStack management server logs. They do not include the deployment time of CloudStack SVMs and the OS startup time. The deployment time grows as the network latency increases. This can be explained as follows. A cloud agent on a compute node communicates with the CloudStack managemnt server to generate a VM configuration file. In the meantime multiple messages are exchanged through an inter-data center network. This makes a negative impact on the deployment time.

Table 2: Elapsed time of user VM deployment [seconds].

(a) one zone

	IaaS	HaaS
0 ms	11.88	11.89
5 ms	-	15.19
10 ms	-	18.84
100 ms	-	86.50

(b) two zones

	HaaS
0 ms	31.02
5 ms	32.16
10 ms	33.45
100 ms	36.56

In the next experiment, we created two availability zones. One zone is located in an IaaS data center; the other zone is located in an HaaS data center. Each zone has its own primary storage. Here we compare two configurations: SVMs on HaaS and SVMs on IaaS. The former is a configuration where SVMs are running in a HaaS data center, and this is the default behavior of CloudStack. The latter is a configuration where SVMs are running in an IaaS data center. Table 2b compares the UVM deployment time with two availability zones. The result is larger and more uniform than that of Table 2a. This is mainly because of the overhead of nested virtualization. While a UVM is deploying, a virtual router works and communicates with the management server to generate the network configuration. In this case, a virtual router runs not on an L_1 VM but on an L_2 VM. This problem can be avoided by migrating a virtual router from a HaaS data center to an IaaS data center. However, the outgoing traffic from a HaaS data center is routed via an IaaS data center. This causes a decrease in the network performance.

In addition, we have confirmed multi-tenancy, that is, Iris enables us to provide resources even with multiple IaaS data centers. To do that, we divided an IaaS data center into two 4-node data centers and set up CloudStack in each data center. CloudStack assigns VLAN IDs to each IaaS user. Even if the same VLAN ID is assigned among different IaaS users, there is no problem because Iris can completely isolate their networks by using OpenFlow.

5.3 User VM Migration

An IaaS administrator can migrate UVMs from a compute node to another node. A HaaS system enables us to migrate VMs between data centers using the same IaaS API, i.e., a CloudStack API called `migrateVirtualMachine`. Shown in Figure 5 are the data from the migratation time between an IaaS data center and a HaaS data center. These numbers are obtained from CloudStack management server logs.

The baseline is VM migration within IaaS data center. In this case, the VM migration time is 2.62 ms. The migration times increases approximately linearly as the network latency increases. During VM migration, the memory pages are copied over data centers. Therefore, the migration time grows as the network latency increases. Notice that VM migration inside a HaaS data center is also affected by the network latency. This is because compute nodes, i.e., L_1 VM, communicate with the management server via the inter-data center network.

Table 3: Relative performance of the BYTE UNIX benchmark normalized to the physical machine on two settings [%]. The performance of HaaS UVM (L_2 VM) on Virtage is mostly comparable to IaaS UVM (L_1 VM) on KVM.

(a) AGC

	IaaS UVM	HaaS UVM
dhrystone	77.16	57.07
whetstone	86.29	70.08
execl	157.00	4.71
file copy 256	48.93	37.75
file copy 1024	45.96	35.51
file copy 4096	56.87	43.01
pipe	49.02	38.49
context switch	205.67	9.43
spawn	256.80	4.82
shell	95.96	4.18
syscall	29.57	22.73

(b) AIST + HCC

	HaaS UVM (Virtage)	HaaS UVM (KVM)
dhrystone	47.27	48.82
whetstone	77.24	74.86
execl	62.44	4.31
file copy 256	125.71	125.00
file copy 1024	119.84	119.10
file copy 4096	113.65	98.05
pipe	128.23	119.91
context switch	1146.68	65.21
spawn	177.39	3.19
shell	71.99	4.71
syscall	165.04	159.55

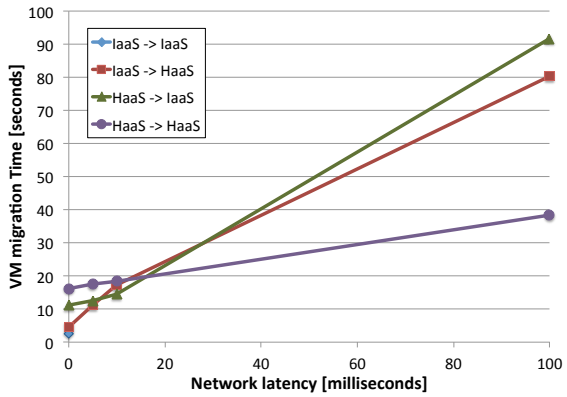


Figure 5: Elapsed time of user VM migration between two data centers.

5.4 User VM Performance

To evaluate the impact of nested virtualization, we have measured the performance of UVMs on both an IaaS cluster and a HaaS cluster.

5.4.1 BYTE UNIX Benchmark

The BYTE UNIX benchark[3] is a micro benchmark suite for evaluating the performance of a UNIX-like system. It includes the following benchmark programs: Dhrystone 2 using register variables (dhrystone), Double-Precision Whetstone (whetstone), Execl Throughput (execl), File Copy 256 bufsize 500 maxblocks (file copy 256), File Copy 1024 bufsize 2000 maxblocks (file copy 1024), File Copy 4096 bufsize 8000 maxblocks (file copy 4096), Pipe Throughput (pipe), Pipe-based Context Switching (context switch), Process Creation (spawn), and Shell Scripts (shell). Shown in Table 3 are the data from the relative performance normalized to the physical machine.

With HaaS UVM in AGC, some tests, including execl, spawn, context switch, and shell, significantly degrade the performance, i.e, less than 10 %, compared with the other tests, including compute and I/O intensive programs. This is caused by a multiplication of VM exits. KVM on an L_1 VM (L_1 KVM) traps and emulates VMX instructions executing on an L_2 VM. To handle a single L_2 VM exit, L_1 KVM performs many operations such as read and write of the VMCS, disable interrupts, page table operations, and so on. Eventually, a single L_2 VM exit causes many L_1 VM exits. This is known as VM exit multiplication. We have confirmed the number of VM exits increases from 64 to 70 times larger on a nested VM environment. The analysis reason for VM exits shows that the overhead of address space operations grows significantly in a nested virtualization environment (Amontamavut et al., 2013).

The performance of compute intensive tests, such as dhrystone and whetstone, is relatively low. In this experiment, we could not use the optimal CPU configuration because CloudStack deploys a VM with the default QEMU64 CPU model. For instance, some special instruction sets could not be used as they are in the physical machine. Moreover, with IaaS UVM and HaaS UVM (Virtage), the results of spawn and context switch tests are not intuitive. The performance of UVM is obviously better than that of the physical node. This can be considered because the overhead of synchronization is reduced due to the smaller number

[3] https://code.google.com/p/byte-unixbench/

of CPU cores.

The performance of L_2 VM on Virtage is mostly comparable to L_1 VM on KVM. This means that a HaaS UVM on Virtage can obtain the equivalent performance of an IaaS UVM. KVM suffers from VM exit multiplication. In contrast, Virtage reduces the frequency of L_1 VM exits by virtualization of the Extended Page Table (EPT) using EPT shadowing.

5.4.2 Network Performance

We measured the effect of nested virtualization on the network performance. In this experiment, we disabled network offloading features, including generic segmentation offload and generic receiver offload, as this can improve network performance and reduce CPU load, and because kernel panics occurred when these features were enabled. As a result, the network goodput between two physical compute nodes was limited to under 4 Gbps.

Table 4 shows the round trip latency between two UVMs by using the ping command. The network latency increases three times due to the nested virtualization. Comparing migration inside a data center with migration between data centers, the latter increases by two routing hops. This incurs extra latency. Table 5 shows the network goodput between two UVMs by using the Iperf command. The message length is set to 32 KB. The goodput drops down to 40 percent due to the nested virtualization. We will discuss this issue in Section 6.

Table 4: Roundtrip latency between user VMs [ms].

src \dest	IaaS UVM	HaaS UVM
IaaS UVM	0.61	1.62
HaaS UVM	1.62	1.85

Table 5: Goodput between user VMs [Mbps].

src \dest	IaaS UVM	HaaS UVM
IaaS UVM	789	405
HaaS UVM	700	310

6 DISCUSSION

In this paper, we have employed a VM technology to provide an L_1 VM. Another approach is OS-level virtualization, such as Linux containers (LXC), Docker, OpenVZ, and Linux VServer. For instance, PlanetLab (Bavier et al., 2004), which is a platform for planetary-scale services, utilizes VServer to provide an isolated resource container. OS-level virtualization is a light-weight method compared to a VM, and the overhead is negligible. In spite of the benefit, we finally decided to employ nested virtualization in terms of isolation. We have met with some serious problems caused by LXC because containers are not perfectly isolated at the kernel level. For instance, an NFS client and Open vSwitch could not run normally when we used an LXC container as an L_1 VM. Although an ad hoc workaround for each problem exists, it is not sufficient.

The experimental results show that current nested virtualization technology suffers from heavy VM exit multiplication, resulting in large performance degradation on an L_2 VM. This is because both the implementation of KVM and the hardware support of the CPU are still immature. Such barriers will be resolved in the near future. The latest Intel architecture supports hardware-based VMCS shadowing; Virtage supports software-based EPT shadowing. These features allow us to significantly reduce the overhead of nested virtualization, and it is certain that KVM will support them. Therefore, we believe nested virtualization is a promising solution to achieve flexible resource management.

Section 5.4.2 shows that nested virtualization introduces large overhead on network performance. To tackle this issue, VMM-bypass I/O technologies, including PCI passthrough and SR-IOV, have been introduced (Dong et al., 2012). We plan to use PCI passthrough with a single-level as well as nested virtualization to improve I/O performance on an L_2 VM.

How to handle the heterogeneous performance between L_1 VM and L_2 VM in context of QoS and SLA is an open issue. An IaaS provider is able to choose wheather he/she discloses the heterogeneity as an operational policy. For instance, L_1 VM and L_2 VM can be categorized into the different VM instance types.

7 CONCLUSION

Inter-cloud federation among heterogeneous systems and/or organizations is a significant challenge. This paper proposes a new cloud computing service model, Hardware as a Service (HaaS), a service that dynamically configures and provides virtual resources, on which IaaS can run, by using nested virtualization and OpenFlow. To demonstrate the feasibility of the proposed model, we have developed Iris on top of the GridARS inter-cloud resource management system. Using Iris, we have confirmed that CloudStack can seamlessly manage resources over multiple data cen-

ters on an emulated inter-cloud network. Iris is independent of the Cloud OS, and it is not limited to working with any Cloud OSs, including OpenStack and even proprietary software. The impact on the usability is negligible when the network latency is less than 10 ms. A 10 ms latency roughly corresponds to 1000 kilometers in actual networks, which covers the major cities in Japan. However, nested virtualization introduces a non-negligible overhead. But, since both software and hardware improvements will lead to getting rid of this issue, we believe nested virtualization is a promising solution to achieving flexible resource management like that provided by our HaaS model.

The evaluation is still limited and we need to consider more realistic scenarios in terms of scalability and workload. We also plan to explore use cases such as IaaS migration for disaster recovery. Using Iris, an entire IaaS can be migrated to a differently configured data center.

ACKNOWLEDGEMENTS

This work was partly funded by the FEderated Testbeds for Large-scale Infrastructure eXperiments (FELIX) project of the National Institute of Information and Communications Technology (NICT), Japan. We would like to thank the Hitachi Harmonious Computing Center for conducting a performance evaluation of nested virtualization technologies on their equipment.

REFERENCES

Amontamavut, P., Takano, R., and Hayakawa, E. (2013). Performance evaluation of nested virtualization based on KVM. In *PRAGMA 24 students workshop*.

Bavier, A., Bowman, M., Chun, B., Culler, D., Karlin, S., Muir, S., Peterson, L., Roscoe, T., Spalink, T., and Wawrzoniak, M. (2004). Operating system support for planetary-scale services. In *Proc. of the First Symposium on Network Systems Design and Implementation (NSDI)*.

Belter, B., Krzywania, R., Kudoh, T., and van Malenstein, G. (2013). Federating SDN-enabled islands with an extended NSI Framework. In *Proc. of the TERENA Networking Conference (TNC2013)*.

Ben-Yehuda, M., Day, M. D., Dubitzky, Z., Factor, M., Har'El, N., Gordon, A., Liguori, A., Wasserman, O., and Yassour, B.-A. (2010). The turtles project: design and implementation of nested virtualization. In *Proc. of the 9th USENIX conference on Operating systems design and implementation (OSDI)*, pages 1–6.

Buyya, R., Ranjan, R., and Calheiros, R. (2010). Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *Proc. of the 10th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP)*, pages 13–31.

Carr, N. (2006). Here comes HaaS. [online] http://www.roughtype.com/?p=279.

Demchenko, Y., Ngo, C., Makkes, M., Stgrijkers, R., and de Laat, C. (2012). Defining inter-cloud architecture for interoperability and integration. In *Proc. of the IARIA Third International Conference on Cloud Computing, GRIDs, and Virtualization*, pages 174–180.

DMTF (2013). Open Virtualization Format Specification. [online] http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_2.0.1.pdf.

Dong, Y., Yang, X., Li, J., Liao, G., Tian, K., and Guan, H. (2012). High performance network virtualization with sr-iov. *Journal of Parallel Distributed Computing*, 72(11):1471–1480.

Guok, C., Robertson, D., Thompson, E. C. M., Johnston, W., and Tierney, B. (2008). A User Driven Dynamic Circuit Network Implementation. In *Proc. of the 3rd IFIP/IEEE Distributed Autonomous Network Management Systems*.

Kodama, Y., Kudoh, T., Takano, R., Sato, H., Tatebe, O., and Sekiguchi, S. (2004). GNET-1: Gigabit Ethernet Network Testbed. In *Proc. of the International Conference on Cluster Computing*, pages 185–192.

Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., Bursell, M., and Wright, C. (2013). Vxlan: A framework for overlaying virtualized layer 2 networks over layer 3 networks. *IETF Internet Draft draftmahalingam-dutt-dcops-vxlan-01.txt*.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *SIGCOMM Computer Communication Review*, 38(2):69–74.

Nyren, R., Edmonds, A., Papaspyrou, A., and Metsch, T. (2011). Open Cloud Computing Interface – Core. [online] http://ogf.org/documents/GFD.183.pdf.

Pfaff, B., Pettit, J., Koponen, T., Amidon, K., Casado, M., and Shenker, S. (2009). Extending networking into the virtualization layer. In *8th ACM Workshop on Hot Topics in Networks (HotNets-VIII)*.

SNIA (2012). Cloud Data Management Interface (CDMI) Version 1.0.2. [online] http://snia.org/sites/default/files/CDMI v1.0.2.pdf.

Takefusa, A., Nakada, H., Takano, R., Kudoh, T., and Tanaka, Y. (2011). GridARS: A Grid Advanced Resource Management System Framework for Intercloud. In *Proc. of the 1st International Workshop on Network Infrastructure Services as part of Cloud Computing (NetCloud)*.

Ueno, H., Hasegawa, S., and Hasegawa, T. (2010). Virtage: Server Virtualization with Hardware Transparency. In *Proc. of Euro-Par 2009*, pages 404–413.

Williams, D., Jamjoom, H., and Weatherspoon, H. (2012). The Xen-Blanket: Virtualize Once, Run Everywhere. In *Proc. of the ACM European Conference on Computer Systems (EuroSys)*.

A Survey on Approaches for Interoperability and Portability of Cloud Computing Services

Kostas Stravoskoufos, Alexandros Preventis, Stelios Sotiriadis and Euripides G. M. Petrakis
Department of Electronic and Computer Engineering, Technical University of Crete, Chania, Greece
{*kgstravo, apreventis, s.sotiriadis, petrakis*}*@intelligence.tuc.gr*

Keywords: Cloud Computing, Cloud Interoperability, Cloud Portability.

Abstract: Over the recent years, the rapid development of Cloud Computing has driven to a large market of cloud services that offer infrastructure, platforms and software to everyday users. Yet, due to the lack of common accepted standards, cloud service providers use different technologies and offer their clients services that are operated by a variety of proprietary APIs. The lack of standardization results in numerous heterogeneities (e.g., heterogeneous service descriptions, message level naming conflicts, data representation conflicts etc.) making the interoperation, collaboration and portability of services a very complex task. In this work we focus on the problems of interoperability and portability in Cloud Computing, we address their differences and we discuss some of the latest research work in this area. Finally we evaluate and point out relationships between the identified solutions.

1 INTRODUCTION

According to the NIST (U.S. Government's National Institute of Standards and Technology) Cloud Computing definition (Mell and Grance, 2009), the Cloud is composed by three Service models: The Infrastructure as a Service (IaaS) model, which is used to manage the underlying hardware in order to provide the consumers with computer resources such as storage, processing power, or network. The Platform as a Service (PaaS) model which virtualizes a computer environment supporting the development and deployment of consumer applications and, finaly, the Software as a Service(SaaS) model which gives consumers the capability to use applications running on the cloud.

The term of interoperability has many definitions in literature (Petcu, 2011) and is often misused to include the term of portability. In order to get a more clear view we focus on the three cloud service models and the different conceptions interoperability refers to in each model. This also applies for applications that are not already ported to the cloud: developers who want to port an application to the cloud must consider all the different options offered by cloud service providers and find the most suitable for their application. On the other hand, in case where a client combines two SaaS (this leads to more complex service composition), interoperability refers to the ability of the services to communicate and be compatible by overcoming message level heterogeneities like naming conflicts or data representation conflicts. the services to communicate and be compatible by overcoming message level heterogeneities like naming conflicts or data representation conflicts.

The NIST makes a clear distinction between interoperability and portability by defining service interoperability, system portability and data portability (Bohn et al., 2011). *Service interoperability* is defined as "the ability of cloud consumers to use their data and services across multiple cloud providers with a unified management interface" while *portability* is further distinguished into *Data Portability* and *System Portability*. Data Portability is defined as "the ability of cloud consumers to copy data objects into or out of a cloud or to use a disk for bulk data transfer" while, System Portability is defined as "the ability to migrate one of a fully stopped virtual machine(VM) instance (that represents a cloud service or an application) or a machine image from one cloud infrastructure to another cloud, or migrate applications and services and their contents from one service provider to another".

The NIST also defines the concept of the "hybrid cloud" which is a composition of two or more clouds that remain as distinct entities but are bound together by standardized or proprietary technology that enables data and application portability. In general, portability and interoperability between the various cloud service models resort to standards.

In the following sections we present the various approaches for solving the problem. In Section 2 we discuss interoperability and portability and in Section 3 approaches on this. In Section 4 we present a cross comparison study of the discussed approaches to characterize the most crucial requirements for future developments. At last, in Section 5 we highlight the future research steps of this work.

2 CLOUD SERVICE INTEROPERABILITY AND PORTABILITY

Service Interoperability allows customers to use services across multiple clouds using common management API. The general requirement to achieve service interoperability, since world-wide standards haven't been defined yet (Petcu et al., 2013), is the semantic description of services which is vital to create the middleware towards a common management API to override the proprietary ones.

Service Interoperability applies to all three Cloud Computing service models but the meaning of it (and also the requirements) varies in each model. Achieving interoperability in the IaaS model means that clients have the ability to use the infrastructure of different clouds and control them as if they were one. A simple example is when the client has control of several virtual machines that encapsulate the computational resources from different clouds. In PaaS, service interoperability is about enabling the clients to use different APIs, tools, libraries etc. from different platforms ported in different clouds in order to create applications. Finally, in the SaaS model, interoperability refers to enabling cloud applications to exchange messages or data. Below we present some of the latest research work on this subject.

System portability refers to the ability of cloud services to be deployed on other cloud services of a lower service model (e.g. the ability of a SaaS service to be deployed on different PaaS services). In the IaaS model, portability refers to the ability to migrate and run Virtual Machines along with data and configurations, across different infrastructure providers. In the PaaS model, portability refers to the ability to deploy applications across different platforms (across different virtual machines). Finally portability in the SaaS model refers to the ability of clients to move their data across equivalent SaaS applications. This is also known as data portability and is further analyzed below.

Another case of portability is when an external application needs to be ported in the cloud. In such case, an application description is required along with the description of the platform services. According to (Nagarajan et al., 2006) there are 4 types of semantics describing applications:

1. System semantics are semantics pertaining system characteristics (deployment and load balancing).
2. Data semantics are semantics pertaining to data (typing, storage and manipulation restrictions).
3. Non functional semantics which are semantics pertaining to QoS characteristics (performance, security).
4. Logic and process semantics which semantics pertaining to the core functions of the application (programming language, runtime, exception handling).

The basic requirement to achieve system portability, as in service interoperability, is the description of services. The concept of data portability applies only on the SaaS model and refers to the ability of customers to move their data along equivalent cloud applications hosted on different providers. Since the "vendor lock-in" also affects data portability, the main problem in this task is the variety of data representations used among the different cloud applications. This requires transforming the representation of data which is the output of one cloud application to the representation of data required as input by the target application. Another important problem to deal with is the different import and export functionalities of cloud applications (the mechanisms used to export or import data from and into cloud applications). The description of those functionalities is required to enable migration of data from one application to another (standardization plays an important role in this part).

3 SERVICE DISCOVERY IN CLOUDS

This section describes the analysis of the related approaches in the area of service discovery in cloud systems. The works deal a) with the service description problem (a vital requirement to achieve service Interoperability) and b) with message level heterogeneities. Although messages may be described in the same way (i.e., using SOAP) they may differ in the domain level (Nagarajan et al., 2006). This kind of differences is described as heterogeneity at the message level. Dealing with message heterogeneities is very important if we take into consideration all the

Table 1: A summary of the cloud interoperability and portability approaches for service discovery.

Approach	Focus	Key operations
WSDL-S, OWL-S	Approaches that provide the vocabulary for describing services.	Introduce semantics for the description of services (including cloud).
Unified Cloud Interface (UCI)	Standardization for interacting between clouds using an interface.	Common management API for cloud services.
Open Cloud Computing Interface (OCCI)	Standardization approach that proposes a common management API.	API for common management tasks e.g. cloud resource deployment.
OWL-S based Broker	Cloud service description and discovery approach using a broker.	OWL ontology for service discovery, providers need to be compliant.
Cloud Computing Ontology (CoCoOn)	Cloud service description and discovery using a broker.	OWL ontology for service description (IaaS model) and discovery.
Cloud Service Generic Search Engine	Description and discovery of services using a broker.	OWL ontology that uses a service discovery mechanism.
mOSAIC	Service description approach and service discovery and composition using a broker.	OWL-S ontology for service description. Interoperability through a common management API (OCCI).
PSIF	Framework for semantic interoperability conflicts on the PaaS layer.	Description of services for capturing and representing the conflicts.

different proprietary APIs that are used in cloud computing. In Table 1 we present a summary of the cloud interoperability and portability approaches for service discovery.

The **Unified Cloud Interface (UCI)**[1] Project's goal is to create an interface to interact with various cloud API's. The main idea is to create an API to handle all other API's. The UCI abstracts the usage of any cloud API and unifies them in one layer that is agnostic to cloud providers. It uses ontologies and OWL (van Harmelen and McGuinness, 2004) in order resolve any heterogeneities between the cloud based APIs and the existing protocols and standards. OWL is also used for describing the cloud data models and the UCI uses this description to make resources from multiple cloud providers available.

The **Open Cloud Computing Interface (OCCI)**[2] is an API and Protocol that supports different kinds of management tasks (e.g. deployment, monitoring etc.). The API acts as a service front-end to a provider's internal management framework, for controlling the cloud hardware resources and it can also be used as a management API for all kinds of resources (virtualization, networking and storage). It is highly extensible and focuses both on interoperability and portability. OCCI is one of the first approaches towards standardization and was first introduced as a remote management API for IaaS model based services. Since then, with the support of the Open Grid Forum[3] community it evolved to support all three cloud service models. Many academic and industry members currently support OCCI (e.g. OpenStack, GoGrid etc.).

OWL-S (Martin et al., 2004) is an OWL ontology that provides the vocabulary for describing services. It enables automatic service discovery, composition and invocation. The main OWL-S ontology consists of three sub-ontologies, the "ServiceProfile", the "ServiceModel" and the "ServiceGrounding". The service profile is an upper level description of the service. It is mostly used for describing what the service does, in order to be discovered by agents and determine if it fits their needs.

The service model gives a detailed description of how the service works. It describes a) the semantic content of the service's inputs and outputs b) the conditions that have to be satisfied under which the service can be performed c) step by step processes that lead to the outcome of the service and d) the result of the service, meaning the effects of the service in the world (i.e., change in the balance of an account). Service grounding describes how a service can be accessed. It specifies the communication protocols, the message formats and other specific details such as the port that is used for accessing the service. OWL-S has such a structure that can be used with WSDL (Christensen et al., 2001) by extending the existing bindings of the later.

The **Cloud Computing Ontology (CoCoOn)** (Zhang et al., 2012) is an OWL ontology for describing cloud infrastructure services. Although they currently focus on the IaaS model, CoCoOn creators claim that the ontology will be extended in the future

[1] http://code.google.com/p/unifiedcloud/

[2] http://occi-wg.org/

[3] http://www.gridforum.org/

to discribe the PaaS and SaaS models. The ontology proposed in this work consists of: a) functional Cloud service configurations information parameters and b) non-functional service configuration parameters.

In the functional cloud service configuration information part, the concepts defining the IaaS model are defined as a taxonomy. For example, the IaaS concept is divided into three subclasses: Compute, Network and Storage, which in turn are further analyzed and categorized. In the non-functional cloud service configuration information part, properties of the ontology concepts are described. A distinction is made here between properties of Cloud resources that are known at design time (non-functional properties) like the provider or the deployment model and attributes that can only be recorded after at least one execution cycle of a Cloud service (QoS attributes) like durability or performance. Based on the ontology describing the services, a service discovery system has been implemented that uses the CoCoOn Ontology to search for available infrastructure services. The ontology has been populated with well-known cloud providers (Amazon, Microsoft Azure, GoGrid, etc).

OWL-S Cloud Broker (Ngan and Kanagasabai, 2012) is an OWL-S based semantic cloud service brokering system that enables dynamic service discovery. The broker uses the OWL-S ontology to describe the services and SWRL [4] to apply complex constraints on them. The user describes the desired service with a set of preferences and constraints that the service should satisfy. The preferences are represented by OWL entities that are inserted into the system's ontology. The broker uses reasoning techniques to match the inputs and outputs of the requested service to the inputs and outputs of services available. The services that do match the user's preferences are checked for satisfying also the constraints of the preference. In the final stage, the services that have passed the previous steps are ranked using a scoring function.

The **Cloud Service Generic Search Engine** (henceforth referred to as CSGSE) (Nagireddi and Mishra, 2013) is an approach towards cloud service discovery. The work is based on an OWL ontology which provides the description of services and their attributes. Also it includes a search engine for services discovery in the ontology directory. The search engine uses the SPARQL query language [5] to retrieve information from the ontology and to match services based on user requests. Cloud services must be registered in an ontology registry and are available to the search mechanism.

mOSAIC Project's (Moscato et al., 2011) vision is to provide a platform that will enable interoperability between different cloud services, portability of cloud services in different platforms, automatic service discovery and composition and management of Service Level Agreement. It provides a set of APIs that enable developers to build vendor independent applications that consist of multiple cloud components. Each of these components perform a simple operation. On run-time, the applications are decomposed into these components and the mOSAIC platform automatically will decide which cloud implementation is better for each component to be deployed on. The mOSAIC API serves as an intermediate layer between the developers and the cloud platforms. It lets the developers deploy their applications without getting involved with APIs.

The **PaaS Semantic Interoperability Framework (PSIF)** (Loutas et al., 2011) is a framework used to capture and represent semantic interoperability conflicts on the PaaS layer. Semantic interoperability conflicts may arise a) during the migration of an application from one PaaS to another, b) during the deployment of an application to a PaaS or, c) on the message exchange level between two PaaS systems. PSIF is structured according to three dimensions. Firstly, the fundamental PaaS entities (i.e., PaaS system, PaaS offering, management interface, software component, IaaS system and application) it is used to determine which entities are involved in the conflict and secondly, the types of semantics on the PaaS layer (functional, non-functional and execution) it is used to identify the semantic conflict. Thirdly, the levels where semantic conflicts occur (either on the information model or the data).

4 SERVICE DISCOVERY FOR PORTING APPLICATIONS IN CLOUDS

This section focus on the critical analysis of the related approaches for cloud service discovery. Table 2 demonstrates a cross comparison of the cloud service interoperability and portability approaches in order to define the most prominent. Based on the comparison we conclude that mOSAIC uses OWL-S and OCCI standards as well as provides the most complete ontology for service description. In addition, it is compliant with most of the cloud providers. The CoCoOn and OWL-S also present ontologies, yet there is still place for improvement. Their difference is that CoCoOn is compliant with most common vendors (and we consider it as an advantage), while OWL-S re-

[4] http://www.w3.org/Submission/SWRL/

[5] http://www.w3.org/TR/rdf-sparql-query/

quires vendors to be compliant with it. Finally, CGSE ontology is non detailed and requires from vendors to register their services to the broker (thus no support for most common cloud services).

In Figure 1, we demonstrate the portability and interoperability in the case of porting a service from a legacy system to a cloud. Here portability refers to the porting action while interoperability refers to the translation mechanism for matching purposes. The assumption is that the service to be ported requires to give an input to a broker that interprets and translates the service description. This has been characterized as a vital requirement by (OWL-S, CSGSE, OWL-S Broker and mOSAIC) for service acknowledgment. The broker includes the interoperability functionality in order to interpret and translate the request and match it with available ontological descriptions.

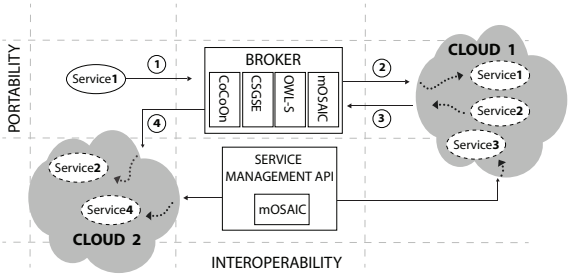


Figure 1: The interoperability and portability in cloud service discovery.

We highlight portability for the SaaS, PaaS and IaaS cases:

- The SaaS user/developer a) decides the service(s) to be ported, b) the broker interprets and translates the service(s) description based on ontologies, c) the broker selects a cloud instrance that matches the service description, and d) the developer uses the service(s).
- The PaaS user/developer a) decides the service(s) to be ported, b) the broker interprets and translates the service(s) description based on ontologies, c) the broker selects a cloud node for instantiation, d) the broker selects a cloud instance that matches the service(s) description and build the platform, and e) the developer accesses the service(s).
- The IaaS user/developer a) decides the service(s) to be ported, b) the broker interprets and translates the service(s) description based on ontologies, c) the broker selects the cloud node that matches the required computational resources, d) the developer accesses and utilizes the node(s).

Figure 1 shows that a service (Service1) could be ported to a cloud (CLOUD1), as shown in steps 1 and 2, through a broker (as implemented from the literature review approaches) based on semantically translation of the service requirements, described as Service Level Agreement (SLA). The broker operation includes the matching process (match SLA description with available ones based on ontologies). The interoperability is demonstrated in steps 3 and 4, where a service (Service2 from CLOUD1) is moved to another cloud (CLOUD2). The service management API is the common control point for different clouds (CLOUD1 and CLOUD2) integrated based on OCCI standards.

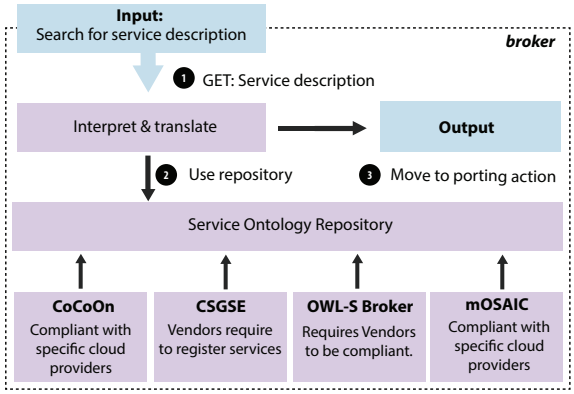


Figure 2: The interoperability and portability cloud service discovery broker.

Figure 2 demonstrates the functionality of the service broker. The first step is to get the service description and interpret and translate the descriptions. The second step is to use the repository where all four approaches add as input their ontologies. For example, CoCoOn is compliant with service descriptions of specific cloud providers (e.g. Microsoft Azure). After the matching, the process moves to step 3 that is the porting action.

5 CONCLUSIONS

This work presented an analysis of the related works that aim to service interoperability and portability on cloud systems. By summarizing the different approaches we concluded to key features such as a) the use of semantics in the description of services and ontologies as the most suitable tool for this task (e.g. CoCoOn, mOSAIC etc.), b) the definition of standards in the description of services (e.g. OCCI) and c) the use of broker to bridge gap of translating services between systems.

Table 2: A cross correlation study of the cloud interoperability and portability approaches.

	CSGSE	**OWL-S Broker**	**mOSAIC**
CoCoOn	CSGSE ontology in not as detailed compared to CoCoOn. Also, CoCoOn is compliant with specific providers, yet CGSE requires vendors to register in a broker.	OWL-S Broker is considered as a more detailed ontology. It uses OWL-S standard. Yet, OWL-S requires compliant providers with its ontology.	Compared with CoCoOn, mOSAIC is more detailed ontology in the level of descitpion of services. It uses the OWL-S standard and OCCI as a service front-end to a providers internal management framework.
CSGSE		OWL-S Broker is a more specialized ontology as it uses standards as OWL-S and SWRL. Also, CSGSE uses databases for storage thus it looses the benefits of semantics.	mOSAIC has a more specialized ontology on the level of service description for IaaS. Also it uses OWL-S, SPARQL unambiguous queries, as an advantage compared with CGSE that uses database for storage. It uses also OCCI.
OWL-S Broker			mOSAIC uses OWL-S, SPARQL and OCCI. In the level of ontology repository, OWL-S Broker requires vendors to be compliant with ontologies, yet mOSAIC is compliant with most providers (e.g., MS Azure).

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Unions Seventh Framework Programme (FP7/2007-2013) under grant agreement no 604691 (project FI-STAR).

REFERENCES

Bohn, R. B., Messina, J., Liu, F., Tong, J., and Mao, J. (2011). Nist cloud computing reference architecture. In *Proceedings of the 2011 IEEE World Congress on Services*, SERVICES '11, pages 594–596, Washington, DC, USA. IEEE Computer Society.

Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. (2001). Web Service Definition Language (WSDL). W3C recommendation, W3C. http://www.w3.org/TR/wsdl.

Loutas, N., Kamateri, E., and Tarabanis, K. A. (2011). A Semantic Interoperability Framework for Cloud Platform as a Service. In *CloudCom*, pages 280–287.

Martin, D. L., Paolucci, M., McIlraith, S. A., Burstein, M. H., McDermott, D. V., McGuinness, D. L., Parsia, B., Payne, T. R., Sabou, M., Solanki, M., Srinivasan, N., and Sycara, K. P. (2004). Bringing Semantics to Web Services: The OWL-S Approach. In *SWSWPC*, pages 26–42.

Mell, P. and Grance, T. (2009). The NIST Definition of Cloud Computing. Technical report.

Moscato, F., Aversa, R., Martino, B. D., Fortis, T.-F., and Munteanu, V. I. (2011). An Analysis of mOSAIC ontology for Cloud Resources annotation. In *FedCSIS*, pages 973–980.

Nagarajan, M., Verma, K., Sheth, A., Miller, J., and Lathem, J. (2006). Semantic Interoperability of Web Services - Challenges and Experiences. In *ICWS '06: Proceedings of the IEEE International Conference on Web Services (ICWS'06)*, pages 373–382, Washington, DC, USA. IEEE Computer Society.

Nagireddi, V. and Mishra, S. (2013). An ontology based cloud service generic search engine. In *Computer Science Education (ICCSE), 2013 8th International Conference on*, pages 335–340.

Ngan, L. D. and Kanagasabai, R. (2012). Owl-s based semantic cloud service broker. In *ICWS*, pages 560–567.

Petcu, D. (2011). Portability and interoperability between clouds: challenges and case study. In *Proceedings of the 4th European conference on Towards a service-based internet*, ServiceWave'11, pages 62–74, Berlin, Heidelberg. Springer-Verlag.

Petcu, D., Macariu, G., Panica, S., and Crciun, C. (2013). Portable Cloud applications - From theory to practice. *Future Gener. Comput. Syst.*, 29(6):1417–1430.

van Harmelen, F. and McGuinness, D. (2004). OWL web ontology language overview. W3C recommendation, W3C. http://www.w3.org/TR/2004/REC-owl-features-20040210/.

Zhang, M., Ranjan, R., Haller, A., Georgakopoulos, D., Menzel, M., and Nepal, S. (2012). An ontology based system for cloud infrastructure services discovery. *CoRR*, abs/1212.0156.

VM Live Migration Algorithm Based on Stable Matching Model to Improve Energy Consumption and Quality of Service

Abdelaziz Kella and Ghalem Belalem
Department of Computer Science, University of Oran, Oran, Algeria
kaa.kella.abdelaziz@gmail.com, ghalem1.dz@gmail.com

Keywords: Energy Efficiency, Quality of Service, Cloud Computing, Datacenter, Energy Consumption, Stable Matching Problem, the Coase Theorem, VM Migration, Live Migration.

Abstract: Cloud Infrastructure is a newly emerged computing platform that builds on the latest achievements of diverse research areas, such as Grid computing, Service-oriented computing, business process management and virtualization. The use of server virtualization techniques for such platforms provides great flexibility with the ability to allocate users and applications per virtual machine, ability to consolidate several virtual machines (VMs) on the same physical machine (PM), to resize a virtual machine as needed and being able to support migration of virtual machines across physical machines (PMs) to meet users and system requirements. However, datacenters hosting Cloud applications consume huge amounts of electrical energy, contributing to high operational costs for the providers' services as well as for the users' services. This paper presents a live migration strategy based on economic model named "stable marriage problem" to find a fair stable matching between VMs' requirements and PMs' requirements. The proposed approach aims to improve the quality of service while reducing the energy consumption. We refine our approach by integrating the Coase Theorem to reduce the number of migrating VMs and maximize the system's performances. Preliminary experiments show that, with a suitable number of PMs and VMs, the strategy can reduce the energy consumption of Physical machines and datacenter while improving the quality of service.

1 INTRODUCTION

Cloud computing (Richa et al., 2011) (Buyya et al., 2011) is modeled to provide utility-based IT services. Services like computation, software, data access and storage are provided to its users without its knowledge about physical location and configuration of the server that is providing the services. To guarantee these specifications, large-scale datacenters are established. These datacenters usually employ Virtual Machines (VMs) technologies for consolidation and environment isolation purposes (Beloglazov et al., 2012) (Barham et al., 2003). The Cloud services are, generally, referred to Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) respectively.These services (Buyya et al., 2011) are not only used but also installed, deployed or replicated with the help of virtualization.

Virtualization is a key technology that has enabled several operating systems to run simultaneously on a single physical machine.Hardware virtualization provides a logical separation between applications and the underlying physical machine resources, thereby enabling a flexible mapping of virtual machines to Physical Machines (PMs) in a datacenter. Further, virtual machine platforms support resizing of VM containers to accommodate changing workloads as well as the ability to live-migrate virtual machines from one physical machine to another without incurring application downtimes (Wood et al., 2011).

VM migration is one of the important capabilities of system virtualization, which allows applications to be transparently migrated along with their execution environments across PMs. Live migration further allows the VM to be migrated almost without any interrupt to its application's execution. VM migration is an important means for managing applications and resources in a large virtualized system. It enables resource usage to be dynamically balanced in the entire virtualized system across physical host boundaries. It, also, allows applications to be dynamically relocated to

improve performance and reliability.

Certainly, virtualization isa very useful technology that allows simulating a variety of different platforms and managing the resources of the system. It offers several advantages, such as flexible and efficient sharing of resources, fault tolerance, portability, cost efficiency, and resource consolidation (Barham et al., 2003). However, Modern resource-intensive enterprise and scientific applications create growing demand for high performance computing infrastructures. This has led to the construction of large-scale computing datacenters consuming enormous amounts of electrical power. Despite of the improvements in energy efficiency of the hardware, overall energy consumption continues to grow due to increasing requirements for computing resources.Such large consumption of electrical energy has increased operating cost for the service providers as well as for the service users. To address this problem, many works had been proposed to decrease energy consumption of datacenters. These researches focused on the resources consolidation, the full utilization of PMs, and switching off the idle PMs. the proposed works bring considerable benefits in energy consumption context, but in the other hand, they affect, negatively, the systems performances. Thus, how to save the energy of the datacenter without affecting the system performance (Quality of service) is an important issue.

Aiming at the problem, this paper presents a stable matching strategy based on dynamic migration of virtual machines. The strategy seeks to improve the energy consumption, based on CPU utilization, without affecting the quality of service. Assuming that the VMs are the side that wants to preserve the quality of service and PMs are the side that looks to optimize the energy consumption, our approach aims to find a stable matching fair to both VMs and PMs instead of favoring either side because of a deferred acceptance procedure. However, the resulting solutions by VMs and PMs may sometimes be different and even conflicting. This is known as the polarization of stable matching (Roth et al., 1989) (Martínez and al, 2003)**.** Some works have been proposed (Hong Xu et al., 2011) to solve this problem, but in the case of polarization, they favor one side at the expense of the other. In this work, we use an extension of the stable matching (Martínez and al, 2003) that is used in economics to find a compromise between both sides (VMs and PMs) in the case of polarization. We presume two dynamics thresholds, one for the energy consumption and the other for the quality of service, to find the appropriate moment to start migrating the VMs hosted on PMs that the load is over one of the two presumed threshold. Our approach incorporates another theorem used in the economy "the Coase Theorem" (R. H. Coase, 1960).This theorem allows us to find the optimal number of VMs to migrate to select the solution among other possible (depending on the number of migrated VMs) that best enhances the energy and quality of service. Rest of this paper is organized as follows: In the second section, we present some research on the VMs migration in cloud computing. An overview of the stable matching problem and the Coase theorem is presented in the third section. The fourth section describes the suggested approach. In the fifth section, we evaluate our approach by a set of experiments. Finally, the sixth section will be reserved for the conclusion and some future work.

2 RELATED WORKS

The academia and commercial companies have conducted numerous research efforts to determine best approaches to improve system performances while reducing the energy consumption. We cite many relevant works in this paper:

Hong Xu and al **(**Hong Xu et al., 2011**)** proposed the Egalitarian stable matching for VM migration in cloud computing. Egalitarian approach finds a stable matching fair to both VMs and servers, instead of favoring either side because of the deferred acceptance procedure. Servers differ in their traffic loads, affecting the performance of migration. VMs also differ in the volume of disk images to be migrated, incurring different amounts of transmission overhead – a critical factor that server needs to consider. Egalitarian stable matching – tries to find the matching that minimizes the total rank sum of the outcome among all stable matching and it applies a polynomial time algorithm to find such egalitarian stable matching; However, the authors ignores the migration costs. In our work, we use the idea of the stable matching and integrate the Coase theorem to decrease the migration costs, and an algorithm to compute the full set of many stable matchings' that allow us to find a fair stable matching.

The paper (Lee et al., 2012) proposed a dynamic energy-saving mechanism in a cloud-computing environment for reducing power consumption. The system consists of three parts: the CPU utilization monitoring, DVFS **(**Pouwelse et al., 2001**)** adjustment, and live migration. CPU utilization on

each host is monitored in the system. According to the measured CPU utilization, an appropriate process will be determined for saving energy consumption. The voltage of the idle or light loaded computer can be reduced and heavy loaded works can be migrated to those machines with lighter loading for achieving the purpose of energy saving. The results obtained was promising in term of power consumption, however, the system degradation, due to the migration process or the VM placement after the migration, has not been taken into consideration. To achieve their objective, the authors propose three policies of continuous optimization of VM placement: (1) Optimization over multiple system resources utilization like CPU, RAM, and network bandwidth. (2) Network optimization: communication between VMs should be considered in reallocation decisions in order to reduce data transfer overhead and network device load. (3) Thermal optimization: take into consideration the temperature of the physical nodes in the reallocation of VMs. This aims to avoid the "hot spots" by reducing workload of the over headed nodes, and decrease the cooling system load.

In the paper (Beloglazov et al., 2010), the authors present a decentralized architecture with and energy efficient resource management system for virtualized cloud datacenters that reduces operational costs and provides required quality of service. The proposed model seeks to minimize the energy consumption by consolidating the VMs according to virtual network topologies (VMs network), current resources utilization, and thermal state of computing nodes. The authors have proposed three stages of continuous optimization of VM placement and presented heuristics for a simplified version. The heuristics have been evaluated by simulation using the extended CloudSim toolkit. The experimentations show that, with a suitable Thresholds values, the policy can provide, generally, good results.

3 BACKGROUND AND PRELIMINARY

3.1 The Stable Matching Problem

In economics, We say that a market situation is stable in a general sense, if there is no set of agents such that all of them are interested in creating a new cooperation (after breaking their other eventual cooperations). Gale and Shapley (Gale et a.l, 1962) introduced and studied the problem of stable marriages. Here a matching, that corresponds to a set of marriages, is stable, if there exists no man and woman, who would both prefer to marry each other (after leaving their eventual partners). Gale and Shapley described a natural algorithm that finds a stable matching for the marriage**.** Three types of SMP are distinguished: one-to-one problem (the stable marriage problem), many-to-one problem (the college admission problem), and many-to-many problem (the firm consultant problem). In this paper, we are interested by the many-to-one problem, but we will present, first, a variant of one-to-one problem (the stable marriage problem) for more simplicity and clearness.

3.1.1 The Stable Marriage Problem

A stable marriage (SM) problem (Gusfield and al, 1989) consists of matching members of two different sets, usually called men and women. When there are n men and n women, the SM problem is said to have size n. Each person strictly ranks all members of the opposite sex. The goal is to match the men with the women so that there are no two people of opposite sex who would both rather marry each other than their current partners. If there are no such pairs (called blocking pairs), the marriage is *"stable"*.

Definition 1 (Marriage). *Given an SM*P *of size* n*, a marriage* M *is a one-to-one matching of the men and the women. If a man* m *and a woman* w *are matched in* M*, we write* M (m) = w *and* M (w) = m*.*

Definition 2 (Blocking Pair). *Given a marriage* M*, a pair* (m,w)*, where* m *is a man and* w *is a woman, is a blocking pair if* m *and* w*are not partners in* M*, but* m *prefers* w *to* M(m) *and* w *prefers* m *to* M(w).

Definition 3 (Stable Marriage). *A marriages* M *is stable if it has no blocking pairs.*

A convenient used SM representation is showed in Figure 1, where each person is followed by his/her preference list in decreasing order and the figure 2 shows a stable marriage for the preference lists given in figure 1.

Men's list	Women's list
A: a,b,c,d	a: A,B,C,D
B: b,a,c,d	b: D,C,B,A
C: a,d,c,b	c: A,B,C,D
D: d,c,a,b	d: C,D,A,B

Figure 1: Sample Preference List for Men and Women.

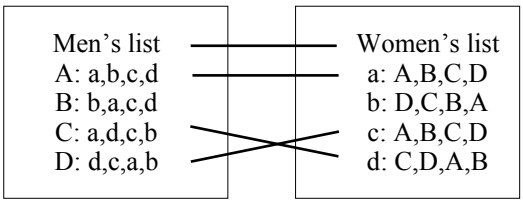


Figure 2: A Stable Marriage for Men and Women.

The Gale-Shapley (GS) algorithm (Gale and al, 1962) (or the deferred acceptance algorithm) is a well-known algorithm to solve the SM problem. At the start of the algorithm, each person is free and becomes engaged during the execution of the algorithm. Once a woman is engaged, she never becomes free again (although to whom she is engaged may change), but men can alternate between being free and being engaged. The following step is iterated until all men are engaged: choose a free man m, and let m propose to the most preferred woman w on his preference list, such that w has not already rejected m. If w is free, then w and m become engaged. If w is engaged to man m', then she rejects the man (m or m') that she least prefers, and becomes, or remains, engaged to the other man. The rejected man becomes, or remains, free. When all men are engaged, the engaged pairs form the male optimal stable matching. It is female optimal, of course, if the roles of male and female participants in the algorithm were interchanged (Silvia and al, 2010).

3.1.2 The College Admission Problem

This problem is a variant of SMP in which students have to be matched to colleges (Gale and al, 1962) (Silvia and al, 2010) based on the students' and the colleges' preferences over the other side of the market and colleges' capacity constraints. An outcome for such a college admissions market, a matching, is an assignment of students to colleges such that each student is matched to at most one college and no college is matched to more students than its capacity allows for.

There are two finite and disjoint sets of agents: a set $\underline{\boldsymbol{S}}$= $\{s_1; : : : ; s_m\}$ of students and a set $\underline{\boldsymbol{C}}$= $\{C_1; : : : ; C_n\}$ of colleges. We denote a generic student by s and a generic college by C. For each college C, there is a fixed quota q_C that represents the number of positions it offers (In marriage model q_C=1).

Each student s has a complete, transitive, and strict preference relation $\geq_s$ over the colleges and the prospect of being *unmatched*. Hence, student s's preferences can be represented by a strict ordering $P(s)$ of the elements in $\underline{\boldsymbol{C}} \cup \{s\}$. If $C \in \underline{\boldsymbol{C}}$ such that $C >_s s$, then we call C an *acceptable college for student* s.

A set of students S', which belongs to $\underline{\boldsymbol{S}}$, *is feasible for college* C if $|S'| \leq q_C$. Each college C has a complete and transitive preference relation $\geq_C$ over feasible sets of students, which can be represented by a weak ordering $P(C)$ of the elements in $P(S; q_C) \equiv \{S'$ belongs to $\underline{\boldsymbol{S}}$: $|S'| \leq q_C\}$ We make two assumptions on the preferences of a college C (Roth and al, 1989):

First, C's preferences over singleton sets of students, or equivalently over individual students, are strict. For notational convenience, we denote a singleton set $\{s\}$ by s. The second assumption describes comparisons of feasible sets of students when a single student is added or replaced. If $s \in \underline{\boldsymbol{S}}$ is such that $s >_C \Phi$ then we call s an *acceptable student for college* C. If s, $s' \in \underline{\boldsymbol{S}}$ are such that $s >_C s'$, then we call student s *a better student than student s' for college* C. We assume that each college C's preferences over feasible sets of students are based on preferences over individual students such that C always prefers to add an acceptable student and it also prefers to replace any student by a better student.

This model is the base of our approach. We study the VM migration as a college admission problem when the VMs represent students and the servers represent colleges.

3.2 The Coase Theorem

The Coase theorem has evolved from an illustrative argument in Ronald Coase's "The Problem of Social Cost" (Coase, 1960) to a centerpiece of the modern law and economics movement. Ronald Coase first invented a new concept of "transaction costs" to lay the groundwork for his insight. "Transaction costs" consist of the time, money, and effort someone loses in obtaining what he wants (Andrew, 2007). The Coase Theorem, developed by economist Ronald Coase, states that, when conflicting property rights occur, bargaining between the parties involved will lead to an efficient outcome regardless of which party is ultimately awarded the property rights, as long as the transaction costs associated with bargaining are negligible. Specifically, the Coase Theorem states that "*if trade in an externality is possible and there are no transaction costs, bargaining will lead to an efficient outcome regardless of the initial allocation of property rights*"(R.H. Coase, 1960).

This theorem was developed by Ronald Coase when considering the regulation of radio

frequencies. He posited that regulating frequencies was not required because stations with the most to gain by broadcasting on a particular frequency would have an incentive to pay other broadcasters not to interfere (INVESTOPEDIA, 2013).
In this paper, we are focused on the principle of the Coase theorem, and we will use it to determine the number of VMs to migrate to have the best possible compromise between energy consumption and quality of service.

4 PROPOSED VM MIGRATION MODEL

4.1 System Model

We consider the cloud computing system model as shown in the Figure 3. The users send their requests towards an environment cloud computing. Then, the system will automatically direct this request towards the suitable datacenter Di among the set D = {D1, D2… D_n} of the datacenters. Each datacenter D_i contains a set PM_i= {PM_{i1}, PM_{i2}… PM_{im}} of physical machines. The broker B_i of the datacenter D_i will choose, thereafter, the virtual machine VM_{ijk} of the unit VM_{ij}= {VM_{ij1}, VM_{ij2}… VM_{ijp}} which will satisfy the request of the user, and which is stored in the physical machine PM_{ij}.

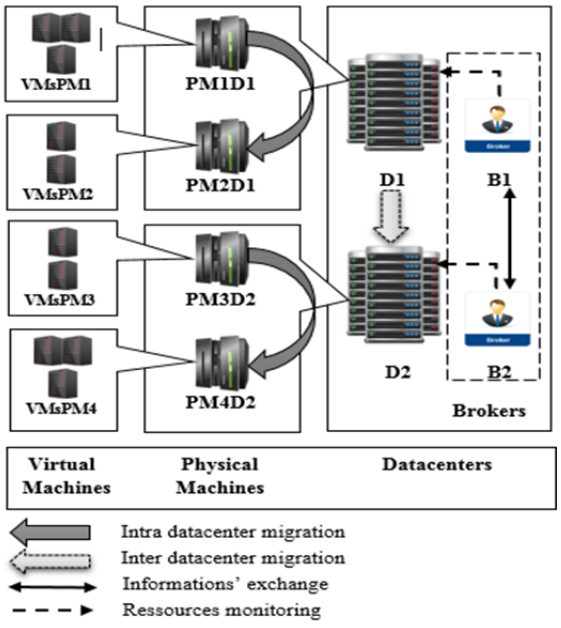


Figure 3: Architecture cloud computing for the VM migration proposed strategy.

The broker of each datacenter periodically checks the internal and external state of the datacenter to decide, then, of the need for the VMs migration type:

1. *Intra datacenter migration*: if the state of one or more PMs of the datacenter is critical.
2. *Inter datacenter migration*: if the global state of the datacenter is critical.

4.2 VM Migration proposed Algorithm

The VM migration proposed algorithm is divided in 4 parts. These parts are based on four questions answers:

- ✓ *When to migrate? Moment of migration*: to know when a migration is necessary.
- ✓ *Where to migrate? VM placement:* to seek PMs being appropriate to host VMs which will be migrated.
- ✓ *How to migrate? The VM migration process.*
- ✓ *How much and which VM to migrate? Determination of VM's that will be migrated:* to determine the number of VMs to migrate for a better improvement of the system performances.

4.2.1 The Migration Moment

Each broker checks periodically the state of its datacenter, and it launches a migration only if one of the two thresholds (*Time response threshold*or *Energy threshold*) were exceeded.

As the suggested model is based on an economic strategy which is "the stable matching", that aims to find a consensus between two parts, two dynamic thresholds were defined:

- ✓ Response time
- ✓ Energy consumption based on CPU utilization.

The broker of each datacenter periodically calculates these two thresholds to determine if a migration is essential or not.
For a datacenter D_i, which has *m* physical machines, the response time threshold (see formula (1)), and the energy consumption threshold (see formula (2)) are calculated, respectively, as follows:

Time-resp(D_i)=

$$\left[\frac{\sum_{j=1}^{j=m} Time-resp\ j}{m}\right] + \left(\left[\frac{\sum_{j=1}^{j=m} Time-resp\ j}{m}\right] * Const\right) \quad (1)$$

S_Energy (D_i) =

$$\left[\frac{\sum_{j=1}^{j=m} EnergyCons\ j}{m}\right] + \left(\left[\frac{\sum_{j=1}^{j=m} EnergyCons\ j}{m}\right] * Const\right) \quad (2)$$

Where:

Const: is a constant (30%≤*const*≤70%).

Time-resp$_j$: the average time needed by VM to execute its cloudlets.
EnergyCons$_j$:the energy consumption of PM $_j$.

A. The Response Time
The response time of one cloudlet (CLDi) is calculated as follow:

$$\text{Time-resp}_{CLDi} = \text{Time-exe}_{CLDi} + \text{Time-trn}_{CLDi} + \text{Time-att}_{CLDi} \quad (3)$$

Where:

$$\text{Time-exe}_{CLDi} = \text{CLDi-Size/VM-Size (MIPS)} \quad (4)$$

$$\text{Time-trn}_{CLDi} = \text{CLDi-Size/BW-VM (Mb/S)} \quad (5)$$

Time-att$_{CLDi}$ is the time that the cloudlet wait in the queue before being executed by the VM.

B. The Energy Consumption
According to (Richa and al, 2011)**:** many studies (Fan, 2007) (Kusic, 2009) show that a linear relation between the power consumption and CPU utilization can describe the energyconsumed by servers. These studies as say that an average power consumed by an idle server is 70% of power consumed by fully utilized server. So, the power consumption is considered as CPU utilization P(u) by as shown in (6):

$$P(u) = Pmax * (0.7 + 0.3*u) \quad (6)$$

Where:
Pmax is 250 W for modern computing server
u is the CPU utilization.

4.2.2 The VM Placement

In all the cases that will be approached, the condition, shown in the formula (7), must be checked:

$$\sum_{j=1}^{m} CapPMj > VSize \quad (7)$$

Such as:
CapPM$_j$: the free space of a physical machine *j* that was selected to host one or more VMs.
VSize: The sum of the sizes of all VMs that will be migrated.

4.2.3 The VM Migration Process

The migration process of the suggested model is summarized in two phases:

A. The Establishment of the Preferences' Lists: It consists in creating:

✓ A preference list *VM_Pre$_k$* for each virtual machine *VM$_k$*, concerned by the migration. This list contains the physical machines, which are selected like potential sites. These PMs are put in an ascending order according to the transfer time *(Time_Trans (VM$_k$, S (VM$_k$), D (VM$_k$)) formula (8)*) of *VM$_k$* towards these physical machines.

$$Time\_Tans\left(VMk, S(VMk), D(VMk)\right) = TS\_VMk * \sum_{c=1}^{q-1} \frac{1}{BP(c,c+1)} \quad (8)$$

$$CPU_{Utilisation}(PMj) = [\sum_{l=1}^{W} Impact\_Utilisation(VMl)]/W \quad (9)$$

Such as:
S_VM$_k$: is the size of VM *VM$_k$*.
S(VM$_k$): is the physical machine containing *VM$_k$*.
D(VM$_k$): is the physical machine of the destination.

✓ A preference list *PM_Pre$_j$*for each machine physical *PM$_j$*, concerned by migration This list contains VMs, which will be migrated, put in an ascending order according to the impact of these VMs on the use of the machines *PM*$_j$ CPU (*CPU_Utilization (PM$_j$)*)

Such as:
W: is the number of virtual machines hosted by the PM *PM$_j$*

$$Impact_{Utilisation}(VMk) = CPU(VMk)/CPU(PMj) \quad (10)$$

Where:
Impact_utilization (VM$_k$): is the utilization ratio of the PM$_j$ machine CPU by VM VM$_k$.
CPU (VM$_k$): is the quantity of MIPS required by the virtual machine VM$_k$.
CPU (PM$_j$): is the quantity of MIPS that the physical machine PM$_j$ can support.

B. The Establishments of the Partners' Lists: in this phase, one of two parts (PM or VM) made proposals for the other, to join him, and this last according to its preferences decides if it will accept or disallow these proposals.

In what follows, we will present two algorithms, one for the side VM (which make proposals) and the other for the side PM (which accept or disagree proposals). At the end of the treatment, the two sides will have their lists of the partners (*VM_list$_k$* and *PM_list$_j$*).

The main steps of a proposal algorithm (VM side) are:
Step 1: if Stop is Boolean with a false as value, go to step 2.

Step 2: if the preference's list of $VM_k$is not empty, go to step 3, else go to step 1.
Step 3: propose to PM_j, the first element in the preferences' list of VM_k(*VM_Pre_k)*, delete PM_j from *VM_Pre_k*, and add it to the partners' list of VM_k(*VM_list_k*), and go to step 4.
Step 4: wait for the PM_j's response while Stop is false. If a response have been received, go to step 5
Step 5: if PM_jaccept the proposal, then VM_k has found his partner and go to step 4, else go to step 6.
Step 6:delete $PM_j$from the VM_k preferences' list. Go to step 7.
Step 7: if PM_j belongs to VM_list_k then delete it from the list, else Stop becomes true. Go to step 2.

The main steps of an acceptance algorithm (PM side) are:
Step 1: if the broker does not send a "Stop" message, go to step 2.
Step 2: wait for a proposal from a virtual machine VM_k. if a proposal have been received, go to step 3.
Step 3: if PM_j has a suitable capacity (Storage, RAM, CPU) to put in the VM_k, so go to step 4, otherwise go to step 7.
Step 4: Accept the proposal of VM_k, add it to the partner's list of PM_j (*PM_List_j*) by preference's order, and calculate the new capacity of PM_j after hosting VM_k. Go to step 5.
Step 5: if the PM_j's capacity is inferior to VM_k requirements, go to step 6, else go to step 7.
Step 6: send a rejection message to all the virtual machines VM_l classified after VM_k in the preference's list of PM_j, delete them from *PM_Pre_j*, and go to step 7.
Step 7: if VM_k belongs to PM_Pre_j, then go to step 8, otherwise go to step 10.
Step 8: send a rejection to the last element of PM_List_j (VM_{Last}) because VM_k is best classified than VM_{Last} in PM_Pre_j, send a confirmation message to VM_k, add it to the partner's list of PM_j (*PM_List_j*) by preference's order, and delete VM_{Last} from the PM_List_j. Go to step 9.
Step 9: send a rejection message to all the virtual machines VM_l classified after VM_k in the preference's list of PM_j, delete them from *PM_Pre_j*, and go to step 1.
Step 10: send a rejection message to $VM_k$and go to step 1.

When all the VMs' "Stop" values become true, the broker deduces that all the VMs had find their stable matching, therefore, it will send a message to the PMs to incite them to stop waiting for proposal from VMs.

The result of these two algorithms is an optimal stable matching for VMs (Optimal μ_{vm} or VM-optimal). By applying the same algorithms, but this time is, the PMs will be the side that will make the proposals; we obtains an optimal stable matching for the physical machines (Optimal μ_{pm} or PM-optimal).

However, the two stable matching (VM-optimal and PM-optimal) can be different and even contradictory (the best solution for a side can be, at the same time, the worst for the other side). This problem is known as the polarization of the stable matching.

In this paper, to mitigate this problem, we used an algorithm, which was proposed in (Martínez, 2003) to solve the problem of polarization in variant of many to many stable matching problem that is the firms-consultants problem, by computing the full set of many to many stable matching based on the changes makes in preferences lists.

In this algorithm, the side which makes proposals must remove its best choice of its preference list, and start again its procedure to obtain its optimal stable matching (VM-optimal or PM-optimal). After, the other side must make the same thing to obtain, in its turn, its optimal stable matching. These steps will be repeated until the two stable matching converge or in the best case, we will obtain the same stable matching for the two sides (VM-optimal = PM-optimal).

4.2.4 Determination of VMs that Will be Migrated

This step is the last being executed, after having answered all the other questions: What to migrate? When to migrate? How to migrate?

We will use the Coase theorem (*Section 3.2*) to determine the optimal distribution of VMs on the datacenters for having the best possible compromise between the energy consumption and the time response.

If the energy consumption threshold were exceeded then we will use the energy consumption as the income and the quality of service as the social costs and vice versa (*see example in section 3.2*).

Example:

We suppose that we obtain this stable matching after the migration process: {(PM_1, VM_7), (PM_2, VM_6), (PM_3, VM_8)} and the energy threshold is the one that trigger the VM migration. Therefore, the PMs will be put in a list by increasing order based on their energy consumptions.

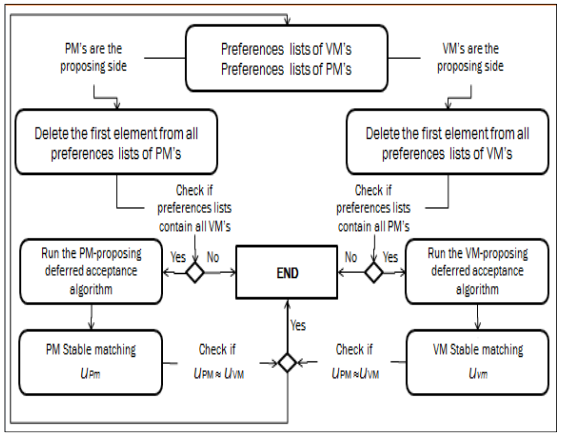


Figure 4: A diagram that presents the principle of the algorithm that compute the full set of many-to-many stable matching.

Be $V_1$the energy consumption of a PM before migration, and V_2 its energy consumption after migration.

We calculate the gain of energy consumption by the formula (11):

$$\text{Gain (PMj)} = \left[\frac{\sum_{i=1}^{i=j} V1\ (PMi)}{\sum_{i=1}^{i=j} V2\ (PMi)}\right] \quad (11)$$

On the other hand, the loss by the formula (12):

$$\text{Loss (PMj)} = \left[\frac{\sum_{i=1}^{i=j} V2\ (PMi)}{\sum_{i=1}^{i=j} V1\ (PMi)}\right] \quad (12)$$

Table 1: Example of an implementation of the Coase theorem.

	Energy Consumption	Response Time	Total Gain
$PM_1(VM_7)$	50 %	-15 %	50 -15 = 35 %
$PM_2(VM_6)$	20 %	20 %	40 %
$PM_3(VM_8)$	25 %	-10 %	15 %

The Energy consumption gain of PM_7 is calculated by dividing the V_1 (PM_7) on V_2 (PM_7) while the energy consumption gain of $PM_6$is calculated by dividing the sum of V_1 (PM_7) and V_1 (PM_6) on the sum of V_2 (PM_7) and V_2 (PM_6). The same method is used to obtain the energy consumption gain of the others PMs

We remark in the table that the line 2 is the best line with 40% of total gain. Therefore, the final distribution of VMS will be as follows: {(PM_1, VM_7), (PM_2, VM_6)}. The VM_8 will not be migrated to the PM_3.

The Coase theorem is used to reduce the migration overhead while improving the energy consumption and the QOS, by decreasing the number of VM to migrate.

5 EXPERIMENTATIONS AND RESULTS

In order to evaluate the approach suggested, we performed three series of experiments on CloudSim (a simulator developed in Java language). Java language is used for develop and implement the proposed strategy. In these experiments, we focus on the impact of the number of physical machines (servers), virtual machines, and cloudlets (requests) on the response time and the energy consumption. Our approach and a strategy without migration will be compared to show the benefits that it (our approach) brings in response time and energy consumption. Assuming the application is deployed in one datacenter.

5.1 The Impact of the Physical Machines Number

In these simulations, we arranged one datacenter that contains 100 same virtual machines stored in a set of same physical machines. The set's size changes every simulation. It varies between 20 and 100. The datacenter's broker distributes a set of 1000 cloudlets in every simulation (*Table 2*).

The Figures 5 and 6 show, respectively, the comparison of response time and energy consumption between the proposed mechanism and the default mechanism. The response time and the energy consumption without applying the proposed mechanism are presented by the blue color, and the red color presents the response time and the energy consumption by applying the proposed mechanism.

Table 2: Simulation's parameters of the first experimentation.

Parameter	**Value**
Datacenter's number	1
PM's number	[20,40,60,80,100]
VM's number	100
Cloudlets number	1000
PM's CPU	4
VM's CPU	1
PM's RAM	2 Go
VM's RAM	1 Go

About 24-25% of response time can be minimized by applying the proposed model.

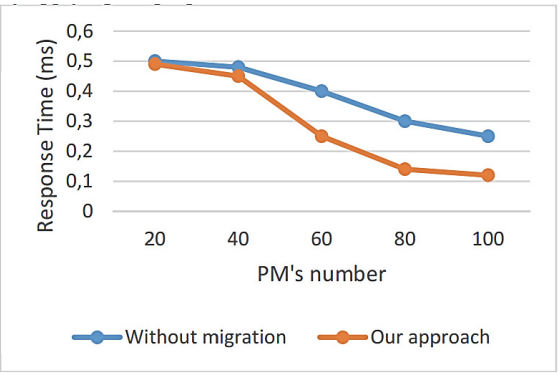


Figure 5: The impact of PM number on response time.

About 23-24% of energy consumption can be saved by applying the proposed model even if the results of the two mechanism are too close in the end (the reason will be explained the *section 5.3*)

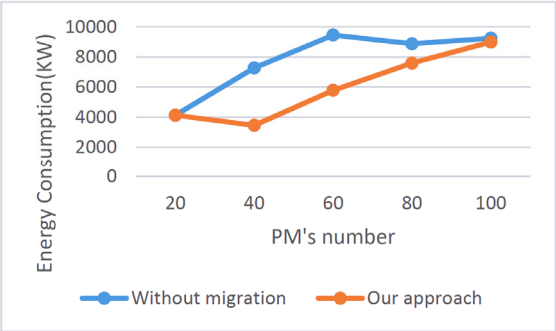


Figure 6: The impact of PM number on energy consumption.

5.2 The Impact of the Virtual Machines Number

These experiments were performed with 200 physical machines, which form 1 datacenter, 1000 cloudlets, and a variable set of virtual machines *(Table 3).*

Table 3: Simulation's parameters of the second experimentation.

Parameter	Value
Datacenter's number	1
PM's number	200
VM's number	[50,75,100,125,150,175, 200,225,250,275,300]
Cloudlets number	1000
PM's CPU	4
VM's CPU	1
PM's RAM	2 Go
VM's RAM	1 Go

As shown in figure 7, our proposed model provide the best results because of the periodic migration of the VM's to the PM's with low CPU utilization.

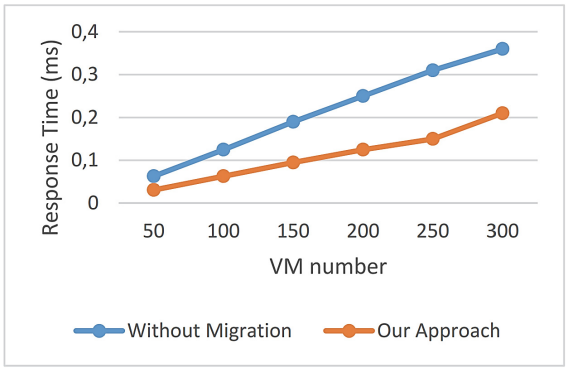


Figure 7: The impact of VM number on response time.

We remark in the figure 8 that the two can be divided into two intervals [50… 200] and [200 … 300]. In the first interval, the results provided by our proposed approach are the worst because of the effort of migration that is unnecessary in a mechanism without migration. This effort is a consequence of the behaviour of our approach (for more details *section 5.3*

5.3 Discussion

From these three series of simulation, we can conclude that the only parameters that have a real impact on the two measured metrics (response time and energy consumption) are the PM number and the VM number and the import is more important on the energy consumption. In this section, we will discuss the reasons the impact of the VM number and the PM number on the calculated energy.

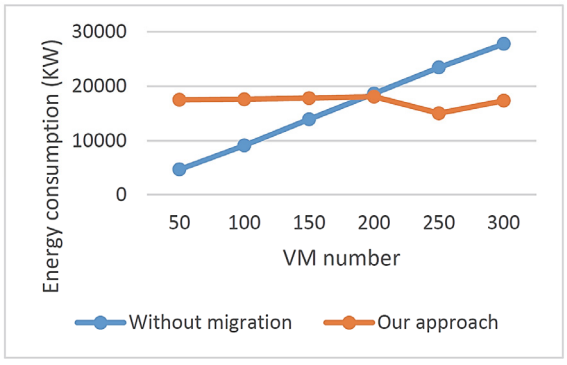


Figure 8: The impact of VM number on energy consumption.

We remark from all the previous simulation and their results, concerning the energy consumption, that the best results are provided when the number of PMs is between 30%VM number < PM number <70%VM number, why?

This is because of the behavior of our approach. The proposed model must calculate periodically two thresholds (energy consumption and response time). When the PM number do not respect this condition one of the two thresholds will be useless, how?

1. If PM number < 30% VM number, the two thresholds will be exceeded every time, so the proposed model will migrate VM each period. The migration here will become inconvenient because the PM will be all the time full utilized (Figure 8. The first interval).
2. In the second case, where PM number > 70% VM number, the energy consumption will not be exceeded, so the proposed approach will favor the response time thresholds by migrating VMs into different PMs to improve the response time. Therefore, after a small number of migrations our approach will be almost identical to an approach without migration (Figure 6. At the end of the curve).

6 CONCLUSION AND FUTURE WORKS

This paper has presented a stable matching algorithm for VM live migration in Cloud infrastructures. The proposed approach aims to improve the energy consumption without degrading the quality of service (Qos). We proposed, also, an economical algorithm to compute the full set of stable matchings, in case of polarization, to find the ideal compromise (matching). Moreover, the Coase theorem reduces significantly the cost of migration by determining the number of VMs to migrate that maximizes system performances. We perform other experiments and we remarked that our approach is independent of the workload. As well, the experiments show that, with a suitable number of VMs and PMs, the proposed model has a very good effect on reducing the power consumption and the quality of service (response time).

We propose a plan for the future works that consists of several steps:

1. Completion of the algorithm by introducing the SLA Constraints and improving the Compute full set algorithm for the polarization problem.
2. Making other experiments to show the importance of the use of the Coase theorem on the migration cost and the system performances.
3. Real implementation of the system, and experimental evaluation, in a cloud platform such as Eucalyptus **(**Eucalyptus Systems, 2010**).**

REFERENCES

S. Richa,.,P. Nidhi,, D. Hiteishi., Dec 2011. "Energy Efficient Dynamic Integration of Thresholds for Migration at Cloud Data Centers ".*Special Issue of International Journal of Computer Applications (0975 8887) on Communication and Networks, No.11.*

R. Buyya, CS Yeo,S. Venugopal, J. Broberg, I. Brandic. 2011. "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility". Future Generation Computer Systems.

P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield. 2003. "Xen and the art of virtualization", in: *Proceedings of the 19th ACM Symposium on Operating Systems Principles, SOSP 2003, Bolton Landing, NY, USA*, p. 177.

A. Beloglazov, J. Abawajy, R. Buyya, 2012. "Energy-aware resource allocation heuristics for efficient management of datacenters for Cloud computing", *Future Generation Computer Systems 28*, 755–768.

T. Wood, P. Shenoy, K. K. Ramakrishnan, J. Van der Merwe. March 2011. "CloudNet:DynamicPooling ofCloudResourcesbyLiveWANMigrationofVirtualMa chines". *VEE'11, ACM.*

R. Martínez, J. Massó, A. Neme, and J. Oviedo. July 2003. "An algorithm to compute the full set ofmany-to-manystable mathings".*Journal of Economic Literature Classification Number: C78.*

Hong Xu, Baochun Li. April 2011."Egalitarian Stable Matching for VM Migration in Cloud Computing".*Department of Electrical and Computer Engineering, University of Toronto, Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference, page 631-636.*

R. H. Coase. Oct. 1960. "The Problem of Social Cost". *Journal of Law &1 Economics, III, pp. 1-44*

L. T. Lee, K. Y. Liu, and H. Y. Huang. May 2012." Dynamic resource management for energy saving in the cloud computing environment",*Proceedings of the International Conference on Information Science and Industrial Applications, pp. 176-181.*

J. Pouwelse., K. Langendoen, H. Sips. 2001. "Dynamic Voltage on a Low-Power Microprocessor System". *In: Proceedings of the International Conference on Mobile Computing and Networking, pp. 251-259.*

A. Beloglazov, R. Buyya, 2010. "Energy Efficient Resource Management in Virtualized Cloud DataCenters".*10th IEEE/ACM International Conf.on Cluster, Cloud and Grid Computing; Cloud*

Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computer Science and Software Engineering, the University of Melbourne, Australia.

D. Gale and L. S. Shapley. 1962. "College Admissions and the Stability of Marriage". *Amer. Math. Monthly, 69(1):9–15.*

D. Gusfield and R. W. Irving. 1989. "The Stable Marriage Problem: Structure and Algorithms".*MIT Press, Boston, MA.*

M. Silvia Pini, F. Rossi, K. B. Venable, and T. Walsh. 2010. "Stable marriage problems with quantitative preferences".*In Informal Proc.of COMSOC'10 Third International Workshop on Computational Social Choice.*

A.E.Roth, and Sotomayor, M.A.O.1989. "The College Admissions Problem". *Revisited,"Econometrica, 57, 559-570.*

INVESTOPEDIA, Seen September, 25th, 2013. "http://www.investopedia.com/terms/c/coasetheorem.asp".

L. Andrew Schlafly. 2007. "The Coase Theorem: The Greatest Economic Insight of the 20th Century", *Journal of American Physicians and Surgeons V 12 Number 2.*

X. Fan. 2007. "Power provisioning for a warehouse-sized computer". *In proc. Of the 34th Annua Intl. Symp. On Computer Architecture.*

D. Kusic. 2009. "Power and performances management of virtualized computing environments via lookahead control". *Cluster Computing.*

"Eucalyptus, Cloud Computing Platform, User's Guide, Enterprise Edition 2.0"; Eucalyptus Systems, Inc; 2010.

An Open-source Framework for Integrating Heterogeneous Resources in Private Clouds*

Julio Proaño, Carmen Carrión and Blanca Caminero
Albacete Research Institute of Informatics (I3A), University of Castilla-La Mancha, Albacete, Spain
julio.proano@alu.uclm.es, {mariablanca.caminero, carmen.carrion}@uclm.es

Keywords: Cloud Computing, FPGA, Hardware Acceleration, Energy Efficiency.

Abstract: Heterogeneous hardware acceleration architectures are becoming an important tool for achieving high performance in cloud computing environments. Both FPGAs and GPUs provide huge computing capabilities over large amounts of data. At the same time, energy efficiency is a big concern in nowadays computing systems. In this paper we propose a novel architecture aimed at integrating hardware accelerators into a Cloud Computing infrastructure, and making them available as a service. The proposal harnesses advances in FPGA dynamic reconfiguration and efficient virtualization technology to accelerate the execution of certain types of tasks. In particular, applications that can be described as Big Data would greatly benefit from the proposed architecture.

1 INTRODUCTION

The popularity of Cloud computing architectures is growing over the Information Technology (IT) services. Cloud computing is based on an on-demand approach to provide to the user different computing capabilities "as-a-service". Three Cloud usage models can be categorized in this computational paradigm, namely Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS), depending on the type of capability offered to the Cloud user which in turn is related to the degree of control over the virtual resource that support the application.These cloud models may be offered in a public, private or hybrid networks.

Nowadays, the main problem to infrastructure providers are concerned about the relationship between performance and cost. One of the solutions adopted has been adding specialized processors that can be used to speed up specific processing tasks. The most well-known accelerators are Field Programmable Gate Arrays (FPGAs) and Graphics Processors (GPUs) but paradoxically this use also directly related to the energy consumption of the system (Buyya et al., 2013), and cost.

Focusing on FPGAs, which have large resources of logic gates and RAM blocks to implement complex digital computations. This hardware devices are an order of magnitude faster for non-floating point operations than state of the art CPUs. Furthermore, FPGAs provides very good performance and power efficiency in processing integer, character, binary or fixed point data. And also deliver competitive performance for complex floating-point operations such as exponentials or logarithms. Additionally, the FPGA component can be reconfigured any number of times for new applications, making it possible to utilize the heterogeneous computer system for a wide range of tasks (Intel, 2013). In particular, many life science (LS) applications, such as genomics, generate immense data sets that require a vast amount of computing resources to be processed. FPGAs provide an intrinsic degree of parallelism amenable of being exploited by these applications. Moreover, LS applications change very quickly, thus making impractical the use of application specific integrated circuits (ASICs). To this end, the use of FPGAs seems a promising option. Finally, FPGAs exhibit a high computation/power consumption ratio, which turns them into an interesting option to lower operational expenses as well as minimize the carbon footprint of the data center.

Thus, the main objective of this work is to achieve an efficient low power architecture able to provide on-demand access to a heterogeneous cloud infrastructure, which can be especially exploited by big-data applications. FPGA-based accelerators are key computational resources in this cloud infrastructure. The proposal is aimed at enabling the use of FPGA-

*This work was supported by the Spanish Government under Grant TIN2012-38341-C04-04 and by Ecuadorian Government under the SENESCYT Scholarships Project.

based accelerators without dealing with the complexity of hardware design. Furthermore, the vision is to provide users with a repository of bitstream[2] files of common applications, which can be used as a service but also as building blocks to develop more complex applications. It must be noted that Amazon AWS already offers GPU instances (AWS, 2013), and FPGA-based accelerators are offered by Nimbix within its JARVICE service. But, to the best of our knowledge, this is the first open-source proposal of an IaaS architecture in which the FPGA accelerators (among others) are considered as computational virtual resources.

The rest of the paper is organized as follows. In Section 2 some related works are reviewed. Then, the architecture of our proposal is described in Section 3, while some implementation details are given in Section 4. Finally, Section 5 concludes this work, and outlines some guidelines for future work.

2 RELATED WORK

Hardware FPGA devices used as co-processors can offer significant improvement to many applications (Guneysu et al., 2008) (Dondo et al., 2013). So, there are several attempts to integrate FPGAs into traditional software environments. Many solutions are provided by the industry, such as PicoComputing, Convey and Xillybus. These products connect software to FPGAs via a proprietary interface with their own languages and development environments. The Mitrion-C Open Bio Project (Mitrion-C: The Open Bio Project, 2013) tries to accelerate key bioinformatics applications by porting critical parts to be run on FPGAs. Open source proposals such as (Jacobsen and Kastner, 2013) provide a framework that uses FPGAs as an acceleration platform. It must be highlighted that in this work FPGAs are connected to PCs through PCIe. However, these examples lack the framework for developing parallel and distributed applications on clusters with multiple nodes.

There are systems where FPGA devices are used as the only computing elements forming the cluster (Guneysu et al., 2008). However, not all applications can be accelerated effectively using FPGAs. For example, the high clock rate and large numbers of floating point units in GPUs make them good candidates for hardware accelerators.

The use of GPUs as computational resources on cloud infrastructures has emerged in the last years (Suneja et al., 2011). Hence, nowadays several commercial companies offer GPU service to their clients (Nvidia, 2013). For example, Amazon EC2 (AWS, 2013) supports general purpose GPU workflows via CUDA and OpenCL with NVIDIA Tesla Fermi GPUs. The use of FPGAs in cloud systems has not been so successful until now. At this point it must be highlighted that Nimbix (Nimbix, 2013) offers a commercial cloud processing system with a variety of accelerated platforms. Recently, the company has launched JARVICE, a PaaS platform that includes the availability of GPUs, DSP and FPGAs for an application catalog and an API or command-line access to submit jobs.

Moreover, there are some efforts that combine accelerators (FPGAs and GPUs) and CPUs in cluster nodes such as Axel (Tsoi and Luk, 2010). Axel combines heterogeneous nodes with the MapReduce programming model resulting in a low-cost high-performance computing platform. The authors in this work highlight that the largest drawback of FPGA design is the difficulty of implementation compared with CUDA programming time.

Also, recent researches focus on dynamically reconfigurable FPGAs. Reconfigurable FPGAs are devices where software logic can be reprogrammed to implement specific functionalities on tunable hardware. Hence, in (Dondo et al., 2013) a reconfiguration service is presented. This service is based on an efficient mechanism for managing the partial reconfiguration of FPGAs, so that a large reduction in partial reconfiguration time is obtained. Providing this kind of service is a must in order to attach FPGA resources in cloud infrastructures. Nevertheless, more in-depth research is required to provide a virtual machine monitor able to manage the reconfigurable capabilities on FPGA devices.

The closest research to our vision is the HARNESS (Hardware- and Network-Enhanced Software Systems for Cloud Computing) European FP7 project (HARNESS FP7 project, 2013). The objective of this project is to develop an enhanced PaaS with access to a variety of computational, communication, and storage resources. The application consists of a set of components that can have multiple implementations. Then, an application can be deployed in many different ways over the resources obtaining different cost, performance and usage characteristics. However, for the time being no framework proposal nor proof-of concept has been published yet.

[2]A *bitstream* is the data file to be loaded into a FPGA.

3 ARCHITECTURE PROPOSAL

In this section we describe an heterogeneous architecture for building on-demand infrastructure services on Clouds systems. The architecture attempts to address the challenges of deploying an infrastructure as a service (IaaS) on a private datacenter in which FPGA devices are considered as computational virtual resources. The virtual FPGAs are computing resources managed by the system together with the virtual machines running on CPUs. To be more precise, FPGA devices are used as accelerators for some codes to be run. In the remainder of this section, we describe the main characteristics of the proposal.

First of all, some arrangements must be fixed to orchestrate the physical computational components (CPUs, FPGAs and GPUs) of the proposed infrastructure. This proposal is based on uniform nodes composed of heterogeneous resources, as shown in Figure 1. Each node is composed of at least one CPU, and some extra COTS components can be included, which means that a FPGA on its own cannot be a compute node and every node must be provided with a CPU resource. The CPU of a node accesses and controls the task assigned to the accelerators. Therefore, a physical connection is required between the VM running on the CPU and the COTS.
Secondly, in order to support the heterogeneous cloud IaaS, some extra functionalities must be provided by the Virtual Machine Manager (VMM). The VMM is in charge of the deployment of VMs, and monitoring and controlling their state. Hence, since a mechanism is required to manage the heterogeneous architecture, our proposal makes use of a Hardware Accelerator Manager (HAM). HAM is an independent open-source component that makes use of the functionality of the VMM and extends it by providing the management of accelerators, namely, FPGAs and GPUs. When a user requests the use of the cloud infrastructure, HAM is in charge of deploying the cluster infrastructure, for example, for a Hadoop application (Apache, 2013), to run on it. This also includes the proper configuration of the accelerators so that they can be effectively exploited by the VMs. So, the system has different states which include the set-up state and the running one. To be more precise, in the mentioned case, during the set-up state, HAM deploys the master VM together with the worker VMs required to attend the request of the user on time.

As the integration of GPUs into a virtual infraestucture has already been addressed and even ofered by some Cloud providers (i.e.,Amazon), we will focus on the description on the integration of FPGAs.

3.1 Overall Description

In our system, we propose that the users use FPGAs to execute certain tasks as hardware acceleration modules integrating FPGAs as part of a private cloud. In general, the Hardware Acceleration Manager controls the communication and full processing of preparing and deploying hardware / software elements.

A general view of the proposed architecture is shown in Figure 1. Our system is composed of two elements, namely a **Hardware Accelerator Manager (HAM)** and a **Catalog**. On one hand, HAM is responsible of receiving requests from clients and deploying virtual hardware/software acceleration elements.

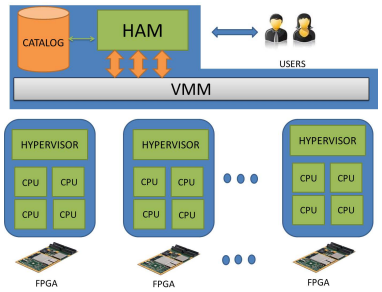


Figure 1: Arquitecture Overview.

HAM is supported by the Catalog, which consists of a set of files where all the meta-data and information about software, hardware, bitstream files, hosts, IP address, FPGA resources are stored, that is, the information stored in the Catalog consists of the state of the resources and the code of all the tasks loaded in the system up to now. Note that the repository of bitstreams is not fixed and can be increased over time. HAM will access this information when needed.

HAM is implemented as a software component which uses the functionalities of a Virtual Machine Manager (VMM) to monitor and deploy virtual infrastructures. It also harnesses the Intel VT-d technology support within the hypervisor to communicate hardware and software through the PCIe host port.

By using HAM, users will be able to execute jobs that can be partitioned into simple tasks, which can be implemented as hardware modules into an FPGA device, and other tasks which will be deployed in virtual CPUs. To illustrate the use of our architecture, let us assume that a user wants to run an application based on matrix-vector multiplication. Firstly, HAM processes the request from the user and tries to supply the virtual infrastructure required to run the matrix multiplication. To be more precise, HAM searches for efficient resources, including virtual machines with FPGA accelerators. Then, the system provides the user with the list of available resources, including the

corresponding FPGA IDs. HAM is responsible for preparing and deploying the virtual environment to process the request. Focusing on the FPGA accelerator, deploying the virtual infrastructure consists on providing access to the resource by using the Intel VT-d technology and loading the multiplication bitstream file in the FPGA. The multiplication bitstream file can be provided by the user or loaded from the Catalog. When the virtual infrastructure is successfully created, the system sends the data (matrix to be multiplied) and waits for the results. Finally, the results are sent back to the user. Also, the multiplication bitstream file can be stored in the Catalog so that it is available for future use.

HAM uses four modules (referred to as Controllers) to provide full deployment of virtual acceleration elements in the cloud environment. HAM Controllers will be described next.

4 IMPLEMENTATION DETAILS

In this section, we discuss the details of each part of the HAM architecture focusing on the integration of FPGAs as accelerators. The key software controllers in HAM are depicted in Figure 2, and are the following:

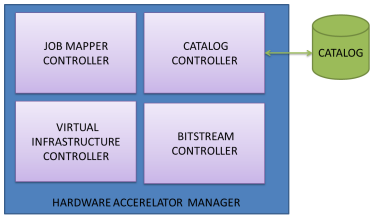


Figure 2: Hardware Accelerator Manager (HAM) Controllers.

- **The Catalog Controller (CC)** keeps the dataset of the **Catalog Module** updated.
- **The Bitstream Controller (BC)** is in charge of programming the FPGA, so that a given task can be executed on it.
- **The Virtual Infrastructure Controller (VIC)** deploys the virtual infrastructure required by an application.
- **The Job Mapper Controller (JMC)** interacts with the user and controls the execution of the tasks in the resource of the system.

When a request arrives, the Job Mapper Controller analyzes the state of the system by consulting the Catalog and supplies different options to the user. Then, the user will select the best computational resources

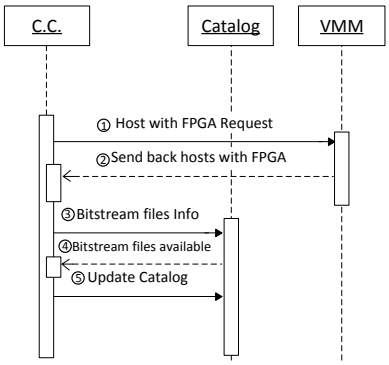


Figure 3: Catalog Controller.

suitable for running his/her application and the Virtual Infrastructure Controller will go ahead with the set-up of the infrastructure. This controller will be in charge of the deployment of the virtual resources and will interact with the Bitstream Controller each time an FPGA is required as computational resource. After the set-up, the tasks of the application will be executed on the provided resources. At the end of the execution, the Job Mapper Controller will collect the results and will send them to the user.

A more detailed description of the functionality and the interaction of the controllers with the environment (the Catalog, the VMM, FPGAs,...) is given next. Note that to get a clear explanation only one VM and one FPGA accelerator are considered in the following description, in spite of the fact that HAM is able to manage all the datacenter resources.

The Catalog Controller (CC) charge of monitoring available hardware/software resources and updating the Catalog. Figure 3 shows a sequence diagram illustrating the functionality of this controller. Requests to the Virtual Machine Manager (step 1) are used to collect all the information about FPGAs, bitstream files and Virtual Machines. This information is then collected and transferred to the Catalog (step 3). Also, the Catalog Controller is used to register new bitstream files corresponding to new computational resources being accelerated (step 5).

The Bitstream Controller (BC) is responsible for programming FPGAs with bitstream files, as shown Figure in 4. The mechanism used to program FPGAs consists of transferring the bitstream file to a host with an associated FPGA (which is known through a query to the Catalog, steps 1 and 2) via `ssh` (step 3). Each host with an associate FPGA contains a FPGA driver, which is used to program the FPGA device (step 4).

The Virtual Infrastructure Controller (VIC) is responsible for preparing and deploying the virtual environment which will be used to communicate the software (i.e., the application running in the virtual

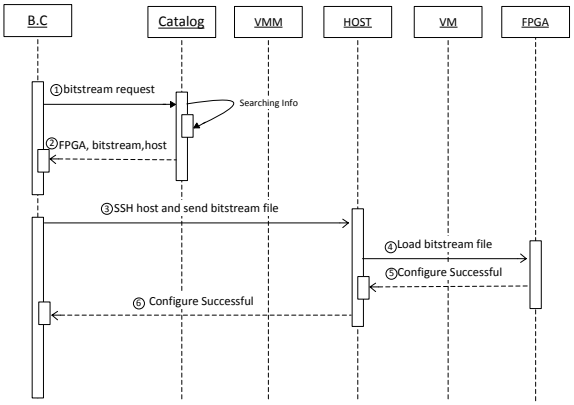


Figure 4: Bitstream Controller.

CPU) with the hardware accelerator. For example, for a Hadoop application, a cluster composed of several VMs will be deployed. Figure 5 depicts how this controller works. First of all, the Virtual Infrastructure Controller uses the Catalog Controller to collect all the necessary information to deploy a virtual environment (step 1). For example, a template which contains a definition of a Virtual Machine, the host name which contains the FPGA device associated, FPGA Id device and finally the PCIe port information.

The Virtual Machine Manager is used then to create a virtual machine (step 3) and then, when the virtual machine has been successfully created (step 5), the controller uses an "attach" functionality (steps 6 and 7) to create the access between the virtual machine (software) and the FPGA (hardware). Finally, the full system (composed of both the software and the hardware) is started up and ready to use (step 9).

It is important to clarify that the Intel VT-d technology (Intel, 2013) is used to communicate virtual machines and FPGA hardware because, it allows an important improvement on the performance achieved by accessing I/O devices in virtualized environment.

Finally, **The Job Mapper Controller (JMC** is responsible for the management of data sharing, monitoring the execution of jobs and sending files and reports to the clients. Additionally, it also makes use of an efficient scheduling algorithm to assign task to resources. This algorithm must have into account the requirements of the client applications, the available hardware accelerator resources and the state of the CPUs. The main characteristic that enable job acceleration is that data should be allowed to be partitioned among the available computing resources (i.e., CPU and FPGA) and be processed independently in order to improve performance. When the Job Mapper Controller (JMC) receives a job from a client, it adds the job to the global scheduler queue. Then, the Catalog Controller (CC) is consulted to discover the FPGA

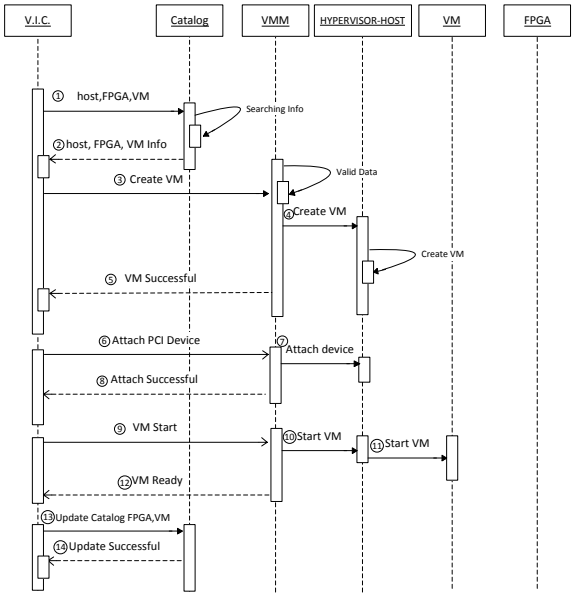


Figure 5: Virtual Infrastructure Controller.

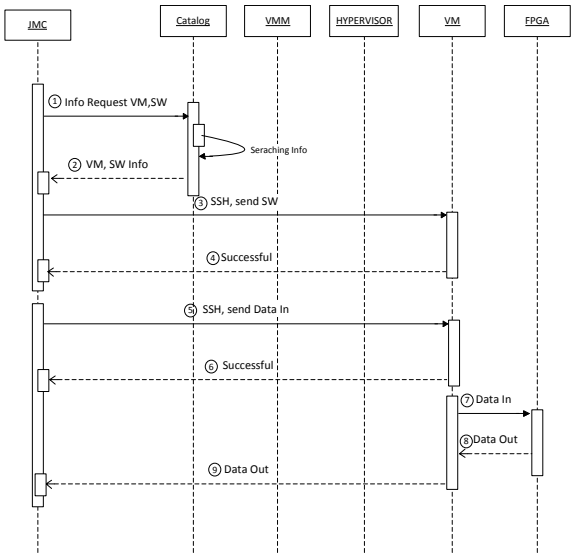


Figure 6: Job Mapper Controller.

resources available on where the data related to the job reside, to split the job into tasks, and to distribute these tasks according to the resources and data availability at each FPGA. These process is depicted in figure 6. The Catalog Controller (CC) is used by the Job Mapper Controller to retrieve information relative to previously deployed software application and virtual machine identification, which users can use (step 1). The Catalog sends back the necessary information identification for preparing software acceleration elements. Then, the job is transferred to the allocated Virtual Machine (step 3) through ssh. The application (bitstream file) programmed into the FPGA is de-

signed to receive a stream of data as ”input” as shown in (step 5) the (Data-In). Next, the FPGA processes data and, when the results are ready, the FPGA sends back a stream containing data results as ”output” (step 8). Finally, the JMC sends the results to the user.

5 CONCLUSIONS AND FUTURE WORK

We envision a heterogeneous cloud infrastructure architecture in which accelerators such as FPGAs and GPUs are used to reduce the power consumption of the datacenter. This will be possible when they get significantly easier to interact with. Hence, in this work an open-source IaaS architecture has been described focused on providing FPGA accelerators as easy-to-use virtual resources.

The future work will mainly consist on implementing the proposed architecture. As a matter of fact, a preliminary prototype built on top of the OpenNebula (Moreno-Vozmediano et al., 2012) Virtual Machine Manager is near to completion. Next, the focus will be placed on the problem of effective resource scheduling, addressing the issue of FPGA sharing among different virtual machines running in the same node.

Also, the feasibility of exploiting standard Single Root-I0 Virtualization (SR-IOV) (Intel LAN Access Division, 2011), which allows a PCIe device to appear as multiple virtual devices that can be accessed by a guest making use of VM passthrough will also be explored.

REFERENCES

Apache (Last access: 18th December, 2013). Hadoop. Available at: http://hadoop.apache.org/.

AWS (Last access: 10th December, 2013). Amazon EC2 Instances. Web page at http://aws.amazon.com/ec2/instance-types/.

Buyya, R., Vecchiola, C., and Selvi, S. T. (2013). *Mastering Cloud Computing: Foundations and Applications Programming*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition.

Dondo, J., Barba, J., Rincon, F., Moya, F., and Lopez, J. (2013). Dynamic objects: Supporting fast and easy run-time reconfiguration in FPGAs. *Journal of Systems Architecture - Embedded Systems Design*, 59(1):1–15.

Guneysu, T., Kasper, T., Novotny, M., Paar, C., and Rupp, A. (2008). Cryptanalysis with COPACABANA. 57:1498–1513.

HARNESS FP7 project (Last access: 12th December, 2013). HARNESS: Hardware- and Network-Enhanced Software Systems for Cloud Computing. Web page at http://www.harness-project.eu/.

Intel (2013). Intel Virtualization Technology of Directed I/O, Architecture Specification, Rev. 2.2. Available at http://www.intel.com.

Intel (Last access: 13rd December, 2013). Heterogeneous Computing in the Cloud: Crunching Big Data and Democratizing HPC Access for the Life Sciences. Web page at http://www.intel.com/.

Intel LAN Access Division (2011). PCI-SIG SR-IOV Primer: An Introduction to SR-IOV Technology. Available at http://www.intel.com/.

Jacobsen, M. and Kastner, R. (2013). RIFFA 2.0: A reusable integration framework for FPGA accelerators. In *23rd International Conference on Field programmable Logic and Applications, FPL 2013*, pages 1–8.

Mitrion-C: The Open Bio Project (Last access: 13rd May, 2013). Web page at http://mitc-openbio.sourceforge.net/.

Moreno-Vozmediano, R., Montero, R. S., and Llorente, I. M. (2012). IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures. *IEEE Computer*, 45.

Nimbix (Last access: 13rd May, 2013). Nimbix Accelerated Compute Cloud (NACC). Web page at http://www.nimbix.net/.

Nvidia (Last access: 14th December, 2013). GPU Cloud Computing . Available at: http://www.nvidia.com/object/gpu-cloud-computing-services.html.

Suneja, S., Baron, E., Lara, E., and Johnson, R. (2011). Accelerating the Cloud with Heterogeneous Computing. In *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'11, pages 23–23, Berkeley, CA, USA. USENIX Association.

Tsoi, K. H. and Luk, W. (2010). Axel: A Heterogeneous Cluster with FPGAs and GPUs. In *Eighteenth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'10)*.

A Green Model of Cloud Resources Provisioning

Meriem Azaiez[1], Walid Chainbi[1] and Hanen Chihi[2]
[1]*National Engineering School of Sousse, University of Sousse, Sousse, Tunisia*
[2]*Institute of Computer Sciences of Ariana, Ariana, Tunisia*
{*azaiez.myriam, walid.chainbi, hanen.chihi*}*@gmail.com*

Keywords: Cloud Computing, Optimization, Scheduling, Green Computing, CloudSim, Genetic Algorithm.

Abstract: The evolution of network technologies and their reliability on the one hand, and the spread of virtualization techniques on the other hand, have motivated the use of execution and storage resources allocated by distant providers. These resources may progress on demand. Cloud computing deals with such aspects. However, these resources are greedy in energy because they consume huge amounts of electrical energy, which affects the invoicing of Cloud services which depends on run-time and used resources. The environment is affected too due to the emission of greenhouse gas. Therefore, we need Green Cloud computing solutions that reduce the environmental impact. To overcome this Challenge, we study in this paper the relationship between Cloud infrastructure and energy consumption. Then, we present a genetic algorithm based solution that schedules Cloud resources and optimizes the energy consumption and CO_2 emissions of Cloud computing infrastructure based on geographical features of data centers. Unlike previous work, we propose to optimize the use of Cloud resources by scheduling dynamically the customers applications and therefore reduce energy consumption as well as the emission of CO_2. The optimal solution of scheduling is found using multi-objective genetic algorithm. In order to test our model, we extended the CloudSim simulator with a module implementing the dynamic scheduling of customers applications. The experiments show promising results related to the adoption of our model.

1 INTRODUCTION

Cloud computing is an emerging field which becomes increasingly popular. But, this technology is identified as one of the fastest growing consumers of energy. This consumption of energy will reach in 2020, more than three times compared to today (Relaxnews, 2010). This problem is caused by the energy consumed by data centers. Indeed, the data centers energy consumption increases with the number of centers and with data center workload. This consumption is amplified when the cooling infrastructure and auxiliary equipment are included which represents more than 50% of the power consumption (Zhang et al., 2012).

Another problem with energy is the emission of greenhouse gas which reached the 2% of the total amount of CO_2 emissions in the world (TUAL, 2013) and will quadruple in 2020 (Thrash, 2012). This problem brings a huge impact to the environment. Therefore, a new challenge appears to deal with the energy consumption and greenhouse gas emissions.

Many studies have addressed the green provisioning of resources to reduce energy consumption in Cloud environment. Most of these works use static allocation methods such as FCFS (First-Come-First-Serve) to ensure performance and quality of services (Calheiros et al., 2011). These strategies are very simple, but the problem with them is the need of many available resources. The energetic efficiency of these resource provisioning methods depends on the number of customers applications. Other methods used in the Cloud environment are deployed pre allocation strategies (Nair and Jayarekha, 2012). But the problem here is the prediction of required resources, which is difficult.

Unlike theses works, we propose to include green aspect in Cloud environment to support the Green Cloud computing. Indeed, the present study allows finding a solution to the green provisioning by scheduling customers applications to optimize the use of Cloud resources and therefore reduce energy consumption as well as the emission of CO_2.

We use the information of the Cloud infrastructure resources and their relations with energy consumption and CO_2 emissions. The main objectives are to min-

imize these two factors on the one hand, and to reduce the cost of services on the other hand. To optimize these objectives, we include our parameters in a multi-objective genetic algorithm and we execute the optimal solution in our Cloud which is tested as an extension of the CloudSim framework.

The remainder of this paper is organized as follows. Section 2 discusses related work. A detailed description of our solution is presented in section 3. Then in section 4, we provide technical details of our work. Section 5 deals with the experiments and the results of the simulations which are produced with comparisons and detailed analysis. Finally, the paper ends with brief conclusive remarks and discussion on future studies directions.

2 RELATED WORK

Cloud resources optimization is difficult to meet because of the uncertainty of future consumer demand and resource prices. It has been a topic of research interest and development for many years. To address the growing challenge, techniques from many disciplines were integrated synergistically. Next, we present the state of the art of cloud resources optimization methods.

Cloud computing's usage-based pricing model creates an incentive for subscribers to optimize the utilization of the rented resources. Borovskiy et al. (Borovskiy et al., 2011) devise a formal approach for distributing workload among a minimum number of servers. They model this problem as a linear programming problem and describe two solution approaches. The first one generates a set of candidate blocks and then composes an optimal partition by solving an integer programming problem. The second approach solves the set partitioning problem with column generation technique. The disadvantage of such method is its difficulty to consider the purpose of Cloud resources optimization because of the nonlinear characteristics of users' demands distribution.

Chaisiri et al. (Chaisiri et al., 2012) propose a method to optimize Cloud resources cost. The under provisioning problem can occur when the reserved resources are unable to fully meet users' demands due to its uncertainty of the workload distribution. To address this problem, the authors propose an optimal cloud resource provisioning algorithm based on stochastic programming model.

Regarding the problem of the description of the Cloud resources characteristic with nonlinear equation, some researchers propose the use of a stochastic optimization approach. For example, Li proposes a model based on stochastic integer programming for Cloud resources optimization (Li, 2012). He proposes to address the SLA-aware resource composition problem. He defines a stochastic integer programming model for resource composition and provides an algorithm that implements Grbner based theory to solve this problem (Buchberger, 2001).To solve the minimization problem of Cloud infrastructure cost, Zhao et al. developed a deterministic model for resource reservation planning, using a mixed integer linear algorithm, to generate optimal decisions given fixed parameters (Zhao et al., 2012). In addition, they proposed a stochastic model of resource rental planning which explicitly takes into account the uncertainty of resources and users' demand in the decision making process. One major disadvantage of such approaches, especially in dynamic environments where the optimal solution changes over time, is that the parameter estimation phase significantly delays the implementation of an optimal solution.

Other researchers use the constraint satisfaction problem (CSP) approach to solve the problem of Cloud resources optimization. Van et al. propose a two-level based architecture that defines a clear separation between application specific functions and a generic global decision level (Nguyen Van et al., 2009). They use utility functions to map the current state of each application for a scalar value that quantify the "satisfaction" of each application in terms of its performance targets. These utility functions are also means of communication with the layer of global decision which builds a global utility function including the costs of resource management. The stage of provisioning of virtual machines has been separated from the stage of placement of virtual machines within the global decision layer loop and formulates both problems as constraint satisfaction problem. Doughertya et al. propose a model driven approach to optimize the configuration, the energy consumption and the cost of infrastructure for Cloud infrastructure self-scaling to create green IT environments that reduce emissions resulting from the use of redundant resources unused (Dougherty et al., 2012), (Dougherty et al.,). They proposed to decompose the model to four sub-problems to ensure infrastructure auto-scaling: explaining how virtual machine configurations can be captured; describe how these models can be transformed in constraint satisfaction problems for the configuration and optimization of energy consumption; showing how optimal auto-scale configurations can be derived from these CSPs with a constraint solver and present a case study of energy consumption and cost reduction of production of this model-driven approach. The main drawback of these methods is

there exponential complexity.

Bio-inspired scheduling algorithms are often used in heterogeneous computing environments. Chaisiri et al. propose an optimal VM placement algorithm which optimally allocates VM to multiple cloud providers and follows optimally in advance the resources reservation (Chaisiri et al., 2009). This algorithm minimizes the cost virtual machines hosting in an environment of multiple cloud providers. Van et al. show that the ability to automate the provisioning and dynamic placement of virtual machines, taking into account both the software-level' SLA and resource cost with high-level handles for the monitor to specify compromise between the two (Van et al., 2009). The model defines a support for heterogeneous applications and workloads including both enterprise online applications with stringent QoS requirements and batch-oriented CPU intensive applications. It is not focused on optimization problems that are NP-hard in their general form.

Kessaci et al. (Kessaci et al., 2011) propose to optimize the allocation of VM requests using a Pareto-based meta-heuristic approach. In fact, they use a multi-objective genetic algorithm and propose to adopt new mutation and crossover strategies to produce new generations. The three objectives are considered in the optimization process: minimize both energy consumption and CO2 emissions of the cloud infrastructures and maximize the profit of the suppliers. To formulate the problem, they use a real encoding. Each individual is a vector representing the result of processing a pool of applications received during the scheduling cycle. The used encoding identifies three main features: the index of the vector represents the applications, the value of each cell identifies the VM on which the application will be scheduled and the maximum number of application. The Initialization of the MOGA population is divided into three steps: read the application pool with the greedy method (Black,), initializes one or two elements of the population by the result of the first step and randomly initializes the rest of the population. The problem of this approach is the use of the greedy algorithm to initialize the population of the MOGA algorithm. Despite their simplicity, greedy algorithms can be subtle and they are costly and mostly provide a local minimum. All of the presented approaches take into account the optimization of the Cloud resources but they do not consider the relationship between the satisfaction of users' requests and the optimization of providers' infrastructures cost. They do not pay attention on how each one of those parts can affect the other. In fact, the optimization of cost and response time of clients' requests are closely related to the number of available and active resources. Our objective is to minimize the number of the active resources that minimize energy consumption. To resolve this problem, we propose to use a multi-objective optimization. The initialization of the MOGA population is a real time process. It amounts to read the application pool and extracts useful information for the optimization algorithm.

3 THE PROPOSED APPROACH

Cloud computing can be represented in three main categories which are based on the capacity of abstraction and the paradigm of services. Thus we have the SaaS, PaaS and IaaS. In our work, we focus on IaaS as our goal is Cloud requests scheduling.

Iaas provides processing capacities and storage as well as network components as standardized services. These services manage a workload requested by customers applications.

3.1 The Mathematical Model

The Cloud model adopted by this study is IaaS with two-tier architecture as shown in figure 1. On the one hand, we have the Cloud services provider and on the other hand, the customer applications which need services.



Figure 1: The architecture of our Cloud model.

The optimization of our objectives, which are the energy consumption and the CO_2 emission, is owed to the exploitation of features offered by geographical distribution of data centers. Indeed, the parameters of energy model as well as CO_2 model are different from one geographical site to another.

In the Cloud, there are two sources of energy consumption: energy from computing equipment, which is the energy required for calculation and energy from auxiliary equipment, which is the energy required for cooling.

To make the energy equation, we use the model of CMOS (Complementary Metal-Oxide Semiconductor) processors by adding two constraints of the problem which are run time (t_{run}) and number of processors required to run the customer application (nb_{pr}). Thus, the final formula of the energy required for calculation is

$$E_{cal} = (\alpha f^3 + \beta) \times t_{run} \times nb_{pr} \quad (1)$$

The energy performance coefficient (COP) is the ratio between the produced heat and the energy consumed in the treatment. This factor leads us to deduct the cooling energy which is

$$E_{aux} = \frac{E_{cal}}{COP} \quad (2)$$

Therefore, the energy to minimize is the total energy:

$$E_{total} = E_{cal} + E_{aux} \quad (3)$$

Regarding the second objective, which is the emission of greenhouse gas, we use in its formula CO_2 rate, which depends on geographical locations. The formula of this second objective is

$$CO_2 = E_{total} \times Rate_{CO_2} \quad (4)$$

To optimize simultaneously these two objectives, we use the multi-objective optimization techniques (Goldberg, 1996).

3.2 The Multi-objective Optimization

A multi-objective optimization problem is to optimize several objective functions simultaneously. Our optimization problem is defined as follows:

Minimize $F(x) = (f_1(x), f_2(x))$ where both functions to optimize f_1 and f_2 are respectively the function of energy and the function of CO_2, $x = (x_1, , x_n)$ is the vector of decision variables which are the run time and the number of required processors, $F(x)$ is the vector of objectives which will be optimized.

The single optimal solution in the mono-objective optimization problem is replaced by the concept of Pareto optimal solutions in multi-objective optimization problem. Therefore, the optimal solution is not a single solution but a set of solutions. To find the right balance of solutions, it is necessary to identify the relation between these objectives. The most used relation is the relation of Pareto dominance. For this relation, all efficient solutions are called the Pareto Front. This set of solutions is a balance where no improvement can be made on an objective without degradation of at least another objective. So the purpose is to obtain the Pareto front or converge as much as possible on this front. Figure 2 shows an example of dominance relation in case there are two objectives to be maximized.

To solve our problem of multi-objective optimization based on Pareto-solving methods, we use the heuristic algorithms. These algorithms are used to explore the possible solutions space seeking the best solution. Among these algorithms, we use multi-objective genetic algorithm (MOGA).

Different models have been proposed for multi-objective genetic algorithms including VEGA (Vector

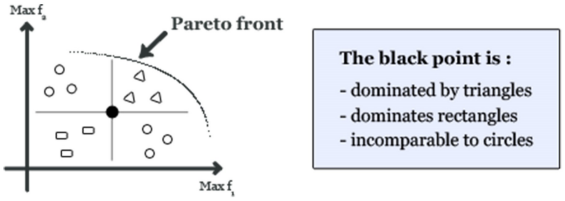


Figure 2: The relation of dominance.

Evaluated Genetic Algorithm), NPGA (Niched Pareto Genetic Algorithm), NSGA (Non Dominated Sorting Genetic Algorithm) etc. We adopt in our project, the NSGA-II because it uses an elitist approach that saves and injects the best solutions found in previous generations in the new generations (Melcher, 2007),(Deb et al., 2002). It uses a sorting procedure based on the non-dominance which is faster. It requires no parameter setting. It uses also a comparison operator based on a calculation of the Crowding distance. This distance is calculated from nearby solutions.

In our study, this algorithm makes the scheduling of the execution of customers applications with resource optimization. And to assess the effectiveness of the algorithm as well as the solution, we calculate the energy consumption and the emission of CO_2 after the execution of the applications.

4 SIMULATION ENVIRONMENT

4.1 CloudSim Extension

Since CloudSim provides an extensible framework for modeling and simulation of Cloud computing infrastructures and services, the present work is included in CloudSim as an extension. In fact, to create our application, we leverage some basic functions of CloudSim and we extend some characteristics and features.

In CloudSim, the management of Cloud resources and all allocation policies are standard and don‘t consider some characteristics of data centers. So, in the present work, we use this specification to investigate new model for resource allocation. This new technique is in accordance with ecological standards.

4.2 Class Diagram

Our application divided into two classes diagram:

The first class diagram is presented in figure 3. In this diagram we have three types of classes.

To meet the needs of our application, we add two classes. The first one is *Localisation*. This class contains all locations specifications of Datacenter. The

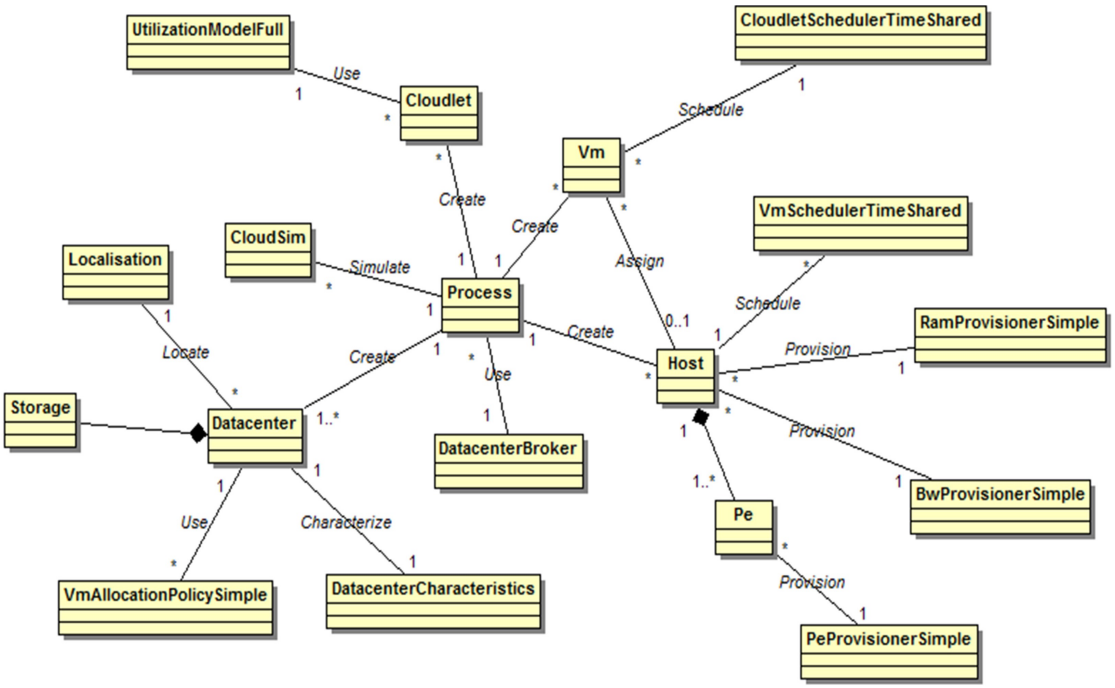


Figure 3: Project infrastructure class diagram.

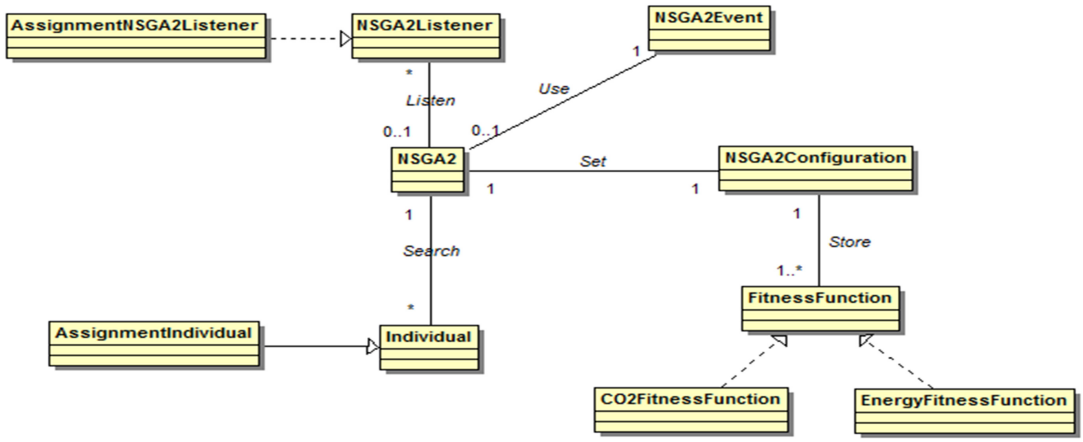


Figure 4: MOGA implementation class diagram.

second class is *Process*. This class is in charge of all communications between graphical interfaces, data access, Cloud configuration and simulation. Also, it allows to manage all Cloud components.

We modify the class *DatacenterBroker* to adapt it to our application. This class denotes a Cloud broker. It is a mediator between users requests and Cloud infrastructure. Since *DatacenterBroker* contains the process of creation of Cloud infrastructure and management of resource allocation policies, we change some details and we implement our new model in this class.

The other classes are those that we use in this work to create the environment of our new extension. The infrastructures of Cloud are represented by the class Datacenter which manages physical machines. Technical static properties of datacenters are in *DatacenterCharacteristics* class and desired functionalities of a storage system are in *Storage* Class.
The class *Host* represents a physical machine which has one or more processing element. Also, it has memory and bandwidth, managed by allocation policies, storage capacity and provisioning policy for assignment to one or more virtual machines. Each processing element of hosts, represented by *Pe* class, has a processing capacity. The allocation in virtual machines depends on physical characteristics of hosts.
User applications, represented by the class *Cloudlet*,

are deployed in virtual machines by provisioning policy *CloudletSchedulerTimeShared*. Virtual machines are distributed in Datacenters using the *VmAllocationPolicySimple* class.

The class diagram presented in figure 4, shows the classes used to implement MOGA namely NSGAII. The *DatacenterBroker* class from the first diagram is used for linking this group of classes to the other and is used also for running the algorithm.

In this diagram, we have 2 types of classes.

Classes of library for the NSGA-II algorithm that we use: the core of the algorithm is implemented in *NSGA2* class. In this class, we create an instance of *NSGA2Configuration* to specify and store all necessary parameters for genetic algorithm.
The instance of the class *NSGA2Event* is created in each generation during the run of the algorithm. In this class, we store information about the current status of the algorithm. We use this class in *AssignmentNSGA2Listener* to extract information and print it.

Classes that we use to personalize our algorithm: *AssignmentIndividual* class implements the class *Individual*. Its used to describe the populations of genetic algorithm. Each individual present a possible solution of the resource assignment problem.
For each fitness function, we create a specific class which implements *FitnessFunction* interface. Since we have 2 fitness functions, we implement 2 classes. The first one is *EnergyFitnessFunction*, which contains the energy function and the second is *CO2FitnessFunction* which contain CO_2 function.
To observe the performance of our genetic algorithm, we implement the *NSGA2Listener* interface using *AssignmentNSGA2Listener* class. This class shows fitness function values and other detailed data of the best individuals found during the run of the algorithm.

When the genetic algorithm finish, it returns the best found populations which are non-dominated solutions.

5 RESULTS ANALYSIS

In this section, we present the experiments and the evaluation that we undertook in order to study the efficiency of our extension of CloudSim in terms of Cloud computing environments optimization.
We deploy two sets of tests. In the first one, we decide on the value of MOGA parameters. Then, in the next test, we simulate and we analyze the Cloud environment by taking into account the extension of CloudSim and we compare the results with the initial existing approach in CloudSim.

5.1 Parameters of the Algorithm

Genetic algorithms have four parameters. To maximize the efficiency of our algorithm, we have to make a good choice of its parameters values.

Regarding the first parameter which is the size of the population, it is equal to the number of customers applications to optimize. By varying the values of stop criterion of the algorithm, which is the maximum number of generation, we noticed that the best individuals are always found before the 1000th generation. Accordingly, we consider this value as the maximal generation number and as sufficient to find the solution.

In theory, it was found that the values of the parameters of evolutionary algorithms vary in a specific interval. Indeed, former studies (Mais et al.,) have shown that the best results are achieved by a value of crossover probability between 0.45 and 0.95, and a value of mutation probability between 0.01 and 0.005. To fix these two parameters, we kept the same structure of the Clouds environment, the same resources and the same customers applications. Then, we vary the two remaining parameters of MOGA to choose the values that give the best results.

To test the effect of the different values of mutation probability on our Cloud environment, we assume that the value of crossover probability is 0.9 which is a predetermined value. Also for the test of crossover probability, we use the predetermined value of mutation probability which is 0.05.

The evaluation of tests show that the variation of probability values of the two parameters in the theoretical range gives almost stable results. Hence, we keep the predefined value of crossover probability and we choose the upper border of the interval of mutation probabilities. Consequently, values of 0.9 as crossovers probability and 0.01 as mutations probability may give a satisfactory result. We keep these values to simulate the different test cases.

5.2 Simulation Results

The purpose of this experiment is to discuss the performance analysis of our approach compared with static allocation method in terms of the efficiency of resource utilization for the same workload. The static allocation method used is the method deployed in the framework CloudSim before extension.

In these experiments, we calculate 2 metrics. The first one is the total energy consumption by the physical resources of data centers caused by customer applications workloads which is presented in figure 5.

The second one is the total emission of greenhouse gas caused by energy consumption of data centers which is shown in figure 6.

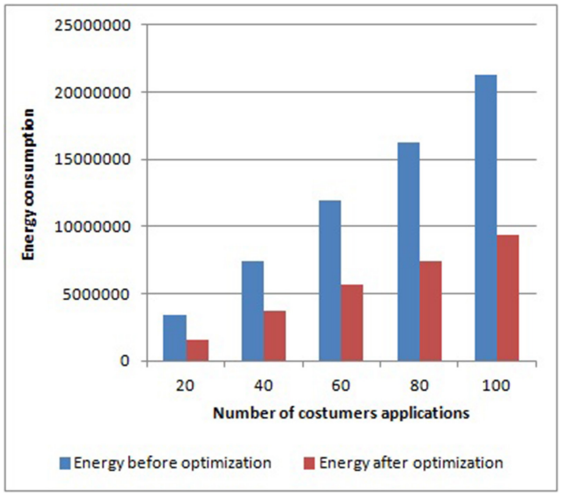


Figure 5: Energy consumption.

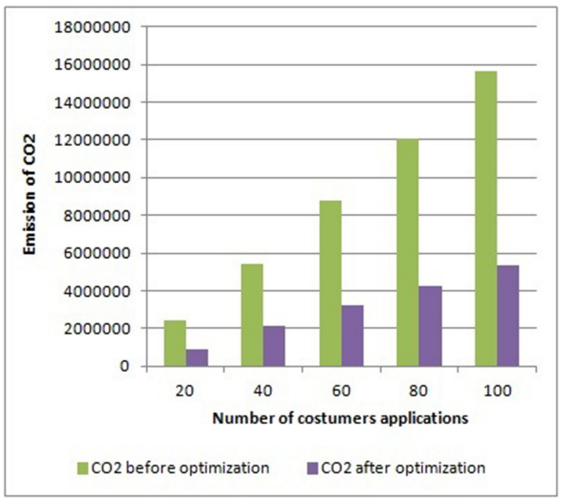


Figure 6: The emission of CO_2.

To demonstrate the amount of this optimization, we calculate the percentage of decrease of energy consumption as well as the emission of CO_2. The result of this demonstration is presented in figure 7.

The experimental results show that our proposed approach provides better results compared to results obtained by static method of resource allocation. Indeed, from these results we can conclude that for the use of our solution on Cloud environment, the energy consumption is reduced to 50% and CO_2 emissions up to 60%. This reduction depends strongly on the characteristics of available resources and the amount of applications which will be run on the Cloud. The obtained results are due to the efficient scheduling of customers applications, and to the reduction of the use of energy-intensive resources.

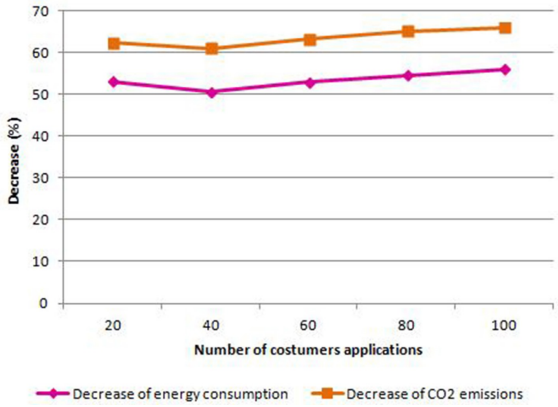


Figure 7: Percentage of decrease.

6 CONCLUSION

To address the problem of energy efficiency, we have proposed in this paper a new approach for a Cloud computing environment that schedule resources allocation based on energy optimization functions. More precisely, the presented work has optimized the resources in the Cloud, and has minimized the rate of energy consumption and CO_2 emissions by the management of Cloud computing resources and the effective choice of our genetic algorithm parameters. The experimental results have shown that the proposed approach leads to a significant reduction in energy consumption and CO_2 emissions in comparison with static techniques of resource allocation.

This study opens more challenges to reduce the impact of new technologies on the environment, encourage the ecosystems, and support energy efficiency.

In future work, we plan to integrate in our project, an agent system in order to make the Cloud environment auto-adaptive. A study is underway in order to fulfill this objective.

REFERENCES

Black, P. Greedy algorithm. dictionary of algorithms and data structures (online), us national institute of standards and technology, february 2005.

Borovskiy, V., Wust, J., Schwarz, C., Koch, W., and Zeier, A. (2011). A linear programming approach for optimizing workload distribution in a cloud. In *CLOUD COMPUTING 2011, The Second International Conference on Cloud Computing, GRIDs, and Virtualization*, pages 127–132.

Buchberger, B. (2001). Gröbner bases: A short introduction for systems theorists. In *Computer Aided Systems TheoryEUROCAST 2001*, pages 1–19. Springer.

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., and Buyya, R. (2011). Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50.

Chaisiri, S., Lee, B.-S., and Niyato, D. (2009). Optimal virtual machine placement across multiple cloud providers. In *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, pages 103–110. IEEE.

Chaisiri, S., Lee, B.-S., and Niyato, D. (2012). Optimization of resource provisioning cost in cloud computing. *Services Computing, IEEE Transactions on*, 5(2):164–177.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197.

Dougherty, B., White, J., and Schmidt, D. C. Model-driven configuration of cloud computing auto-scaling infrastructure.

Dougherty, B., White, J., and Schmidt, D. C. (2012). Model-driven auto-scaling of green cloud computing infrastructure. *Future Generation Computer Systems*, 28(2):371–378.

Goldberg, D. (19 janvier 1996). *Algorithmes génétiques: exploration, optimisation et apprentissage automatique*. Addison-Wesley France, SA, Kluwer Academic Publisher.

Kessaci, Y., Melab, N., Talbi, E.-G., et al. (2011). Optimisation multi-critère pour l'allocation de ressources sur clouds distribués avec prise en compte de l'énergie. In *Rencontres Scientifiques France Grilles 2011*.

Li, Q. (2012). Applying stochastic integer programming to optimization of resource scheduling in cloud computing. *Journal of Networks*, 7(7):1078–1084.

Mais, H.-R., Bloch, C., RAMDANE-CHERIF, W., and Chatonnay, P. Différentes opérateurs évolutionnaires de permutation: sélections, croisements et mutations.

Melcher, J. (11 September 2007). Jnsga2 1.1 - a free java library for the nsga-ii algorithm.

Nair, T. and Jayarekha, P. (2012). Pre-allocation strategies of computational resources in cloud computing using adaptive resonance theory-2. *arXiv preprint arXiv:1206.0419*.

Nguyen Van, H., Dang Tran, F., and Menaud, J.-M. (2009). Autonomic virtual resource management for service hosting platforms. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 1–8. IEEE Computer Society.

Relaxnews (02-04-2010). Data centers et cloud computing mettent trop de co2 selon greenpeace.

Thrash, B. (27-02-2012). The green data center opportunity.

TUAL, M. (14-04-2013). Data centers: la donne colo.

Van, H. N., Tran, F. D., and Menaud, J.-M. (2009). Sla-aware virtual resource management for cloud infrastructures. In *Computer and Information Technology, 2009. CIT'09. Ninth IEEE International Conference on*, volume 1, pages 357–362. IEEE.

Zhang, Y., Wang, Y., and Wang, X. (2012). Electricity bill capping for cloud-scale data centers that impact the power markets. In *Parallel Processing (ICPP), 2012 41st International Conference on*, pages 440–449. IEEE.

Zhao, H., Pan, M., Liu, X., Li, X., and Fang, Y. (2012). Optimal resource rental planning for elastic applications in cloud market. In *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 808–819. IEEE.

A Simple and Generic Interface for a Cloud Monitoring Service

Augusto Ciuffoletti
Dipartimento di Informatica — Università di Pisa, Pisa, Italy
augusto@di.unipi.it

Keywords: Cloud Monitoring, Open Cloud Computing Interface (OCCI), RESTful Service, Cloud Ontology, Cloud Interoperability.

Abstract: The paper addresses the definition of an ontology for cloud monitoring activities, with the aim of defining a standard interface for their configuration. To be widely adopted, such ontology must be extremely flexible, coping with a wide range of use cases: from the minimalist plug-and-play user, to the one governing a complex infrastructure.
Our work is based on the Open Cloud Computing Interface, that is an open, community driven OGF standard allowing boundary-level interfaces to be built using RESTful patterns over HTTP. Among others, OpenStack and OpenNebula adopt OCCI.
Using the OCCI ontology we define two *kinds* that are associated with the basic components of a monitoring infrastructure: the *collector link*, that performs measurements, and the *sensor resource*, that aggregates data and undertakes actions.
This paper is a compact and self-contained revision of a document currently under discussion inside the OCCI community.

1 INTRODUCTION

In the current terminology a Cloud is a collection of *resources* that are dynamically provisioned to a user: see for instance the NIST definition in (Mell and Grance, 2011). Whenever the task such resources are involved in becomes critical, for any of the many reasons and ways a computing task may become critical, the user wants to evaluate its performance: the NIST definition referenced above is explicit about this fact, and indeed a growing attention is paid to the topic of *cloud monitoring*.

The purpose of cloud monitoring is to extract quantitative performance evaluations of cloud resources in the form of *metrics*. During the operation, cloud monitoring produces *measurements*, that are instances of given metrics.

The measurements produced during monitoring are used for several reasons. Among them:

- Fault Treatment — historically this is the main purpose for resource monitoring. However a Cloud Computing environment gives a different perspective, since hardware failures now are softly coped with by the cloud provider (VMware, 2007). Fault treatment is still needed for marginal conditions that damage the application while being admitted by the Service Level Agreement (SLA) stipulated with the provider;
- Billing — the provider expects a revenue that is related with the amount of resources consumed by the user. A billing rule that is adherent to the effective use, and not to the reserved resources, is more effective from both the user and the provider side: the user pays only for what it uses, and the provider has further optimization margins;
- Service Level Agreement Verification — the user that signs a service level agreement with the provider wants to be able to verify that the service meets the terms of the contract. The provider that meets this demand increases its credibility;
- Quality of Service Implementation — the user that implements a service in the cloud may want to ensure a given QoS whose terms are different from the SLA signed with the cloud provider: the availability of resource performance metrics enables the user to undertake actions to ensure the QoS.

The above non-exhaustive taxonomy justifies the claim that the monitoring activity deeply depends on user needs: the provider is in charge of making available the resource measurement tools, but the user is

left the final word about their configuration to meet specific use cases.

This paper aims at the definition of an ontology for the domain of cloud monitoring, thus defining the contents crossing the interface between the provider and the user, in view of the definition of a widely accepted standard.

In the next section we give a taxonomy of use cases, and then we explore the monitoring tools currently offered by a selection of cloud providers.

2 AN ONTOLOGY FOR CLOUD MONITORING

The monitoring of cloud resources is a complex task, and many entities contribute to its unfolding. It is therefore appropriate to identify a limited set of fundamental concepts — an ontology — that helps the coordination among the involved parties. This approach has been already adopted in cloud computing (Youseff et al., 2008; Di Modica et al., 2012; Bernstein and Vij, 2010), and it is tightly related with interoperability and service discovery.

We first observe that the final user of measurements does not always identify with the user of the monitored infrastructure. We distinguish three alternatives:

- the user of resource measurements is the cloud provider. In this case monitoring is directed either to billing purposes using raw usage metrics on real hardware resources, or to implement a Service Level Agreement,
- the user of resource measurements is the cloud service end-user. In this case the user is mostly interested in checking Service Level Agreement Compliance, or in the implementation of application-specific resource optimization,
- the user of resource measurements is a user that, in his turn, provides critical services based on resources leased by another provider. The case is similar to the above, but with a number of challenging variants since the measurements may come from dynamically provisioned virtual resources from both a private and public clouds.

The way raw measurements are processed before being delivered introduces further variance in the scenario: resource performance is usually expressed as a multi-dimensional metric, while an aggregate unidimensional metric, tailored for a specific application, is more appealing for the user. For business related reasons, also the provider prefers to expose an aggregated metric, instead of a fine grain set of raw metrics.

An example is the measurement of network performance in an IaaS cloud, since performance measurements should be dynamically configured on demand, depending on a changing user activity: the provider has to figure out the filters to be applied to networking devices so to select and measure relevant traffic characteristics.

The *description* of resource monitoring capabilities becomes relevant when it is considered as a part of the provided service, considering the monitoring infrastructure itself as a leased resource. Its description should cross the same interface used to convey the description of the service.

There are currently a lot of such interfaces, which is one of the problems a potential cloud user finds on his way. Cloud providers want to make their services more attractive, and thus they design straightforward interfaces for their specific provision. Such an approach tends to fragment the access to cloud services into a number of narrow, vendor-specific interfaces: let us explore the offer.

2.1 Cloud Monitoring in the Market

The majority of cloud providers offer resource monitoring as part of the provided service: at a point in time, the user may decide to subscribe to an additional service that reports measurements or that undertakes actions based on the monitoring activity.

The AWS monitoring service is named **CloudWatch**. It is available for all of the AWS resources, like compute resources, various forms of database and storage facilities, and other infrastructure components like queues and load balancers. In total, more than one hundred metrics are available. The typical frequency is of one measurement every 5 minutes, but periods as low as 1 minute are allowed. One notable fact is that, among the available metrics, there is the financial cost of the cloud provision. The measurements may be directly used to trigger compensating actions in a closed control loop, or they may be presented for human inspection. To improve flexibility, AWS allows the user to perform custom measurements, and to submit them as an input to CloudWatch, for instance to shut down an unused resource.

AzurWatch solution has a typical period of one hour, and around twenty metrics are offered on the various components of a cloud provisioning, that are either used as a trigger for elastic behaviors, or directly presented.

RackSpace approach consists of providing a software package that implements a *Monitoring Agent*. Once installed on a compute instance in the cloud, it is able of producing measurements from that server.

The user controls which measurements are collected and how they are used through a management page, which is embedded in the cloud management interface. In addition, measurements feed automatic actuators represented with *checks*: for instance, one check consists in sending a series of ping to an IP address and reporting problems by email.

Private cloud platforms may live without a structured way to deal with monitoring: for instance, the open cloud computing platform **OpenStack** does not provide a structured solution, but its *open source* nature lets the user free to implement a custom solution.

In contrast, **OpenNebula** offers both a *nuts and bots* solution based on software probes that *push* measurements to the cloud monitoring front-end, and a closed loop mechanism, that implements automatic scaling decisions based on measurements. OpenNebula offers also a less scalable *pull* mechanism based on the same software probes.

A totally different approach consists of the provision of a stand-alone monitoring service. For instance, **StackDriver** provides an agent that is installed on the user premises to monitor cloud resources: it is designed to inter-operate with the monitoring services offered by AWS and RackSpace clouds. The Stackdriver dashboard integrates the know-how that helps the user to design an effective analysis of the measurements, thus obtaining sensitive and reliable alarms.

CompatibleOne is a project that follows a similar approach, and aims at providing the widest possible compatibility with existing cloud computing platforms: a unique tool to interact with all major providers. Therefore they embed in an interface the tools to describe a monitoring framework, that envisions the presence of a specialized *monitoring agent* coordinating the monitoring *probes* activity.

After this partial overview, we understand that there are a lot of ways to provide a cloud monitoring service, which is regarded as a positive symptom of vitality, and we expect new and innovative ways to emerge. An attempt to contain them within a rigid standard would be a mistake, ultimately unsuccessful. On the other hand, the existence of an agreement about a few relevant aspects allows interoperability, with all of the advantages that we learned from the history of the Internet.

The challenge is in finding the balance between what is specified in the standard, and what is left unspecified, so that a new service can be both innovative and complying.

2.2 The OCCI Way to Cloud Standards

The OCCI approach to the definition of a standard for cloud computing (Edmons et al., 2012) consists in defining the interface to the service, giving as little detail as possible to the definition of the service itself. The interface is defined in a way that is easily implemented with tools that, in their turn, are firmly based on widely used and accepted standards.

The document that describes the core of the OCCI standard (OGF, 2011a) lays the foundations, binding the cloud management interface to be RESTful, according with the principles defined in (Fielding and Taylor, 2002). This gives an HTTP framework to the interface between the cloud user and the provider, and restricts the interaction between the two so to preserve the advantages of the HTTP infrastructure (like proxying, caching etc.). According to the REST paradigm, the *resource* with its representation are subject of the communication between a *client* and the *server*.

Besides the core standard, designed to be long-living as valuable standards need to be, there is a series of satellite documents, that enrich the core with the details of a specific interface and provisioning. One series of standards specifies the language used to represent entities, i.e. its *rendering*. At this time, there are three such renderings: embedded as HTTP attributes, using the JSON language and, the last to appear and not yet fully specified, using XML. Another series of standards is bound to a specific kind of cloud provisioning: at this time there is one such document that focuses on an IaaS provisioning, defining entities like compute and storage resources.

The proposal addressed in this document is in the spirit of defining resource monitoring as just another kind of cloud service provisioning[1]. Since we use the OCCI ontology, we summarize the features of the OCCI API scheme before going into the details of our proposal.

3 OCCI — AN OPEN CLOUD COMPUTING INTERFACE

The OCCI working group of the Open Grid Forum (OGF) has produced the description of an interface for the description of a Cloud Computing infrastructure. It representes the front gate of a cloud provider: the users willing to obtain resources from a provider

[1]The concepts exposed in this paper are the basis of a more technical paper, that can be found browsing the repository of the OCCI project at http://redmine.ogf.org, in git branch named monitoring

need to submit a request that follows a given protocol. The intent of the OCCI working group is to foster the convergence towards a widely adopted standard.

Both users and providers have an interest in the existence of a standard, since its introduction usually carries a development in the market as well as technology advances. The ideal place for its development is not a private enterprise, since it is subject to pressures from the market. A better result is expected from an independent forum, where the suggestions coming from industries may meet with the results of independent research: the OGF, together with other standardization bodies, played an important role on this respect, and the OCCI working group is a branch of it.

The OCCI interface is based on a server that operates following the Representational State Tranfer (REST) paradigm. According with it, the client and server exchange messages that contain the description of *resources*: in this context, the *resource* is any concept that admits a formal representation, and its is not directly related with a computing *resource*, whose monitoring is the matter of our investigation.

In a REST framework (Fielding and Taylor, 2002), the client and the server exchange *request* and *response* messages that contain representations of the state of REST resources. In this respect the REST framework defends a coherent utilization of the HTTP protocol, against new communication tools like WebSockets, by introducing four fundamentals constraints:

- the communication between the client and the server uses a uniform interface carrying the representation of resources
- there is no session-related state in the server, and each request-response pair is a self-contained operation
- the existence of an intermediate processing of the messages (like caching) is trasparent
- the client may provide code to extend server capabilities

The request message contains the indication of an operation to be performed on the resource: the REST paradigm indicates four operations that correspond to the well known HTTP verbs: GET, PUT, POST and DELETE.

The OCCI interface uses a REST interface to describe the interaction between the user and the service provider aimed at the specification of the infrastructure the user wants to obtain from the server. A great deal of attention is paid to the structure of the information that may be included in the message, and that describes the operation of infrastructure resources.

According with OCCI proposal, the representation of a cloud infrastructure is carried out by describing REST resources represented as instances of an *entity* type. An *entity* instance is characterized by a unique identifier, but it is otherwise left abstract: it needs to be related with a *kind* and to one or more *mixin*s in order to be fully specified. The *kind* gives an entity its basic features, described as *attributes*. *Kind*s are arranged in a tree structure, where each *kind* is put into relationship with another higher level one, in a sub-typing hierarchy.

The OCCI working group has defined two core *kind*s: the *resource*, here intended as an IT resource[2], and the *link*, that represents a relationship between *resources*. Each core type can be sub-typed in its turn to take into account the multitude of IT resources and their relationships, thus generating the *kind*s hierarchy.

The association of a *mixin* to an *entity* instance corresponds to a further characterization of it. A *mixin* can be used to bind attributes already defined by the *kind*, or to introduce new attributes, like a root filesystem with a preconfigured OS. *Mixin*s are related with a many-to-many relationship: in particolar, a *mixin* can be defined as a *tag* with an associated semantic, but no attributes. The UML class diagram that describes the OCCI model is in figure 1, and its exhaustive definition is in (OGF, 2011a).

The core ontology is in fact more general than what strictly needed for the definition of an IaaS cloud. In a distinct document (OGF, 2011b) the OCCI working group defines a specialization of the core model that addresses such task. In a nutshell, three sub-kinds of the *resource* kind are defined that model IaaS resources (Compute, Network and Storage), and two sub-kinds of the *Link* kind to describe relationships among them (NetworkInterface and StorageLink).

In the end, the task of describing a cloud infrastructure is carried out in a natural with the instantiation of a number of *Compute*, *Storage* and *Network resources*, configuring them with appropriate *attribute* values, and interconnecting them using *StorageLink*s and *NetworkInterface*s links. Appropriate mixins are associated to those *entities* that need further specification.

Note that the hierarchical nature of *kind*s and *mixin*s allows the user to discover the capabilities of a provider: for instance, the availability of a Celeron

[2]in this document the term *resource* is used in two quite different but close meanings: one is the content addressed with an URI, as defined in sec. 1.1 of (Berners-Lee et al., 2005), the other is the representation of a cloud resource. To disambiguate we use the term *REST resource* for the former

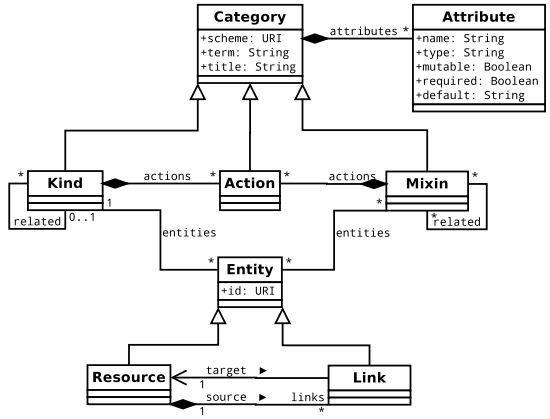


Figure 1: The OCCI core model – A UML class diagram Copyright © Open Grid Forum (2009-2011). All Rights Reserved.

CPU might be discovered browsing the *mixins* that can be associated with a *Compute* entity. In fact, a provider may define provider-specific *mixin*s, thus leaving the overall structure open to unlimited extension.

All operations involved in the instantiation of *Entities* in the above scheme are carried out using the basic HTML verbs: once a system is populated with *Entities*, it can be browsed using associations. For instance, the discovery of available resources is implemented with a `GET /-/`, while the instantiation of new *entities* is obtained with a `POST`. The descriptions of the available mixins is learned using the above discovery mechanism, and mixin instances are associated to *Entity* instances using `POST` requests.

The next step is to introduce an OCCI extension to describe a monitoring infrastructure.

4 DEFINITION OF A MONITORING INFRASTRUCTURE USING OCCI

In our view, a monitoring infrastructure can be defined as a service using an extension of the OCCI *core* model: this approach follows the one used to define an IaaS in the OGF document (OGF, 2011b). The adherence to the OCCI model is deemed appropriate, since that standard has the favorable properties of being open, and of being designed to be long-living. There is no need to introduce a dependency from the IaaS *entities* defined in (OGF, 2011b), and so our monitoring framework may extend to other computational models, different from a plain IaaS.

In a simplistic view, a monitoring capability might be associated to a generic *resource* using a *mixin* describing the available metrics, in analogy with the OpenNebula *pull* mechanism. However this scheme is able to capture only very simple cases, and it is unable to model a metric that aggregates several raw metrics, like the average load of a pool of servers. In addition, the inclusion of measurements in the rendering of a resource introduces scalability issues, since such attributes are frequently updated, and devoids caching: it is definitely in contrast with the RESTful paradigm.

Instead, our option is to introduce a specialized *resource* kind, that we call *sensor*, that embeds the capabilities of processing and delivering the measurements: it is not directly involved with their production. It can be approached to the *monitoring agent* as introduced by CompatibleOne and Stackdriver, and that we successfully adopted in a Grid-oriented research prototype (Ciuffoletti et al., 2003; Ciuffoletti and Polychronakis, 2007). A single sensor is able to collect measurements from many sources, and to deliver metrics obtained processing the inputs in many ways.

The user defines across the interface the timing of the *sensor resource*, and operational details are defined using provider specific mixins: the definition of the available monitoring capabilities is thus left to the provider. This is appropriate since the provider wants to define monitoring capabilities according with its own business strategy: as shown in the introductory overview, each provider has a distinguished one.

We consider that a representative ontology must be powerful for the demanding user, and simple for minimal tasks: on one end we want a simple metric (in the values {green, yellow, red}) for a large infrastructure moving the complexity in the aggregation of raw metrics (in Rackdriver style), on the other we want a raw metric on a single resource (Mb/s on a network interface) (in RackSpace style). A truly generic ontology must capture both use cases, as well as all those in between. So we define a designated *resource* whose task is to represent and coordinate the monitoring activity.

The next step is the description of the relationships between the *sensor* and the resources from which it receives measurements: it is a way to associate a metric to a (resource,sensor) pair. The *link* entity is the tool offered by the OCCI core model for this case. The metric is thus attached to a *link* between a *sensor* and a generic *resource*: we call *collector* the kind of this *link*. This solution has the favorable property of making discoverable the association between sensors and monitored resources. It can be approached with Rackspace and CompatibleOne *probe*.

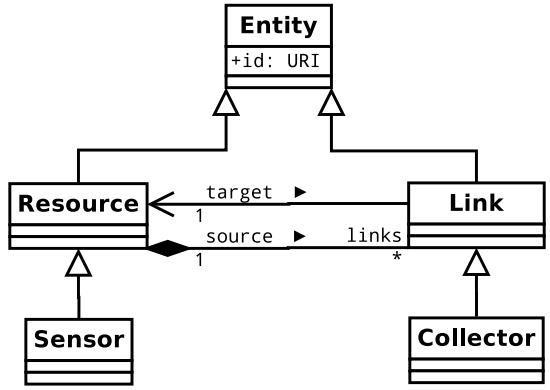


Figure 2: A model for monitoring *entities* – UML class diagram.

Like in the case of the sensor, the generic interface transfers only the timing aspects of the collector, and leaves to provider-specific *mixins* the work of giving the operational definition of the collector.

The introduction of the *sensor* and of the *collector* subtypes can be described in the UML diagram that is shown in figure 2.

In conclusion, to introduce a monitoring service in an existing infrastructure we proceed as follows:

1. we attach an instance of a *collector* link to each of the monitored resources, and define the collected metrics by way of *mixins*;
2. we define a *sensor* that receives the measurements from the collectors and associate processing and publishing funcionalities by way of *mixins*;

The approach defined above does not allow a plain *pull* mechanism, consisting in a direct access of metric attributes inside the *resource* description. Such option has an heavy footprint on the provider, and it is not RESTful compliant: so we consider this restriction as justified.

With some effort, it is possible to treat very simple use cases by unifying the monitored resource and the *sensor*, so that the two functionalities live together inside the same resouce: this is near to the RackSpace solution. However, the same solution can be represented in a cleaner way using two distinct entities.

5 DESCRIPTION OF REST RESOURCES FOR CLOUD MONITORING

This section is devoted to the description of the two abstract REST resources in our ontology: the *sensor resource* and the *collector link*.

Table 1: Definition of the *Sensor Resource* Kind.

Model attribute	**value**			
scheme	http://schemas.ogf.org/occi/monitoring#			
term	sensor			
title	*Sensor Resource*			
attributes	**name**	**type**	**mut.**	**req.**
	occi.sensor.period	number	true	true
	occi.sensor.periodspec	string	true	false
	occi.sensor.timebase	number	false	true
	occi.sensor.timestart	number	true	true
	occi.sensor.timestop	number	true	true
	occi.sensor.timespec	string	true	false

A *sensor resource* defines the timing of the measurements (see table 1): how frequently measurements are collected, during which time period, the accuracy and the granularity of the time scale. Any further specification is left to mixins that describe the relevant aspects of the sensor, namely:

- the way input metrics are aggregated to produce an output metric and
- how output metrics are published

To clarify the semantics associated to the two types of *mixin*, we describe the role of their attributes.

In the case of the *mixins* that define the aggregation function, we have *input attributes* that bind the measurements provided by one of the ingress collectors to variables in the aggregation algorithm, and *output attributes* that bind the results of the aggregation to a mixin that defines the way they are published. Other attributes indicate the value to assign to constant parameters used in the aggregation algorithm.

In the case of *mixins* that describe a publishing method, we have *input attributes* that bind the metrics computed locally or coming from ingress collectors to items in the communication protocol. The binding between *input* and *output* attributes is implemented with labels, that are assigned as values to input and output attributes. The scope of such labels is limited to the sensor and to the set of input collectors. Example of publishing mixins are those that deliver the measurements through a graphical interface, but also those that store the data in a database, or that trigger *cloud management* activities in a *closed loop* scheme.

Table 2: Definition of the *Collector Link* Kind.

Model attribute	**value**			
scheme	http://schemas.ogf.org/occi/monitoring#			
term	collector			
title	*Collector Link*			
attributes	**name**	**type**	**mut.**	**req.**
	occi.collector.period	number	true	true
	occi.collector.periodspec	string	true	false

A *collector link* (see table 2) defines a relationship between a *sensor resource* and another generic cloud *resource*. Like the *sensor resource*, also the *collector link* has attributes that define the timing of the measurement activity. The operation of a *collector* instance is further defined by *mixin*s that indicate the collected metrics. Such *mixins* are characterized by a *title* that is reminiscent of the measured metric and of the methodology used to perform the measurement. Other parameters control the application of the specific methodology (e.g, the length of a ping packet), and *output attributes* convey the measurements to the sensor.

Table 3: The *Mixin* tags defined for the monitoring API.

Model attribute	value
scheme	http://schemas.ogf.org/occi/monitoring/collector#
term	metric
scheme	http://schemas.ogf.org/occi/monitoring/sensor#
term	aggregator
scheme	http://schemas.ogf.org/occi/monitoring/sensor#
term	publisher

As seen, *mixins* for resource monitoring fall into three categories: those that describe how metrics are aggregated, those that indicate how they are published and those that stand for the metrics themselves. To give a representation of this fact, we introduce three tagging *mixins* that are used to apply structural constraints that are specific for each set of mixins. The three tags are defined in figure 3.

These *mixins* play a fundamental role, since their semantics define the monitoring that is introduced in the system. To leave the provider free to define a business strategy for the monitoring service, they are left defined as not fully qualified entities. In this way the applicability of the proposed ontology is not limited to the compute/storage/network triad: for instance, the scheme described above can be easely adapted to a PaaS infrastructure, where the monitored resource is a NoSQL repository and the metric is the number of queries per second.

Finally, we see how to represent a simple monitoring activity that consists of an alarm that sends an email when the load of a CPU is steadily above a threshold, using a filter to avoid false alarms.

6 A SIMPLE CASE: ALARM FOR AN OVERLOADED CPU

The user wants to create an alarm that sends an email to a given address when the load of a compute resource (id=`urn:uuid:1111`) is steadily over 80%.

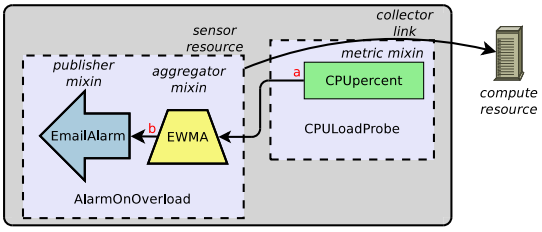


Figure 3: Block representation of a CPU overload alarm.

The check is run evey minute starting ten minutes from now, and it lasts during one hour.

The block diagram of the alarm is shown in figure 3, and we now explain how the user can incrementally implement it using the API defined in this paper.

The first step in the workflow is the instantiation of a blank sensor with the following attributes (optional attributes are omitted):

name	value
title	”AlarmOnOverload”
period	”60”
timebase	”1386925386”
timestart	”600”
timestop	”3600”

`AlarmOnOverload` sensor attributes

The server returns an id=`urn:uuid:2222` for the sensor. The next step is the instantiation of the *collector link*, with timing attributes consistent with those of the sensor:

name	value
title	”CPULoadProbe”
source	`urn:uuid:2222`
target	`urn:uuid:1111`
period	”60”

`CPULoadProbe` collector attributes

Now the user agent browses the capabilities of the provider looking for one that measures the CPU load, and finds the `CPUpercent` one. The mixin is added to the `CPULoadProbe` with a POST carrying the added attribute:

name	value
out	a

`CPUpercent` metric attributes

indicating a label attached to the output stream.

Next the user explores provider’s capabilities looking for a robust average, an it finds the exponentially weighted moving average inplemented by the `EWMA` mixin: three attributes are indicated, respec-

tively for the gain, the input and the output stream. The mixin is added to the `AlarmOnOverload` sensor with a POST and the attributes are given a value:

name	value
gain	”16”
instream	a
outstream	b

`EWMA` aggregator attributes

The same is for the `EmailAlarm` mixin, that has threshold, input and email attributes. It is posted to the `AlarmOnOverload` sensor with:

name	value
threshold	“80”
email	”myself@example.com”
input	b

`EmailAlarm` collector attributes

The operation, here split in a series of 5 POST operations, can be aggregated in a unique POST, using a suitable syntax defined for OCCI resources.

The user interface can be simplified by defining a template that embeds the whole structure, thus allowing the presence of a single button on an hypothetical graphic user interface. In the same spirit, the email alarm might be replaced with a WebSocket (Fette and Melnikov, 2011) connection driving a green/red light on the graphic user interface.

7 CONCLUSIONS

One conclusion of this work is about the OCCI ontology in itself. The fact that the *core* model can be used for a purpose distant from the original one (that was the description of IaaS provisions) confirms that it is a good standard, potentially stable in time. This is relevant as a guarantee of the return of an investment in compliance.

Monitoring has recently emerged as a relevant aspect of a cloud provisioning. This reflects on the interest for an extension of the OCCI *core* to express a monitoring infrastructure, which is the subject of this paper. Being based on the *core* model, it is potentially extensible to applications beyond the current horizon.

In this paper we have validated our ontology with respect to a number of providers that offer a cloud monitoring, showing that their services can be described using our model. By the time this paper will be published, an formal XML description of the model will be available, and we consider the implementation of a proof of concept prototype.

REFERENCES

Berners-Lee, T., Fielding, R., and Masinter, L. (2005). Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (INTERNET STANDARD). Updated by RFC 6874.

Bernstein, D. and Vij, D. (2010). Using semantic web ontology for intercloud director ies and exchanges. In *International Conference on Internet Computing (ICOMP)*.

Ciuffoletti, A., Ferrari, T., Ghiselli, A., and Vistoli, C. (2003). Architecture of monitoring elements for the network element modeling in a grid infrastructure. In *Proc. of Workskop on Computing in High Energy and Nuclear Physics*, La Jolla (California).

Ciuffoletti, A. and Polychronakis, M. (2007). *Architecture of a Network Monitoring Element*, volume 4375 of *Lecture Notes in Computer Science*, chapter 2, pages 4–14. Springer.

Di Modica, G., Petralia, G., and Tomarchio, O. (2012). A semantic framework to support cloud markets in interoperable scenarios. In *Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on*, pages 211–214.

Edmons, A., Metsch, T., Papaspyrou, A., and Richardson, A. (2012). Toward an open cloud standard. *IEEE Journals and Magazines*, (4):15–25.

Fette, I. and Melnikov, A. (2011). The WebSocket Protocol. RFC 6455 (Proposed Standard).

Fielding, R. T. and Taylor, R. N. (2002). Principled design of the modern web architecture. *ACM Trans. Internet Technol.*, 2(2):115–150.

Mell, P. and Grance, T. (2011). The NIST definition of cloud computing. Technical Report Special Publication 800-145, US Department of Commerce.

OGF (2011a). *Open Cloud Computing Interface - Core*. Open Grid Forum. Available from www.ogf.org. A revised version dated 2013 is available in the project repository.

OGF (2011b). *Open Cloud Computing Interface - Infrastructure*. Open Grid Forum. Available from www.ogf.org.

VMware (2007). Vmware high availability: Concepts, implementation and best practices. White paper.

Youseff, L., Butrico, M., and Da Silva, D. (2008). Toward a unified ontology of cloud computing. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1–10.

Model-based Approach to Automatic Software Deployment in Cloud

Franklin Magalhães Ribeiro Junior and Tarcísio da Rocha
Departamento de Computação, Universidade Federal de Sergipe (UFS), São Cristóvão, SE, Brazil
{franklin.mr3, tarcisiorocha}@gmail.com

Keywords: Software Deployment in Cloud, Model-driven Deployment, Human-Computer Interaction, Cloud Computing.

Abstract: Cloud computing provides resources to reduce software processing costs in IT companies. There are automatic mechanisms to software deployment in cloud providers, however it demands manual coding. In this paper we present a model-based approach to automatic software deployment in cloud environment. We show a brief literature review of existent proposals to automatic software deployment in cloud. We analyzed the proposals, where five used deployment mechanisms based on script or programming language, two proposals based on manual mechanisms and two proposals use a model-based approach to software deployment in cloud, however one is still strongly tied to manual aspects and other complex to modelling. This paper presents a new detailed architecture, a use case and the conceptual view of our model-based approach to automatic software deployment in cloud. This approach aims to reduce the human efforts and time to deploy services in cloud, using UML deployment diagrams as input, in order to deploy it as much as possible on the highest abstraction layer.

1 INTRODUCTION

A way to attend the demand of large amount of data is the resources provided by cloud computing. IT Companies are considering software deployment in cloud providers, because processing distribution and larger storage capacity reduces infrastructure costs.

To implement a software system in a cloud provider requires a reconstruction of existing requirements, because cloud environments have their own architectures (Cala and Watson, 2010). An approach to automatic software deployment makes easier this process. In addition, it reduces human efforts (Cala, 2010).

Aiming investigate solutions referred to services deployment in clouds, we present a brief discussion of related proposals in (Ardagna et al., 2012), (Cala and Watson, 2010), (Chieu et al., 2010), (Juve and Deelman, 2011), (Konstatinou et al., 2009), (Li et al.,2012), (van der Burg, 2009) and (Zhang et al., 2013), to characterize and analyze the software deployment mechanisms in cloud environments, under vision of human effort employed to achieve the deployment.

This paper proposes a model-driven approach to automatic software deployment in cloud, which uses UML diagrams as input, to minimize the impact of human efforts and IT costs. In addition, we describe the architecture and the conceptual view of our approach.

The rest of paper is structured as follows. Section 2 describes the deployment in cloud. We review the related works in Section 3. Section 4 compare and analyze our proposal against the related works. In Section 5 we present our model-based approach to automatic software deployment in cloud, followed by a discussion describing the architecture, a conceptual view of proposal and the benefits to reduce human efforts. Finally, we present the conclusion and future works in Section 6.

2 DEPLOYMENT

Cloud computing is a mix between grid computing, distributed computing and virtualization concepts (Kalagiakos and Karampelas, 2011). Among the existing cloud providers, we can list some, such as Amazon EC2 (Amazon, 2013) and Azure Cloud (Microsoft, 2013).

To deploy and run an application in a cloud provider, services to support the application are necessary. In this context, services could be seen as components.

The OMG (OMG, 2006) defines the deployment process of components in five steps:

- Installation: which publishes software packages in a control repository;
- Configuration: configure the repository to support applications;
- Planning: provides an implementation plan to decide the part of the application that will run;
- Preparation: decides where software will run;
- Launch: decides which roles will connect the component instances in order to run the entire software.

To deploy software in cloud manually is expensive, because it is necessary to understand the cloud architecture, the security mechanisms (Savu, 2011) and others specific configurations of the cloud (Li et al., 2012).

Among the features related to software deployment, the main are:

- Coordinators: manage a software stack (Cala and Watson, 2010);
- Software stack: contains the services to support an application;
- Virtual machine (VM): where services run to support the application (Armstrong et al., 2011) and
- Client node: encapsulates the VM (Juve and Deelman, 2011).

3 RELATED WORK

The research in (Li et al., 2012) proposed encapsulation of services in virtual machines (VMs) and the deployment of services in a range of cloud scenarios, such as private cloud, busted cloud, federated cloud and multi cloud brokering. They concluded that the time to deploy depends on the scenario of the cloud.

The investigation in (Juve and Deelman, 2011) evaluated a system called Wrangler. Wrangler sends an XML description of a web service implementation to manage virtual machines provisions and to interact with several cloud providers (such as, Amazon EC2, Eucalyptus and OpenNebula) in order to deploy applications. The proposal used four concepts: clients, coordinator, agents and plugins. They realized that Wrangler makes the deployment process easier.

The research in (Ardagna et al., 2012), the authors presented a model-driven approach for design and execution of applications on multiple clouds called MODAClouds, They proposed architecture is divided in two views, design and run time to software deployment.

Aiming homogeneity access across different physical devices in a hospital that were connected using different networks topologies and complex security policies, such as MRI machines, computers, among others, in (van der Burg, 2009) the authors proposed an architecture which software components are automatically deployed in cloud, using a declarative model creating an extension of Nix.

The authors in (Cala and Watson, 2010) presented a platform for automatic software deployment in Azure Cloud. The work was motivated by the need for execution of an application that analyzes the chemical structure of a drug, the QSAR, used by a multi-agent system, called Discovery Bus. The authors build the deployment in two plans, one to install the services by workers nodes, and other to define spatial and temporal dependences, including library dependencies. They concluded that their platform was successfully applied to OS virtualization levels and can model well the spatial and temporal constrains of deployment process.

The research in (Chieu et al., 2010) presented a virtualization for building and deploying software stack in virtual machines (VM) using XML. They proposed a framework called Vega that provides mechanisms to enable administrators to configure the dependencies between services to achieve deployment in cloud. Also it makes possible to manage hardware and network resource. However, because a limitation of this approach is that it does not deal well with complex cloud scenarios.

To decouple VM from software, (Zhang et al., 2013) developed a framework for deploying applications in cloud, which aims to reduce the time and costs related to deployment. The proposal differs from the others approaches presented so far, because the software would not be pre-installed on a VM image.

The authors in (Zhang et al., 2013) presented the framework and its functionality divided into three stages: (i) in the software preparation, to the customer stores the software in a repository; (ii) in the selection stage, of the customer can select a software in the repository – the software is installed in a VM and this VM is sended and installed on the local machine of the customer, and finally (iii) the software deployment, instead of storing the application in a VM image, it runs on a local machine on the client side without installing it.

In their work, authors in (Konstatinou et al., 2009) presented an architecture that supports model driven software deployment. The technique shows some phases: the virtual solution declarative model (VSM), the model of virtual solution deployment (VSDM) and deployment plan virtual solution (VSDP). However, for each phase, it is necessary human interaction with the proposed system.

4 DEPLOYMENT MECHANISMS

Aiming to discover and analyze the mechanisms used by solutions related to automatic software deployment, we selected some relevant characteristics proposed by related work.

The existing mechanisms for the deployment of services are: manual, script-based, language-based and model-based. In (Talwar et al., 2005) the authors observed that model-based mechanisms deals well with time metric and increases the complexity of the solution (see Figure 1).

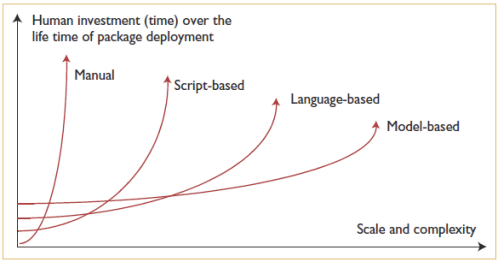


Figure 1: Deployment Mechanisms (Talwar et al., 2005).

In the solutions presented by the related work, we identified the same mechanisms (see Table 1) found in (Talwar et al., 2005).

We perceived that the mechanism presented in (Konstatinou et al., 2009) and (Ardagna et al., 2012) was the approaches that present a model-based software deployment (see Table 1).

We observed that solutions in (Cala and Watson, 2010), (Li et al., 2012) and (Zhang et al., 2013) used the language-based mechanism, which in contrast to the approaches presented in (Chieu et al., 2010); (Juve and Deelman, 2011) and (van der Burg, 2009) it leads better with time investments to the deployment process (see Figure 1). While the code solution of the approaches presented in (Chieu et al., 2010); (Juve and Deelman, 2011) and (van der Burg, 2009) are less complex, they require more time to the deployment process. However, according to (Fazziki et al., 2012) and (Talwar et al., 2005) these aspects have a lower degree of advantages, because they increase the costs with respect to the code and to human efforts.

Table 1: Deployment Mechanisms used in Proposals.

	Deployment Mechanism			
Proposals	Manual	Script-based	Language-based	Model-based
(Ardagna et al., 2012)				X
(van der Burg et al., 2009)	X	X		
(Konstantinou et al., 2009)	X	X		X
(Chieu et al., 2010)		X		
(Cala and Watson, 2010)			X	
(Juve and Deelman, 2011)		X		
(Li et al., 2012)			X	
(Zhang et al., 2013)			X	
Our Approach				**X**

Furthermore, we perceived that approaches mentioned in (Konstantinou et al., 2009) and (van der Burg, 2009) presented semi-automatic mechanisms to the deployment. In other words, in these works there is still the need for manual intervention in the deployment process.

We also perceived that the proposal in (Ardagna et al., 2012) presented a model-based mechanism to deployment, however this approach requires some understanding of the end user about details of the cloud computing structure.

Our proposal consists on a model-based approach to automatic software deployment, aiming to deploy in a higher abstraction layer to reduce the human investment and time as much as possible. Moreover, the use of model-based approach is the best way to increase productivity (Fazziki et al., 2012).

5 MODEL-BASED PROPOSAL TO DEPLOYMENT IN CLOUD

This section explains the architecture and the conceptual view of proposal to automatic software deployment in cloud, moreover, is also presented the use case of proposal. On the use case, we can observe the reduction of human efforts in deployment, because on solution is only used UML deployment diagrams as input.

Our proposal for automatic software deployment does not require additional coding efforts, it only requires as input two UML deployment diagrams: the first one is cloud provider independent (to handle software legacy), that only contains virtual machines

(VMs) and the services dependencies allocated to VMs, the second one has specific aspects related to the cloud, but only the access keys and the cloud provider name. This is possible, because our approach is supported by the 14BIS (SWX Software, 2013) and Chef (Opscode, 2013), that provides a platform to deploy third party software, where the final user of our approach (IT companies that uses the deployment as a service from 14BIS) only uses the software repository (as a general model, to handle the software legacy, being cloud provider independent), the database *url*, the name of cloud provider, the operational system and the access key as inputs.

5.1 Architecture

The proposed architecture is divided into three views (see Figure 2):

- The system view: showing the five modules of the system (*Control*, *Associator*, *Stacker*, *Allocator* and *Preparator*);
- Local view: that corresponds to the user's view, it means, the human-computer interaction (which uses as input passwords and UML deployment diagrams);
- Remote view: which includes the creation of a Chef server instance (Opscode, 2013) in the cloud and the final stage of deployment that is the software installation in the cloud.

The routine of *Associator* module starts when the user input two UML deployment diagrams (specific and general), as well as the hosts and passwords (cloud access keys). Moreover, the *Associator* module interprets UML diagrams to collect information related to UML components (a more detailed concept of each module is described in Section 5.3 and Figure 4).

The *Stacker* module is responsible for creating the software stacks, as previously specified in UML deployment diagram by the user.

The *Allocator* accesses the cloud and creates a Chef server (Opscode, 2013) instance in the cloud.

The *Preparator* module receives an acknowledgment from the *Allocator* module that the server instance was created in the cloud. So *Preparator*, prepares a client with a virtual machine (VM), it installs the operating system on the VM, allocates the services (the software stack) on VM and finally deploys the application in the cloud. Thereafter, the application can be executed by the provider.

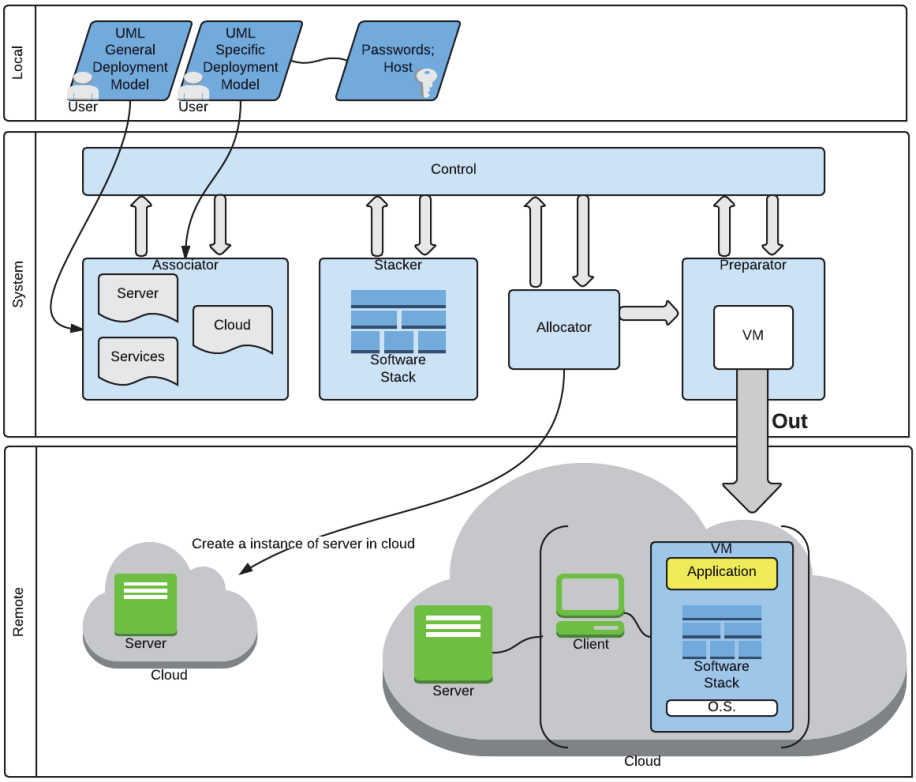


Figure 2: Proposed architecture.

The *Control* block manages the other four modules, acting as an intermediary in the communication between modules and is responsible for:

- Send *Associator* services to the *Stacker* and the list of dependency relationships between services (software stack), from *Stacker* to *Preparator* module;
- Enable communication between *Allocator* and the cloud;
- Inform to *Allocator* about creating the server instance in the cloud;
- Inform *Preparator* which operating system must be installed on the VM;
- Inform *Preparator* which software stack must be allocated on its VM;
- Manage the transmission in order to inform and perform tasks between the blocks.

5.2 Reduction of Human Efforts

The main advantage of the proposal is to focus on reducing the costs of time and human efforts to deployment. The reduction of this metrics values (time and human investment) may come from the deployment mechanism through the use of UML deployment diagrams as input (without coding). In addition, it is known that the use of model-driven development to generate code increases software quality, reduces costs and possible coding errors (Fazziki et al., 2012).

The use case (see Figure 3) aims to show the reduction of human efforts in the deployment process. In use case, it can be observed that the workload is summarized in creating UML deployment diagrams and implementation of services. The user would have just to use as input two deployment models, a general (for VMs and services) and other specific (corresponding to aspects of chosen cloud).

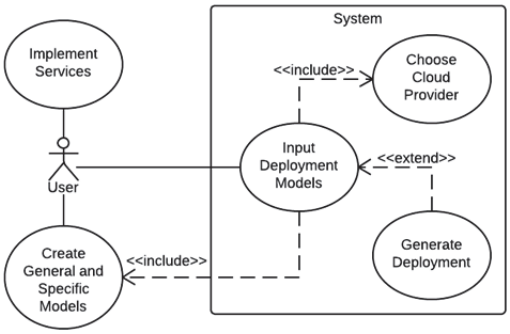


Figure 3: Use case.

5.3 Conceptual View

To observe the proposal main aspects, we have developed a conceptual view (see Figure 4), using the Amazon EC2 (Amazon, 2013) as the cloud provider, the Chef (Opscode, 2013) as server management and StarUML tool (Plastic Software, 2013) to create UML diagrams. The conceptual view is composed of four modules mentioned in the proposed architecture see Subsection 5.1), where each of these modules corresponds to a state in a state machine (see Figure 4).

5.3.1 Associator Module

In the conceptual view, the state that represents the *Associator* module is responsible to interpret the meta-language generated by the UML deployment diagram (created by user in StarUML tool). After the analysis of generated meta-language (see Figure 4), we observed that the Virtual Machine node is represented as a array element *OwnedViews[],* which has a *type = UMLNodeView*, an identifier *guid* and the coordinates as integer values by the *Left, Top, Width and Height* attributes (see Figure 4).

We perceived that the dependency relationship between the artifacts *Service [n]* and *Service [n-1]* is identified by *type = UMLDependencyView* and that this relationship has two attributes named by *head* and *tail,* which corresponds to the identifiers of UML artifacts: *Service[n]* and *Service [n-1],* where *Service[n]* depends on *Service[n-1].*Moreover, the *guid* assigned to *Service [n]* corresponds to the ID attribute *head* of the dependency relationship (see Figure 4).

5.3.2 Stacker Module

The *Stacker* module identifies which components (UML artifacts) are inside the UML node envelopment, in other words, which services belong to the node and stores them in lists of artifacts (software stacks corresponding to the respective VM node). For it be possible, it is necessary to compare the coordinates of each node with each artifact, however, this technique complexity would be *O(m*n),* where *m* is the number of nodes and *n,* the number of artifacts.

To reduce the complexity of the algorithm *O(m*n),* we developed an algorithm (see Figure 4) of complexity *O((n + m) + (m *log m) + (n *log n)),* which presents a lower complexity compared to *O(n*m),* but only if *n>= 4 and m>= 3.* However, this solution uses the algorithm complexity *O (m* n) if m<3* and *n<4.*

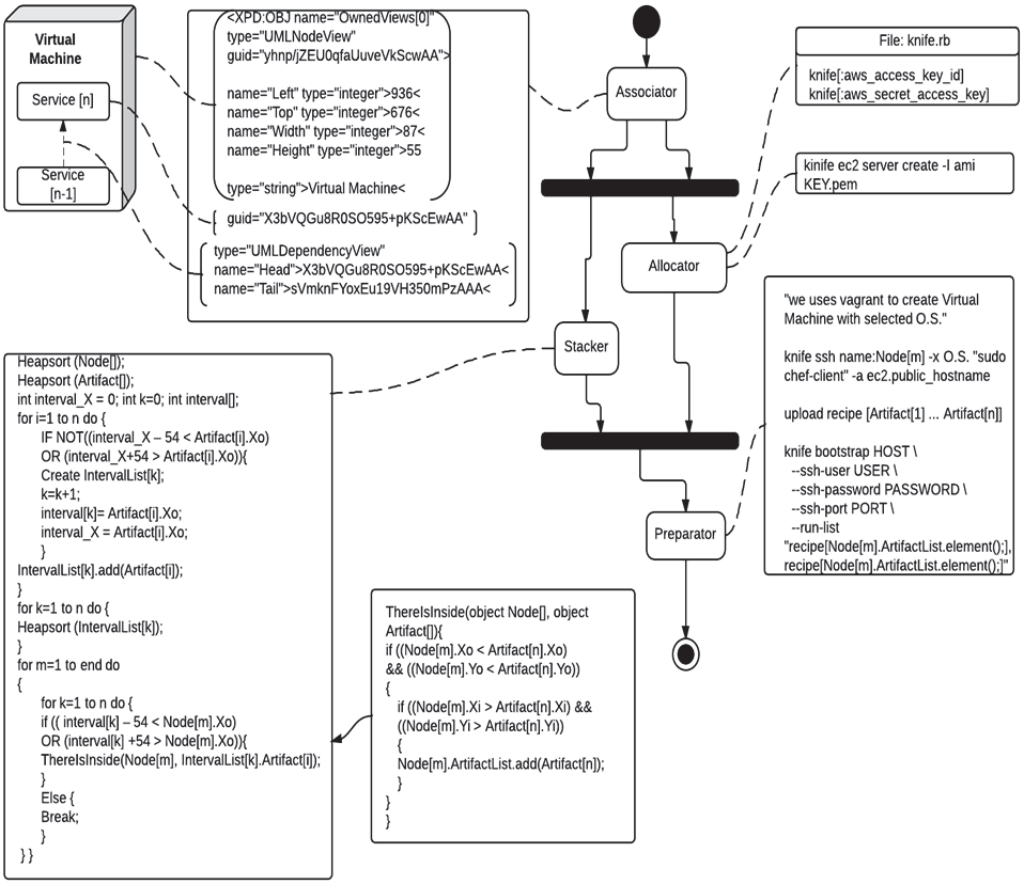


Figure 4: Conceptual view.

5.3.3 Allocator Module

The *Allocator* module, on this example, using the Chef (Opscode, 2013) as a tool for creating a server instance in the Amazon EC2 cloud (Amazon, 2013). In it is used access keys and passwords, written in the file called knife.rb, which is required by EC2 provider (see figure 4).

5.3.4 Preparator Module

The *Preparator* module is responsible for creating the virtual machine and for the installation of selected operating system in the VM. After it, this module creates a client node and allocates the virtual machine to this client.

The solution uploads all services (elements of *ArtifactList*) to the VM with the command *"upload recipe [Artifact[n-1]]…[Artifact [n]]"*. Finally it *bootstrap* services related to VM node (see Figure 4), with command *"recipe [Node[m].ListArtifact.element();]"* (see Figure 2).

6 CONCLUSIONS

In this study, we presented an approach to automatic software deployment in the cloud. Through a literature review, we identified and analyzed the properties of existing solutions. In the proposed approaches, we found four software deployment mechanisms, such as: manual, script-based, language-based and model-based.

We perceived that two researches (Konstatinou et al., 2009) and (Ardagna et al., 2012) presented a model-based proposal, but first one with strongly manual aspects and second to expert deployment users.

In this research we proposed an automatic model-based approach to software deployment in cloud to reduce the human efforts and time, using only UML diagrams as input. Furthermore, we described the architecture of our proposal, which is divided into three visions and five modules, as well as a conceptual view of the proposal was also described.

As future work, we will verify and analyze human investment and time in the company *SWX Softwares* about deployment in cloud (using our proposal against traditional proposals) to evaluate our approach.

ACKNOWLEDGEMENTS

Thanks to *CAPES* for supporting this work and *SWX Softwares* for the partnership.

REFERENCES

Amazon, 2013. Elastic Compute Cloud (EC2), Available in: http://aws.amazon.com/ec2. Access in 10, June, 2013.

Ardagna, D.; Di Nitto, E.; Mohagheghi, P.; Mosser, S.; Ballagny, C.; D'Andria, F.; Casale, G.; Matthews, P.; Nechifor, C.-S.; Petcu, D.; Gericke, A.; Sheridan, C., "MODAClouds: A model-driven approach for the design and execution of applications on multiple Clouds," *Modeling in Software Engineering (MISE), 2012 ICSE Workshop on* , pp.50,56, 2-3 June 2012.

Armstrong, D.; Djemame, K.; Nair, S.; Tordsson, J.; Ziegler, W., 2011. "Towards a Contextualization Solution for Cloud Platform Services," Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, pp.328,331.

Cala J., 2010. Adaptative Deployment of Component-based Applications in Distributed Systems. A dissertation for the degree of Doctor of Philosophy. University of Science and Technology, Krakrów, Poland.

Cala J., Watson P., 2010. Automatic Software Deployment in the Azure Cloud. Distributed Applications and Interoperable Systems Lecture Notes in Computer Science Volume 6115, pp 155-168.

Chieu T.; Karve, A.; Mohindra, A.; Segal, A., 2010. "Simplifying solution deployment on a Cloud through composite appliances," Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium., pp.1,5, 19-23.

Fazziki, A.E.; Lakhrissi, H.; Yetognon, K.; Sadgal, M., 2012. "A Service Oriented Information System: A Model Driven Approach," *Signal Image Technology and Internet Based Systems (SITIS), 2012 Eighth International Conference on*., pp.466,473, 25-29.

Juve, G.; Deelman, E., 2011. "Automating Application Deployment in Infrastructure Clouds," Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, pp.658,665.

Kalagiakos, P.; Karampelas, P., 2011. "Cloud Computing learning," Application of Information and Communication Technologies (AICT), 2011 5th International Conference on , vol., no., pp.1,4, 12-14.

Konstantinou A. V., Eilam T., Kalantar M., Totok A. A., Arnold W., Snible E., 2009. An architecture for virtual solution composition and deployment in infrastructure clouds, Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing.

Li W., Svard P., Tordsson J., and Elmroth E., 2012. A General Approach to Service Deployment in Cloud Environments. 2012 Second International Conference on Cloud and Green Computing, UmeaUniversity, Sweden.

Microsoft Corporation. 2013. Azure Cloud, Available in: http://www.windowsazure.com/en-us/. Accessed: 10, June, 2013.

Object Management Group, 2006. Inc.: Deployment and Configuration of Component-based Distributed Applications Specification, Version 4.0.

Opscode, 2013. Chef. Available in: http://www.opscode.com/chef/. Access in 09, May, 2013.

Plastic Software, 2013. StarUML. The Open Source UML/MDA Platform, Available in: http://staruml.sourceforge.net/en/ Access in: 08, August, 2013.

Savu, L., 2011. "Cloud Computing: Deployment Models, Delivery Models, Risks and Research Challenges," Computer and Management (CAMAN), 2011 International Conference on , vol., no., pp.1,4, 19-21.

SWX Softwares. “14BIS - Software como Produto na Nuvem” Avaliable in: http://swx.com.br/ and http://ciomarket.com.br/ . Access in: May, 2013.

Talwar V., Milojicic D., Wu Q., Pu C., Yan W., and Jung G., 2005. Approaches for Service Deployment. IEEE Internet Computing, 9(2):70–80.

van der Burg, S.; de Jonge, M.; Dolstra, E.; Visser, E., 2009. "Software deployment in a dynamic cloud: From device to service orientation in a hospital environment," Software Engineering Challenges of Cloud Computing, 2009. ICSE Workshop on, vol., no., pp.61,66, 23-23.

Zhang Y., Li Y., Zheng W., 2013. Automatic software deployment using user-level virtualization for cloud-computing, Future Generation Computer Systems, Volume 29, Issue 1, Pages 323-329.

Cost-efficient Capacitation of Cloud Data Centers for QoS-aware Multimedia Service Provision

Ronny Hans[1], Ulrich Lampe[1], Michael Pauly[2] and Ralf Steinmetz[1]
[1]*Multimedia Communications Lab (KOM), TU Darmstadt, Rundeturmstr. 10, 64283 Darmstadt, Germany*
[2]*T-Systems International GmbH, Pascalstr. 51, 52076 Aachen, Germany*
ronny.hans@kom.tu-darmstadt.de

Keywords: Cloud Computing, Data Center, Capacitation, Quality of Service, Multimedia, Service.

Abstract: Cloud infrastructure is increasingly used for the provision of sophisticated multimedia services, such as cloud gaming or Desktop as a Service, with stringent Quality of Service demands. Serving these service demands results in the need to cost-efficiently select and capacitate data centers. In the work at hand, we introduce the corresponding *Cloud Data Center Capacitation Problem* and propose two optimization approaches. Through a quantitative evaluation, we demonstrate that an exact solution approach is only practically applicable to small problem instances, whereas a heuristic based on Linear Program relaxation achieves significant reductions in computation time of about 80% while retaining a favorable solution quality, with cost increases of approximately 5% or less.

1 INTRODUCTION

Since the term was first coined in the mid-2000s, cloud computing has received increasing attention by both IT practitioners and researchers. In this context, a focus has often been on Infrastructure as a Service, given that it is the most flexible among the three cloud computing service models (Briscoe and Marinos, 2009) and that its successful application has been widely documented in the literature, e. g., (Armbrust et al., 2009). However, with the increasing maturity of cloud computing, the focus is shifting toward the cloud-based delivery of sophisticated multimedia services. Such software-oriented services include, for example, cloud gaming / Games as a Service (Chang, 2010) or Dekstop as a Service (Erdogmus, 2009). Due to their nature, such multimedia services pose high demands concerning Quality of Service (QoS) attributes. Unfortunately, past empirical research has shown that the current cloud infrastructure is partially insufficient to meet those demands, most notably due to the latency that arises from the massive centralization of cloud data centers in few geographical locations (Choy et al., 2012).

Accordingly, in the work at hand, we examine how cloud data centers can be appropriately selected and capacitated in order to serve QoS-aware multimedia services. In our previous work (Hans et al., 2013), we have addressed the selection of cloud data centers for single service types and time-invariant service demands. The work at hand expands our past research through the consideration of multiple service types, as well as fluctuating service demands, and also regards the resulting distinction between variable costs for operation and reservation of cloud infrastructure. In this context, we propose an exact solution approach, based on an Integer Program (IP) formulation, as well as a heuristic approach based on Linear Program (LP) relaxation.

The remainder of this paper is structured as follows: In Section 2, we briefly explain the specific problem that is addressed in this paper. In Section 3, we introduce formal notations, based on which we specify two optimization approaches. These approaches are quantitatively evaluated in Section 4. An overview of related work is given in Section 5. Section 6 concludes the paper with a brief summary and outlook on future work.

2 PROBLEM STATEMENT

In this work, we assume the role of a service provider, who aims to deliver multimedia services to a distributed set of users. Specifically, we consider a set of so-called user clusters, each of which represents a predefined number of users in a certain geographi-

cal location, e. g., a state or county. Each user cluster exhibits specific demands for a given set of service types, with the demand fluctuating over a predefined number of time slots. If these demands cannot be met, certain penalties accrue. Furthermore, each service type is associated with certain QoS requirements, e. g., concerning permissible latency.

In order to deliver his/her services, the provider has the choice among a given set of data centers. The selection of a data center incurs certain fixed costs, e. g., for construction or long-term lease. In addition, the operation of a server within each data center results in certain variable costs. Furthermore, the reservation of a number of servers over the planning period may incur certain variable reservation costs. Due to the geographical distribution, each data center makes different QoS guarantees with respect to each user cluster.

The aim of the provider is to choose among the data centers and further take a capacitation decision, i. e., decide on the number of reserved servers, such that the overall cost of the solution is minimized. In the following, we refer to this problem – which is a generalization of a research issue we previously examined (Hans et al., 2013) – as *Cloud Data Center Capacitation Problem* (CDCCP).

3 OPTIMIZATION APPROACHES

In the following, we first introduce a set of notations to formally represent the CDCCP (cf. Section 3.1). Subsequently, we introduce two optimzization approaches, namely an exact approach based on Integer Programming (cf. Section 3.2) and a heuristic approach based on LP relaxation (cf. Section 3.3).

3.1 Formal Notations

In order to represent the CDCCP in the form of a mathematical model, a few formal notations are required. To begin with, we formally define the basic entities within the CDCCP using the following symbols:

- $D = \{1,2,...,D^{\#}\}$: Set of (potential or existing) data centers
- $U = \{1,2,...,U^{\#}\}$: Set of user clusters
- $S = \{1,2,...,S^{\#}\}$: Set of available services
- $Q = \{1,2,...,Q^{\#}\}$: Set of considered QoS attributes
- $T = \{1,2,...,T^{\#}\}$: Set of discrete time slots within the planning period

Based on the previously introduced basic entities, the parameters that are associated with the individual entities can be defined as follows:

- $SD_{u,s,t}$: Service demand of user u for service s at time t
- $K_d^{min} \in \mathbb{R}$: Minimal capacity of data center d
- $K_d^{max} \in \mathbb{R}$: Maximal capacity of data center d
- $CF_d \in \mathbb{R}$: Fixed cost of selecting data center d
- $CVO_d \in \mathbb{R}$: Variable cost for operating one server unit for one time unit in data center d
- $CVR_d \in \mathbb{R}$: Variable cost for reserving one server unit in data center d
- $CP_{u,s} \in \mathbb{R}$: Penalty cost per service unit not provided to user u w.r.t. service s
- $QG_{d,u,q} \in \mathbb{R}$: QoS guarantee of data center d w.r.t. user u for QoS attribute q
- $QR_{u,s,q} \in \mathbb{R}$: QoS requirement of user u w.r.t. service s for QoS attribute q

Lastly, in order to model the CDCCP as optimization problem, we use the following decision variables:

- x_d: Selection of a data center d
- $y_{d,u,s,t}$: Capacity provided by data center d to user cluster u concerning service s at time t
- $y'_{u,s,t}$: Penalty-bound capacity not provided to user cluster u concerning service s at time t
- z_d: Capacity reserved in data center d

3.2 Exact Optimization Approach CDCCP-EXA.KOM

Based on the notations from the previous section, the CDCCP can be modeled as an optimization problem in an intuitive manner. The result is provided in Model 1 and will be explained in detail in the following. To begin with, Equation 1 defines the objective function, aiming at a minimization of total costs, depending on the values of the decision variables. Equation 2 ensures that all service demands will be satisfied or that corresponding penalties will accrue. Equation 3 links the decision variables y and z, ensuring that only the reserved capacity in each data center may be used in each time slot. Equations 4 and 5 make sure that the capacity constraints for each data center are held. Equation 6 ensures that the QoS requirements of each user cluster are matched by the corresponding data center guarantees, depending on the value of the auxiliary variable p from Equation 7. Lastly, Equation 8 defines the decision variables as binary and natural.

As can easily be seen, Model 1 constitutes an IP. Such problems can be solved using off-the-shelf algorithms, most notably the branch-and-bound algorithm (Domschke and Drexl, 2004). However, despite its efficiency in many application scenarios, branch-and-bound is based on the principle of enumeration (Hillier and Lieberman, 2005). Hence, in the worst case, the time complexity of computing a solution to a given problem instance grows exponentially with the number of decision variables, i. e., the number of entities in the model. Accordingly, the practical applicability of CDCCP-EXA.KOM is likely limited to smaller problem instances and situations where the computation time requirements play an inferior role.

3.3 Heuristic Optimization Approach CDCCP-REL.KOM

The brief qualitative analysis from the previous section indicates a potentially high computational complexity for the exact approach CDCCP-EXA.KOM. Based on this notion, we introduce a heuristic approach that is based on the common concept of LP relaxation (Domschke and Drexl, 2004). Specifically, the binary and integer decision variables in the initial model (cf. Equation 8) are substituted by corresponding natural variables (cf. Equation 9).

The resulting LP formulation of the initial problem can be solved using another set of off-the-shelf algorithms, such as interior point methods. In contrast to branch-and-bound, such algorithms are characterized by polynomial, rather than exponential worst case time complexity (Hillier and Lieberman, 2005). This renders them potentially applicable to larger problem instances, even under relatively rigid time constraints. From the LP-based solution, a final solution can simply be deduced by rounding all natural values of the decision variables to the next-highest integer.

4 EVALUATION

4.1 Setup

In order to assess the applicability of our proposed optimization approaches, we prototypically implemented them in Java 7. As solver framework, we used IBM ILOG CPLEX 12.5[1], which was accessed through the JavaILP middleware[2].

[1]http://www.ibm.com/software/integration/optimization/cplex-optimizer/

[2]http://javailp.sourceforge.net/

Model 1. Cloud Data Center Capacitation Problem.

$$\begin{aligned}\text{Min. } C(x,y,z) = &\sum_{d\in D} x_d \times CF_d \\ &+ \sum_{d\in D,u\in U,s\in S,t\in T} y_{d,u,s,t} \times CVO_d \\ &+ \sum_{d\in D,u\in U,s\in S,t\in T} y'_{u,s,t} \times \times CP_{u,s} \\ &+ \sum_{d\in D} z_d \times CVR_d\end{aligned} \tag{1}$$

$$y'_{u,s,t} + \sum_{d\in D} y_{d,u,s,t} \geq SD_{u,s,t} \quad \forall u \in U, \forall s \in S, \forall t \in T \tag{2}$$

$$\sum_{u\in U,s\in S} y_{d,u,s,t} \leq z_d \quad \forall d \in D, \forall t \in T \tag{3}$$

$$z_d \leq x_d \times K_d^{max} \quad \forall d \in D \tag{4}$$

$$z_d \geq x_d \times K_d^{min} \quad \forall d \in D \tag{5}$$

$$y_{d,u,s,t} \leq p_{d,u,s} \times K_d^{max} \quad \forall d \in D, \forall u \in U, \forall s \in S, \forall t \in T \tag{6}$$

$$p_{d,u,s} = \begin{cases} 1 & \text{if } QG_{d,u,q} \leq QR_{u,s,q} \quad \forall q \in Q \\ 0 & \text{else} \end{cases} \tag{7}$$

$$\begin{aligned} x_d &\in \{0,1\} \quad \forall d \in D \\ y_{d,u,s,t} &\in \mathbb{N} \quad \forall d \in D, \forall u \in U, \forall s \in S, \forall t \in T \\ y'_{u,s,t} &\in \mathbb{N} \quad \forall u \in U, \forall s \in S, \forall t \in T \\ z_d &\in \mathbb{N} \quad \forall d \in D \end{aligned} \tag{8}$$

. . . . . .

$$\begin{aligned} x_d &\in \mathbb{R}, 0 \leq x_d \leq 1 \quad \forall d \in D \\ y_{d,u,s,t} &\in \mathbb{R}, y_{d,u,s,t} \geq 0 \quad \forall d \in D, \forall u \in U, \forall s \in S, \forall t \in T \\ y'_{u,s,t} &\in \mathbb{R}, y_{u,s,t} \geq 0 \quad \forall \forall u \in U, \forall s \in S, \forall t \in T \\ z_d &\in \mathbb{R}, z_d \geq 0 \quad \forall d \in D \end{aligned} \tag{9}$$

In accordance with Silver (Silver, 2004), our evaluation focuses on two dependent variables, namely computation time and solution quality (i. e., total cost associated with the computed solution). As independent variables, we considered the number of data cen-

ters ($D^{\#}$), user clusters ($U^{\#}$), service types ($S^{\#}$), and time slots ($T^{\#}$), since they have a direct impact on the number of decision variables, and hence, the size of the solution space. We used a fractional factorial design, varying the value of each independent variable separately while treating the remaining variables as controlled, i. e., assuming a fixed value.

In accordance with our previous work (Hans et al., 2013), we employed data from the 2010 United States census[3] as the basis for problem generation. In order to model data centers and user clusters, we randomly drew US counties from the census data, and set the service demands and different cost parameters based on the according county population and median income. As the only QoS requirement, we considered latency and set it to represent different multimedia service types, ranging from cloud gaming to Desktop as a Service. The QoS guarantees were finally computed based on the geographical distance between data centers and user clusters.

For each *test case*, i. e., distinct combination of values for the independent variables, we randomly created 50 problem instances. Problems that could not be successfully solved by the heuristic approach CDCSP-REL.KOM were removed from the sample; such invalid solutions may result from certain capacity constraints not being met due our simplistic next-highest integer rounding approach (cf. Section 3.3). Based on the samples, we subsequently computed the observed mean absolute computation times, as well as the macro-averaged ratios of computation time and total cost between CDCCP-REL.KOM and CDCCP-EXA.KOM, along with the respective 95% confidence intervals based on a t-distribution (Kirk, 2007). The evaluation was conducted on a desktop computer, equipped with an Intel Core 2 Quad Q9450 processor and 4 GB of memory, operating under Microsoft Windows 7.

4.2 Results and Discussion

The results of our evaluation are presented in Figures 1 through 3. As can be seen in Figure 1, the observed mean absolute computation times strikingly confirm the different computational complexity of CDCCP-EXA.KOM and CDCCP-REL.KOM. Even for the smallest considered test cases, the computation time for CDCCP-EXA.KOM ranges in the order of magnitude of 1 s, quickly growing to 10 s or even 100 s with an increasing size of the problem instances. In contrast, the mean computation times for CDCCP-REL.KOM remain in the order of magnitude of 10 s, even for the largest problem classes. These findings are also confirmed by the macro-averaged ratios of computation times, as given in Figure 2. Except for the smallest problem classes, CDCCP-REL.KOM consistently reduces the computation time by about 80% or more to CDCCP-EXA.KOM. The reduction is statistically significant across all test cases at the assumed confidence level of 95% (i. e., $\alpha = 0.05$).

On the downside, Figure 3 indicates that the application of LP relaxation in CDCCP-REL.KOM comes at a certain amount of additional cost, i. e., degradation in solution quality. Compared to CDCCP-EXA.KOM, the increase ranges between approximately 0.4% and 4.3%; however, it does not exceed 1.5% for all considered test cases except one. Thus, while the slight increase is statistically significant for practically all test cases at the 95% confidence level, it can be considered quite marginal and most likely acceptable in practical applications. In addition, as can be seen from the given sample sizes, CDCCP-REL.KOM is able to provide valid solutions to essentially all considered problem instances, except in six test cases, where one instance respectively could not be solved.

In conclusion, we find that the exact optimization approach CDCCP-EXA.KOM is associated with high computational complexity and hence, its practical application is limited to small problem instances. However, the approach can also serve as a benchmark for the assessment of alternative solution approaches, such as CDCCP-REL.KOM. The latter has presented a much more favorable performance in our experiments with respect to computational demands. Nevertheless, the development of custom-tailored optimization approaches for the CDCCP that do not rely on LP formulations may provide further improvements concerning the trade-off between computational requirements and solution quality.

5 RELATED WORK

In recent years, there has been vivid research in the area of cloud computing. In the following, we briefly discuss selected works that are most closely related to our research.

(Goiri et al., 2011) present an approach for efficient data center placement based on several factors, e. g., available network backbones and proximity of population centers. To find a solution for the placement problem, the authors use a combination of exact and approximate approaches. Thereby, Goiri et al. focus on design time, i. e., construction planning for new data centers. In contrast to our work, they do not

[3] http://www.census.gov/geo/maps-data/data/gazetteer.html

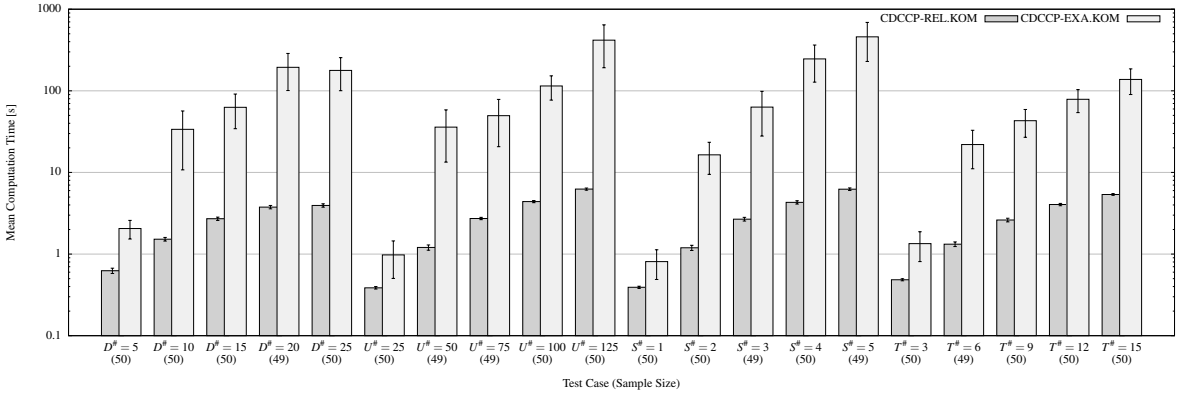


Figure 1: Observed mean computation times (with 95% confidence intervals) by test case. Please note the logarithmic scaling of the ordinate. If not specified differently, we use $D^{\#} = 15$, $U^{\#} = 75$, $S^{\#} = 3$, and $T^{\#} = 9$ for the independent variables.

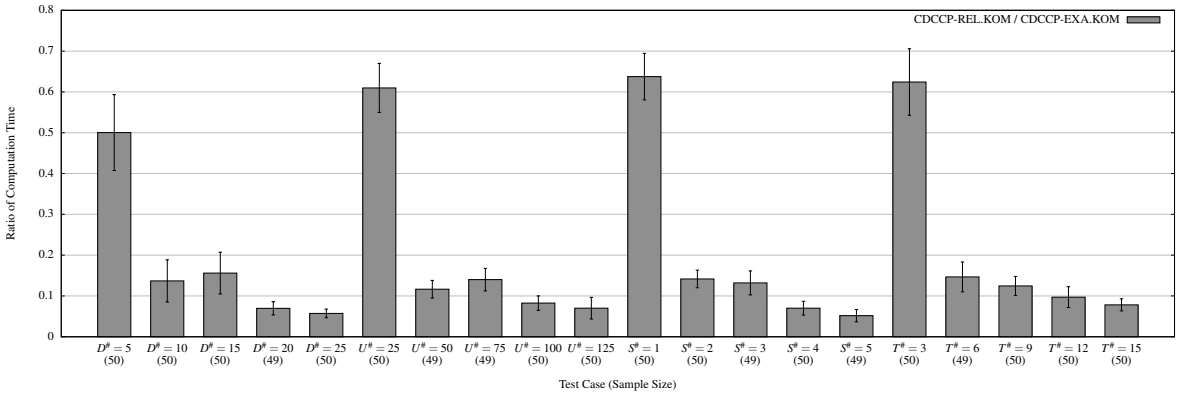


Figure 2: Ratio of computation times (based on macro-average; with 95% confidence intervals) between the two optimization approaches by test case. Configuration identical to Figure 1.

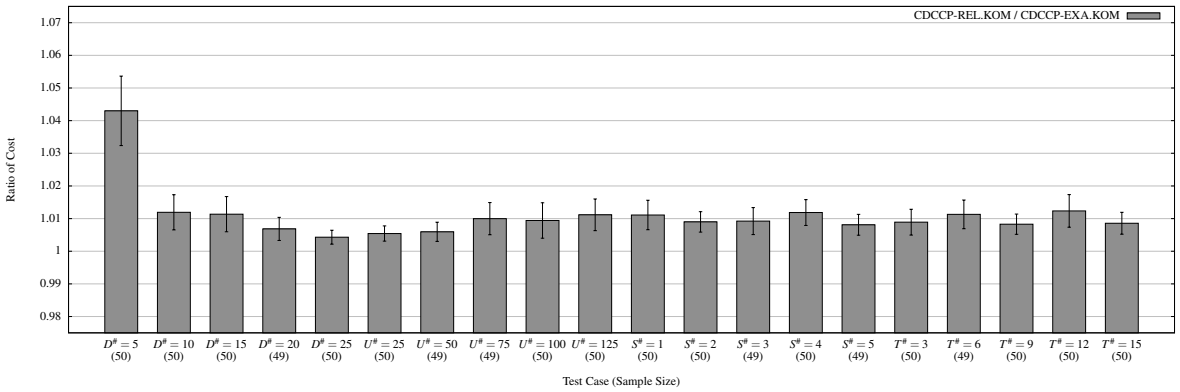


Figure 3: Ratio of costs (based on macro-average; with 95% confidence intervals) between the two optimization approaches by test case. Configuration identical to Figure 1.

consider time-variant service demands.

(Choy et al., 2012) study the current cloud infrastructure with respect to cloud gaming. The authors demonstrate that the current Amazon EC2 data centers could only serve 70% of the US population with adequate latency. Based on this finding, they propose the use of so-called edge servers to extend the current existing infrastructure, and validate their proposal using simulation approaches. In contrast to us, Choy et al. do not propose an exact approach for data center capacitation, and do not consider time-variant service demand.

(Larumbe and Sansò, 2012) present an optimization approach that addresses three distinct, yet inter-

linked problems: the geographical location of data centers, the location of software components that are hosted in network nodes and routing. Because the authors see a close connection between these problems, they integrated them in one mathematical framework using an optimal approach. Similar to the two aforementioned papers, this work only considers static service demand. Also, it exclusively provides an exact, but not a heuristic solution approach.

In summary, to the best of our knowledge, our work is the first to address the cost-efficient capacitation and placement of cloud data centers for QoS-aware services under consideration of time-variant service demand. In this context, this paper not only provides the exact solution approach CDCCP-EXA.KOM but also an initial heuristic solution, CDCCP-REL.KOM, which features substantially reduced computation times.

6 SUMMARY AND OUTLOOK

Cloud-based delivery of multimedia services, such as cloud gaming or Desktop as a Service, offers great economic potential. However, the adequate design of the underlying cloud infrastructure is a challenging task that has been only insufficiently addressed in research so far. In this work, we introduced the according Cloud Data Center Capacitation Problem (CDCCP). We proposed an exact solution approach, named CDCCP-EXA.KOM, based on Integer Programming. We further proposed a basic heuristic, called CDCCP-REL.KOM, which is based on the principle of Linear Program relaxation. Based on a quantitative evaluation, we showed that CDCCP-EXA.KOM is only practically applicable to smaller problem instances due to its exponential computational complexity. In contrast, CDCCP-REL.KOM features polynomial time complexity, thus significantly reducing the required computational effort for solving individual problem instances by 80% or more. At the same time, the heuristic maintains a favorable solution quality, with cost increases generally amounting to less than 5% compared to an exact solution.

Our future work primarily aims at the development of further heuristic approaches, which provide an even more favorable tradeoff between computational complexity and solution quality. Furthermore, we will extend the proposed approaches to account for stochastic, rather than just deterministic parameters, e. g., uncertain service demands or QoS properties.

ACKNOWLEDGEMENTS

This work has partly been sponsored by the E-Finance Lab e.V., Frankfurt a.M., Germany and by the German Research Foundation (DFG) in the CRC 1053 – MAKI.

REFERENCES

Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, H., Patterson, D., Rabkin, A., Stoica, I., et al. (2009). Above the Clouds: A Berkeley View of Cloud Computing (TR UCB/EECS-2009-28). Technical report, UC Berkeley.

Briscoe, G. and Marinos, A. (2009). Digital Ecosystems in the Clouds: Towards Community Cloud Computing. In *DEST 2009*.

Chang, T. (2010). Gaming Will Save Us All. *Communications of the ACM*, 53(3):22–24.

Choy, S., Wong, B., Simon, G., and Rosenberg, C. (2012). The Brewing Storm in Cloud Gaming: A Measurement Study on Cloud to End-User Latency. In *NetGames 2012*.

Domschke, W. and Drexl, A. (2004). *Einführung in Operations Research*. Springer, 6th edition. In German.

Erdogmus, H. (2009). Cloud Computing: Does Nirvana Hide Behind the Nebula? *IEEE Software*, 26(2):4–6.

Goiri, I., Le, K., Guitart, J., Torres, J., and Bianchini, R. (2011). Intelligent Placement of Datacenters for Internet Services. In *ICDCS 2011*.

Hans, R., Lampe, U., and Steinmetz, R. (2013). QoS-Aware, Cost-Efficient Selection of Cloud Data Centers. In *CLOUD 2013*.

Hillier, F. and Lieberman, G. (2005). *Introduction to Operations Research*. McGraw-Hill, 8th edition.

Kirk, R. (2007). *Statistics: An Introduction*. Wadsworth Publishing, 5th edition.

Larumbe, F. and Sansò, B. (2012). Optimal Location of Data Centers and Software Components in Cloud Computing Network Design. In *CCGRID 2012*.

Silver, E. (2004). An Overview of Heuristic Solution Methods. *J. of the Operational Research Society*, 55(9):936–956.

Internet of Things in the Cloud
Theory and Practice

Philip Wright and Andrea Manieri
Researchers, Engineering Ingegneria Informatica S.p.A., Rome, Italy
{*philip.wright, andrea.manieri*}*@eng.it*

Keywords: Cloud, Internet of Things, Smart City, Arduino, Sensors, Raspberry PI, Pachube, Cosm, Xively, Nimbits, oVirt, NoSQL DB, CouchDB, ZABBIX.

Abstract: The digital convergence of IT, Network and Content produced during the last years has created a new landscape for the definition and delivery of ICT services to citizens. The massive introduction of hypervisors and virtualisation techniques has allowed the (self-service) provision of any resource (software, network, data) in a seamless way to Users and Applications. The term Future Internet was coined in EU to synthesize these concepts and refer to the novel architectures enabling the next generation of Internet applications. The digital convergence includes also the so-called embedded systems that through Gateways connected to the Internet to provide a technical bridge to interact with sensors and actuators. This paper introduces the initial findings from experimental work done in the context of ClouT[a], a EU-funded project aiming at defining and developing a common virtualisation layer, allowing the access and management of IoT devices, as well as Cloud Services, in the same way as any other data.
The authors demonstrate and provide a simple way to design and implement a real infrastructure that satisfies the following requirements: cheap, easy to maintain, open source based, compatible and interoperable with different platforms and services. It is an example of how to make a public or a private Cloud capable of hosting data that comes from different kinds of sensors. Moreover this architecture allows the interconnection of all devices (Internet of Things, IoT) and its implementation is illustrated in detail.

[a]ClouT: Grant agreement, for "Collaborative project", number 608641, FP7 (Seventh Framework Programme). Project full title: "ClouT: Cloud of Things for empowering the citizen clout in smart cities".

1 INTRODUCTION

The ClouT project regarding the "Cloud of Things" (http://clout-project.eu/) is the first project that merges the IoT and the Cloud Computing concepts. One of the main focus' of the ClouT project, and of this paper as well, is not only to provide a system capable to send data coming from different sensors or actuators into the Cloud, but that can also work with legacy devices and different communication protocols.
In order to respond to the described objectives, a research in the open market has been made, looking for a device that can give us the possibility to collect data coming from sensors and we have opted for a microcontroller. Among the many offerings, we encountered one, called Arduino (http://arduino.cc), an Italian open-source device often used for different kind of applications, easy to use and very cheap.
The microcontroller makes it possible to find different solutions interfacing for example a database service like Xively[1] that allows the user to simply add data, coming for example from a sensor, and store them into a Cloud, accessible from any Cloud service (i.e.: web-portal or other). This example is very useful to understand how to implement a local testbed and demonstrates the possibility to match all lead technologies to build our experiment.

1.1 The Cloud Architecture

A typical Cloud has three possible service models: Infrastructure as a Service, Platform as a Service and Software as a Service[2], and they must be interconnected satisfying both the compatibility with all the desired services and the performance requirements in

[1]Xively (xively.com), previously known as Cosm and before as Pachube, is an on-line database service.

[2]For more details we suggest to read Nist's documentation (see reference and resources at the end of this paper).

a big data environment.
In particular, the software that will be deployed in or consumed from the Cloud must be compatible with the most used operating systems and devices; it must be user friendly and stable. The Platform tools and middleware must be engineered to have the best performance when receiving and storing huge amounts of data. The Infrastructure must also be stable, easy to maintain, and through with future implementations in mind. Last, but not least, security is an important issue for all the three mentioned service models of the Cloud, even though it will not be thoroughly covered in this paper.

2 STATE OF THE ART AND THE ClouT APPROACH

The first step approached in the ClouT project has been to analyse all the solutions available in commerce that respond to the project requirements. The second step was to build a case study based on capturing information from a temperature sensor; through the Arduino device. In fact it was tested and selected also because there are many different Cloud providers compatible with it, allowing to use a web page to store and read data, and the implementation just requires modifying few lines of code.
In building the prototype, confirmation was received that Arduino is indeed friendly and useful for the purposes of the experiment.
Below an analysis of the components and the technology that were planned to be used in the first phase.

2.1 Arduino

Arduino is an open-source electronics prototyping platform, available to create interactive objects or environments, easy to use (hardware and software), as shown in a simple schema in Fig.1. The microcontroller on the board must be programmed using its own language that allows the developer, for example, to read a value coming from a sensor in a specific time interval.

Arduino projects can be stand-alone or can communicate with software on a computer; in this case a stand-alone project with a dedicated Arduino's Ethernet Shield, which allows networking, was opted for making outgoing connections to a dedicated storage for the data coming from the sensor; in this way, for this case study, an Ethernet Shield and the chosen sensor was needed to be connected to the Arduino (See Fig.2 below).

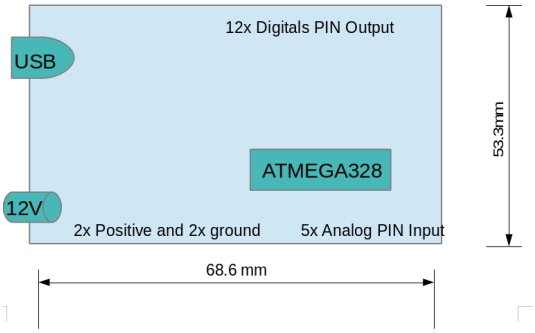


Figure 1: Arduino (Simple schema).

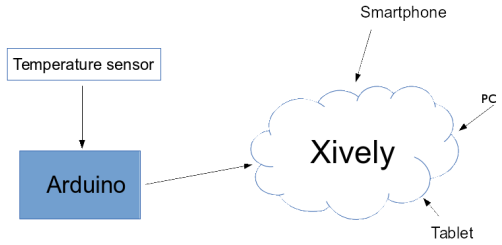


Figure 2: Sensor-Arduino-Xively.

2.2 Xively

Xively is a public Cloud for the Internet of Things that permits to connect different private or public devices like Arduino with dedicated API keys, in order to allow the user to put and get the data coming from a device, and have simple access to it from the website with a smartphone, a tablet or a computer, as shown in Fig.3 below.

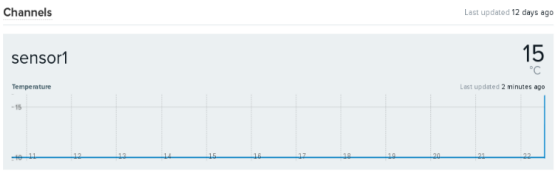


Figure 3: Xively (an example of visualization of the data sent).

To carry out the example mentioned before, it is only needed to: sign in on the Xively website, add a new project, specify the utilization of the Arduino, save the received APIKEY and FEEDID keys, and modify the example code on the "File - Example - Ethernet - Pachube Client"[3].

[3]Example of the Arduino program for a simple access data to the Cloud, created on 15 March 2010, modified on 9 Apr 2012 by Tom Igoe with input from Usman Haque and Joe Saavedra, http://arduino.cc/en/Tutorial/PachubeClient; this code is in the public domain.

2.3 Nimbits

Like Xively there is also Nimbits made by Google; both can be used to collect data, because they are all based on the same structure and logic: the data coming from the sensor are all processed, and then sent to the Cloud's services through SQL. The data are collected into a storage, through a web page and afterwards they can be visualized on a thin (tablet or smartphone) or thick (PC) client.

2.4 Raspberry Pi

Similar to Arduino, as well, it is also available an open-source device named Raspberry Pi, that can connect and use both Xively and Nimbits services. In order to use it, an Operating System (OS)[4] must be installed onto an SD card. In that sense Raspberry Pi represents a real microcomputer while Arduino is a microcontroller; this characteristics gives the possibility to use both the devices.

2.5 Raspberry Pi and Arduino

The possibility of connecting the two devices and of installing the Arduino software tools inside the Raspberry Pi, can solve an important management issue represented for instance by the remote update of the distributed Arduino devices. In fact, as more then one Arduino microcontroller can be connected to one Raspberry Pi microcomputer, it is possible to modify its configuration through internet, i.e. without any cable connection. Others interesting solutions can be added like, for example, storing the data locally, remote monitoring of the hardware status and performance, changing the interval of data acquisition, and so on.
The Arduino and the Raspberry Pi devices can easily communicate with each other in different ways: they can communicate through Ethernet, Bluetooth or Wireless communication, but it is also possible to the Raspberry Pi to use an Arduino Shield, or just convert the I/O pin tension[5].

2.6 Raspberry Pi as a Data Storage

An interesting aspect could be to make the Raspberry Pi device such as to behave like a server and data storage, installing on it a NoSQL DB, like CouchDB, and connect it to the Arduino on internet; in this way one obtains a real "Internet of Things" communication. However we have to consider that such configuration does not provide a real Cloud service, like, for example, an Infrastructure as a Service model may. Therefore we have discarded this hypothesis, because the main object of the project is "infinity processing and storage capacity of data from trillions of things and people that are integrated via virtual services in the Cloud"; for this reason, we will employ a Virtual Machine (VM) running inside a private Cloud. The mentioned ability of the Raspberry Pi to store a limited amount of data could be useful for backup functions in case of a short interruption of the communication between the device and the VM.

3 MANAGING DATA STORAGE IN A NoSQL DB

This chapter described a practical example of how to make a real Infrastructure as a Service, and how to use it.
Among the different open-source solutions, it has been chosen and tested one that appeared to be one the most interesting: oVirt. It is an open-source Infrastructure as a Service solution that allows the client, using an interface (the oVirt engine, see Fig.4), to manage hardware nodes, storage and network resources, and to deploy and monitor virtual machines running on the data center.
Given that the main strength of the Cloud approach is in the optimisation of the hardware resources through virtualisation, enabling a meaningful cut on total costs of ownership, we decided to choose a VM, instead of a simple dedicated machine, as a server. Another really useful and powerful Cloud oriented virtualisation solution that we considered was OpenStack, but in our test case oVirt was fit for the task.
It is possible to manage your own VMs using oVirt, simply accessing a web interface known as User Portal (see Fig.4).
oVirt provides a high availability mechanism that permits the automatic migration of a VM to another hardware node in case of failure on the current hosting node. That happens without losing any data or information, and no manual action on the VM is needed.

In our test case we employed a VM with CentOS 6.4 as OS and CouchDB installed as database to store data; in this way it is possible to send a string through TCP/IP and store data on the NoSQL DB (in this case CouchDB). For performance reasons it is highly recommended to store data this way, and not, for example, to employ a mySql DB, especially since a "terabyte torrent" coming from trillions of devices is ex-

[4]Raspberry Pi can be operated by an optimised Linux OS chosen among list.

[5]Arduino needs 5V instead of Raspberry Pi 3.3V, the respective Pins must be connected.

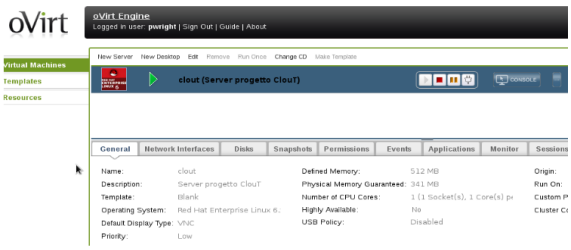


Figure 4: oVirt Engine.

pected. Once CouchDB is installed it is possible to have an idea on how to read and manage the data accessing an URL like: http://VM_ ADDRESS:5984/_utils/index.html, where WM_ ADDRESS is the IP ADDRESS of the VM previously made. To provide a unified way to access and store data we choose the CDMI standard, that offers a standard data interface installed in the Cloud. In our case we prefer to use a software called CDMI-Proxy.
Whatever the solution chosen, CouchDB or CDMI-Proxy, it is only needed to modify the sample code created for the Arduino to communicate with Xively, previously created[6], changing the server name, instead of "api.pachube.com", insert "your.VM.host.name", and specify port number 5984 if using CouchDB; otherwise, if you use CDMI-Proxy, insert port 8080 or 2365[7]. In addition, few code lines must be modified in the main sendData function, because CoachDB follows the JSON format (see official website guide for the format to use)[8].
At this point all the data, that has been automatically stored in the chosen DB, must be read. In order to do this, we have evaluated a well-known and standard procedure based on a web server package like httpd. In this case it requires a workload to carry out "manually", therefore there is a risk of making mistakes and reduces the security. Therefore, we examined other possibilities, with the aim of finding a solution that is still based on an open-source system, but is easier to implement and maintain, and, lastly, is more stable and efficient. ZABBIX, seems to respond to the above mentioned requirements.

[6]See paragraph on Xively

[7]Using one of these two ports (8080/2365), instead of the standard CouchDB port is useful for security reasons, but it is not enough to be safe from packet sniffing. In this case using SSL protocol to encrypt data is recommended.

[8](These conclusions, regarding the infrastructure side of how to use a NoSQL DB, need additional work on the platform side of the Cloud. Such work will be one of our next tasks.)

4 MONITORING WITH ZABBIX

All signals, coming from the variety of sensors installed in many different locations, must be captured, analysed, stored and represented in a quite understandable form and possibly, in case of alarms, in real time. It would be interesting, and quite useful for our purposes, if there is the possibility to control and monitor in real time all the items previously mentioned using a software available in the open source market. Therefore we have tested some solutions and finally found one, named ZABBIX, that can carry out the needed controls. It consists of two different elements in connection: the agent and the server. If, for example, we want to control all the sensors with a device like Arduino, and we want to connect more of these devices to only one microcomputer, that in our test is a Raspberry Pi, it sufficient to install ZABBIX agent on the Raspberry Pi and ZABBIX server on a Virtual Machine. The agent sends all the data to the ZABBIX server so that the data can be not only stored, but also shown on a map and monitored in a very easy way. In a test we made, the information shown was an alarm for an exceptional high temperature captured by some sensors connected to one of the devices.
The monitoring options can be added during the first installation, but they can be also modified later, remotely, by an administrator who can also add more users having each one different powers of access to the system. This feature allows not only the simple monitoring of the sensors, but also of the devices used, the Virtual Machine, the Raspberry Pi and all the network connected, as it is shown in the enclosed fig.5.

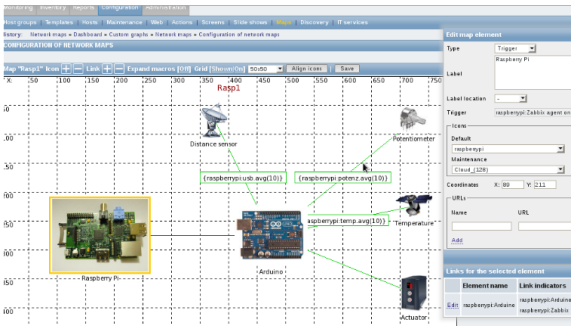


Figure 5: A screen of a web page interface of ZABBIX server running on Engineering's VM.

In Fig.5 there is also an actuator, which will be studied in the next ClouT project phases.
Another example of the data coming from a sensor is in Fig.6, where no filter was applied, and for this reason there are different range of values in spite of a

small time interval. The continue line is because the Raspberry Pi and the Arduino were connected, but, after some time, we manually stopped the code running on the Raspberry Pi.

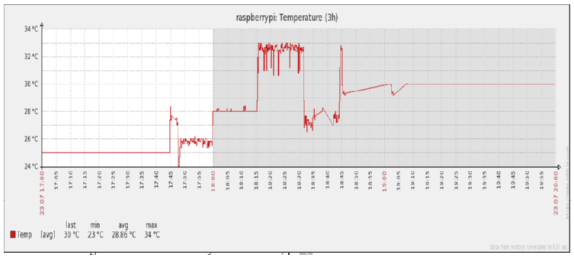


Figure 6: An example of a temperature graph of data coming from a sensor.

5 POSSIBLE APPLICATIONS FOR SMART CITIES

The ClouT project aims at empowering the citizen clout in smart cities.

What has been designed so far in the course of the project, and here briefly described, can cover many of the different features that are now included in what is called a Smart City. Reference is made to the possibility of collecting data, producing statistics, monitoring services, etc. and, for example, two applications aimed at improving the citizen's quality of life. All this is obtained by capturing data from many sensors and devices positioned in different places and each one devoted to a specific function. One application drafted regarded an intelligent fire alarm system, involving temperature sensors and web cams automatically switching on, in case of rapidly rising temperatures, and sending images of the scene to a command and control center or to supervisor, on his smartphone.

Another example of a possible and not expensive applications using the devices described in this paper, we can cite for instance the monitoring of people inflows and outflows from a room (i.e.: for security purposes or marketing), or, the presence of persons close to a particular area of a hall and the duration of their presence (i.e.: for marketing purposes; cultural purposes; etc.). Such data, can be easily provided also in time series format, with statistics and graphs.
The two applications have required an in-depth study and implementation, to simulate different situations, to detect all the problems that may occur, and, to make the application completely automatic.

The interesting feature, in our opinion, is that in using the suggested hardware, configuration and connections between the different technologies, it means to create the Internet of Things. The benefits of Cloud Computing are that the end users need not worry about the exact location of servers and they can switch to their application by connecting to the server on Cloud and start working without any trouble and no installation is required: just an internet connection and the right credentials. All the data are stored on a dedicated database, that permits big data storage, like the Cloud Platform allows.

Several other applications may be conceived to empower the citizens in smart cities: to improve mobility, to help social services, to save energy, etc.

6 CONCLUSION AND NEXT STEPS

As we wrote on the first chapter of this document, the first step we approached in the ClouT project was to analyse all the things previously implemented and existing in commerce. We started using open-source technology, hardware and software, because they are cheap and can be used for different purposes; but this document wants to be useful also using other similar technology, because it wants to be as general as possible. Our idea on using a gateway or a little device to control all the microcontrollers that collect data from different kind of sensors, makes it easy to control and to monitor the system as we can see on fig. 7.

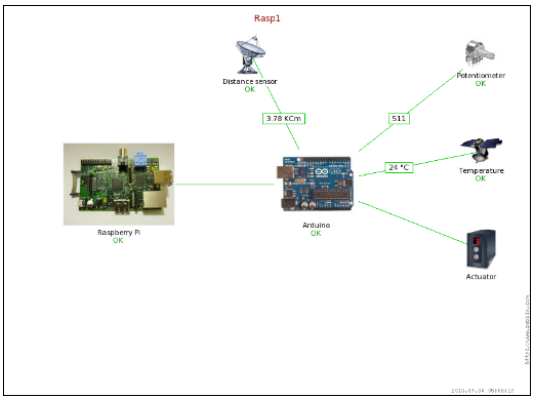


Figure 7: An example on the monitoring of the hardware used.

All the technologies used are already existing, but all their functions are not used at all.
The prototype we have implemented, has also given us the opportunity to understand that the infrastructure can be applied for the production of applications in domotics, medical environment, etc. In addition, the prototype showed us that it can manage a large multitude of sensors, like those obtained from social

network. Another possibility that came out during our experiments is that the Arduino device has demonstrated its ability to read sensors data in real time, while the Raspberry Pi device has greater potentialities because it hosts an operating system. Such features can be used to manage and update the software in a high number of distributed Arduino devices.
Therefore both devices can provide a useful tool for future implementations.
In conclusion, the project is still in the first phases, but since now a new and ample number of applications can be defined and implemented, on the basis of the described architecture.

ACKNOWLEDGEMENTS

Special thanks to our colleagues Marco Emanuel Palazzotti and Daniele Pavia for their contribution.

REFERENCES

Arduino, *official website*, http://www.arduino.cc.

M. Banzi (2011), *"Getting Started with Arduino"*, O'REILLY.

Nist, *National Institute of Standards and Technology*, http://csrc.nist.gov/publications/_ nistpubs/800-145/SP800-145.pdf.

Snia, *Cloud Data Management Interface (CDMI)*, http://cdmi.sniacloud.com/.

CDMI-Proxy *provides CDMI-compliant proxy server to public cloud backends*, https://github.com/livenson/vcdm.

C. Pfister (2011), *"Getting Started with the Internet of Things"*, O'REILLY.

Cosm, *official website*, http://www.xively.com.

Raspberry Pi, *official website*, http://www.raspberrypi.org.

Nimbits, *official website*, http://www.nimbits.com.

oVirt, *official website*, http://www.oVirt.org.

CouchDB, *official website*, http://couchdb.apache.org/.

Zabbix, *official website*, http://www.zabbix.com/.

OpenStack, *official website documentation*, http://docs.openstack.org.

IoT-A, *Internet of Things Architecture*, http://www.iot-a.eu/public.

C. Doukas (2012), *"Building Internet of Things with the Arduino"*.

K. Ashton (2011), *"That 'Internet of Things' Thing"*. RfiD Journal.

ZigBee, *as a 6LoWPAN, low power wireless, IEEE 802.15.4*, http://www.zigbee.org/Standards/_ZigBeeNetworkDevices/Overview.aspx.

T. Igoe (2011), *"Making Things Talk"*. O'REILLY.

A Framework for Ambient Computing

Mohammed Fethi Khalfi and Sidi Mohamed Benslimane
Computer Science Department, University of Djilali Liabes, Sidi Belabess, Algeria
_fethi_khalfi@yahoo.fr, sm@univ-sba.dz

Keywords: Pervasive Information Systems, Ambient Intelligence, Ubiquitous Computing, Smart Space.

Abstract: The proliferation of mobile computing and wireless communications is producing a revolutionary change in our information society. Ubiquitous Computing is a recent paradigm whose objective is to support users in accomplishing their tasks, accessing information, or communicating with other users anytime, anywhere. In other terms, Pervasive Information Systems (PIS) constitute an emerging class of Information Systems where Information Technology is gradually embedded in the physical environment, capable of accommodating user needs and wants when desired. PIS differ from Desktop Information Systems (DIS) in that they encompass a complex, dynamic environment composed of multiple artefacts instead of Personal Computers only, capable of perceiving contextual information instead of simple user input, and supporting mobility instead of stationary services. In this paper, as an initial step, we present PIS novel characteristics compared to traditional desktop information systems; we explore this domain by o ering a list of challenges and concepts of ubiquitous computing that form the core elements of a pervasive environment. As a result of this work, a generic framework for intelligent environment has been created. Based on various and related works concerning models and designs. This framework can be used to design any PIS instance.

1 INTRODUCTION

Mark Weiser was the first to describe the vision of ubiquitous computing, which has as its goal the enhancing computer use by making many computers available throughout the physical environment, and making computers e ectively invisible to the user (Weiser, 1991). The essence of Weisers vision is that persons use many computers embedded in the environment, allowing technology to recede into the background. The first era was defined by the mainframe computer, a single large time-shared computer owned by an organization and used by many people at the same time. Second, came the era of the PC, a personal computer primarily owned and used by one person, and dedicated to them. The third era, ubiquitous computing, representative of the present time, is characterized by the explosion of small networked portable computer products in the form of smart phones, personal digital assistants (PDAs), and embedded computers built into many of the devices we own resulting in a world in which each person owns and uses larger numbers of computers becoming integrated into everyday life. In addition communications extend beyond the classic concept man to man or man to machine, to include direct communication between machines, (Fig. 1).

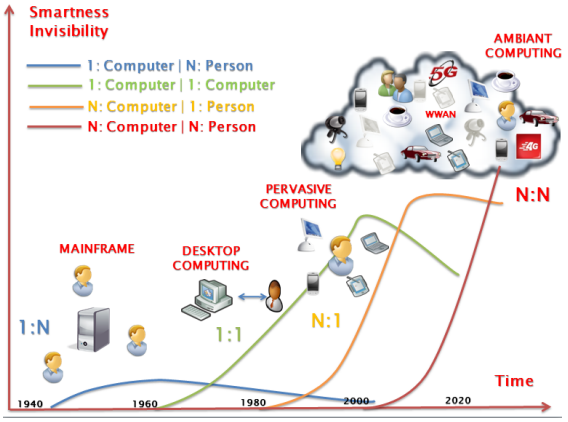


Figure 1: Evolution of computers, from the beginning to the ambient computing.

The remarkable recent progress in embedded devices, smart phones, wireless communications and networking technologies (Fig. 2) has enabled us to create pervasive computing systems and services with diverse applications and global accessibility (Escoffier, 2008), and promote their mobility in a transparent way without the explicit user intervention. This technological progress offers an opportunity to focus on its main task instead of configuring and managing all IT equipment at their disposal and ac-

cess to various services offered by these objects, anywhere, at any various devices (Cheikh, 2012), (Saha and Mukherjee, 2003), (Satyanarayanan, 2001).



Figure 2: Wireless networks.

This paper is organized as follows, after reviewing the state of the pervasive computing. Section 2 formulates the definition and evolution of information systems and the main constraints of pervasive information systems, after we present PIS novel characteristics compared to traditional desktop information systems (DIS), Section 3 identify and describe fundamental challenges, properties and characteristics of ubiquitous computing environments that form or are part of those environments. Section 4 reviews the background and related works for different architectural models of pervasive systems. In section 5, we describe out the proposed system architecture. Finally, we conclude this paper in section. 6, ans we described future possibilities works.

2 ISSUES AND CHALLENGES OF PERVASIVE COMPUTING:

Based on an analysis of literature in the field of ubiquitous computing we present an overview of fundamental properties and characteristics of ubiquitous computing.

2.1 Enabling Concepts and Technologies:

2.1.1 Wireless Networks

Wireless networks infrastructures can form the platform, which enable clients to transparently connect and share context with remote entities. The geographic scope defines the coverage capacity of a wireless network; we can distinguish several categories, (Fig. 3).

2.1.2 Wireless Sensor Networks (WSN)

consist of a large number of tiny devices called sensor nodes that pervade an area and collaborate together

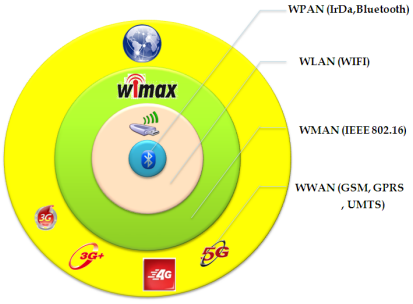


Figure 3: Pervasive Wireless Network.

to collect the information necessary to adjust system behavior, pervasive systems usually integrate wireless sensors, which consist of small devices capable of sensing, processing, and communicating different types of sensory data. A WSN comprises a number of components, and though the terminology may change according to different architectures, (Fig. 4):

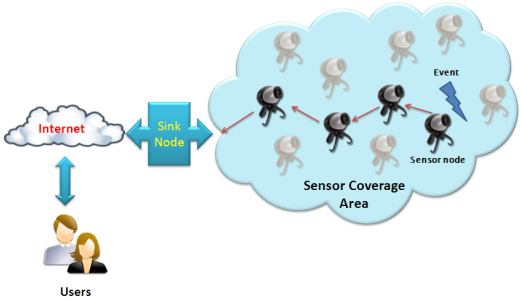


Figure 4: Wireless Sensor Networks.

2.1.3 Pervasive Access Devices

Pervasive access devices constitute the front end of PIS (Senn, 2007; Mattern and Sturm, 2003) and are likely to contain a multitude of different device types that differ in size, shape (more diverse, ergonomic, and stylistic), and functional diversity (mobile phones, laptops, pagers, PDAs) (fig.5, fig.6). In essence, pervasive devices dictate the interaction between the user and the pervasive system (Beigl et al., 2003). A major requirement for participation of a device in a pervasive environment is connectivity. Devices may include one or more connectivity options depending on their functionality.



Figure 5: Aibo7.



Figure 6: Nabaztag.

2.1.4 Human-Computer Interaction (HCI)

Is the interaction between users and computers via the user interfaces (Dnovan, 2010; Gallissot, 2012). HCI with ICT systems has conventionally been structured using a few relatively expensive access points. This interaction primarily uses input from Keyboard and pointing devices which are fairly obtrusive to interact with. Nowadays, Interaction is via natural user interfaces and physical interactions that can involve gestures input, multiple touch, voice control, eye gaze control, etc. It can improve performance and user experience through anticipated actions and user goals in relation to the current context, past user context and group context.

2.2 Core Properties of Pervasive Information Systems

2.2.1 Distributed Systems

Are considered the core of pervasive systems (Siddiq and Ali, 2010). It consists of multiple autonomous computers that communicate through a computer network (Bourcier), each device may have its own user with individual needs, and the purpose of the distributed system is to coordinate the use of shared resources or provide communication services to their users (Tanenbaum and M. Van Steen, 1999). A distributed system can operate across different homogeneous environments (Li, 2010; Dugénie; Petit) and seamlessly integrate devices with environments.

2.2.2 Mobility

Will be an important characteric of a ubiquitous system (Saadi, 2009; Louberry, 2010). There are, however, different kinds of mobility schemes, such as terminal mobility, personal mobility, session mobility and service mobility. Users shall be supported in such a way that they can move from one place or terminal to another and still get a personalized service (Tigli et al,. 2009; Hoareau, 2007; Reignier, 2010; Kouici).In the future, the networks may also be mobile and dynamic, and therefore, full mobility is an essential requirement in the ubiquitous wireless world.

2.2.3 Interoperability

Is one of the most essential requirements of ubiquitous software. Today, application developers' use a wide ranges of programming models, languages and development environments (Vallecillo et al.). The need to network the embedded products of different vendors with cyber world applications is increasing in the ubiquitous environment. the infrastructure for pervasive computing must support diverse types of software component. Applications in pervasive computing environments will be required to respond to novel tasks and situations, applications will increasingly be formed dynamically from available software components. This will require dynamic interoperability at the component level, in addition to interoperability that overcomes the heterogeneity of the environment and of components.

2.2.4 Scalability

Is also crucial in pervasive computing scenarios (Duboc et al,. 2006) as smart environments may lead to highly intensive interactions between the computing infrastructure and the user space (Escoffier, 2008; Najar et al., 2012; Sarr, 2010). This, in tum, can have serious implications for the user with regards to computing and network resources such as bandwidth, memory and energy. The ever increasing levels of communication and computing interactions between different users also adds to the pressure on resources (Bourcier) and making scalability an absolutely critical requirement for pervasive systems (André; Sancho, 2010).

2.2.5 Heterogeneity

Devices present in pervasive environments can be of various kinds (Duboc et al,. 2006; Shaout and Srinivasan, 2009; Jouve, 2009): simple devices (light or clock), mobile assistant devices (PDA or smartphones), multimedia devices (PC or TV), etc. These devices have different hardware and are based on various operating systems (Flissi et al., 2005; Ferry et al,. 2008; Louberry, 2010). They can have different communication technologies (wired/wireless) (Bourcier), and implement different network protocols (UPnP, Jini, SLP, GSM). Thus, we need to handle this heterogeneity and provide users with an interoperable system. In the same network, computers with large storage can coexist with other devices with limited resources (Cheung-Foo-Wo, 2009; Hoareau, 2007).

2.2.6 Integration

Pervasive computing systems by its nature require integration of many different subsystems (Saha and Mukherjee, 2003) with very different characteristics. These subsystems include computational facilities, communication devices, mechanical or chemical sensors and actuators, smart appliances, and existing control systems.

2.2.7 Dynamicity

The mobility of some pervasive devices and the limited resources of others increase the dynamics of pervasive computing environments. This dynamics is perceived in terms of the number and lifetime of pervasive functionalities a user can access to at a specific time and location. (Najar et al., 2012) In particular, new devices may appear in the environment while other devices may become out of reach due to a lack of resources (e.g., battery down), or due to the range of radio transmissions.

2.2.8 Autonomy

Is a property of a system that allows itself to organize and operate its own actions without human intervention (Bantz et al., 2003; Wikipedia, 2010). The Ubiquitous Computing systems are naturally autonomous, because it is always self-governing and capable of its own independent decisions and actions. The system can operate without human intervention (Sarr, 2010), and both the input/output and computation of the System are completely embedded in the device it controls.

2.2.9 Proaction

The system needs to be self-triggered to capture a priori what its users want in order to increase the overall quality of service (Laforest, 2007).

2.2.10 Invisibility

is another key requirement for pervasive computing. In the ideal sense, invisibility implies that the underlying technology is completely hidden from the user's perspective (Liu, 2006). In real environments, invisibility can exist only in the approximation sense. This could be facilitated by smart environments that constantly adapt to meet the user's requirements with minimum user intervention. This problem becomes more difficult when the user is in a dynamically changing environment.

2.2.11 Context Awareness

One of the most important novel characteristics that PIS introduce is the notion of context awareness. This term was formally defined and used for the first time by (Schilit et al., 1994; Brown et al., 1997) to describe applications that "adapt according to their location of use, the collection of nearby people and objects, as well as the changes to those objects over time". (Abowd et al., 1999) Define context-aware computing as "a system that uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task". (Pascoe, 1998) de?ne context as the subset of physical and conceptual states of interest to a particular entity. (Jameson, 2001) extend the previous definitions by adding the user's behavior and current interactions with the pervasive system. (Laerhoven and Kofi, 2001) emphasize the importance of sensors embedded in the environment in order to sense the location and current movement of the user and add it to the properties of a context-aware system. (Abowd et al., 1999) approach context as the user's emotional state, focus of attention, location and orientation, date and time, objects, and people in the user's environment. Other definitions have simply provided synonyms for context, referring, for example, to context as the environment or situation. (Cousins and Varshney, 2009) defines context as the elements of the user's environment that the user's computer knows about. The most generic de?nition for the features of a context-aware system has been provided by (Dey, 2001): "a system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.

2.2.12 Context Management

Users in pervasive environments must access to device functionalities. Moreover, they need a representation of the different elements present in the environment (Dey, 2001). The first requirement of a pervasive system thus is the context management. It demands to represent the context enable users to get an overview of functionalities and to enable users to manage this representation. Moreover, context management also involves to be continuously aware of environmental changes and to relate this according to the context representation.

2.2.13 Adaptability

Adaptation is required in order to overcome the intrinsically dynamic nature of pervasive computing. Mobility of users, devices and software components

can occur, leading to changes in the physical and virtual environments of these entities. Moreover, applications can be highly dynamic, with users requiring support for novel tasks and demanding the ability to change requirements on the fly (Bourcier). It should be the role of the infrastructure for pervasive computing to facilitate adaptation, which may involve adapting individual software components and/or reconfiguring bindings of components by adding, removing or substituting components. Dynamic adaptation can involve complex issues such as managing the adaptation of software components that are used simultaneously by applications with different (and possibly conflicting) requirements, and maintaining a consistent external view of a component that has behavior that evolves over time.

2.2.14 Security

The goal of security is also to guarantee the well functioning of pervasive computing systems. Ubiquitous computing has been designed in order to be deployed anywhere and be accessed by the majority of users. The user mobility and the proliferation of lightweight devices make complex security problems. Recent research on pervasive computing focuses on building infrastructures for managing smart spaces, connecting new devices, and providing useful applications and services. Privacy, trust, and security issues in such environments, Contextual data are generally personal data are static data such as user preferences and habits, and dynamic data such as its location and the tasks it performs. Thus, these data have confidentiality and exchange of data between these applications must comply with the security measures defined by the user.

2.2.15 Service Discovery

Is an important and challenging issue in pervasive environments. Service discovery is the task of locating which services are available (Brown, 1996), usually followed by a selection activity whose main goal is the choice of the most appropriate (Zhu et al., 2005), or best solution based on user-defined metrics (e.g. accessibility, cost, available bandwidth, load, etc.) in the presence of multiple instances of the same service scattered throughout the network.

2.2.16 Quality of Service Management

(UIT) describes the quality of service as a set of requirements on the collective behavior or multiple objects. The aim is to provide users with applications that work best and as long as possible on their favorite mobile device and that whatever the changes in the environment (Louberry, 2010).

3 RELATED WORK

In this section, we briefly present some research projects related to this topic, which are also based on the ubiquitous concepts. We are positioning our contribution in relation to this work. Though much progress has been achieved, current architectures provide very few of abstraction and often generic and limited support. Several studies on the architecture of pervasive systems are proposed, even if they differ, all of these works are in fact complementary. (Satyanarayanan, 2001; Moitra, 2004; Afyouni, 2010; Achour et al., 2012; Em and Yoo, 2005; Siddiq and Ali, 2010; Cousins and Varshney, 2009; Conti et al., 2012; Zhou et al., 2010) Adopt an architecturally model based on new technologies (infrastructure networks and communicating devices). (Saha and Mukherjee, 2003) detailed description of ubiquitous computing, In their opinion, the sensors provide information about the context of ubiquitous systems, what makes them different from traditional systems. (Saha and Mukherjee, 2003) Introduced a model of the pervasive computing environment by differentiating between devices, pervasive networking, pervasive middleware, and pervasive applications. (Satyanarayanan, 2001) explains that ubiquitous computing based on distributed systems and mobile computing, adding features such as smart spaces and invisibility. (Hoh, 2006) In his initial work presents a user-centric model. The authors (Cheung-Foo-Wo, 2009; Afyouni, 2010; Achour et al., 2012; Hoh, 2006) illustrate the importance of smart spaces where digital and physical world are related in a natural and transparent to the user. Work (Satyanarayanan, 2001; Afyouni, 2010; Em and Yoo, 2005; Siddiq and Ali, 2010) integrate the field of intelligent interfaces that allows users to control and interact with objects intuitively. Finally (Moitra, 2004; Em and Yoo, 2005; Hoh, 2006) address the security which is a major challenge in building pervasive environment.
Most of the approaches proposed in the literature are specific to an application or to a particular domain. They are not sufficiently generic to be reusable in other domains. The key contribution of the paper is a reference model architectural has been proposed for analyzing the functionalities and key of pervasive computing. Tables 2 offer an overview of the proposals that have inspired our proposed model.

Table 1: Summaries of research work, (-) and (+) correspond to an absence and a presence of the layer

	HCI	Devices	Smart Space	Infrastructure	Security
Debashis SAHA, 2003	+	+	-	+	-
Punnet GUPTA, 2004	-	+	+	+	+
Natalia V,Em, 2005	+	+	-	+	+
S Hoh, 2006	-	+	+	+	-
Karlene C, 2009	-	+	-	+	-
Shahid SIDDIQ, 2010	+	+	-	+	+
LAYFOUNI, 2010	+	+	+	+	-
Jiehan ZHOU, 2010	-	+	-	+	-
Fatma ACHOUR, 2012	-	+	+	+	-
Marco CONTI, 2012	-	+	+	+	+
Our model	+	+	+	+	+

4 ARCHITECTURAL MODEL PROPOSED

The global architecture of PIS is illustrated in Fig. 7. It is based on five layers. The smart interaction Layer is called for to combine many individual activity interactions for smart devices and smart environments to support the core properties of Ubiquitous Computing. In other words smart interaction is a unified and continuous interaction model between Ubiquitous Computing applications and their Ubiquitous Computing infrastructure, physical world and human environments. The smart devices layer can be considered as a single entry to access sets of popular multiple application services on the device or remotely on servers. Most of them usually have one specified user. The smart environment layer comprises a set of networked devices connected to the physical environment. Different from smart devices, the devices including a smart environment normally complete a single pre-defined task. Integrated environment components automatically react to or anticipate user interaction using iHCI (implicit human computer interaction). The infrastructure layer contains all physical hardware such as devices and network equipment. This layer collects the raw data which are provided by the physical devices. The security layer must ensure basic security services. Privacy refers to the protection of the users' identities and information from non-authorized parties. Confidentiality is required to protect the users' information in the whole system, access control; it is a restriction to access or performance of an action on some resources, Authentication: used to identify and authenticate an entity in order to verify its source or to estimate the trust level that it can be granted.

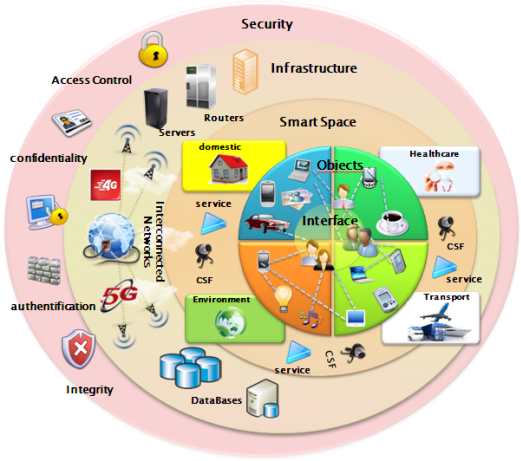


Figure 7: Our system architecture for pervasive computing.

4.1 Infrastructure Layer

Represents the level of communication channels, data and sensor networks, and broadband internet access, but also the intelligent design of databases, storage, and network applications (Fig. 8). The term infostructure reflect the change in complexity over the traditional view of infrastructures, but also to emphasize the increased interdependency with the other levels of the value chain. Pervasive infostructure is all about access to information and services beyond the traditional client-server paradigm. The merger of computing and communications is leading the way towards pervasiveness, and infrastructures are the level where we place the property of this merger. System infrastructure provides the systems needed for using the services (Cheung-Foo-Wo, 2009). Infrastructure providers have advantage about ubiquitous computing as products and services are provided through multiple channels (Moitra, 2004).

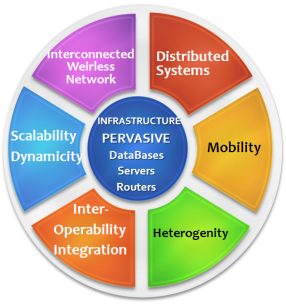


Figure 8: Layer Infrastructure for the Intelligent Environment.

4.2 Smart Devices Layer

Smart devices, personal computer, mobile phone, tend to be multipurpose ICT devices (Sartor), operating as a single portal to access sets of popular multiple application services that may reside locally on the device or remotely on servers. There is a range of forms for smart devices. Smart devices tend to be personal devices, having a specified owner or user. In the smart device model, the locus of control and user interface resides in the smart device. Devices are often designed to be multi-functional because these ease access to, and simplify the interoperability of, multi functions at run time. The main characteristics of smart devices are as follows (Fig. 9).

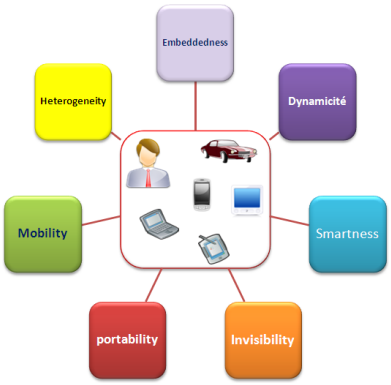


Figure 9: The main characteristics of smart devices.

4.3 Smart Interface Layer

Smart Interaction is called for to combine many individual activity interactions for smart devices (Miraoui, 2009) and smart environments to support the core properties of Ubiquitous Computing. In other words, smart interaction is a unified and continuous interaction model between Ubiquitous Computing applications and their Ubiquitous Computing infrastructure, physical world and human environments.

4.4 Smart Space Layer

A smart environment comprises a set of networked devices connected to the physical environment (Satyanarayanan, 2001). It allows inhabited services and tasks with different objects available (Fontaine, 2006). The collection of information from the physical environment system using communicating objects via using iHCI (implicit human computer interaction) or objects capable of capturing information (sensors). The collected information is then interpreted, filtered and aggregated by various applications to enrich the contextual information in order to provide appropriate services. The intelligence of the systems depend on their way to exploit the context data acquired. The data received from the environment and the user must be interpreted to perform the appropriate action (Fig. 10).

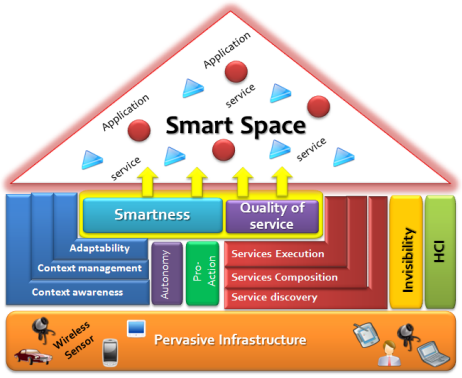


Figure 10: Smart Space.

4.5 Security layer

In pervasive computing, services are more open, accessible, distributed, and close to the user. This proximity introduces new threats and vulnerabilities for the systems. Now, systems and services are assimilated to supermarkets open for everybody (Bourcier), in which merchandise is directly accessible to clients in a self-service way. Of course, such displayed goods are vulnerable and can be easily stolen, but new sophisticated guard systems appeared to protect the merchandise. new techniques were developed to adapt the security requirements to the environment.

5 CONCLUSION AND PERSPECTIVES

In this paper, we have validated our approach in the domain of pervasive computing from a complete domain analysis. Based on the concepts of Ubiquitous Computing, we define a list of challenges and concepts that form the core elements of a pervasive environment. Pervasive information systems (PIS) constitute an emerging class of information systems (IS) where information technology (IT) is gradually embedded in the physical environment (Kouruthanassis and Giaglis, 2006). Therefore, PIS introduce the property of context awareness as a result of the pervasive artifacts capability to collect, process, and manage environmental or user-related information on a real-time basis. other research issues that we would like to work on are: context awareness in pervasive computing.

REFERENCES

Mark Weiser, The computer for the 21st century, *Scientic American*, tome 265, nř 3, pages 66-75, septembre 1991.

Ansem Ben Cheikh, E-CARe: une méthode d'ingénierie des Systèmes d'Information ubiquitaires, 2012.

Clement Escoffier, iPOJO: Un modèle à composant à service flexible pour les systèmes dynamiques, 2008.

Debashis Saha and Amitava Mukherjee, "Pervasive computing: a paradigm for the 21st century", *IEEE Computer*, 25-31, March 2003.

M. Satyanarayanan, Pervasive Computing: Vision and Challenges IEEE Personal Communications, 2001.

Puneet Gupta Deependra Moitra, Evolving a pervasive IT infrastructure: a technology integration approach, 2004, p33.

Cristiano da Costa André, A context aware service based software infrastructure for pervasive computing.

Adnan Shaout, Hari Srinivasan , Framing Pervasive Computing, 2009.

Imad Afyouni, Indoor space modelling, mobility, and positioning in ubiquitous information systems, 2010.

Jinshan Liu, Découverte de services sensible à la qualité de service dans les environnements de l'informatique diffuse, 2006.

Salma Najar, Manuele Pinheiro, Luiz Angelo and Carine Souveyet, Analyse des Mécanismes de Découverte de Services avec Prise, 2012.

Fatma Achour, Anis Jedidi and Faiez Gargouri, The Generic Model for Pervasive Information System, 2012.

Daniel Cheung-Foo-Wo, Adptation dynamique par tissage d'aspects d'assemblage, 2009.

Moeiz Miraoui, Architecture logicielle pour l'informatique diffuse, 2009.

Natalia V. Em, Keedong Yoo, Extracting requirements for ubiquitous computing technology-based is deploying factor analysis, 2005.

Kouruthanassis, P. E. and Giaglis, G. M., A design theory for pervasive information systems, 3rd Int. *Workshop on Ubiquitous Computing*, pp. 62-70, 2006.

Wilfried Jouve, Approche déclarative pour la génération de canevas logiciels dédiés à l'informatique ubiquitaire p7, 2009.

Didier Hoareau, Composants ubiquitaires pour reseaux dynamiques, p8-9, 2007

L. Duboc, et al., "A framework for modelling and analysis of software systems scalability," published in the proceedings of the *International conference on Software Engineering*, 2006, pp 949-952.

D. F. Bantz, C. Bisdikian, D. Challener, J. P. Karidis, S. Mastrianni, A. Mohindra, D. G. Shea and M. Vanover, "Autonomic personal computing". *IBM Systems Journal*, 42(1):165-176, 2003.

A. K. Dey, "Understanding and Using Context". Personal and Ubiquitous Computing, 5(1):4-7, 2001.

Boxiao Li, Design and Devleopment of an "Intelligent Environment", p10-11, 2010.

Jared Dnovan, Human computer interaction, 2010.

Wikipedia. Autonomy. http://en.wikipedia.org/wiki/Autonomy, 2010.

P. Senn, "Objets communicants et nanotechnologie", *Journées Scientifiques du CNRFS Nanosciences et Radioélectricité*, 2007.

F. Mattern and P. Sturm, "From Distributed Systems to Ubiquitous Computing - The State of the Art, Trends, and Prospects of Future Networked Systems", In Klaus Irmscher and Klaus-Peter Fähnrich, editors, Proceedings KIVS, pp. 3-25, Springer- Verlag, February 2003.

M. Beigl, A. Krohn, T. Zimmer and Christian Decker, Situation Aware Context Communication", *International conference on Ubiquitous Computing*, October 12-15, Seattle, USA, 2003.

A. S. Tanenbaum and M. Van Steen. Distributed Systems: Principles and Paradigms. Prentice Hall, 1999.

Idrissa Sarr, Routage des Transactions dans les Bases de Données à Large Echelle, 2010.

Rachid Saadi, The Chamelon: Un système de sécurité pour utilisateurs nomades en environnement pervasif et collaboratif, 2009, p2-15.

Jean-Yves Tigli, Stéphane Lavirotte, Gaëtan Rey, Vincent Hourdin, Daniel Cheung- Foo-Wo, Eric Callegari, and Michel Riveill. WComp Middleware for Ubiquitous Computing: Aspects and Composite Event-based Web Services. *Annals of Telecommunications (AoT)*, 64(3-4):197-214, Apr 2009.

F. Zhu, M. W. Mutka and L. M. Ni. Service Discovery in Pervasive Computing Environments. IEEE Pervasive Computing, 4(4):81-90. *IEEE Educational Activities Department*, 2005.

Frédérique Laforest, De l'adaptation à la prise en compte du contexte Une contribution aux systèmes d'information pervasifs, p10, 2007

Areski Flissi, Christophe Gransart and Philippe Merle, Une infrastructure à composants pour des applications ubiquitaires. UbiMob 2005: 45-48.

Nicolas Plouznikoff, Jean-Marc Robert, Applications Ubiquitaires, Caractéristiques, enjeux et défis de l'informatique portée, 2005.

Antonio Vallecillo, Juan Hernandez and José M. Troya, Component Interoperability.

Nicolas Ferry, Stéphane Lavirotte, Gaëtan Rey and Jean-Yves Tigli, Adaptation Dynamique d'Applications au Contexte en. *Informatique Ambiante*, 2008.

Germán Sancho, Adaptation d'architectures logicielles collaboratives dans les environnements ubiquitaires. Contribution à l'interopérabilité par la sémantique, 2010, p20.

Christine Louberry, Adaptation au Contexte pour la Gestion de la Qualité de Service,2010.

UIT. X.902, Technologies de l'information, traitement reparti ouvert, modèle de référence fondements. Recommandation x.902 (11/95), *Union Internationale des Télécommunications*.

Patrick Reignier, Intelligence Ambiante Pro-Active de la Spécification à l'Implémentation. 2010.

Nabil Kouici, Gestion des déconnexions pour applications réparties à base de composants en environnements mobiles.

Johann Bourcier, Auto-Home: une plate-forme pour la gestion autonomique d'applications pervasives, 94.

Pascal Dugénie, Espaces collaboratifs ubiquitaires sur une infrastructure à ressources distribuées, p52.

Mathieu Petit, Approche spatiale pour la caractérisation du contexte d'exécution d'un système.

Fabien Sartor, Modélisation de l'interopérabilité d'objets communicants et de leur coopération Application à la domotique.

Shahid Siddiq and Aasim Ali, Gap Analysis between Ubiquitous Computing Requirements and Features of an Open Source Operating System (Openmoko) for Hand Held Devices, *International Journal of Electrical & Computer Sciences IJECS-IJENS* Vol: 10 No: 05, 2010.

Mehdi Loukil, Gestion de contexte pour l'optimisation de l'acces et l'adaptation des services sur des environnements heterogenes, 137, 2012.

Mathieu Gallissot, Modéliser le concept de confort dans l'habitat intelligent: du multisensoriel au comportement, 2012, p16.

Emeric Fontaine, Programmation d'espace intelligent par l'utilisateur final, p12, 2006

Schilit, N. B., Adams, N. and Want, R., (1994). Context aware computing applications. In *Workshop on Mobile Computing Systems and Applications*. Sta Cruz, EtatsUnis, décembre 1994. p 8590.

Brown, J. P., Bovey, D. J. and Chen, X., (1997). Contextaware Applications: from the Laboratory to the marketplace. In Institute of Electrical and Electronics Engineers (IEEE) Personnal Communications. v. 4, p. 5864.

P. J. Brown. The Stick-e Document: a framework for creating Context aware applications. In Electronic Publishing 96, (1996), Document Manipulation and Typography, 24-26 September 1996, Pala-Alto, California, USA, pp.259-272

Pascoe. Adding generic contextual capabilities to wearable computers. In Proceedings of *2nd International Symposium on Wearable Computers*, October 1998, pp. 92-99

Kristof Van Laerhoven and Kofi Aidoo, Personal and Ubiquitous Computing vol. 5 issue 1 February, 2001. p. 46 - 49.

Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., and Steggles, P., 1999. Towards a Better Understanding of Context and Context- Awareness (*Lecture Notes In Computer Science*, Vol. 1707, pp. 304- 307), 1999

Karlene C. Cousins and Upkar Varshney Designing Ubiquitous Computing Environments to Support Work Life Balance, 2009,117-122.

S. Hoh, J. S. Tan, M. Hartley, Context-aware systems a primer for user-centred services, BT Technology Journal 186 o Vol 24 No 2 o April 2006

Marco Conti, Sajal K. Das and Chatschik Bisdikian, Looking ahead in pervasive computing: Challenges and opportunities in the era of cyber-physical convergence, Pervasive and Mobile Computing 8 (2012) 2-21.

Jiehan Zhou, Ekaterina Gilman, Mika Ylianttila and Jukka Riekki, Service Computing: Visions and Challenges, 2010 *10th IEEE International Conference on Computer and Information Technology*, p1335-1339.

Anthony Jameson, "Modeling Both the Context and the User", Personal Technologies, 5 (2001).

Construction and Verification of the Trusted Cloud Service

Dexian Chang[1,2], Hongqi Zhang[1,2] and Yingjie Yang[1,2]
[1]*Zhengzhou Institute of Information Science and Technology, Zhengzhou, China*
[2]*Henan Key Laboratory of Information Security, Zhengzhou, China*
changdexian@126.com

Keywords: Trusted Cloud Service, Trustworthy Property, Formal Verification, Logic of Secure System.

Abstract: Trust assurance of cloud service is a central focus of the application of the cloud computing. We present a novel approach for constructing trusted cloud service based on the trusted computing technology, which is able to provide reliable data and service for the cloud users. Utilizing the extended LS^2, we model the programs loaded in the trusted cloud and reason its trustworthy property with the invariants. Result shows the validity and high efficiency of our approach.

1 INTRODUCTION

Currently, cloud computing is changing the landscape of corporate computing and is developing rapidly with its special technology and service model. Investigation shows that more than 25% business organizations hope to transplant their business into the cloud architecture(Research, 2013). Among the commercial cloud computing models, the IaaS (Infrastructure as a Service) cloud has more advantages for the large enterprises, because this cloud model provides resources as the virtual machines (VMs) and supports the complex application environments for cloud users.

However, during the process of using IaaS cloud services, users can not control the data in the cloud computing environment as before. Due to the multi-tenant attribute of cloud computing environment, it is easy to cause the leakage of user data. Recently massive data loss events in the cloud computing environment make many users pay more attention to the data security and privacy(IDC, 2013), which also affect the cloud computing application and promotion. Therefore, the cloud user needs to verify the trustworthiness of the cloud service to assure his data be really protected, i.e., cloud managers must provide users the trusted proofs of the cloud computing environment. The trusted proofs must be based on the starting point-the root of trust, which will transfer the trust from the running environment of the cloud service, eventually to the remote cloud users. At the same time, the cloud require that the resource and service must be on-demand scheduled and fast deployed, so the root of trust in the cloud computing environment must also be able to be flexible, rapid deployed.

For enforcing the reliability of the cloud service, the existing approaches strengthen the servers' operation system(i.e.,the virtual machines) and control the data access through the network security measures. But, there are some questions unresolved,

1. The trustworthy of the cloud data or service is depended on the software system,which is unable to defend the malice attackers fundamentally.
2. Lack of the valid formal analysis method for verifying the trustworthy property of the cloud service. The present verification methods for security properties are not applied directly to cloud environment.

Trusted computing technology based on the hardware root of trust can be used for constructing the trusted runtime environment of the user platform. We utilizes the trusted computing technology to build the trusted cloud service platform, and verifies its trustworthy property based on the extended LS^2.

2 BACKGROUND

The core challenge for establishing the trusted IaaS cloud service platform is to utilized the single hardware TPM to protect multiple user virtual machines. For this, the research are mainly focus on the following three aspects.

Virtualized Trusted Platform(VTP). A common approach to virtualizing the TPM is to emulate the

TPM in software. M Strasser presented the TPM emulator(Strasser and Sevnic, 2004) which was a software TPM simulator. But it is mainly used for development purposes. Later, Researchers have proposed various solutions to virtualize the TPM. The typical and widely used one is the vTPM Scheme(Perez et al., 2006)which was firstly presented by Stefan Berger et al. The vTPM can be bound to a physical TPM. Now the vTPM is already integrated in the Xen virtualization platform.However, the usage and management of vTPM are located in the AdminDom, which cause more running load and vulnerabilities of AdminDom. Besides, Para-virtualization TPM (England and Loeser, 2008) was proposed by Microsoft to provide several VMs to access the hardware TPM simultaneously, but it is not applied because it required to modify the TPM specification. The classical schemes of VTP are the TVD(Trusted Virtual Domain) proposed by IBM(Griffin et al., 2005) and TVP(Trusted Virtual Platform) proposed by HP(Dalton et al., 2009). Moreover, based on the vTPM, some other schemes were proposed (Sadeghi et al., 2008; Stumpf and Eckert, 2008). The VTP schemes put forward the research of the trusted cloud.

Trusted Cloud Service. To construct the trusted cloud service, Krautheim, etc proposed the PVI(Private Virtual Infrastructure)(Krautheim, 2009) and TVEM(Trusted Virtual Environment Module)(Krautheim et al., 2010) , which utilized the trusted agent to make the cloud users participate in the process of establishing trust,to ensure users to verify the trustworthy of the cloud. The TCCP(Trusted Cloud Computing Platform) scheme(Santos et al., 2009) was proposed to build the construction and migration mechanisms for the cloud service. But these cloud service approaches still depend on the administrative domain (*Domain*0), so they cannot be used directly on the existing IaaS cloud platform.

Trustworthy Verification. Formal verification should be provided for representing the security property of target system. The trustworthy property of trusted cloud service mainly includes two aspects, one is local trust, the other is remote trust. Abbadi(Abadi and Wobber, 2004) utilized a authorization logic to describe the new secure computing platform NGSCB(Next Generation Secure Computing Base), but not focused on the specific security properties of the platform. Zhang X(Zhang Xing, 2010) modelled and analyzed the trust chain of trusted computing platform based on noninterference model, which is unsuitable for the trusted cloud service due to its abstract. The remote trust verification mainly utilized the remote attestation protocol based on the TCG specification, such as (Gürgens et al., 2007; Millen et al., 2007), which are focus on the correctness of the attestation protocol itself.

While the prior work has addressed different aspects of rooting trust in the IaaS cloud platform, including sharing the TPM for the VMs rented by the cloud user and build the trust cloud environment for the cloud provider, they still has some shortcomings for applying and popularize the cloud. In this paper, we propose a scheme based on the TSD(Trused Service Domain) to construct the trusted cloud, which can be easily deployed and migrated in the IaaS cloud platform. In addition, we model and formally verify its trustworthy property based on the extended the logic of secure system(LS^2).

3 TRUSTED CLOUD ARCHITECTURE

3.1 System Model

Considering the special requirements above for securing the cloud service, we propose a novel architecture of the trusted IaaS cloud platform as shown in Figure 1. Based on the desegregation of the trusted functionalities, we design the independent virtual machine domain-TSD(Trusted Service Domain)-as the new root of trust for the cloud virtualization platform.

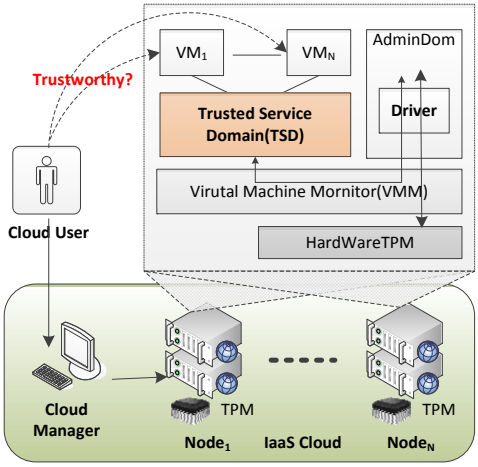


Figure 1: The trusted IaaS cloud based on TSD.

We consider a setting where an IaaS cloud provider possesses several (>2) virtualized servers that are equipped with physical TPMs and wishes to offer the trusted service for the cloud user based on the TSD. The cloud provider uses the cloud manager to manipulate the resource nodes in the cloud, for example, scheduling the cloud resources according to the user requirement. In the IaaS cloud, the VM is

provided for cloud users as resource where the data can be stored or the service can be located.

3.2 Trusted Cloud Service

To ensure the trust service for the cloud user, the VMs rented by the user must be trusted in order to guarantee the reliability of the user's data and service. As the root of trust of the virtualization trusted platform, the trusted service domain(TSD) is a special functional domain of the virtualization platform. It can establish the trusted chain for the user domains(VMs) based on the hardware TPM, through the mapping relationship between the hardware TPM and the TSD. The TSD provides the corresponding instance of root of trust for each VM, which includes the keys and command sets, so as to establish the complete trusted chain from the TPM to the user domains. The proxy in the user domain should interact with the daemon of the TSD to utilize the trust services which are provided by the TSD and the TPM. The trust chain which is established based on the TSD is shows in Figure 2.

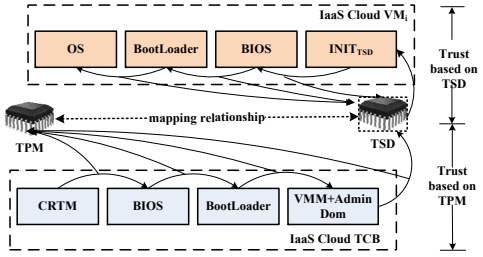


Figure 2: The cloud trust chain based on TSD.

Compared to other present schemes, it is more flexible and available. The trustworthy property (TP) should include two aspects, one is local trust, that is to say, the programs must be properly loaded in the virtulized server locally. The other is remote trust, which represents that the local trust must be proved by remote verifier (R), such as the cloud user. The trustworthy property is represented in Figure 3.

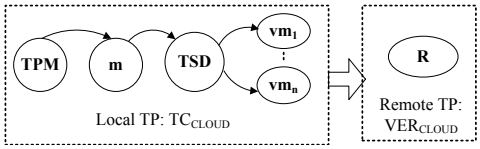


Figure 3: Trustworthy property of trust cloud.

We define the trustworthy property of the cloud as follows,

TP_{CLOUD}:=$\{TC_{CLOUD}, Ver_{CLOUD}\}$

TC_{CLOUD} is the boot sequence of the trust transitive of the VMs in the IaaS cloud, Ver_{CLOUD} shows the verification of the local boot sequence of the VMs.

4 TRUSTWORTHY VERIFICATION

Logic of secure system(LS^2) is a logic inspired by the Protocol Composition Logic(PCL) for reasoning about properties of secure systems(Datta et al., 2009). The logic is built around a concurrent programming language with constructs for modeling machines with shared memory, a simple form of access control on memory, machine resets, cryptographic operations, network communication, and dynamically loading and executing unknown (and potentially untrusted) code. LS^2 has been used for modelling and reasoning the correctness of the trusted chain of the common trusted computing platform, but is unable to be used directly for verifying the trustworthy of the IaaS cloud platform because of its characteristic.

We model the trusted chain of the IaaS cloud platform based on the previous research,and extend the present LS^2 to reason the trustworthy property of the trusted cloud service.

4.1 The extended LS^2

The original LS^2 was proposed as a common analysis method for verifying the security property of the secure systems, causing that its syntax, operation semantic and proof system are not able to satisfy the verification requirements of trust property of the trusted cloud platform, such as, unable to model several VMs, unable to distinguish the operation between VMs and the physical machine. For this, we extend the LS^2 aiming at the IaaS cloud platform as following.

The extended LS^2 program language,

$$Machine\ m, vm, TSD$$
$$Location\ l := m.disk.k | m.pcr.k | m.dpcr.k | m.vpcr.vm.k$$
$$Thread\ id\ I, J ::= < \hat{X}, \eta, m > | < \hat{X}, \eta, vm_i >$$

The extended LS^2 semantic,

$$(reset_{vm}) \quad \iota, \sigma, T_1|...|T_n \rightarrow \iota[locs(vm) \mapsto \_], \sigma[(locs(vm) - locs(vm, disk)) \mapsto init], (T_1|...|T_n) - \{vm\} | [INIT(vm)]_I (I = \langle v\hat{m}, \eta, vm\rangle, \eta\ fresh)$$
$$(jump\ vm) \quad [jump\ P]_I \rightarrow [P]_I$$

The extended LS^2 syntax,

$$M ::= Mem(l,e) | IsLocked(l,I) | Reset(m,I) | Reset(vm,I) | Reset(TSD,I) | Jump(I,e) | Latelaunch(m,I) | Contains(e,e') | e = e' | t \geq t' | Honest(\hat{X}, \vec{P})$$

The extended LS^2 proof rules,

$$\frac{(\forall Q \in IS(Boot(vm)) \wedge (\forall t'.(\neg Reset(TSD)on(t_b,t_e))}{\vdash Reset(vm,I)t \supset \forall t'.(t' > t) \supset A(t,t')} \vdash [Q]_I^{t_b,t_e}A(t_b,t_e), Reset_{vm} :=$$

$$\frac{(\forall Q \in IS(P) \wedge (\forall t'.(\neg Reset(TSD)on(t_b,t_e))}{\vdash Jump(I,P)t \supset \forall t'.(t' > t) \supset A(t,t')} \vdash [Q]_I^{t_b,t_e}A(t_b,t_e), Jump_{vm} :=$$

The extended semantic and syntax are used for describing and interpreting the actions and functions of the cloud service, and the extended proof rules are used for reasoning the security properties. The correctness of the extended rules has been proved formally before submitting this paper, we will present the proof in the future long paper considering the limited number of the pages.

4.2 Formal Verification

According to 3.2, we know that the verification should include two aspects which are local TC_{CLOUD} and remote verification Ver_{CLOUD}.

4.2.1 Local Verification

In the IaaS cloud, local programs of the VMs are loaded as follows (referring to Figure 2),

$$\begin{aligned}
Latelaunch(m) \equiv\ & TSD = read\ m.SLB;\\
& extend\ m.dpcr.k, TSD;\\
& jump\ TSD;\\
& .....\\
INIT(vm) \equiv\ & b = read\ vm.bl_loc;\\
& extend\ m.vpcr.vm, b;\\
& jump\ b\\
BL(vm) \equiv\ & o = read\ vm.os_loc;\\
& extend\ m.vpcr.vm, o;\\
& jump\ o\\
OS(vm) \equiv\ & a = read\ vm.app_loc;\\
& extend\ m.vpcr.vm, a;\\
& jump\ a\\
APP(vm) \equiv\ & ...
\end{aligned}$$

Based on the trust chain of the trusted cloud service shown in 3.2, we model its trustworthy property definition according to the LS^2,

$$\begin{aligned}
&MeasuredBoot_{SRTM}(vm,t) \equiv \forall I \in m.\exists t_T.\\
&\quad \exists t_B.\exists t_O.\exists J \in vm.(t_T < t_B < t_O < t)\\
&\wedge (\neg Reset(TSD,I)on(t_T,t]) \wedge (Reset(vm,J)@t_T)\\
&\wedge (Jump(J,BL(vm))@t_B) \wedge (Jump(J,OS(vm))@t_O)\\
&\wedge (\neg Reset(vm,J)on(t_T,t]) \wedge (\neg Jump(J)on(t_T,t_B))\\
&\wedge (\neg Jump(J)on(t_B,t_O))
\end{aligned}$$

That is to say, if the local trusted cloud service has been built, the boot sequence of the VMs in the cloud platform must be from $BL(bootloader)$ to OS, and during the jump process no other program is executed. The one-to-one map relationship between the program of boot process and the vPCR value must be proved. We model this as follows,

$$\begin{aligned}
&J_{DRTM} + (TSD_{DRTMBasedTPM} \succ VM_{SRTMBasedTSD}) +\\
&\quad ProtectedSRTM(vm) \vdash\\
&Mem(m.vpcr.vm,\\
&seq(INIT(vm), BL(vm), OS(vm), APP(vm)))\\
&\supset MeasuredBoot_{SRTM}(vm,t)
\end{aligned}$$

Proof. At t, there is $Mem(m.vpcr.vm,\ seq\ (\ INIT(vm),\ BL(vm),\ OS(vm),\ APP(vm))\ )$, using axioms (PCR1) and (PCR2) we can obtain

$$\begin{aligned}
&\exists t_T, t_1, t_2, t_3, J.(t_T \le t_1 < t_2 < t_3 < t)\\
&\wedge (Mem(m.vpcr.vm, seq(INIT(vm), BL(vm),\\
&\quad OS(vm), APP(vm)))@t)\\
&\wedge (Mem(m.vpcr.vm, seq(INIT(vm), BL(vm),\\
&\quad OS(vm)))@t_3)\\
&\wedge (Mem(m.vpcr.vm, seq(INIT(vm), BL(vm)))@t_2)\\
&\wedge (Mem(m.vpcr.vm, seq(INIT(vm)))@t_1)\\
&\wedge (Reset(vm,J)@t_T)\\
&\wedge (\neg Reset(vm)on(t_T,t]) \qquad (L-1)
\end{aligned}$$

Then, considering the the executed programs. The first one is booting the $vm(Reset(vm,J))$. Then, $INIT(vm)$ is triggered by the vm based on the TSD. The invariant of $INIT(vm)$ can be got utilizing the extended rule $Reset_{vm}{:}A_{INIT(vm)}(t_b,t_e)$. That is to say,

$$\begin{aligned}
&[Q]_J^{t_b,t_e}\forall t,b,o.((Reset(vm,J)@t_b) \wedge (\neg Reset(vm)\\
&\quad on(t_b,t_e]) \wedge (t_b < t < t_e) \wedge (Mem(m.vpcr.vm,\\
&\quad seq(init,b,o))@t)) \supset\\
&\exists t'.((t_b < t' < t) \wedge (Jump(J,b)@t')\\
&\wedge (\neg Jump(J)on(t_b,t'))\\
&\wedge (Mem(m.vpcr.vm, seq(init,b))@t')\\
&\wedge (IsLocked(m.vpcr.vm,J)@t'))
\end{aligned}$$

According to the definition of program prefix of $IS(INIT(vm))$ in LS^2, the variants of its subprogram should be proved utilizing the axioms (ActN1),

(MemR), (LockI), (MemIP)(Datta et al., 2009) Based on the extended rule $Reset_{vm}$, the modal formula is reduced as follows,

$$\begin{aligned}
&Reset(m,J)@t_T \supset (t > t_T) \supset \\
&((Reset(vm,J)@t_T) \wedge (\neg Reset(vm) on(t_T,t]) \\
&\wedge (t_T < t_3 \leq t) \wedge (Mem(m.vpcr.vm, \\
&seq(init,b,o)))@t_3)) \supset \\
&\quad \exists t_B.((t_T < t_B < t_3) \wedge (Jump(J,b))@t_B) \\
&\quad \wedge (\neg Jump(J) on(t_T,t_B)) \\
&\quad \wedge (Mem(m.vpcr.vm, seq(init,b)))@t_B) \\
&\quad \wedge (IsLocked(m.vpcr.vm,J)@t_B))
\end{aligned}$$

That is to say, at t_3, if the measurement sequence is $seq(init,b,o)$, there must be another time point $t_B < t_3$ when the program jumped b and the current measurement sequence is $seq(init,b)$ without any other program jumping. After Combination and substitution with $(L-1)$, the formula becomes,

$$\begin{aligned}
&\exists t_T, t_B, J.(t_T < t_B < t) \\
&\wedge (Mem(m.vpcr.vm, seq(INIT(vm), BL(vm), OS(vm)))@t) \\
&\wedge (\neg Reset(vm) on(t_T,t]) \\
&\wedge (Mem(m.vpcr.vm, seq(INIT(vm), BL(vm)))@t_B) \\
&\wedge (Reset(vm,J)@t_T) \\
&\wedge (Jump(J, BL(vm))@t_B) \\
&\wedge (\neg Jump(J) on(t_T,t_B)) \\
&\wedge (IsLocked(m.vpcr.vm,J)@t_B) \qquad (L-2)
\end{aligned}$$

$(L-2)$ shows that the fact is correct, that is to say, after INIT(vm) is executed, the vm first jumped to the program BL(vm) indeed, and there was no other program executed before.

Similarly, the correctness of the transitive process from BL(vm) to OS(vm) should be proved through its invariant, $A(t_b,t_e)$, which represents that at some time after loading program b, the program jumped to a without any other program's jumping. The modal factor of its modal formula is eliminated by utilizing the extended rule $Jump_{vm}$. We can get,

$$\begin{aligned}
&Jump(J, BL(vm))@t_b \supset \forall t_e.(t_e > t_b) \\
&\quad \supset \forall t, o, a.((\neg Reset(vm) on(t_b,t_e]) \\
&\quad \wedge (Mem(m.vpcr.vm, seq(init,b)))@t_b) \\
&\quad \wedge (IsLocked(m.vpcr.vm,J)@t_b)) \\
&\quad \wedge (t_b < t \leq t_e) \\
&\quad \wedge (Mem(m.vpcr.vm, seq(init,b,o,a))@t)) \\
&\quad \supset \exists t'.((t_b < t' < t) \\
&\quad \wedge (Jump(J,o)t') \wedge (\neg Jump(J) on(t_b,t')))
\end{aligned}$$

After combination with $(L-2)$ and substitution by $t_b = t_B, t_e = t, b = BL(vm), o = OS(vm), a = APP(vm)$, we get the conclusion,

$$\begin{aligned}
&\exists t_T, t_B, t_O, J.(t_T < t_B < t_O < t) \\
&\quad \wedge (\neg Reset(vm) on(t_T,t]) \wedge (Reset(vm,J)@t_T) \\
&\quad \wedge (Jump(J, BL(vm)t_B)) \wedge (\neg Jump(J) on(t_T,t_B)) \\
&\quad \wedge (Jump(J, OS(vm)t_o)) \\
&\quad \wedge (\neg Jump(J) on(t_B,t_O))
\end{aligned}$$

This is what we set out to prove.

4.2.2 Remote Verification

With the remote attestation specification(Group, 2006) and the present implementation scheme(Cabuk et al., 2010), the remote verification includes some programs as follows,

$$\begin{aligned}
&vRT_{SRTM}(vm) \equiv \\
&\quad w = read\ m.vpcr.vm; \\
&\quad r_{vm} = sign(vPCR(vm), w), AIK^{-1}(vm); \\
&\quad send\ r_{vm} \\
&Verifier(vm) \equiv \\
&\quad sig = receive; \\
&\quad v = verify\ sig, AIK(vm); \\
&\quad match\ v, (vPCR(vm), \\
&\quad seq(INIT(vm), BL(vm), OS(vm), APP(vm)))
\end{aligned}$$

The IaaS cloud should provide the proof of its trustworthy property to the remote cloud users, that is to say, $MeasuredBoot_{SRTM}(vm,t)$ must be proved. According to the remote attestation protocol, we give some premises as follows,

$$\begin{aligned}
\Gamma_{SRTM} = \{&Honest(A\hat{IK}(vm)), (TSD_{DRTMBasedTPM} \succ \\
&VM_{SRTMBasedTSD}), V \neq A\hat{IK}(vm)\}
\end{aligned}$$

That is to say, during the verification process, the virtual root of trust must be loaded before the VMs' boot to ensure establishment of the trust chain, meantime, the remote verifier is different entity from the attestator.

We define the remote verification of the trustworthy property as follows, if the remote verifier verify the correctness of the measurements, the PCR values are definitely sequence with $seq(INIT(vm), BL(vm), OS(vm), APP(vm))$, meanwhile the corresponding trust transitive process is affirmative.

Based on the extended LS^2, we present the above property formally with two aspects as follows,

$$\begin{aligned}
&\Gamma_{SRTM} \vdash [Verifier(vm)]_V^{t_b,t_e} \exists t.(t < t_e) \wedge \\
&\quad (Mem(m.pcr.vm, \\
&\quad seq(INIT(vm), BL(vm), OS(vm), APP(vm))@t)(i)
\end{aligned}$$

$$\Gamma_{SRTM}, Protected_{SRTM}(vm) \vdash$$
$$[Verifier(vm)]_V^{t_b,t_e} \exists t.(t < t_e)$$
$$\wedge MeasuredBoot_{SRTM}(vm,t) \qquad (ii)$$

Proof (i). Based on the premises and $[Verifier(vm)]_V^{t_b,t_e}$, we deduce using axiom (VER),

$$[Verifier(vm)]_V^{t_b,t_e} \exists t_S, e, I.(t_S < t_e) \wedge \hat{I} = AI\hat{K}(vm)$$
$$\wedge Contain(e, SIG_{AIK(vm)^{-1}}\{|PCR(s),$$
$$seq(INIT(vm), BL(vm), OS(vm), APP(vm)))|\})$$
$$\wedge ((Sent(I,e)@t_S) \vee \exists l.(Write(I,l,e)@t_S))$$

Then, we prove the invariant of program $TSD_{SRTM}(vm)$, for the prefix $Q \in IS(TSD_{SRTM}(vm))$, there is a fact as follows,

$$[Q]_I^{t_b,t_e}(\forall l, e, t.(t \in (t_b, t_e]) \supset \neg Write(I,l,e)@t)$$
$$\wedge (\forall t', e'.((t' \in (t_b, t_e]) \wedge Send(I,e')@t')$$
$$\supset (\exists e'', t_R.(t_R < t') \wedge (Read(I, m.vpcr.vm, e'')@t_R)$$
$$\wedge e' = SIG_{AIK(vm)^{-1}}\{|PCR(s), e''|\}))$$

We can prove the invariant above using rule (SEQ) and axiom (Act1). Combining (i) and rule (Honest) to omit the modal fact $[Q]_I^{t_b,t_e}$, we deduce,

$$[Verifier(vm)]_V^{t_b,t_e} \exists t_R, e, e'', I.(t_R < t_e) \wedge \hat{I} = AI\hat{K}(vm)$$
$$\wedge Contain(e, SIG_{AIK(vm)^{-1}}\{|PCR(s),$$
$$seq(INIT(vm), BL(vm), OS(vm), APP(vm)))|\})$$
$$\wedge (Read(I, m.vpcr.vm, e'')@t_R) \wedge e$$
$$= SIG_{AIK(vm)^{-1}}\{|PCR(s), e''|\}$$

Using the axiom (Eq) and (Read), we deduce,

$$[Verifier(vm)]_V^{t_b,t_e} \exists t_R, e'', I.(t_R < t_e)$$
$$\wedge Contain(e,$$
$$SIG_{AIK(vm)^{-1}}\{|SIG_{AIK(vm)^{-1}}\{|PCR(s), e''|\},$$
$$seq(INIT(vm), BL(vm), OS(vm), APP(vm)))|\})$$
$$\wedge (Mem(m.vpcr.vm, e'')@t_R) \qquad (1)$$

Here, the value of e'' should be two probabilities,

$$(e'' = seq(INIT(vm), BL(vm), OS(vm), APP(vm)))$$
$$\vee Contain(e'', SIG_{AIK^{-1}(vm)}\{|PCR(s),$$
$$seq(INIT(vm), BL(vm), OS(vm), APP(vm)))|\})(2)$$

Based on axiom (PCRC) and $Mem(m.vpcr.vm, e'')$, we know that the second in (2) is impossible, so we get,$e'' = seq(INIT(vm), BL(vm), OS(vm), APP(vm))$.

Using the axiom (Eq) to (1), we obtain,

$$[Verifier(vm)]_V^{t_b,t_e} \exists t_R.(t_R < t_e) \wedge$$
$$(Mem(m.pcr.vm,$$
$$seq(INIT(vm), BL(vm), OS(vm), APP(vm))@t_R)$$

This is what we set out to prove.

Proof (ii). With (i) and the definition of local trustworthy property, we get the proof of (ii) directly.

Local and remote verification show that the VMs in the trusted IaaS cloud based on the TSD are trust, that is to say, the cloud users who utilize the VMs as their data or service should confirm the trustworthy property of the cloud service. We have established the trusted IaaS cloud based on the TSD(De-Xian Chang, 2012), and the experiment result shows that, the efficiency of cloud platform based on the TSD is reduced a little (below 15%) because of the mechanisms for trust establishment, so that our scheme has more flexibility and availability.

5 CONCLUSIONS

Trustworthy of the cloud service is very important for applying and spreading the cloud computing. We propose a new approach for constructing trusted cloud platform based on the TSD which is developed as the virtual root of trust. To verify the correctness and validity of trustworthy property, we utilized the extended the LS^2 to analyze the trust chain of trusted IaaS cloud.

In future work, we will model the specific applications of IaaS cloud, and reason their security properties based on the extended LS^2 in order to promote the development and application of cloud computing.

ACKNOWLEDGEMENTS

The authors would like to thank their colleagues and the anonymous reviewers for their helpful feedback and comments.

REFERENCES

Abadi, M. and Wobber, T. (2004). A logical account of ngscb. In *Formal Techniques for Networked and Distributed Systems–FORTE 2004*, pages 1–12. Springer-Verlag.

Cabuk, S., Chen, L., Plaquin, D., and Ryan, M. (2010). Trusted integrity measurement and reporting for virtualized platforms. In *Trusted Systems*, pages 180–196. Springer-Verlag.

Dalton, C. I., Plaquin, D., Weidner, W., Kuhlmann, D., Balacheff, B., and Brown, R. (2009). Trusted virtual platforms: a key enabler for converged client devices. *ACM SIGOPS Operating Systems Review*, 43(1):36–43.

Datta, A., Franklin, J., Garg, D., and Kaynar, D. (2009). A logic of secure systems and its application to trusted computing. In *The 30th IEEE Symposium on Security and Privacy*, pages 221–236. IEEE.

De-Xian Chang, Xiao-Buo Chu, Y. Q. D.-G. F. (2012). Tsd: A flexible root of trust for the cloud. In *The IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications(TrustCom'12)*, pages 119–126. IEEE.

England, P. and Loeser, J. (2008). Para-virtualized tpm sharing. pages 119–132.

Griffin, J. L., Jaeger, T., Perez, R., Sailer, R., Van Doorn, L., and Cáceres, R. (2005). Trusted virtual domains: Toward secure distributed services. In *Proceedings of the 1st IEEE Workshop on Hot Topics in System Dependability (HotDep'05)*. Citeseer.

Group, T. C. (2006). Tcg infrastructure working group architecture part ii-integrity management version 1.0, https://www.trustedcomputinggroup.org.

Gürgens, S., Rudolph, C., Scheuermann, D., Atts, M., and Plaga, R. (2007). Security evaluation of scenarios based on the tcg's tpm specification. In *Computer Security–ESORICS 2007*, pages 438–453. Springer-Verlag.

IDC (2013). The single biggest reason public clouds will dominate the next era of it, http://blogs.idc.com/ie/?p=345.

Krautheim, F. (2009). Private virtual infrastructure for cloud computing. In *Proceedings of the 2009 conference on Hot topics in cloud computing*, pages 1–5. USENIX Association.

Krautheim, F., Phatak, D., and Sherman, A. (2010). Introducing the trusted virtual environment module: a new mechanism for rooting trust in cloud computing. pages 211–227.

Millen, J., Guttman, J., Ramsdell, J., Sheehy, J., and Sniffen, B. (2007). Analysis of a measured launch. Technical report, DTIC Document.

Perez, R., Sailer, R., and van Doorn, L. (2006). vtpm: virtualizing the trusted platform module. In *Proc. 15th Conf. on USENIX Security Symposium*, pages 305–320.

Research, F. (2013). Conventional wisdom is wrong about cloud iaas, http://www.forrester.com.

Sadeghi, A.-R., Stüble, C., and Winandy, M. (2008). Property-based tpm virtualization. *Information Security*, pages 1–16.

Santos, N., Gummadi, K. P., and Rodrigues, R. (2009). Towards trusted cloud computing. In *Proceedings of the 2009 conference on Hot topics in cloud computing*, pages 3–6.

Strasser, M. and Sevnic, P. (2004). A software-based tpm emulator for linux. *Semesterarbeit, ETH Zurich*.

Stumpf, F. and Eckert, C. (2008). Enhancing trusted platform modules with hardware-based virtualization techniques. In *Emerging Security Information, Systems and Technologies, 2008. SECURWARE'08. Second International Conference on*, pages 1–9. IEEE.

Zhang Xing, Huang Qiang, S. C.-X. (2010). A formal method based on noninterference for analyzing trust chain of trusted computing platform. *Chinese Journal of Computers*, 33(1):960–971.

Storage CloudSim
A Simulation Environment for Cloud Object Storage Infrastructures

Tobias Sturm, Foued Jrad and Achim Streit
Steinbuch Centre for Computing (SCC)
Department of Informatics, Karlsruhe Institute of Technology, Karlsruhe, Germany
tobias.sturm@student.kit.edu, {foued.jrad, achim.streit}@kit.edu

Keywords: Cloud Computing, Storage as a Service, STaaS, Cloud Broker, CloudSim.

Abstract: Since Cloud services are billed by the pay-as-you-go principle, organizations can save huge investment costs. Hence, they want to know, what costs will arise by the usage of those services. On the other hand, Cloud providers want to provide the best-matching hardware configurations for different use-cases. Therefore, CloudSim a popular event-based framework by Calheiros et al., was developed to model and simulate the usage of IaaS (Infrastructure-as-a-Service) Clouds. Metrics like usage costs, resource utilization and energy consumption can be also investigated using CloudSim. But this favored simulation framework does not provide any mechanisms to simulate todays object storage-based Cloud services (STaaS, Storage-as-a-Service). In this paper, we propose a storage extension for CloudSim to enable the simulations of STaaS-components. Interactions between users and the modeled STaaS Clouds are inspired by the CDMI (*Cloud Data Management Interface*) standard. In order to validate our extension, we evaluated the resource utilization and costs that arise by the usage of STaaS Clouds based on different simulation scenarios.

1 INTRODUCTION

Cloud computing is one of the most emerging technologies of the past few years. Start-ups, big companies and the scientific community explore the benefits of this kind of resource utilization. These Cloud users exploit the advantages of less configuration overhead, less investments, less operational costs, highly flexible systems that scale out to large systems, or scale in to a minimal resource provisioning, depending on the current needs.

The main benefit from using Cloud services is to reduce the cost for computation and storage by sourcing these services out to a third-party provider. The crucial questions are about the costs and the performance of the usage. Users want to know, how much money they are going to spend, if they switch to a Cloud solution. On the other side, providers have to know, what is the best hardware constellation and configuration for them. These questions can be answered by simulating Cloud environments.

CloudSim (Calheiros et al., 2009) is a popular simulation environment, which offers capabilities to simulate computing Clouds that provision computing infrastructures as a service. The focus of *CloudSim* is on the modeling of so-called *Cloudlets* which are jobs, that can be scheduled on VMs (Virtual Machines). Models for hard disks, SAN and files are already included in *CloudSim*. However, capable interfaces to simulate object STaaS are missing, as well as a fine-grained model for the size of files.

Object storage systems pool multiple physical devices together and provide one logical medium to store and retrieve many different pieces of information called objects. Besides user data, an object contains so called *metadata*, like timestamps or number of replicas. Objects can be accessed by their server-wide unique ID and can be created, updated, read (complete or partially) by all authorized clients (Azagury et al., 2003). Containers are virtual collections of objects and may be hierarchical like folders on conventional file systems (CDMI, 2012).

Considering that there is no known simulation environment for object STaaS, which includes a proper modeling of the interfaces, this work extends the existing framework *CloudSim* by adding components for an accurate modeling of object storage.

The rest of this paper is organized as follows: In the next section we present the existing simulation framework CloudSim and review the fundamentals of object storage, before we discuss prior works related to STaaS Cloud simulation. In Section 3 we propose

the extension to enable STaaS Cloud simulation, followed by Section 4, in which we describe a set of experiments we investigated on the extended simulation framework. Section 5 concludes the paper with a brief summary and describes our future research directions.

2 RELATED WORK

In this section we describe the existing *CloudSim* framework, explain the needed changes to simulate STaaS Clouds and then present related works to STaaS simulation.

2.1 CloudSim

CloudSim is a time discrete simulation framework for Cloud computing. The framework consists of three layers as described by the authors (Calheiros et al., 2011) (from bottom to top):

1. *Core Simulation Engine:* Queuing and processing of events, management of Cloud system entities such as host, VMs, brokers, etc.
2. *CloudSim:* Representation of network topology, message delays, VM provisioning, CPU, storage and memory allocation, etc.
3. *User code:* General configuration such as Cloud scenarios and user requirements, User Broker, etc.

Users of the framework can either modify the top layer to model simulation scenarios, or extend the second layer, to test different allocation policies in a Cloud system. The *user code* layer defines so-called *cloudlets* that define a specific amount of computation requirements similar to a Cloud job. Datacenter Brokers forward the cloudlets to Cloud providers and manage and monitor their execution. These jobs are dispatched on available VMs by the *CloudSim* layer. Communication between the entities is done via message events which are sent to the *core simulation engine*, which handles all events in the correct order and manages the simulated time. Events between two remote entities are automatically delayed, if the network topology is predefined (Garg and Buyya, 2011).

CloudSim can simulate SAN storage, hard drives and files, that are stored on hard drives directly or via SAN storage. But their modeling lacks support for object storage as follows:

- *File size magnitude:* CloudSim models the file size in megabyte, but 90% of all web objects fit within 16KB (Dukkipati et al., 2010), which means that a content delivery scenario scenario can not be modeled accurately.
- *Hard drive models:* The hard drive models do not provide all metrics that are required to model the read and write durations accurately.
- *No storage controller:* CloudSim does not offer a controller that determines the storage location of objects.
- *No appropriate object storage interface:* No model for any STaaS interface, as for example CDMI.

2.2 STaaS Simulation

The only work known to us that deals with storage modeling in CloudSim was performed by Zhao and Long (Long and Zhao, 2012). Even after intensive search we have not found any other simulation environment that suits STaaS. Zhao and Long propose an extension of the existing Datacenter model by introducing a *Replica Catalog* and *Block Catalog* (One piece of information is striped and distributed on different physical locations to gain a parallelized and therefore faster access, catalogs can provides a query interface to retrieve those locations) to benchmark different configurations. All operations are started from within a Cloudlet, which means that the VM is the requesting entity. Overall their work presents a mixture of datacenters that offer computing power and storage capacities. This work in contrast focuses on the strict separation of these two capabilities, as currently done in real-world IaaS providers.

In a previous paper (Jrad et al., 2012), we investigated the use of multiple IaaS Compute providers interfaced by the Open Cloud Computing Interface[1] (OCCI). One user assigns a *Meta Broker* to choose between multiple available Clouds based on *Service Level Agreements* (SLA). We provide similar SLA mechanisms for STaaS providers like: Available capacity, bandwidth of Cloud interfaces, restrictions on file sizes, etc.

3 STORAGE CLOUDSIM

We extended the existing simulation framework *CloudSim* to enable the modeling of STaaS Clouds. Therefore we provide additional models in the user code layer and the layer beneath as we can see in Figure 1. *CloudSim* consists of three layers. Our additions to the framework build on top of the *CloudSim Core Engine*, which is responsible for the time discrete message passing between simulation entities. We provide new models for hard disks, servers and

[1] http://occi-wg.org

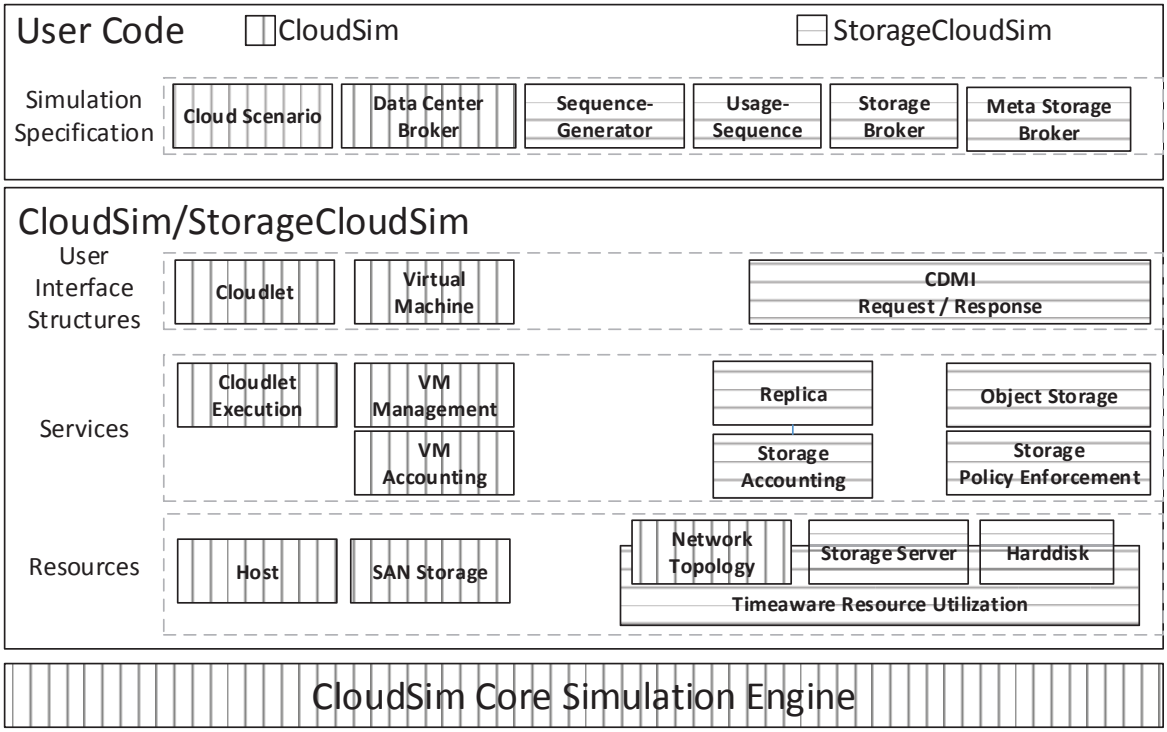


Figure 1: Storage CloudSim architecture Overview.

a more accurate model resource utilization over time. Beside data replication and accounting, we implemented different storage policies enforcements. Users of the simulation framework can generate sequences of CRUD (Create, Read, Update, Delete) operations on objects and containers and implement different multi-Cloud broker policies.

3.1 User Code Extension

The usage of STaaS components is modeled as *usage sequence* - a sequence of CRUD operations on both containers and objects. The order of these operations is fixed (an object can only be put into a container if the container was created before), but the execution of two sequences never interfere or depend on each other. We provide tools for automatically generating such sequences in a way that they are random, defined by statistical distribution constraints[2] and are repeatable by storing them in XML files.

Each sequence is attached to a SLA requirement, where hard constraints and soft requirements can be defined. The hard constraints serve to exclude STaaS Clouds that do not fulfill required features (e.g. capability to store user metadata, available capacity, etc.), whereas the soft requirements lead to a scoring of providers whenever there is more than one provider that fulfills all hard requirements in order to choose the one with the highest score (e.g. prefer providers with the lowest storage cost).

The sequences and their requirements are forwarded to the *Meta Storage Broker*, which query characteristics and capabilities of each available STaaS provider and chooses the best suitable provider for every sequence and dispatches each one to a *Storage Broker*. The former is acting as a proxy for a single user Cloud interaction by dispatching the requests of the sequence and storing all responses from the Cloud for later analysis.

The entire communication between broker and Cloud is wrapped in CDMI calls, which is an industry-wide standard for accessing STaaS Clouds.

3.2 STaaS Provider Models

Every Cloud is described by some characteristics as for example networking capabilities, supported operations, billing information and hardware specifications or limits as for example maximum size of an entity or maximum number of objects in one container. Each instance features a CDMI endpoint, a set of servers and a sets of hard disks. On top of the

[2]size of objects, delay between requests, balance between read and write operations etc.

Table 1: Defined Usage Sequence Patterns.

Name	**Pattern A**	**Pattern B**
Traffic / Sequence	1GB .. 100GB (normal distributed)	1MB .. 16GB (gamma distribution 3,2)
Size / Object	1MB..100GB (normal distributed)	1MB .. 1GB (normal distributed)
Read Write Ratio	0:1	5:11
Hard Constraints	no container size limit sufficient boundary for object size limit enough storage capacity	enough storage capacity
Soft Constraints	low upload and storage costs	all costs low

hardware models is the representation of the logical organization such as each object might have multiple replica, called *blob*s. Communication between brokers and STaaS Clouds is accomplished through CMDI messages, which organize data as *CDMI Objects* whithin Containers (both described as entity in the following). Every object must be stored in one container - containers can not be nested. Each user of the Cloud (represented by a broker) can only see his entities. They can be accessed via their Cloud-wide unique *CDMI ID* or via a human readable name that was chosen by the user. Entities feature so-called *Metadata*, which is a key-value storage, which holds for example the size of an entity, date of creation, number of desired replica or date of last-access. Depending on the capabilities of a Cloud, a user can modify or extend these metadata (e.g. the number of desired replicas).

The storage controller matches certain policies, such as the physical distribution of blobs that belong to one object (replica). A blob can only be stored on a disk, when the disk holds no other blob of the same object, so that the failure of a single disk does not cause the loss of data if replication is used.

Since read, write and update operations involve the transport of data through the internal network, we modeled the payload size of these operations on the most fine-grained level (bytes). As we model the storage of data to the level of disks, we have to model also the data transport between hard disks. Therefore, we assigned a bandwidth to network and disk connections (see Figure 2) inside the STaaS CLoud. The complete data path begins with the broker, which sends a CDMI request to the STaaS provider. The data is then sent to the storage server and then to the hard disk. Every segment of the data path has a certain bandwidth[3] to introduce a certain delay. In addition we model the current utilization of these path sections.

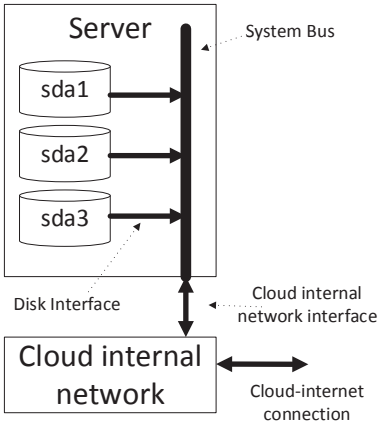


Figure 2: IO limitations on the data path.

The duration of each operation depends on the bandwidth of the path section that has the lowest available bandwidth. Since concurrent operations might interleave or overlap, the available bandwidth might change during one operation. Thus, the total duration of each operation depends on the least available bandwidth on the path over time and thus the following terms must be fulfilled:

$$payloadSize = \int_{t_{start}}^{t_{end}} \operatorname*{argmin}_i availableBandwidth_i(t)\,\mathrm{d}t. \tag{1}$$

and

$$\bigwedge_{i \in pathSections, t} availableBandwidth_i(t) < bandwidthLimit_i \tag{2}$$

Read and write operations are dispatched by first-scheduled-first-served strategy.

4 SIMULATION EXPERIMENTS

4.1 Meta Broker Evaluation

We conducted two sets of experiments, one with only one StaaS Cloud provider (*Amazon S3*[4]) and another set with three providers, each with a mixture of patterns of *usage sequence* (see Table 1). Pattern A features only uploads of large data chunks. Therefore the costs for downloads from the Cloud have no impact. The other pattern features up- and downloads.

[3] uni-directional bandwidth specification is possible.

[4] http://aws.amazon.com/de/s3

Table 2: Modeled Cloud Specifications.

	S3	**Private**	**Swift**
max. Object size	unlimited	16GB	unlimited
write rate	156MB/s	64MB/s	156MB/s
read rate	156MB/s	156MB/s	156MB/s
capacity	2TB	1TB	2TB
read latency	8.5ms	9ms	8.5ms
write latency	9.5ms	11ms	9.5ms
capacity	72TB	3TB	32TB
# replica	3	3	1
$/GB stored	0.05$	0.04$	0.01$
$/GB up	0.0002$	0.0002$	0.0002$
$/GB down	0.01$	0.0002$	0.1$

We modeled three different Cloud providers *Amazon S3*, a private Cloud and a *Swift Cloud*[5], see Table 2), each with different pricing, storage capacity and throughput. The private Cloud has a hard 16GB limit for objects, but offers the lowest prices.

The meta broker checks all hard constraints against every available Cloud and then calculates a score for each potential Cloud. The score in our example is calculated by the inverse of the relevant costs, e.g.

$$score = \frac{1}{costsperstoredGB} + \frac{1}{costsperuploadedGB}$$

for sequences of pattern A. One hard constraint that is shared among both patterns is the requirement for sufficient storage capacity. Since users in most use cases do not know the exact volume of data they want to store or the size of every single object, we added a 25% uncertainty to these values. A sequence is declined, if no Cloud provider can fulfill all hard requirements.

We can verify the correct behavior of the meta broker by plotting the used storage of each Cloud provider over time, as in Figure 3. Sequences that do not contain objects that are bigger than 16GB can be dispatched on the private Cloud, because it is the cheapest STaaS provider. Sequences that contain objects that are bigger than 16GB will be dispatched on the *Swift Cloud* because it is cheaper than *Amazon S3*, which is never chosen for any sequence. We can see that the *Swift Cloud* usage rises faster since all big objects (greater than 16GB) are stored there. We conducted the above described experiments multiple times and made sure that the randomly generated input sequences did not contain any artifacts. A mixture of both sequence types was used (2486 sequences of type A and 2514 of type B). First, we observed that both experiments (single and multi-Cloud) with 50 and 500 sequences succeeded, but the experiment with 5000 mixed sequences can not be executed on the single-Cloud environment completely (4150 sequences are declined due to capacity exhaustion). If we observe the number of succeeded and failed requests as well as the declined sequences, we observe that at one point of this experiment, the number of succeeded requests stops to rise, the failed requests spike and the number of declined sequences starts to rise. These are declined because the hard requirement for sufficient storage can not be fulfilled. In total 15 requests fail in the single Cloud experiment with 5000 mixed sequences. Three reasons (or any combination of them) explain this behavior:

- *Inaccurate SLA*: *UsageSequence* requests less capacity than it will consume. Cloud storage may be used by 99.99%, but it accepts the request, because the requested size fits into the remaining storage.
- *Unsatisfiable policies*: One reason could be the policy, that forces the storage controller to locate *StorageBlob*s of one object not to be on the same disk. Maybe there is enough physical storage available, but it is not distributed as required.
- *Concurrent access*: Clouds use the currently available storage, when responding to a Cloud discovery request. Requests do not allocate any storage space. Two storage capacity requests can be met, but one of them fills all of the remaining storage, so the other request can not be fulfilled.

One of the most interesting points are the costs that are billed when the sequences are run on the different constellations. We see that multi-Cloud experiments generate less costs than the single Cloud experiment for the runs with 50 and 500 sequences (60% less costs in the 500 sequence experiment, see Table 3). This can be explained by the capacity of the private Cloud, which provides resources for a fraction

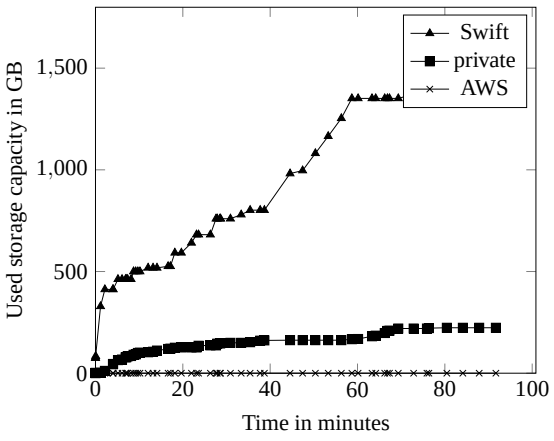


Figure 3: Storage usage for the modeled STaaS Clouds.

[5]http://swift.openstack.org

Table 3: Average Price per accepted Sequence.

# of input Seq.	Single Cloud	Multi-Cloud
50	0.0150$	0.0070$
500	0.0149$	0.0058$
5000	0.0149$	0.0128$

Table 4: Total Costs for 250 Input Sequences.

	Single Cloud	Multi-Cloud
Pattern A	2.62$	2.62$
Pattern B	4.82$	0.30$

of the costs of the *Amazon S3* Cloud. But the total costs for the 5000 sequence experiment shows that the multi-Cloud constellation can generate more costs, which is in the interest of the user, because in this case more sequences are processed, compared to the single Cloud experiment. However, the multi-Cloud setup provides lower prices per accepted sequence in each experiment, as seen in Table 3.

4.2 Effect of the Used Sequence Type

In order to study the impact of the chosen sequence type, we conducted four experiments each of them with a different sequence type and on a single Cloud or a multi-Cloud environment. As we submitted 250 sequences, no sequence was rejected in any experiment. Table 4 shows that the multi-Cloud variant generates less or the same costs as the single Cloud variant. The costs for single and multi-Cloud for the sequences of type A is the same, whereas the difference for sequences of type B is 4.52$ (saving of factor 16). This can be explained by the fact, that the sequences of type A have tighter restriction on the SLA: The minimal accepted value for the maximum object size depends on the largest object of the sequence, which can be between 1 and 100GB. Thus, the majority of A sequences can not be scheduled on the private Cloud, which is deciding for the costs for B sequences

5 CONCLUSIONS

We extended the popular Cloud simulation framework *CloudSim* to enable the simulation of STaaS Clouds. The concurrent use of resources is modeled accurately in order to gain realistic simulation results. Detailed models for storage disks, servers and storage controller are provided and can be easily extended. Monitoring allows users to investigate a broad spectrum of metrics from each simulated STaaS Cloud as well as from the user's perspective.

Different access patterns can be modeled and used to generate sequences of requests. These sequences can be stored and read from file to provide random, but repeatable experiments. In addition, different SLAs can be modeled, such as certain capabilities of a Cloud or some restrictions. A *Meta Broker* enables simulations of multiple Clouds, where different *usage sequences* can be dispatched on the best-matching Clouds. Compared to single Cloud usage users can save huge costs if the SLAs of the request sequences are not very strict.

In a future work, we will investigate more realistic pricing models (no linear price functions) and more complex SLA mechanisms. The framework will be extended to enable the simulation of the storage allocation policies and mechanisms used in current STaaS Clouds, for example mechanisms to prevent bit rotting of failover reactions to outages of hardware. We will also introduce more heuristic parameters to make the simulation more realistic. Also brokers could not only compare requirements against the static SLA but could also calculate metrics based on previous bandwidth usage and then decide which Cloud to use for further requests.

REFERENCES

Azagury, A., Dreizin, V., Factor, M., Henis, E., Naor, D., Rinetzky, N., Rodeh, O., Satran, J., Tavory, A., and Yerushalmi, L. (2003). Towards an object store. In *In Proceedings of the 20th IEEE Conference on Mass Storage Systems and Technologies*, pages 165–176.

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., and Buyya, R. (2011). Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50.

Calheiros, R. N., Ranjan, R., De Rose, C. A., and Buyya, R. (2009). Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services. *arXiv preprint arXiv:0903.2525*.

CDMI (2012). *Cloud Data Management Interface (CDMI) Version 1.0.2*.

Dukkipati, N., Refice, T., Cheng, Y., Chu, J., Herbert, T., Agarwal, A., Jain, A., and Sutin, N. (2010). An argument for increasing tcp's initial congestion window.

Garg, S. K. and Buyya, R. (2011). Networkcloudsim: Modelling parallel applications in cloud simulations. In *Proceedings of the 2011 Fourth IEEE International Conference on Utility and Cloud Computing*, UCC '11, pages 105–113, Washington, DC, USA. IEEE Computer Society.

Jrad, F., Tao, J., and Streit, A. (2012). Sla based service brokering in intercloud environments. In *Proceedings of the International Conference on Cloud Computing and Services Science CLOSER2012*, pages 76–81, Porto, Portugal.

Long, S. and Zhao, Y. (2012). A toolkit for modeling and simulating cloud data storage: An extension to cloudsim. In *Control Engineering and Communication Technology (ICCECT), 2012 International Conference on*, pages 597–600.

Robust Cloud Monitor Placement for Availability Verification

Melanie Siebenhaar, Ulrich Lampe, Dieter Schuller and Ralf Steinmetz
Multimedia Communications Lab (KOM), Technische Universität Darmstadt, Darmstadt, Germany
firstname.lastname@KOM.tu-darmstadt.de

Keywords: Cloud Computing, Service Level Agreements, Verification, Monitoring, Placement, Performance, Availability.

Abstract: While cloud computing provides a high level of flexibility, it also implies a shift of responsibility to the cloud provider and thus, a loss of control for cloud consumers. Although existing means such as service level agreements or monitoring solutions offered by cloud providers aim to address this issue, there is still a low degree of trust on consumer side that cloud providers properly measure compliance against SLAs. A solution lies in designing reliable means for monitoring cloud-based services from a consumer's perspective. We already proposed such a monitoring approach in our former work. However, our experiments revealed that our approach is sensitive to network impairments. Hence, in the work at hand, we introduce the Robust Cloud Monitor Placement Problem and present a formal optimization model. Based on the model, we propose an initial optimization approach, that allows to obtain an exact solution using off-the-shelf algorithms.

1 INTRODUCTION

Cloud computing provides highly configurable computing resources on-demand and with minimal management effort via the Internet (Mell and Grance, 2011). Hence, it promises a high level of flexibility similar to utilities like electricity or water (Buyya et al., 2009). However, using services from the cloud implies a shift of responsibility to the cloud provider and thus, a loss of control for cloud consumers. A negotiation of so-called service level agreements (SLAs) between a cloud consumer and a cloud provider addresses this issue by providing quality guarantees to the cloud consumer and specifying corresponding penalties for the cloud provider in case of violating a part of the contract. The quality guarantees typically refer to certain quality levels such as thresholds for performance parameters like availability. Although a negotiation of SLAs provides a certain level of control to cloud consumers, this solution does not seem to be sufficient. Still, there is only a low degree of trust on consumer side that providers conduct sufficient measurements of performance against SLAs (CSA and ISACA, 2012). However, cloud providers often additionally provide some means for monitoring to their customers. Nevertheless, solely relying on these solutions cannot be perceived as a reliable evidence base for detecting and documenting SLA violations from a consumer's perspective, particularly in the context that providers often put the burden of SLA violation reporting on their customers (Patel et al., 2009).

A solution lies in the design and provision of reliable means to enable consumers to verify compliance with SLAs from their perspective. Such an approach has already been proposed in our former work in (Siebenhaar et al., 2013). Our approach involves the placement of monitoring units within both, the cloud provider's and consumer's infrastructure in order to verify the availability of cloud applications. However, our experiments revealed that our solution is very sensitive to network impairments. Hence, we examine the robust placement of monitoring units depending on current network reliability in the work at hand. This paper introduces the *Robust Cloud Monitor Placement Problem* (RCMPP) as a new research problem and presents a formal optimization model. Furthermore, we propose transformations that can be applied to this nonlinear, multi-objective optimization problem in order to obtain an exact solution using off-the-shelf optimization algorithms.

The remainder of this paper is structured as follows: Section 2 gives an overview of related approaches to our work. Section 3 briefly summarizes our former approach for availability verification. In Section 4, we describe the RCMPP in detail and introduce a formal optimization model. Based on the formal model, we propose an optimization approach in Section 5. The paper closes with a summary and directions for future work in Section 6.

2 RELATED WORK

Only a few approaches have been proposed so far for monitoring the performance of cloud resources from a consumer's perspective. For example, (Sharma et al., 2013) propose a network monitoring framework that enables tenants to monitor the connectivity between their allocated compute nodes. Other related publications focus on robust placement problems in networks in general. For example, (Natu and Sethi, 2008) try to minimize the number of monitor locations while accounting for a maximum of k node/edge failures, and (Bin et al., 2011) consider relocatable VMs in the presence of up to k host failures. In the field of Operations Research, the fault-tolerant facility location problem, first studied by (Kamal and Vazirani, 2000), where each city must at least be connected to a certain number of distinct facilities, is related to the work at hand. But none of the related approaches addresses robust monitor placement by jointly considering current network reliability, monitor redundancy, and resource as well as location constraints.

3 AVAILABILITY VERIFICATION OF CLOUD APPLICATIONS

In this section, we give an overview of our former work for verifying the availability of cloud applications introduced in (Siebenhaar et al., 2013), which serves as a foundation for the work at hand. Our former work proposed a hybrid monitoring approach for verifying the availability of cloud applications from a consumer's perspective, since availability is one of the very few performance parameters that are part of the SLAs of today's cloud providers (e.g., Elastic Compute Cloud (EC2) by (Amazon, 2008)). However, further performance parameters can be easily incorporated in our approach as well. Basically, our former approach combines consumer- and cloud-side monitoring. In doing so, consumers not only obtain means to assess the status of a cloud application independently from a cloud provider, but also obtain visibility of the end-to-end performance of a cloud-based service. Hence, our approach enables consumers to attribute downtimes to causes either on consumer or provider side. Three different component types are considered in our design (cf. Figure 1): Monitoring units placed on VMs on consumer- and cloud-side observe the availability of a cloud application and the VM, where the application is running on. For our approach we make the assumption that a consumer has access to the VM where a cloud application is hosted. However, a monitoring unit on cloud-side observing a specific application is never placed on the VM where the application is hosted. In this case, a monitoring unit would not be able to report any downtimes if the underlying VM crashes. Nevertheless, a lightweight software component, a VM monitor, is placed on each VM where a cloud application is hosted in order to gain access to predefined processes that are essential for running the cloud application. A periodical pull model is applied by the monitoring units in order to invoke predefined services of the cloud application or request the status of essential processes from VM monitors. The set of monitored services and essential processes of a cloud application can be defined in advance, e.g., as part of the SLAs. Finally, a broker component is introduced which collects and aggregates the monitoring data of all monitoring units. The monitoring units apply an event-based push model to send data to the broker. The broker maintains a list of the downtimes reported by each monitoring unit and periodically computes the overall availability. For more information on the computation of the overall availability, the interested reader is referred to (Siebenhaar et al., 2013). Since failures in the network between a consumer and provider could prevent monitoring units on consumer-side from detecting a downtime of a cloud application, monitoring units should be placed on different network domains. We assume that an enterprise running several data centers at different branches could assign independent consumer-side monitors to different locations.

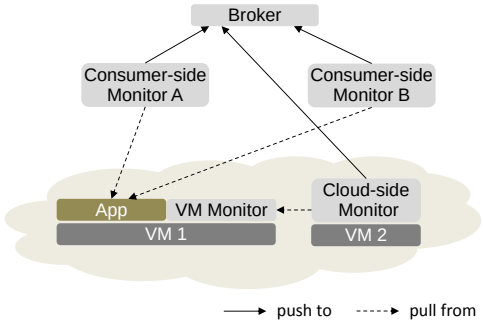


Figure 1: Overview of the monitoring framework.

4 ROBUST CLOUD MONITOR PLACEMENT PROBLEM

4.1 Problem Statement

Our work focuses on a scenario, where an enterprise cloud user has several regional branches worldwide, each running a private data center, and makes use of applications running in the data centers of a cloud

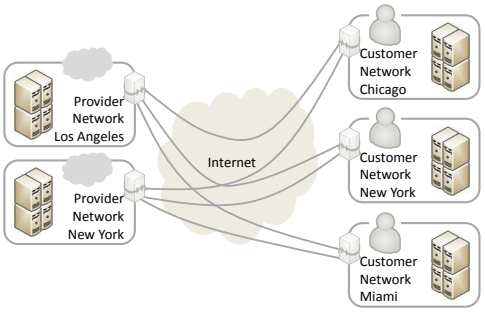


Figure 2: Robust Cloud Monitor Placement scenario.[1]

provider (cf. Figure 2). We take the perspective of the enterprise cloud user, who wants to verify the availability of the running cloud applications. For this purpose, the enterprise cloud user applies our hybrid monitoring approach as described in the previous section. In doing so, s/he places monitoring units for each cloud application on consumer- and provider-side. However, since the experiments in (Siebenhaar et al., 2013) revealed that our approach is sensitive to network impairments, the placement of the monitoring units must be conducted in such a way that the robustness of our monitoring framework is maximized. Table 1 gives an overview of the symbols introduced in the following and used in the formal model. The upper part of the table describes the basic entities, whereas the lower part contains the parameters.

The scenario considered in this work consists of a set of data center sites, formally denoted as $S = \{1,...,n\}$. This set S is a union of the set $S'' = \{1,...,d\}$) of data center sites of the enterprise cloud user and the set $S' = \{d+1,...,n\}$) of data center sites of the cloud provider. A set $V_s = \{1,...,i\}$ of VMs is running on each data center site $s \in S$. All VMs V_s are potential candidates for monitor placement. On the data center sites $s' \in S'$ of the cloud provider, each VM $v' \in V_{s'}$ is provided to the enterprise cloud user, e.g., as part of a virtual private network, running a set of cloud applications $C_{s'v'} = \{1,...,j\}$ to be monitored. The VMs $v \in V_s$ representing potential monitor candidates are interconnected with the VMs $v' \in V_{s'}$ of the cloud applications $C_{s'v'}$ via a set of links $L = \{l(sv \rightleftharpoons s'v')\}$. Since the enterprise cloud user is not aware of the underlying network topologies of the Internet service provider and the cloud provider when connecting to a cloud application, we assume that the enterprise cloud user is only able to measure the end-to-end performance between a given pair of VMs represented by a single link $l \in L$ between this pair of VMs in the model. Each monitoring unit for a cloud application $c \in C_{s'v'}$ exhibits a certain resource demand of $rd_{s'v'cr} \in \mathbb{R}^+$ for a specific resource type $r \in R = \{1,...,k\}$ such as, e.g., CPU power or memory. Furthermore, each VM $v \in V_s$ on a site $s \in S$ is able to provide a specific resource supply of $rs_{svr} \in \mathbb{R}^+$. For reasons of fault-tolerance, the enterprise cloud user specifies a redundancy factor $rf_{s'v'c} \in \mathbb{N}_{>1}$ for each cloud application $c \in C_{s'v'}$ indicating that application c has to be monitored by $rf_{s'v'c}$ different monitoring units. We further assume that the enterprise cloud user is able to utilize traditional network measurement tools in order to estimate the reliability of a given VM $v \in V_s$ on a site $s \in S$ or a link $l \in L$ between a given pair of VMs. With respect to these measurements, $p_{sv} \in \mathbb{R}^+$ denotes the observed reliability for a VM $v \in V_s$ on a site $s \in S$ and $p_{l(sv \rightleftharpoons s'v')} \in \mathbb{R}^+$ denotes the observed reliability for a given link $l \in L$.

Table 1: Used symbols in the formal model.

Symbol	Description
$S = \{1,...,n\}$	set of n data center sites
$S'' = \{1,...,d\})$	consumer sites, $S'' \subset S$
$S' = \{d+1,...,n\})$	provider sites, $S' \subset S$
$V_s = \{1,...,i\}$	VM candidates for monitor placement on site $s \in S$
$C_{s'v'} = \{1,...,j\}$	cloud applications to monitor on VM $v' \in V_{s'}, s' \in S'$
$L = \{l(sv \rightleftharpoons s'v')\}$	links interconnecting VM monitor candidates V_s and VMs of applications $C_{s'v'}$
$R = \{1,...,k\}$	set of k considered VM resource types
$rd_{s'v'cr} \in \mathbb{R}^+$	resource demand for monitoring application $c \in C_{s'v'}$ for resource $r \in R$
$rs_{svr} \in \mathbb{R}^+$	resource supply of VM $v \in V_s$ for resource $r \in R$
$rf_{s'v'c} \in \mathbb{N}_{>1}$	redundancy factor for monitoring application $c \in C_{s'v'}$
$p_{l(sv \rightleftharpoons s'v')} \in \mathbb{R}^+$	observed reliability for each link $l \in L$
$p_{sv} \in \mathbb{R}^+$	observed reliability for each VM $v \in V_s$

The challenge for the enterprise cloud user now consists in assigning the monitoring units of all cloud applications to VMs on consumer and provider sites. In doing so, the objective of the enterprise cloud user is to maximize the reliability of the monitoring framework expressed by the probability that at least one of the redundant monitoring units for each cloud application is working. The resource supplies of all VMs must not be exceeded by the assigned monitoring units. Furthermore, all monitoring units for a

[1]Based on http://docs.aws.amazon.com/AmazonVPC/latest/NetworkAdminGuide/images/Branch_Offices.png

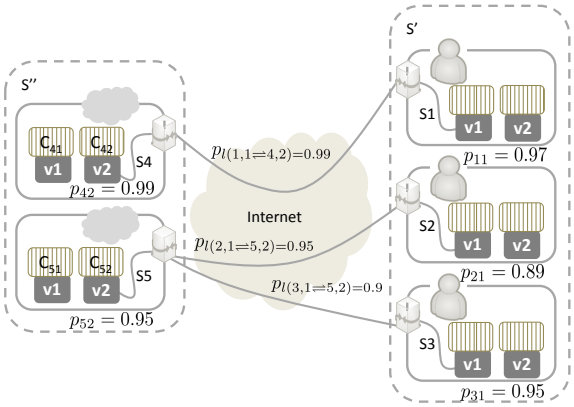


Figure 3: Robust Cloud Monitor Placement example.

specific cloud application must be placed on different sites and, following our hybrid monitoring approach, at least one monitor must be placed on consumer and provider side, respectively. In addition, a monitoring unit for a specific cloud application is not allowed to be placed on the VM where the cloud application is running. As already mentioned, this is due to the fact that a monitoring unit is not able to report a downtime when the underlying VM is currently down.

We denote this problem as the *Robust Cloud Monitor Placement Problem* (RCMPP). A simplified example for a RCMPP instance neglecting resource demands and supplies for ease of clarity is depicted in Figure 3. The instance exhibits two provider sites S_4, S_5 and three consumer sites S_1, S_2, S_3 each running two VMs. A set of applications is running on each VM. For example, C_{42} denotes the set of applications running on VM no. 2 on site S_4. The corresponding link between this VM and VM no. 1 on consumer site S_1 exhibits a reliability of $p_{l(1,1=4,2)} = 0.99$ and VM no. 1 exhibits a reliability of $p_{11} = 0.97$. In our work, we are only interested in monitoring the cloud applications on provider side. However, the applications running on consumer side are also considered in the model in the form of remaining resource supplies of their VMs. If we assume that only one application is part of each set $C_{s'v'}$ on provider side in the example and that the enterprise cloud user specifies a redundancy factor $rf_{s'v'c} = 2$ for each application, 8 monitoring units will be assigned to VMs in total, thereby placing one monitoring unit on consumer side and one on provider side for each application.

4.2 Formal Model

In order to develop strategies to solve the RCMPP, we transform the problem into an optimization model. The resulting optimization model is depicted in Model 1 and will be described in the following.

Model 1: Robust Cloud Monitor Placement Problem.

$$Maximize \quad \{p^{mon}_{s'v'c}(x) | s' \in S', v' \in V_{s'}, c \in C_{s'v'}\} \quad (1)$$

$$p^{mon}_{s'v'c}(x) = 1 - \prod_{s \in S, v \in V_s} (q^{path}_{svs'v'})^{x_{svs'v'c}} \quad (2)$$

$$q^{path}_{svs'v'} = [(1 - p_{sv}) + (1 - p_{l(sv=s'v')}) - (1 - p_{sv})\,(1 - p_{l(sv=s'v')})] \quad (3)$$

subject to

$$\sum_{s \in S, v \in V_s} x_{svs'v'c} = rf_{s'v'c} \quad (4)$$
$$\forall s' \in S', v' \in V_{s'}, c \in C_{s'v'}, rf_{s'v'c} \geq 2$$

$$\sum_{s' \in S', v' \in V_{s'}, c \in C_{s'v'}} rd_{s'v'cr}\, x_{svs'v'c} \leq rs_{svr} \quad (5)$$
$$\forall s \in S, v \in V_s, r \in R$$

$$\sum_{v \in V_s} x_{svs'v'c} \leq 1 \quad (6)$$
$$\forall s \in S, s' \in S', v' \in V_{s'}, c \in C_{s'v'}$$

$$\sum_{s \in S, v \in V_s} x_{svs'v'c} \geq 1 \quad (7)$$
$$\forall s' \in S', v' \in V_{s'}, c \in C_{s'v'}, s = \{d+1, ..., n\}$$

$$\sum_{s \in S, v \in V_s} x_{svs'v'c} \geq 1 \quad (8)$$
$$\forall s' \in S', v' \in V_{s'}, c \in C_{s'v'}, s = \{1, ..., d\}$$

$$x_{svs'v'c} = 0 \quad (9)$$
$$\forall c \in C_{s'v'},\ s = s' \text{ and } v = v'$$

$$x_{svs'v'c} \in \{0, 1\} \quad (10)$$
$$\forall s \in S, v \in V_s, s' \in S', v' \in V_{s'}, c \in C_{s'v'}$$

As already stated, the objective of the RCMPP is to assign the monitoring units of all cloud applications to VMs on consumer and provider side so that the resulting placement maximizes the probability that at least one of the monitoring units for each cloud application is working. The objective can be described by a set of multiple potentially conflicting objective functions that we want to maximize simultaneously (cf. Equation 1). Each objective function expresses the probability $p^{mon}_{s'v'c}(x)$ to reliably monitor a specific application, i.e., at least one of the redundant monitoring units is working properly. $p^{mon}_{s'v'c}(x)$ equals 1 minus the probability that all monitors for a specific cloud application $c \in C_{s'v'}$ fail (cf. Equation 2). The decision variable $x_{svs'v'c}$ is defined in Equation 10 as binary and indicates whether VM $v \in V_s$ running on

Model 2: Optimization of Complementary Objectives.

$$Minimize \quad \{q^{mon}_{s'v'c}(x) | s' \in S', v' \in V_{s'}, c \in C_{s'v'}\} \quad (11)$$

$$q^{mon}_{s'v'c}(x) = \prod_{s \in S, v \in V_s} (q^{path}_{svs'v'})^{x_{svs'v'c}} \quad (12)$$

$$q^{path}_{svs'v'} = [(1 - p_{sv}) + (1 - p_{l(sv \rightleftharpoons s'v')}) - (1 - p_{sv})\ (1 - p_{l(sv \rightleftharpoons s'v')})] \quad (13)$$

site $s \in S$ is selected to monitor cloud application $c \in C_{s'v'}$ running on VM $v' \in V_{s'}$ on site $s' \in S'$. x denotes the vector of all decision variables $x_{svs'v'c}$ (cf. Equation 1). Equation 3 determines the probability $q^{path}_{svs'v'}$ for a monitoring unit of cloud application $c \in C_{s'v'}$ placed on VM $v \in V_s$ on site $s \in S$ to fail, thereby considering the reliability of the VM $v \in V_s$ and the reliability of the path interconnecting VM $v \in V_s$ and the VM of the cloud application. Equation 4 ensures that a cloud application $c \in C_{s'v'}$ is monitored by $rf_{s'v'c}$ redundant monitoring units. For each application, at least two monitoring units must be assigned to VMs, one on consumer side and one on provider side (cf. Equations 7 and 8). Equation 5 defines that each VM $v \in V_s$ offers sufficient resource supplies to serve the assigned monitoring units and Equation 6 makes sure that all monitoring units for a specific cloud application $c \in C_{s'v'}$ are placed on different sites. Finally, Equation 9 ensures that a monitoring unit for a specific cloud application $c \in C_{s'v'}$ is not assigned to the VM $v' \in V_{s'}$ on site $s' \in S'$ where the cloud application is also running on.

As can be seen from Model 1, the RCMPP constitutes a multi-objective optimization problem. Hence, no unique solution of the problem exists. Furthermore, the RCMPP is nonlinear due to Equation 2. However, the problem can be linearized and transformed into a single-objective optimization problem as described in the next section.

5 OPTIMIZATION APPROACH

We propose an initial solution approach based on linearization and transformation of the RCMPP into a single-objective optimization problem. In doing so, off-the-shelf optimization algorithms, such as branch-and-bound (Hillier and Liebermann, 2005), can be applied in order to obtain an exact solution.

For linearization of the RCMPP, we follow a two-step approach: In the first step, we build a new set of objective functions representing the complementary objectives of the former Model 1 and turn the maximization problem into a minimization problem (cf. Model 2). $q^{mon}_{s'v'c}(x)$ in Model 2 determines the probability that all monitors for a specific cloud application $c \in C_{s'v'}$ fail and thus, represents the complementary probability to $p^{mon}_{s'v'c}(x)$ in the former Model 1. In the second step, we can now linearize the problem by taking the logarithm of both sides (cf. Equation 14), an approach that is also followed by (Andreas and Smith, 2008). Model 3 shows the result of the linearization. Since no system is without failure, we assume $q^{path}_{svs'v'} > 0$. Please note, that the Equations 4 to 10 of Model 1 also belong to Model 2 and Model 3, but have been neglected due to lack of space. Finally, we have to transform Model 3 into a single-objective optimization problem. For this purpose, we apply a so-called *minimax strategy* (Jensen and Bard, 2003), i.e., our transformation is based on a worst-case analysis, where we aim to minimize the worst possible outcome. Hence, the former objective of minimizing the probability $q^{log}_{s'v'c}(x)$ that all monitors fail for all cloud applications simultaneously can be transformed into minimizing the maximum probability of all $q^{log}_{s'v'c}(x)$ (cf. Equation 19 in Model 4). In doing so, the maximum (i.e., worst) probability of all $q^{log}_{s'v'c}(x)$ can be represented by a new decision variable $z \in \mathbb{R}$. Furthermore, we have to add $|C_{s'v'}|$ new constraints $\forall s' \in S', v' \in V_{s'}$ (cf. Equation 20 in Model 4) to the set of former constraints already introduced in Model 1. As can be seen from Model 4, the resulting problem constitutes a mixed-integer linear programming problem that can be solved using branch-and-bound (Hillier and Liebermann, 2005).

Model 3: Linearization of RCMPP.

$$log(q^{mon}_{s'v'c}(x)) = log(\prod_{s \in S, v \in V_s} (q^{path}_{svs'v'})^{x_{svs'v'c}}) \quad (14)$$

leads to

$$Minimize \quad \{q^{log}_{s'v'c}(x) | s' \in S', v' \in V_{s'}, c \in C_{s'v'}\} \quad (15)$$

$$q^{log}_{s'v'c}(x) = \sum_{s \in S, v \in V_s} x_{svs'v'c}\ log(q^{path}_{svs'v'}) \quad (16)$$

$$q^{path}_{svs'v'} = [(1 - p_{sv}) + (1 - p_{l(sv \rightleftharpoons s'v')}) - (1 - p_{sv})\ (1 - p_{l(sv \rightleftharpoons s'v')})] \quad (17)$$

$$q^{path}_{svs'v'} > 0\ \forall s \in S, v \in V_s, s' \in S', v' \in V_{s'} \quad (18)$$

Model 4: Transformation to Single-Objective RCMPP.

$$Minimize \quad z \tag{19}$$

subject to

$$q^{log}_{s'v'c}(x) \leq z \tag{20}$$

$$\forall s' \in S', v' \in V_{s'}, c \in C_{s'v'}, z \in \mathbb{R}$$

6 SUMMARY AND OUTLOOK

Although cloud-based service delivery is a very flexible and convenient way for consumers to obtain computing resources, it is attended with a shift of responsibility to the cloud provider and thus, with a loss of control for consumers. Negotiating SLAs with providers and using their provisioned monitoring solutions to verify SLA compliance later on is not considered as sufficient by consumers. Therefore, we have designed a hybrid monitoring approach for availability verification of cloud applications from a consumer's perspective in our former work in (Siebenhaar et al., 2013). However, since our experiments revealed that our approach is sensitive to network impairments, we examined the robust placement of monitoring units in the work at hand. In this paper, we introduced the *Robust Cloud Monitor Placement Problem* (RCMPP) and a corresponding, formal optimization model. Furthermore, we proposed an initial solution approach based on transformations turning the nonlinear, multi-objective RCMPP into a mixed-integer linear programming problem. An exact solution can then be obtained using the branch-and-bound optimization algorithm.

In future work, we will implement and evaluate our proposed optimization approach. Furthermore, we plan to extend the proposed model to consider other objectives such as the total monitoring costs.

ACKNOWLEDGEMENTS

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) under grant no. "01|C12S01V" in the context of the Software-Cluster project SINNODIUM (www.software-cluster.org), E-Finance Lab Frankfurt am Main e.V. (http://www.efinancelab.com), and the German Research Foundation (DFG) in the Collaborative Research Center (SFB) 1053 – MAKI. The authors assume responsibility for the content.

REFERENCES

Amazon (2008). Amazon ec2 service level agreement. http://aws.amazon.com/ec2-sla/, [last access: 3 December 2012].

Andreas, A. K. and Smith, J. C. (2008). Mathematical Programming Algorithms for Two-Path Routing Problems with Reliability Considerations. *INFORMS Journal on Computing*, 20(4):553–564.

Bin, E., Biran, O., Boni, O., Hadad, E., Kolodner, E., Moatti, Y., and Lorenz, D. (2011). Guaranteeing High Availability Goals for Virtual Machine Placement. In *31st International Conference on Distributed Computing Systems (ICDCS)*, pages 700–709.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. (2009). Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility. *Future Generation Computer Systems*, 25(6):599–616.

CSA and ISACA (2012). Cloud Computing Market Maturity. Study Results. Cloud Security Alliance and ISACA. http://www.isaca.org/Knowledge-Center/Research/Documents/2012-Cloud-Computing-Market-Maturity-Study-Results.pdf, [last access: 28 November 2012].

Hillier, F. S. and Liebermann, G. J. (2005). *Inroduction to Operations Research*. McGraw-Hill, 8th edition.

Jensen, P. A. and Bard, J. F. (2003). Appendix A: Equivalent Linear Programs. In *Supplements to Operations Research Models and Methods*. John Wiley and Sons. http://www.me.utexas.edu/ jensen/ORMM/supplements/units/lp_models/equivalent.pdf [last access: 12 January 2014].

Kamal, J. and Vazirani, V. V. (2000). An Approximation Algorithm for the Fault Tolerant Metric Facility Location Problem. In Jansen, K. and Khuller, S., editors, *Approximation Algorithms for Combinatorial Optimization*, volume 1913 of *Lecture Notes in Computer Science*, pages 177–182. Springer.

Mell, P. and Grance, T. (2011). The NIST Definition of Cloud Computing. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, [Last access: 28 November 2012].

Natu, M. and Sethi, A. S. (2008). Probe Station Placement for Robust Monitoring of Networks. *Journal of Network and Systems Management*, 16(4):351–374.

Patel, P., Ranabahu, A., and Sheth, A. (2009). Service Level Agreement in Cloud Computing. Technical report, Knoesis Center, Wright State University, USA.

Sharma, P., Chatterjee, S., and Sharma, D. (2013). CloudView: Enabling Tenants to Monitor and Control their Cloud Instantiations. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 443–449.

Siebenhaar, M., Wenge, O., Hans, R., Tercan, H., and Steinmetz, R. (2013). Verifying the Availability of Cloud Applications. In Jarke, M. and Helfert, M., editors, *Proceedings of the 3rd International Conference on Cloud Computing and Services Science (CLOSER 2013)*, pages 489–494. SciTe Press.

Optimizing the Stretch for Virtual Screening Application on Pilot-agent Platforms on Grid/Cloud by using Multi-level Queue-based Scheduling

T. Q. Bui[1,2], E. Medernach[1] , V. Breton[1] and H. Q. Nguyen[2]
[1]*Laboratoire de Physique Corpusculaire, Universite Blaise Pascal, 63171 Aubiere, France*
[2]*Institut de la Francophonie pour l'Informatique, Vietnam National University, Hanoi, Vietnam*
{bui, medernach, breton}@clermont.in2p3.fr, nguyen.hong.quang@ifi.edu.vn

Keywords: Virtual Screening, Grid Computing, Cloud Computing, Scheduling, Fairness, Stretch, Online-algorithm, Multilevel Queue Scheduling.

Abstract: Virtual screening has proven very effective on grid infrastructures. We focus on finding platform scheduling policy for pilot-agent platform shared by many virtual screening users. They need a suitable scheduling algorithm at platform level to ensure a certain fairness between users. Optimal criterion used in our research is the stretch, a measure for user experience on the platform. From our latest research (Quang et al., 2013), simulation result and experimentation on real pilot agent platform showed that SPT policy is the best policy in 4 different existing scheduling policies (FIFO, SPT, LPT and Round Robin) for optimizing the stretch. However, research on real grid workload (Medernach, 2005) showed that there are two types of grid user: normal users who submit frequently little jobs to grid and data challenge users who submit occasionally large number of jobs to grid. And SPT policy, in particularly, is not appropriate for data challenge user because they have to wait always normal user. In this paper, we proposed a new policy named SPT-SPT which uses multi-level queue scheduling technique for scheduling in a pilot agent platform. In SPT-SPT policy, the administrator creates two separate user groups in the platform: Normal group and Data Challenge group. Each group has their own task queue in the platform and SPT policy is applied on it. A parameter p ($p \epsilon [0,1]$), the probability that task queue is chosen to send pilot agent their task, is assigned to one task queue and 1-p for the other one. This policy improves user experience for Data Challenge group and do not impact very much for Normal group.

1 INTRODUCTION

In silico drug discovery (*in silico* means computer-assisted) offers an efficient alternative to reduce the cost of drug development and to speed-up the discovery process. Virtual screening is achieved through a pipeline analysis whose first step requires using a docking software such as Autodock, DOCK or FlexX to predict potential interacting complexes of small molecules in protein binding sites. From now, we use VS to denote virtual screening.

Large scale virtual screening, and especially its docking step, consumes large computing resources. As docking is an embarrassingly parallel process where thousands to millions of compounds are tested in silico against a biological target, it was successfully deployed on grid computing to reduce the computation time. Some large scale VS projects in the past have been deployed successfully on grids such as WISDOM-I (Jacq et al., 2006) and WISDOM-II (Kasam et al., 2009) on malaria or Avian Flu Data Challenge (Jacq et al., 2008).

Pilot-agent platforms are tools used for submitting and controlling a large number of user jobs on grid infrastructures. Several pilot agent platforms have been developed such as WPE (Kasam et al., 2009), DIRAC (van Herwijnen et al., 2003), DIANE (Mościcki, 2003), glideinWMS (Sfiligoi, 2008), PanDA(Maeno, 2008). The DIRAC pilot-agent platform is now available to the users of several multi-disciplinary virtual organizations on EGI (the European Grid Initiative). Because many users share the DIRAC pilot-agent platform, it is important to define a scheduling policy to ensure a certain degree of fairness so that users receive a fair share of system resources. The scheduling policies used on the existing pilot-agent platforms on EGI,

are respectively First In First Out (FIFO) policy in WPE platform or Round Robin policy in DIRAC platform. The VS project has specific properties such as divisibility in many independent docking tasks, no order of execution constraints and comparable execution time of all docking tasks. In this paper, we focus on evaluating suitable online scheduling policies for the VS application on the pilot-agent platform to improve user's satisfaction in the system.

Our research is also relevant to applications which have the same properties of VS application (divisibility in many independent tasks, no order of execution constraints and comparable execution time of all tasks) on pilot agent platform on grid/cloud. These applications are used in a variety of scenarios, including data mining, massive searches (such as key breaking), parameter sweeps, simulations, fractal calculations, computational biology, and computer imaging.

1.1 Pull Model, 2-Level Scheduling and Limited Machine Availability Property of Scheduling

A pilot-agent platform uses pull model for efficient submission and controlling of user tasks: tasks are no longer pushed through the grid scheduler but are put in a master pool and pulled by pilot agents running on computing nodes. Scheduling job is the process of ordering tasks in this pool. List scheduling is applied in it. The pilot-agent itself is a regular grid job, which is started through a grid resource manager. It is submitted automatically by platform and run on a computing machine on grid. We can see pilot agent as container of jobs. Pull model adapts to heterogeneous property of grid (faster machine will pull more tasks than the other), reduces faults (resubmission of failed tasks) and improves latency (the waiting time of job in grid scheduler is reduced).

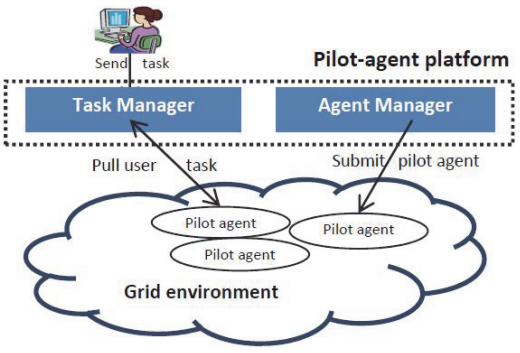


Figure 1: Pull model in pilot agent platform on grid.

As shown on figure 1, a pilot-agent platform has two main modules, the Task Manager and the Agent Manager.

Pilot agent is submitted automatically to grid by Agent Manager. Then it communicates with Task Manager and asks a user task to be executed. Task Manager receives tasks from user and control queue of user tasks. It receives also request of pilot agent, choose some task from queue and sends task to pilot agent.

Scheduling of pilot agent platform on grid takes place at site and platform level. The site level scheduling takes place in the site scheduler. Pilot agents sent by the platform are distributed to the sites according to the grid scheduling policy. The platform level scheduling is done by the platform's Task Manager. User sends the VS project to the Task Manager, where docking tasks are put in the task queue. The Task Manager Scheduler calculates task priorities, and responds to pilot agents requests by sending to them tasks ranked with the highest priority. There are underlying grid architectural scheduling and logical scheduling for the specific grid applications. And in our research, we concern with scheduling issue for many virtual screening application users who share the grid resources given to the same group privilege.

Moreover, platform level scheduling has limited machine availability property. Because each computing center imposes some limits to the maximum computing time for grid job, each pilot agent is available for a limited period. Therefore the number of machines available for the platform changes over time. This specific property makes our analysis directly relevant to cloud infrastructures where users buy computing resources for a limited time.

This paper focuses on finding out the most suitable scheduling policy at platform level to optimize the satisfaction of VS users.

1.2 The Stretch – A Measure for User's Satisfaction in Platform

To work on the fairness of scheduling policy, we need to define a good metric for the satisfaction of an individual user on platform. In the parallel scheduling literature, metrics used to measure the performance of scheduling policies can be classified in two groups: System-centric metrics and Job-centric metrics:

- System-centric metrics to assess platform utilization: C_j denotes the completion time of job j. Makespan (the maximum of the job

termination time, $\max_j C_j$) or the sum of completion time ($\sum_j C_j$) are common objective functions. Minimization of makespan or sum of completion times is conceptually a system-centric approach.

- Job-centric metrics (Flow time, stretch...) to assess user experience: Flow time F is the time an individual job spends in the platform. The stretch S (also called slowdown) is a particular case of weighted flow time: for job j with size W_j and flow time F_j, the stretch is defined as F_j/W_j. In the context of variable job sizes, the stretch is more relevant to describe user experience than the flow time (Legrand et al., 2006).

This paper focuses on user-centric metric. The stretch is here measured for a group of jobs all belonging to the same user. Assume that user has U jobs, F_j is flow time of job j, W is total size of U jobs, the stretch is then defined as $\max_{j \epsilon U} F_j/W$

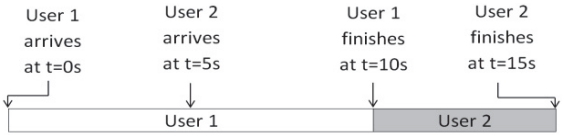


Figure 2: Example of the stretch for two users.

Figure 2 illustrates the mean of stretch in the case of two users sending jobs to a platform (figure 2). User 1 has a job with size 10 that reaches the platform at t=0s. User 2 has a job with size 5 that reaches the platform at t=5s. User 1 job is executed before user 2 job. As in figure 2, user 1 finishes at t=10s. Because user 2 has to wait user 1 job completion, he finishes at t=15s. Although the two users spent the same time on the platform (10s), user 2 satisfaction is worse than user 1's. The stretch of user 1 is equal to 10/10=1 while the stretch for user 2 is equal to 10/5=2. This example illustrates why the stretch S measures the user experience on the platform: the larger the stretch, the lower the satisfaction.

Our goal is to identify the best scheduling policy to minimize the stretch of all VS users of a shared pilot-agent platform. We use the max-stretch (Smax) metrics as measures of fairness in our scheduling problem. In our latest research (Quang et al., 2013), we have compared two well-known scheduling policies, Shortest Processing Time (SPT-the user with the least number of tasks has the highest priority) and Longest Processing Time (LPT: the user with the greatest number of tasks has the highest priority), to the scheduling policies currently used on the existing platforms (FIFO and Round Robin). Simulation result and experimentation result on real platform has shown that SPT is the best policy in these 4 policies for online job-centric stretch optimization with virtual screening application on pilot-agent platform on grid/cloud(Quang et al., 2013).

Although SPT policy is very good online algorithm for optimization of the stretch but this policy has a disadvantage: it has the tendency to push users with many tasks to the end of the task queue. Sometimes these users have to wait a long time for a long series of users with less number of jobs. In the worst case, they have to wait forever. Furthermore, research on grid workloads in (Medernach, 2005) showed that there are two types of grid users: normal users and data challenge users. Normal user submits to the grid little number of tasks, but the number of user in this group is very large. While data challenge user group submit to the grid very large number of jobs, but the number of users in this group is small. If we use the original SPT policy for all of user in pilot agent platform, data challenge user will be negatively affected due to large number of user in normal group.

In this paper, we propose a new scheduling policy named SPT-SPT for platform level scheduling in pilot agent platform which uses multi-level queue scheduling techniques to improve the stretch of VS users. Instead of using one task queue which is implemented by SPT policy for all users, we use two separate task queues: one for normal user group and another one for data challenge user group. These two task queues are both using SPT policy to optimize the user's stretch in each one. Moreover, a task queue is assigned a parameter p ($p \epsilon [0, 1]$) and the rest one with 1 - p. This parameter is the probability that task queue will be selected by Task Manager when Task Manager receives request from pilot agents. We can see that for $p > 0$, this policy ensures that the data challenge group did not have to wait for the normal group to be entirely empty. Therefore all users will get an opportunity to utilize grid resources efficiently. The rest of paper is organized as follows. Section 2 presents the related works. In section 3, the SPT-SPT scheduling policy is proposed. Finally, section 4 is our conclusion and perspective on this research.

2 STATE OF THE ART

2.1 Grid Scheduling

Grid scheduling has been abundantly studied: some

surveys of grid scheduling algorithms are proposed in (Maruthanayagam, 2010); (Jiang et al., 2007) and performance of some priority rule scheduling algorithms is presented in (Azmi, 2011). DIET platform (Marrow et al., 2003) is a GridRPC middleware relying on the client/agent/server paradigm. The scheduling on DIET changes from FIFO, Round Robin and CPU-based scheduling. But the operation of DIET platform is different with pilot-agent platform: DIET use both "push" and "pull" scheduling. Mandatory requests are pushed from clients to resources, whereas optional requests are pulled by resources from clients. Pilot-agent platform takes most scheduling decisions in a centralized agent, in contrast, each client and each server contributes to taking scheduling decisions in DIET. Therefore, the solutions brought by research of scheduling problem on DIET platform are not directly applicable to our problem statement.

In (Berman et al., 1996), author presents a scheduling solution in application level called AppLeS. They describe an application specific approach to scheduling individual parallel applications on production heterogeneous systems. They utilize comprehensive information about application and resource to optimize execution time of application on grid. Our goal is not to optimize the execution time of all users but the quality of service for each user.

Existing pilot agent platforms such as DIANE (Mościcki, 2003), WPE(Kasam et al., 2009) and DIRAC (van Herwijnen et al., 2003) have different scheduling policies: WPE and DIANE platforms use FIFO while DIRAC uses Round Robin policy. The VS projects have specific properties such as divisibility in many docking tasks and no order of execution constraints. Therefore we need to find a suitable online scheduling policy for the VS application on the pilot-agent platform. Fortunately, in some platform such as DIRAC platform, we can configure the specific scheduling policy for a user group sharing the same application. So we can apply suitable policy in a VS user group to improve fairness.

2.2 Cloud Scheduling

As mentioned earlier, the limited machine availability property of the scheduling problem on pilot agent platform is similar with scheduling on cloud environment because on cloud environment, user buys some resources with limited duration. When a VS project is deployed on an IAAS cloud, docking task will be executed on a virtual machine with limited availability.

Some researches on cloud scheduling such as (Pandey et al., 2010); (Li et al., 2011) have presented their scheduling algorithms on cloud to optimize the speed of resources allocation, the price to pay and the utilization of system resource. But our object is optimization of the fairness of users when they share pilot-agent platform together.

In (Luckow et al., 2010), author proposed the design and implementation of a SAGA-based Pilot-Job system, which supports a wide range of application types, and is usable over a broad range of infrastructures from grids/clusters to cloud computing. In (Fifield et al., 2011), author showed also an extension of the pilot agent platform DIRAC on cloud computing by submitting pilot agent on Virtual Machine on cloud such as Amazon EC2. Therefore our research is also relevant to pilot-agent platforms on Cloud environments.

2.3 Scheduling for Stretch Optimization with Limited Machine Availability Constraints

Many groups have conducted research on optimizing job-centric stretch in the context of dedicated machines (i.e. always available). In (Muthukrishnan et al., 1999), S. Muthukrishnan presented the efficiency of the optimal on-line algorithm SPT on uniprocessor and multi-processor. Their objective is optimizing the average of the stretch. In (Legrand et al., 2006), Legrand has shown that SPT is quite effective at max-stretch and sum-stretch optimization in problems with continuous machines. But compared to these studies, our scheduling problem uses a user-centric definition of stretch and adds an additional constraint: machines have limited availability. With this property, the number of machines available for platform changes over time and the complexity of problem increases. In (Schmidt, 2000), authors have reviewed some scheduling algorithm in the context of limited machine availability. LPT is one of the online scheduling algorithms proposed in this research. But these researches are done on system-centric metrics (makespan, sum of completion time…). In our latest research for scheduling for stretch optimization with limited machine availability constraints, we compared two well-known scheduling policies, SPT and LPT, to the scheduling policies currently used on the existing platforms (FIFO and Round Robin). Simulation result and experimentation result on real platform showed that SPT policy is the best policy in these 4 policies for optimization of user stretch in

the context of limited machine availability.

3 SOLUTION PROPOSED

In this section, we briefly explain the proposed solution using multilevel queue technique in Task Manager of pilot agent platform. Administrator of pilot agent platform creates two groups for VS users: Normal group and Data Challenge group. VS user is assigned to Normal group by default. When someone needs to process a big virtual screening project, he will contact with administrator of pilot agent platform to change his role to Data Challenge group in some days or some weeks.

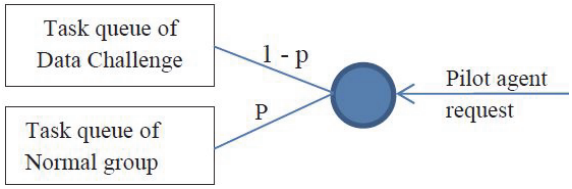


Figure 3: SPT-SPT policy with two task queues.

In Task Manager module, we build two separate task queues: one for normal group and another for data challenge group. We assign a priority p to normal group task queue and 1 – p to the data challenge group one. The Task Manager will use this probability as index in order to choose task queue to send their task when it receives request from pilot agent.

According to our latest research, SPT is better than FIFO, LPT, RR for minimizing the user stretch. Therefore these both task queues use SPT policy for optimizing the stretch of user on each one (We tried with policy SPT-RR: SPT on normal group task queue and Round Robin in data challenge task queue and SPT-FIFO: SPT on normal group task queue and FIFO on data challenge task queue. The result of SPT-RR and SPT-FIFO is worse than SPT policy). We use algorithm SPT-SPT to control user task in Task Manager as algorithm 1. Administrator can change value of parameter p in the configuration of pilot agent platform.

We can see that with $0 < p < 1$ there are always pilot agents which take a task from Data Challenge group. Therefore, Data Challenge user does not have to wait for a long series of normal users which have less number of tasks. We will use our simulator to find out the best value of p to decrease S_{max} of Data Challenge group and do not increase very much S_{max} of Normal group.

Algorithm 1: SPT-SPT policy in Task Manager scheduler.

```
p = parameter of task queue of normal user group
while (1)
    do
        if (receive request from pilot agent) then
            if (data challenge task queue is empty
                AND normal task queue is empty)
                Push pilot agent to pilot agent queue
            else if (data challenge task queue is
                                                empty)
                    Send task of normal task queue to
                                                pilot agent
            else if (normal task queue is empty)
                    Send task of data challenge queue to
                                                pilot agent
            else
                a = random(0,1)
                if (a ≤ p) then
                        Send task of normal task queue
                                                to pilot agent
                else
                        Send task of data challenge task
                                                queue to pilot agent
                end if
            end if
        end if

end while
```

4 CONCLUSIONS

The paper describes a new scheduling policy for virtual screening application on pilot agent platform for optimizing the stretch of user. We proposed SPT-SPT policy which uses multilevel queue technique for platform level scheduling on pilot agent platform. This approach based on the research of grid workload which has showed that there are two types of user (many users submitting small number of tasks and a little user submitting large number of tasks). Because Infrastructure as a Service cloud users buy access to computing resources for a limited time. This is similar with limited availability of pilot agent on grid. Therefore, we can also propose to implement SPT-SPT in deployment of virtual screening application on cloud environments.

REFERENCES

Azmi, B., 2011. Performance Comparison of Priority Rule Scheduling Algorithms Using Different Inter Arrival Time Jobs in Grid Environment. *International Journal of Grid and Distributed Computing*, 4(3), pp.61–70.

Berman, F. et al., 1996. Application-level scheduling on distributed heterogeneous networks. In *Supercomputing, 1996. Proceedings of the 1996 ACM/IEEE Conference on*. p. 39.

Fifield, T. et al., 2011. Integration of cloud, grid and local cluster resources with DIRAC. *Journal of Physics: Conference Series*, 331(6), p.062009.

Van Herwijnen, E. et al., 2003. Dirac-distributed infrastructure with remote agent control. In *Conference for Computing in High-Energy and Nuclear Physics (CHEP 03)*.

Jacq, N. et al., 2008. Grid-enabled virtual screening against malaria. *Journal of Grid Computing*, 6(1), pp.29–43.

Jacq, N. et al., 2006. Large scale in silico screening on grid infrastructures. *arXiv preprint cs/0611084*. Available at: http://arxiv.org/abs/cs/0611084 [Accessed December 31, 2013].

Jiang, C. et al., 2007. A survey of job scheduling in grids. In *Advances in Data and Web Management*. Springer, pp. 419–427.

Kasam, V. et al., 2009. WISDOM-II: screening against multiple targets implicated in malaria using computational grid infrastructures. *Malaria journal*, 8, p.88.

Legrand, A., Su, A. & Vivien, F., 2006. Minimizing the stretch when scheduling flows of biological requests. *Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures*, pp.103–112.

Li, W., Tordsson, J. & Elmroth, E., 2011. Modeling for dynamic cloud scheduling via migration of virtual machines. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*. pp. 163–171.

Luckow, A., Lacinski, L. & Jha, S., 2010. SAGA BigJob: An extensible and interoperable pilot-job abstraction for distributed applications and systems. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*. pp. 135–144.

Marrow, P. et al., 2003. DIET-a scalable, robust and adaptable multi-agent platform for information management. *BT technology journal*, 21(4), pp.130–137.

Maruthanayagam, D., 2010. Grid scheduling algorithms: A survey. *International Journal of Current Research*, 11(2), pp.228–235.

Medernach, E., 2005. Workload analysis of a cluster in a grid environment. *Job scheduling strategies for parallel processing*, (June).

Mościcki, J.T., 2003. Distributed analysis environment for HEP and interdisciplinary applications. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 502(2), pp.426–429.

Muthukrishnan, S. et al., 1999. Online scheduling to minimize average stretch. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*. pp. 433–443.

Pandey, S. et al., 2010. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*. pp. 400–407.

Quang, B.T. et al., 2013. Stretch optimization for virtual screening on multi-user pilot-agent platforms on grid/cloud. In *Proceedings of the Fourth Symposium on Information and Communication Technology*. pp. 301–310.

Schmidt, G., 2000. Scheduling with limited machine availability. *European Journal of Operational Research*, 121(1), pp.1–15.

Migrating Relational Databases to the Cloud
Rethinking the Necessity of Rapid Elasticity

Kevin Williams
Center for Information Systems & Technology,
Claremont Graduate University, 150 E. 10th Street, Claremont, California, U.S.A.
kevin.williams@cgu.edu

Keywords: Transactional Processing Relational Databases, Cloud Computing, Scalability, Rapid Elasticity, Elasticity, Horizontal Scalability, Vertical Scalability, Inelastic Systems, Incarnational Scalability, ACID Compliance, System Efficiency.

Abstract: Rapid Elasticity is often described as an essential characteristic of cloud computing, but there are some good reasons to rethink how it is described and implemented – especially as it relates to transaction processing relational databases, which are broadly used in many organizations. These types of relational databases, which support transaction processing, strictly adhere to what has been called the ACID compliance model, where the Atomicity, Consistency, Isolation, and Durability of transactions are guaranteed to ensure a reliable transaction system. Databases in the cloud often sacrifice one or more of these essential ACID properties to achieve the desired Rapid Elasticity. This conflict between Rapid Elasticity and ACID compliance explains why relatively few existing transactional processing relational databases have been deployed to the cloud without undergoing significant revision. This paper argues for an expanded definition of the essential characteristic of cloud computing on which the underlying goal of Rapid Elasticity is based, but where the ACID compliance remains intact and many of the advantages of cloud computing can be utilized.

1 INTRODUCTION

Rapid Elasticity (nearly automatic unlimited scaling of computer resources upon demand) is often described as an essential characteristic of cloud computing (Mell and Grance 2011), but there are some good reasons to rethink how this quality is described and implemented – especially as it relates to transaction processing relational databases, as broadly used in many organizations. Relational databases which are ACID compliant (supporting the qualities of Atomicity, Consistency, Isolation, Durability) might not be able to support Rapid Elasticity, but does this mean that they are ineligible to be considered cloud computing? This paper will seek to articulate the goal behind Rapid Elasticity and evaluate whether Rapid Elasticity is the only way to meet this underlying scalability goal. The goals for cloud computing should be scalability and efficiency, not Rapid Elasticity as an essential characteristic. This paper will then suggest an alternative method for cloud computing systems to achieve the goal behind Rapid Elasticity that might have value for transaction processing relational databases and other systems, which are not natively rapidly elastically compliant.

This paper will discuss scalability, elasticity, and efficiency as they relate to rapid elasticity. A comparison will be made between rapidly elastic and inelastic systems. Two types of scalability methods will be discussed: Horizontal Scalability (adding additional systems) and Vertical Scalability (adding more power to existing systems), highlighting some of the advantages and limitations of each. A scalability method for transactional processing relational databases in the cloud will be proposed called Incarnational Scalability. A brief discussion will highlight how Incarnational Scalability could be implemented and what its benefits might be. Finally, the paper concludes with the need to broaden the definition of rapid elasticity as an essential characteristic of cloud computing.

2 SCALABILITY, ELASTICITY, AND EFFICIENCY

Scalability is the quality of a system to handle increased workload (Weinstock and Goodenough 2006). Thus as more workload is deployed to a system; the system is able to perform without failure or within acceptable levels. Scalability is a very desirable quality in a system, but overall service quality is based on a number of factors including reliability and responsiveness (Pitt, Watson et al. 1995, Buyya, Yeo et al. 2009).

One of the ways that cloud computing is expected to help solve the challenge of scalability is through the concept of elasticity which is defined by (Herbst, Kounev et al. 2013) as:

> Elasticity is the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manger, such that at each point in time the available resources *match* the current demand as closely as possible.

The appearance of "infinite computing resources" in cloud computing (Armbrust, Fox et al. 2010) seems to hold the promise that all scalability concerns can be resolved through the elasticity that is possible with cloud computing. The distinction between scalability (a goal) and elasticity (a method to reach a goal) can obscure the definition of what a cloud computing system is, since this then makes the claim that all scalability problems can be solved by elastically provisioning and de-provisioning resources.

Alternatively, another goal for systems is that of efficiency, defined as "the amount of resources consumed for processing a given amount of work" (Herbst, Kounev et al. 2013). More work being done by fewer resources has a higher efficiency and also has implications for the ability of the system to scale better. Thus efficiency and scalability are closely related. The greater the efficiency then the greater the ability of the system to handle increased workload.

2.1 Rapid Elasticity

One of the essential characteristics of cloud computing according to (Mell and Grance 2011) is Rapid Elasticity, defined by NIST as:

> Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.

This definition suggests that the essential characteristic or goal behind Rapid Elasticity is the unlimited, rapid, and efficient scalability of system capabilities. The problem that is being addressed by the "Rapid Elasticity" seems to be that the demand on the system is dynamic and random and therefore the cloud computing system must be able to respond quickly to the changes in demand, but without wasting resources. In the NIST definition of Rapid Elasticity there is the statement that "capabilities can be provisioned and released," but the word "capability" may not be the best word for what is happening as capability indicates a potential ability, not the usage of that ability.

2.2 Inelastic Systems

The type of system, which performs well when adding additional resources, can be called rapid elastically compliant. There are however, many systems, which are not elastically scalable including transaction processing relational databases. The relational data model when supporting transactional processing creates some dependencies that often get in the way of rapid scalability, including locking, latching, and deadlocks. Locking is meant to protect the integrity of the system; the integrity is considered to be more important than the system's scalability.

Jim Gray described the set of properties in a transactional processing system necessary for a reliable transactional processing database and these became known as ACID compliance: Atomicity, Consistency, Isolation, Durability (Gray 1981). There are few instances of large transactional processing relational databases as broadly used in organizations being moved to cloud systems and supporting the Rapid Elasticity model.

There is a mistaken idea that adding resources to a slow system will improve its performance. Cary Millsap indicates that more or faster resources will only improve database performance if the initial problem was slow or insufficient resources (Millsap and Holt 2003). Millsap continues that on transactional processing relational databases such as Oracle, adding additional resources can in fact exacerbate the performance and scalability problems as the system gets to the root culprit faster.

While there are many successful and highly elastic NoSQL database systems deployed on cloud systems, they often have to relax one or more of the

ACID compliance guarantees. Cassandra, a popular NoSQL database, describes ACID compliance as:

> Unlike relational databases, Cassandra does not offer fully ACID-compliant transactions. There is no locking or transactional dependencies when concurrently updating multiple rows or column families. But if by "transactions" you mean real-time data entry and retrieval, with durability and tunable consistency, then yes.

When the consistency guarantee is relaxed, this means that at some point in time the database can be in an inconsistent state. For example, in an ATM banking transaction, the money must be removed from the account at the same time that the money is released from the ATM. If either of the parts of the transaction fails, then both parts should fail. However, in a system that is eventually consistent, one part might succeed and the other part could fail. This would be disastrous for banking transactions – either for the client or for the bank.

Some efforts to address this conflict between the ACID compliance model (transactional support) and rapid scalability have been attempted (Das 2011) by developing a relational cloud:

> Statements and transactions spanning multiple nodes incur significant overhead, and are the main limiting factor to linear scalability in practice. (Curino, Jones et al. 2011)

Therefore, the awareness of the dependencies of the data relationships was not present when the applications were originally written. This may give some explanation for the difficulty in migrating existing transactional processing relational databases to the cloud and achieving Rapid Elasticity.

3 APPROACHES TO SCALABILITY

There are three main approaches to scalability in cloud computing: horizontal scalability, vertical scalability, and efficiency improvement. Horizontal and vertical scalability have been discussed, but improving the efficiency of the system is an important method to improve scalability – especially for systems that are inelastically scalable, such as transactional processing relational databases.

3.1 Horizontal Scalability

In the NIST definition of computing the idea of "provisioned and released" resources indicates that things are added or removed from the system, but this presupposes that the inability for the system to handle the additional load can be solved by additional resources. Currently, additional resources are added for horizontal scalability or vertical scalability, but only horizontal scaling is able to scale as to "appear to be unlimited."

Horizontal scaling is described as:

> Horizontal scaling is applicable for applications that have a clustered architecture with a gateway or a master node that distributes requests between the worker nodes (or VMs). If the workload increases, additional nodes are added to the cluster. During decrease in workload intensity, some nodes are removed from the cluster freeing up resources. In typical clustered architectures, the gateway maintains a list of nodes that are part of the cluster. The reconfiguration cost of horizontal scaling varies between applications and depends on the ease with which nodes can join or leave the cluster. (Dutta, Gera et al. 2012)

Therefore, the concept of Rapid Elasticity seems to be synonymous with the concept of horizontal scalability.

3.2 Vertical Scalability

Another method for increasing the ability of a system to handle load is vertical scalability. Vertical scaling is described (Dutta, Gera et al. 2012) as:

> Virtualization enables another way to add or remove resources to a virtual machine. Modern hypervisors support online VM resizing allowing one to add CPU or memory resources to a VM without bringing it down. Vertical scaling is used to denote the addition/deletion of resources to a virtual machine.

One of the features of cloud computing is that instead of buying a certain amount of hardware as capital expense, the resources are leased as operational expenses. This permits the rapid and dynamic movement of the system from less powerful hardware to more powerful hardware. While the scalability is not unlimited, the system can be moved to hardware that is able to handle increased load. Additionally, with vertical scalability, existing software doesn't have to be re-written to scale across multiple nodes (as in the case of horizontal scalability). This is of great benefit to systems that are not elastically scalable – such as transactional processing relational databases.

The vertical scalability available with cloud

computing resolves three important problems in organizations: buying too large a system initially, experiencing a delay when adding capacity, and encountering the problem of sunk costs. With vertical scalability, only the resources that are actually needed are allocated and then changed as needed. Depending on the system, the system might adjust resources throughout the day to handle increases and decreases in load. This is not Rapid Elasticity since the number of resources that can be added to an individual virtual machine bound the scale up.

The second benefit with vertical scalability as available with cloud computing is the relatively short delay in new system deployment and adding capacity. When servers are purchased in a capital expenditure scenario, the delays can be substantial, as the capital expenditure must go through the process of a cost benefit analysis, ordering, hardware delivery, and systems configuration.

The third benefit may be the elimination of sunk costs. Sunk costs will have no place in situations where capital expenditures are replaced with operational expenses, as in cloud computing.

Therefore, even for systems, which are not rapidly elastic, there are some very good reasons to move those systems to the cloud including the ability to take advantage of vertical scalability.

3.3 Improving Efficiency

The operation of a system is a collaboration and/or interaction between the system designers, the programmers, the hardware, the system elasticity, and the users, among others. Much of the efforts in cloud computing have seemed to focus on agnostically enabling Rapid Elasticity and ignoring the changes that can be made in systems to improve their performance, efficiency, and scalability.

Rather than concluding that transactional processing relational databases cannot be migrated to the cloud using current technology, this paper argues for a slightly different scalability model. This model uses the resources and benefits of cloud computing to improve scalability and efficiency of systems through improved testing initiatives.

In many organizations, besides the production systems, there are generally one or more development and test systems. In order to have valid tests, in many cases the development and test systems are equivalently sized to that of production. If the systems are less powerful than the production systems, there is the concern that the testing might not behave identically to the way production behaves in the same situation.

In order to have a reliable test environment, the test systems and the production systems must be made as similar as possible. However, the shrinking or sub-setting of full sized environments is difficult in practice. The cost of production-sized environments for testing and/or development can also be prohibitive.

The demand for the use of test systems is not consistent. Often test systems are some of the most highly scheduled systems in an organization. During some periods, such as pre-release testing, the demand for the test system might peak and multiple identical test systems would be required. Then, after testing has concluded, the demand may drop to zero.

The nature of test system demand seems to fit well with the scalability model of cloud computing, where multiple full-sized copies of the production system can be deployed for testing, but are only available during the actual testing. Once the testing has been completed, the test systems can be de-allocated.

3.4 Incarnational Scalability

The use of cloud copies of production systems for testing in order to improve system efficiency is both economically efficient (as it only incurs necessary expenditures) and scalable (as many or as few copies can be created as needed). Therefore, this paper proposes another method for reaching the goals of scalability and efficiency, different from vertical or horizontal scaling: Incarnational Scalability, where separate incarnations of the full production system are deployed for testing. To do effective testing one must test one change at a time so as to isolate whether that change improves the system or not. Unfortunately, most test environments do not have this luxury and multiple changes are tested at the same time because of system availability limitations. Testing in such an environment always leaves the results in doubt as to what change (or changes) actually have the most benefit. However, testing in the cloud, using as many multiple identical systems (incarnational scalability) as needed, allows one to test each change separately and determine its effect. In this model, the use of the cloud does not benefit existing systems directly through the addition resources, instead helps better use what resources you already have through better testing.

Incarnational Scalability would be implemented by instantiating multiple copies of the system in the cloud and then using a technique called A/B Testing.

A/B Testing is used to evaluate iterative optimization changes in websites:

> Using A/B, new ideas can be essentially focus-group tested in real time: Without being told, a fraction of users are diverted to a slightly different version of a given web page and their behavior compared against the mass of users on the standard site. If the new version proves superior—gaining more clicks, longer visits, more purchases—it will displace the original; if the new version is inferior, it's quietly phased out without most users ever seeing it. A/B allows seemingly subjective questions of design—color, layout, image selection, text—to become incontrovertible matters of data-driven social science (Christian 2012).

Incarnational Scalability would function like A/B Testing, but for whole systems. Multiple incarnations of identical test systems with playback of synthetic or recorded system load can be iteratively evaluated to determine the better version between:

> two versions of an element (A and B) and a metric that defines success (Chopra 2010).

An element might be a new index, statistics change, new release of code, or some other change. By evaluating changes in individual elements instead of as a massive set of changes, it should be possible to find the optimal elements for the success metrics.
Therefore, Incarnational Scalability can be used to solve several important testing challenges including:

1. Availability – test systems can be allocated and de-allocated dynamically.
2. Cost – test systems are only paid for when in use.
3. Granularity – testing can now iteratively test multiple potential configurations to find the optimal ones instead of testing a cluster of interrelated changes.
4. Increased thoroughness – since multiple tests for different elements can be evaluated simultaneously, the testing doesn't need to be prioritized for only the most critical changes.
5. Hardware rightsizing – ability to evaluate the benefits of adding more CPUs, memory, etc.

The experimentation made possible by Incarnational Scalability can support a more scientific approach to system optimization and thereby improves scalability.

4 CONCLUSIONS

The goal for cloud providers is a multi-tenant model that can provide services to a multitude of consumers (Rimal, Eunmi et al. 2009). Analysis of different cloud providers suggests that cloud computing will not be held back by definitions, but will continue to expand, as it has, in the past, as new services are invented and deployed. Nevertheless, where we are able to refine definitions, we can improve the precision of our science.

Some benefits of deploying transactional processing relational databases in a cloud model include the ability to perform vertical scaling (adding resources to a single node), which seems to be different from the widely accepted characteristic of Rapid Elasticity (Mell and Grance 2011). The use of incarnational scalability to support improved testing efforts (and thereby improved system efficiency) also seems to be a powerful alterative method to achieving what has been, until now, the goal of Rapid Elasticity – that is, scalability. Therefore, the essential quality of cloud computing should be broadened from mere Rapid Elasticity to any process that uses cloud-based resources and methods to improve scalability and the efficiency of the system.

REFERENCES

"Planet Cassandra FAQ." Retrieved January 13, 2014, from http://planetcassandra.org/Learn/FAQ.

Armbrust, M., et al. (2010). "A view of cloud computing." Commun. ACM **53**(4): 50-58.

Buyya, R., et al. (2009). "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility." Future Generation Computer Systems **25**(6): 599-616.

Chopra, P. (2010). The Ulitmate Guide to A/B Testing. Smashing Magazine. Online.

Christian, B. (2012). The A/B Test: Inside the Technology That's Changing the Rules of Business. Wired, Condé Nast.

Curino, C., et al. (2011). Relational Cloud: A Database-as-a-Service for the Cloud.". 5th Biennial Conference on Innovative Data.

Systems Research, Asilomar, California.

Das, S. (2011). Scalable and Elastic Transactional Data Stores for Cloud Computing Platforms. Computer Science, UCSB**:** 278.

Dutta, S., et al. (2012). SmartScale: Automatic Application Scaling in Enterprise Clouds. Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on.

Gray, J. (1981). The transaction concept: Virtues and

limitations. VLDB.
Herbst, N. R., et al. (2013). Elasticity in Cloud Computing: What It Is, and What It Is Not. ICAC 13. San Jose, CA.
Mell, P. and T. Grance (2011) The NIST Definition of Cloud Computing. The NIST Definition of Cloud Computing 3.
Millsap, C. and J. Holt (2003). Optimizing Oracle Performance, O'Reilly Media.
Pitt, L. F., et al. (1995). "Service quality: a measure of information systems effectiveness." MIS quarterly: 173-187.
Rimal, B. P., et al. (2009). A Taxonomy and Survey of Cloud Computing Systems. INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on.
Weinstock, C. B. and J. B. Goodenough (2006). On system scalability, DTIC Document.

Beyond Grid Portals and Towards SaaS
A New Access Model for Computational Grids, in the dMRI Brain Context

Tarik Zakaria Benmerar, Mouloud Kachouane, Fatima Oulebsir-Boumghar
ParIMed Team, LRPE, USTHB, BP32, El Alia, Bab-Ezzouar, Algiers, Algeria
tarik.benmerar@acigna.com, mouloud.kachouane@gmail.com, fboumghar@usthb.dz

Keywords: Grid Computing, Grid Portals, Cloud Computing, SaaS, OAuth, MyProxy, Grid-Cloud Interoperability, dMRI Brain, Web Worker, Canvas2D, WebGL, MarionetteJS, BackboneJS.

Abstract: Acigna-G is an ongoing research project to develop a new hybrid Grid SaaS architecture. CloudMRI is our proof-of-concept Acigna-G based SaaS Service for the Brain dMRI field. The main objective of such architecture is to provide local (Browser) and remote (Grid) intensive computational capabilities completly abstracted to the SaaS user. The result is a combination of an in Browser Rendering and Computation engine, interoperable REST-SOAP Grid Services, and interoperable web-grid authentication mechanisms. Such architecture can allow new types of SaaS Services, specifically for the dMRI Brain field.

1 INTRODUCTION

At the higher level of the Cloud Computing architecture, SaaS Services provide ubiquitous access to hosted software stack. These softwares provide an intuitive and an interactive access to user's Cloud Data.

From the computational standpoint, we have observed that the current SaaS Services do not provide the intensive computational capabilities, while remaining highly interactive and intuitive, with rich 2D/3D rendering.

Also, the grid computational capabilities have not been used in conjunction with SaaS Cloud Services, to abstract the underlying job submission mechanisms.

The current grid portals use somewhat intuitive web interfaces, but still follow the job submission paradigm, where the user is provided with job submission forms and basic visualization tools.

Through the ongoing Acigna-G project and platform, we have investigated a hybrid SaaS Cloud Grid architecture. The SaaS Cloud Service provides a rich interactive and intuitive web interface, with in browser 2D/3D rendering and basic computations. The intensive computational part of the service are delegated to the grid infrastructure, while the user is nearly unaware of this fact.

This architecture has been developed under the umbrella of the dMRI Brain field. CloudMRI is the resulting SaaS Service built with this architecture.

The present paper explains the importance of such architecture. Its remaining parts are structured as follows. In Section 2, we introduce the dMRI Brain Tractography and related softwares. In Section 3, we explain the importance of a Hybrid SaaS Grid Architecture. In Section 4, we present the Acigna-G Platform and some parts of the its built upon CloudMRI SaaS Service. In Section 5, we present a comparative study. In Section 6, we conclude the paper.

2 dMRI BRAIN TRACTOGRAPHY

With the Diffusion-weighted imaging (DWI), we can investigate a tissue microstructure in vivo and noninvasively. By combining MRI diffusion with imaging DWI became possible, enabling the mapping of both diffusion constants and diffusion anisotropy inside the brain and revealing valuable information about axonal architectures (Le Bihan and Breton, 1985).

Later, describing the degree of anisotropy and the structural orientation information quantitatively became possible with the introduction of Diffusion Tensor model (Basser et al., 1994). With only six parameters, this Diffusion Tensor Imaging (DTI) was able to provide a simple and elegant way to model this complex neuroanatomical information.

Different regions of the brain are connected through axons, that form the brain white matter. White matter tracts represent a set of axons that share a similar destination. With tractography or fiber-tracking algorithms, major tracts can be clearly ob-

served by DTI with 2-3 mm image (Tournier et al., 2011) (See Figure 1).

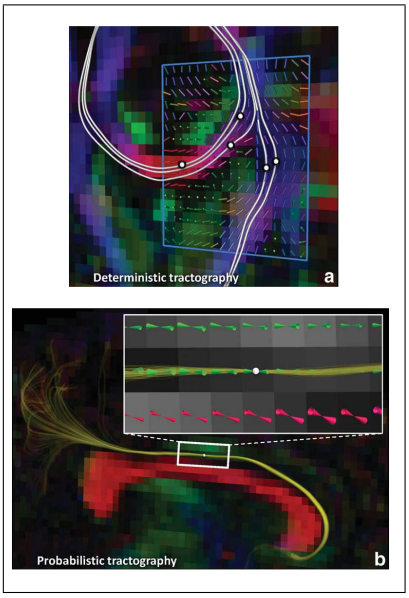


Figure 1: Deterministic VS Probabilistic tractography (Tournier et al., 2011).

Differing in quality and performance, there are mainly two types of tractography:

- **Deterministic Tractography:** This tractography supposes that from a starting point or seed, there exists exactly one tract that goes out from it (See a in Figure 1). This algorithm is fairly fast (a few minutes to less than a hour), but produces limited results.
- **Probablistic Tractography:** This tractography supposes that from a starting point or seed, there can be many tracts that go out from it (See b in Figure 1). This algorithm is heavy in processing (several hours), but produces better results.

From the dMRI Brain Software standpoint, MedInria (MedInria, 2014) provides an intuitive and an interactive desktop application. Neverthless, it is fairly limited as it provides only deterministic tractography, due to the limited ressources of a PC.

On the other hand, FSL (FSL, 2014) provides access to advanced probablistic tractography. But, access to such algorithms are through a non intuitive command line, as it is required for a grid deployment.

3 HYBRID SaaS GRID ARCHITECTURE

From the dMRI Brain Tractography algorithms and related softwares introduced in the previous section, we argue that the existance of a hybrid SaaS Grid architecture is very important for the dMRI Brain field.

In fact, the current SaaS Services such as Google Docs (Google, 2014), provide a visualization and a manipulation tool to act on your Cloud data. These services provide the same features as their desktop counterparts, with the additionnal ubiquitous access to Cloud Data hosted on the Cloud Provider infrastructure.

Neverthless, we have seen that the computational axis of the software is nearly absent in the current services. Hence, the Cloud Providers provide only data hosting and manipulation, and no computational capabilities are provided through the software abstraction level.

In the Grid context, Grid Portals can provide all the computational capabilities required. With a web interface, the user can submit jobs and visualize the end result inside a web browser.

Through the recent years, Grid Portals have tried to cope with the evolution of the web, specifically the web 2.0 trend (Pierce et al., 2006). RSS feeds, social networks, wikis and visualization through interactive maps, have been incorporated.

Neverthless, advanced interaction and 2D/3D rendering that we find in SaaS services haven't been integrated. Also, the job submission paradigm is still the only paradigm followed, in the sense that a Grid Portal is all about job submission and result visualization. No advanced software abstraction exists, where the user is totally unaware about any job submission.

Through our Acigna-G initiative (Benmerar and Oulebsir-Boumghar, 2011) (Benmerar and Oulebsir-Boumghar, 2012), we have tried to provide such a hybrid architecture after issues faced with a previous grid plaform (Oukfif and Oulebsir-Boumghar, 2009). Through iterative improvements, the architecture has evolved from a platform for Linux shell-like interactive web access to computational tasks, to the actual SaaS service completly abstracted from the underlying grid infrastructure.

CloudMRI is our proof of concept, where we use the Acigna-G platform to deliver a dMRI Brain SaaS. It allows to provide the same interactive experience as MedInria, with the additional access to advanced processings such as probabilistic tractography, implemented in FSL.

4 THE ARCHITECTURE OF THE ACIGNA-G PLATFORM AND THE CLOUDMRI SaaS SERVICE

In this section, we present the Acigna-G architecture and its built upon CloudMRI SaaS Service.

The architecture is composed into four parts: In Browser Computing and Rendering, Restful Grid Services, Grid-Cloud Interoperable Authentication and Computing Grid Services. We introduce each part in the following subsections:

4.1 In Browser Computing and Rendering

The CloudMRI web frontend has been built as a Single Page Application (or SPA). Rather than decomposing the application into multiple pages, it is hosted under one url that dynamically fetches the required ressources using AJAX, then emulates passing from one url to another. We usually find this technology in modern web mail clients, we use daily. MarionetteJS (Bailey, 2014) and BackboneJS (DocumentCloud, 2014) have been used as the underlying technology for this purpose. It was required to build the software in this way, as we wanted a high interactivity at the frontend.

The frontend uses the Acigna-G MyThreadJS Javascript library, to manage computations, rendering and data sources. These different axis of the library are explained in the following paragraphs:

Computational Management

The Compute Manager Component provides a central entry point to local and remote job execution. The remote jobs are delegated to the Restful Grid Service presented in 4.2.

Locally, the Compute Manager manages threads that execute either in the local browser thread, or in a separated thread using Web Workers standard (Mozilla, 2014b).

The local threads are managed in a way that avoid blocking user browser interaction. They are built with interruption points specified by the thread developer, that the compute manager uses to schedule threads execution.

Currently, our CloudMRI frontend uses particularly the Computer Manager for parsing NIfti (NIMH, 2014) file format, through a threaded class called ParserNII. This processing can be heavyweight, and the component helped avoid any threading issues.

Rendering Management

The Render Manager Component constitutes a central point for efficient scheduled rendering. Rather than immediatly rendering after a user interaction, it schedules it at a specific time interval to avoid costly reprocessing which can be the case of 2D/3D rendering using Canvas2D (Mozilla, 2014a) or WebGL (Khronos, 2014) standards respectively.

Currently, the CloudMRI fronted uses the Render Manager to render MRI Slices with a component called MRISliceWidget, illustrated in Figure 2.

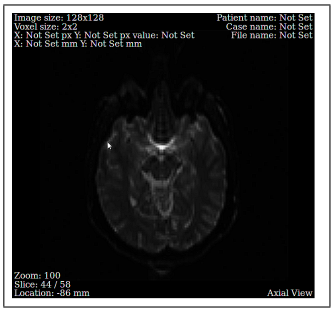


Figure 2: MRI Slice Widget.

Data Source Management

The Source Manager component in myThreadJS library provides a central entry point to local and remote data fetching, caching, persisting and locking to the rest of the application. Two types of data are managed by this component: local data and remote data.

Local data are data hold on the user file system, that stay persistent across sessions. Remote data can be considered as cloud data. The Indexed Database API (W3C, 2014) is used for persisting local data, and for caching remote data.

The data sources are referenced across the application with a point separated string (e.g: 'patient.mri'). The point separation is used to differentiate data namespaces, or for building a hierarchy of data sources processings (e.g 'patient.mri.parsed' has been generated from the 'patient.mri' processing). Such hierarchy is used to avoid later reprocessing of data.

To avoid that two threads process the same data, locking data sources mechanisms exist. For example, if the first thread is processing 'patient.mri' to generate 'patient.mri.parsed', the later is marked as locked. The second thread waits for the availablity of the later data source, after the first thread processed it and unlocked it.

The CloudMRI frontend uses the Source Manager for the user's Brain dMRI Images persistance.

4.2 Restful Grid Service

To submit a job from the SaaS Front-end, we have designed two types of Restful services. They are intermediaries between the SaaS Front-end and the Computing Grid Services. Their role is to avoid the construction of a heavy weight XML SOAP envelope at the browser side, by using JSON data format.

Restful BES Service

Restful BES Service is a REST equivalent to the OGSA-BES (OGF, 2008) standard. This service is based on the proposed solution in (Andreozzi and Marzolla, 2009), but uses JSON for the JSDL document (OGF, 2005). The JSON format construction follows particular rules, that are illustrated in Figure 3 for sample examples.



Figure 3: Conversion examples between JSDL JSON and JSDL XML, for sample fragments.

Task Specification Restful Service

The Task Specification Restful Service avoids the construction of a whole JSDL document, by just requiring a task name and parameters. The Restfull Service retrieves the corresponding JSDL document and inserts the specified parameters. This service can be important when the SaaS service has a predefined set of jobs that can be submitted.

4.3 Grid-Cloud Interoperable Authentication

In the proposed architecture, we have started from the hypothesis that the Cloud Provider is an independant third-party, separated from the Computing Grid. The SaaS Provider should authenticate itself to the grid, before job submission. We have then supposed two different scenarios: Grid User and Non-Grid User, explained in the following paragraphs.

Grid User

In such scenario, the SaaS Service provides computational capabilities of the grid infrastructure the user has access to. The Service Provider should then have access to the user credentials to authenticate itself. These credentials are usually stored in a MyProxy Credential Management Service and protected with a login and a password.

Providing such information (login and password) to the Service Provider is not secure, as it could lead to user's credentials abuse by the provider. A solution proposed in (Basney et al., 2011) for grid portals, avoids enveiling the login and password. The architecture uses an OAuth service in front of the MyProxy service, as depicted in Figure 4.

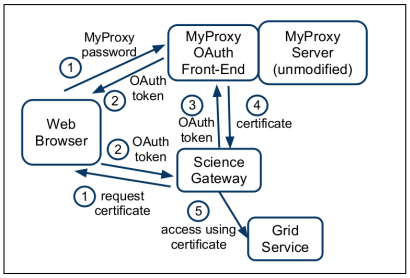


Figure 4: Science Gateways Using OAuth with MyProxy (Basney et al., 2011).

Using OAuth tokens, the user provides access to his MyProxy account, without enveiling his login and password. At a later time, these tokens can be revoked and the third-party will no longer have access to the MyProxy account.

Without change, such solution can be used in our architecture as a means to provide the required computational capabilities to grid users.

Non-Grid User

In the case of a non-grid user, other mechanisms are required for the Service Provider to be allowed to submit jobs to the underlying grid infrastructure.

Dynamic Accounts mechanism (Insley et al., 2009) used in the grid portal context, can be integrated in our architecture. With Dynamic Accounts, the computing grid can allocate dynamically a pool of user accounts to the grid portal. These accounts can be mapped to the real users at runtime.

In the SaaS context, the Service Provider could buy a pool of dynamic accounts from the computing grid. It could then use them to provide the required computational capabilities to the SaaS users.

4.4 Computing Grid Services

At the Grid level of our architecture, a set of Python-based grid services are deployed. These services have been developped using the ZSI (ZSI, 2013) Python library.

The ZSI library is a python library for web services deployment and use. We have successfully deployed a basic WS-BES Grid Service that accepts a JSDL template for the job submitted.

But different extensions such as JSDL SPMD (Savva, 2005) are required for parallel tasks deployment. We are currently working on such extensions, to provide a more complete capabilities at the grid service level.

We have privilegied the use of Python, as we have found it easier for deployment than for example Java based services. There is for example no need for precompilation before deployment. Furthermore, Python-based webservices exhibit comparable performances to Java-based ones (Santos and Koblitz, 2005).

Neverthless, the deployment required a bit of patching on the source code, and we are planning to support the open source project through source code contributions.

HTCondor (HTCondor, 2014) is used as the job submission system at the grid site level. Though other job submissions systems can be used, an adaptor plugin should be developped for each one, at this moment.

5 COMPARATIVE STUDY

NEWT (Cholia et al., 2010) is a RESTFul Service provided by NERSC for building High Performance Computing Web Applications. It provides an easy API for authentication, persistance management and job submission. It uses the lightweight JSON data format for data-exchange between the client application and service.

Neverthless, the service is tightly-coupled to the underlying NERSC infrastracture. When submitting a job, a specific machine queue should be specified. In some cases, the user should even specify the required batch system script (e.g for PBS).

Also, the application examples using this platform, still follows the job submission paradigm we have presented. They don't provide an interactive software environment, abstracted from the infrastructure detail.

The BES-REST (Andreozzi and Marzolla, 2009) exhibits the required service abstraction by providing a Restful interface equivalent to an established Grid Job Submission Standard (WS-BES).

This work has been the basis of our Restful BES Service, with the difference that we use the lightweight JSON data-format rather than XML for the JSDL job specification. This was required for an efficient job submission from the browser.

Cantor et al. (Cantor-Rivera and Peters, 2010) have proposed a web application architecture for remote MRI rendering, using WebGL. The architecture doesn't cover any local or remote computional capabilities.

XTK (XTK, 2014) is a WebGL toolkit, for Scientific Visualization. With XTK, medical images formats can be rendered in the browser, such as NIfTI format.

Compared to our work, XTK neglects threading issues that we have solved with the MyThreadJS library. This constitutes a problem in a heavy software environment.

In (Rings and Grabowski, 2012), the authors investigate the integration of Cloud and Grid infrastructures. The architecture tries to integrate computational ressources situated on an IaaS Cloud or a Computational Grid. It differs significatly from our work, as we have looked into SaaS (rather than IaaS) Clouds and Grids integration.

In (Church et al., 2012), the authors investigate the deployment of an HPC environment in a SaaS Cloud. All the computational ressources are provided by an underlying IaaS provider. The SaaS Service provides a web interface to submit computational jobs in the Cloud Infrastructure. Unlike our architecture, the service follows the job submission paradigm and don't manage local computations.

6 CONCLUSION

Current SaaS Services can evolve to provide computational capabilities, beyond the data capabilities actually provided. On the other hand, Grid Portals can evolve to provide an interactive software environment completly abstracted from the enderlying grid infrastructure, that would be easier and intuitive to use.

The proposed architecture tries to fill in these observed gaps. Particulary, we have emphasied the need of such architecture in the Brain dMRI Field.

Currently, much of the functionalities of the myThreadJS library have been developed and tested. The results have been very satisfying with respect to performance and interface responsiveness, and a indepth comparaison with the XTK library is under preparation. We have also developed and tested the

submission of a grid job into HTCondor through a basic Restful BES and WS-BES Grid Service.

Other parts of Acigna-G are still under development. We have also began the development of the CloudMRI SaaS Service with the first basic user interfaces being setup.

In subsequent papers, we will provide detailed reviews of the architecture and its performances. This is necessary as we are heading away from a developping platform to a testing, and then a production one. This will be characterised by rich feedbacks from the concerned users, particulary our research colleagues.

REFERENCES

Andreozzi, S. and Marzolla, M. (2009). A restful approach to the ogsa basic execution service specification. In *Proceedings of the fourth International Conference on Internet and Web Applications and Services ICIW '09*.

Bailey, D. (2014). Marionettejs website. Retrieved January 13, 2014. http://www.marionettejs.com.

Basney, J., Dooley, R., Gaynor, J., Marru, S., and Pierce, M. (2011). Distributed web security for science gateways. In *Proceedings of the 2011 ACM workshop on Gateway computing environments GCE '11*.

Basser, P., Mattiello, J., and Le Bihan, D. (1994). Mri diffusion tensor spectroscopy and imaging. *Biophysical Journal*.

Benmerar, T. Z. and Oulebsir-Boumghar, F. (2011). Toward a cloud architecture for medical grid applications : The acigna-g project. In *Proceedings of the 10th International Symposium on Programming and Languages ISPS '2011*.

Benmerar, T. Z. and Oulebsir-Boumghar, F. (2012). From grids to cloud : The pathway for brain dmri cloud services. In *Proceedings of the 2nd International Conference on Cloud Computing and Services Science CLOSER '2012*.

Cantor-Rivera, D. and Peters, T. (2010). Pervasive medical imaging applications : current challenges and possible alternatives. Retrieved January 13, 2014. http://imaging.robarts.ca/ dcantor/wp-content/uploads/2010/07/eHealth.pdf.

Cholia, S., Skinner, D., and Boverhof, J. (2010). Newt: A restful service for building high performance computing web applications. In *Proceedings of the Gateway Computing Environments Workshop (GCE)*.

Church, P., Wong, A., Brock, M., and Goscinski, A. (2012). Toward exposing and accessing hpc applications in a saas cloud. In *Proceedings of the 2012 IEEE Nineteenth International Conference on Web Services ICWS '2012*.

DocumentCloud (2014). Backbonejs website. Retrieved January 13, 2014. http://www.backbonejs.org.

FSL (2014). Fsl website. Retrieved January 12, 2014. http://www.fmrib.ox.ac.uk/fsl/.

Google (2014). Google docs. Retrieved January 10, 2014. http://docs.google.com.

HTCondor (2014). Htcondor website. Retrieved January 13, 2014. http://research.cs.wisc.edu/htcondor/.

Insley, J. A., Leggett, T., and Papka, M. E. (2009). Using dynamic accounts to enable access to advanced resources through science gateways. In *Proceedings of the 5th Grid Computing Environments Workshop GCE '09*.

Khronos (2014). Webgl website. Retrieved January 13, 2014. http://www.khronos.org/webgl/.

Le Bihan, D. and Breton, E. (1985). Imagerie de diffusion in-vivo par résonance magnétique nucléaire. *Comptes-Rendus de l'Académie des Sciences*.

MedInria (2014). Mediniria website. Retrieved January 12, 2014. http://med.iniria.fr.

Mozilla (2014a). Canvas. Retrieved January 13, 2014. http://developer.mozilla.org/fr/docs/Web/HTML/.

Mozilla (2014b). Using web workers. Retrieved January 22, 2014. http://developers.mozilla.org/en-US/docs/DOM/Using_web_workers.

NIMH (2014). Nifti format. Retrieved January 13, 2014. http://nifti.nimh.nih.gov/nifti-1/.

OGF (2005). Job submission description language (jsdl) specification. Retrieved January 10, 2014. http://www.ogf.org/documents/GFD.56.pdf.

OGF (2008). Ogsa basic execution service (ogsa-bes) version 1.0 specification. Retrieved January 10, 2014. http://www.ogf.org/documents/GFD.108.pdf.

Oukfif, K. and Oulebsir-Boumghar, F. (2009). Prétraitement parallèle de qsplat sous grille de calcul. In *Journées Scientifiques sur l'Informatique et ses Applications*.

Pierce, M. E., Fox, G., Yuan, H., and Deng, Y. (2006). *Cyberinfrastructure and Web 2.0.*, volume High Performance Computing and Grids in Action. IOS Press.

Rings, T. and Grabowski, J. (2012). Pragmatic integration of cloud and grid computing infrastructures. In *Proceedings of the 2012 IEEE fifth International Conference on Cloud Computing CLOUD '2012*.

Santos, N. and Koblitz, B. (2005). Metadata services on the grid. *Nuclear Instruments and Methods in Physics Research*.

Savva, A. (2005). Jsdl spmd application extension, version 1.0 (draft 008). Retrieved January 10, 2014. http://www.ogf.org/Public_Comment_Docs/Documents/May-2007/draft-ogf-jsdl-spmd-008.pdf.

Tournier, J.-D., Mori, S., and Leemans, A. (2011). Diffusion tensor imaging and beyond. *Magnetic Resonance in Medicine*, 65:1532–1556.

W3C (2014). Indexed database api. Retrieved January 13, 2014. http://www.w3.org/TR/IndexedDB/.

XTK (2014). Xtk website. Retrieved January 13, 2014. http://www.goxtk.com.

ZSI (2013). Zsi website. Retrieved January 10, 2014. http://pywebsvcs.sourceforge.net/zsi.html.

SERVICES SCIENCE FOUNDATION FOR CLOUD COMPUTING

FULL PAPERS

Cloud Asset Pricing Tree (CAPT)
Elastic Economic Model for Cloud Service Providers

Soheil Qanbari[1], Fei Li[1], Schahram Dustdar[1] and Tian-Shyr Dai[2]
[1]*Distributed Systems Group,Technical University of Vienna, Vienna, Austria*
[2]*Institute of Finance, National Chiao-Tung University, Hsinchu, Taiwan*
{*qanbari, li, dustdar*}*@dsg.tuwien.ac.at, d88006@csie.ntu.edu.tw*

Keywords: Cloud Price Elasticity, Asset Pricing, Financial Options, Cloud Federation, Cloud Computing.

Abstract: Cloud providers are incorporating novel techniques to cope with prospective aspects of trading like resource allocation over future demands and its pricing elasticity that was not foreseen before. To leverage the pricing elasticity of upcoming demand and supply, we employ financial option theory (future contracts) as a mechanism to alleviate the risk in resource allocation over future demands. This study introduces a novel Cloud Asset Pricing Tree (CAPT) model that finds the optimal premium price of the Cloud federation options efficiently. Providers will benefit by this model to make decisions when to buy options in advance and when to exercise them to achieve more economies of scale. The CAPT model adapts its structure to address the price elasticity concerns and makes the demand, price inelastic and the supply, price elastic. Our empirical evidences suggest that using the CAPT model, exploits the Cloud market potential as an opportunity for more resource utilization and future capacity planning.

1 INTRODUCTION

Cloud providers offer APIs associated with their pool of configurable computing resources (e.g., virtual machines) so that clients can access and utilize them by deploying their packages in runtime environments (Kurze et al., 2011). In a Cloud market, the right to benefit from these pools of Cloud resources with their utilization interfaces, can be delivered as "*On-Demand*" or "*Reserved*" instances. For clients, the reserved instances (RIs) are more reliable and economic assets. As a proof, the unit of a resource being studied here is an Amazon EC2 Standard Small Instance (US East) at a price[1] of $\$0.060/hour$ for an *on-demand* instance and for a *reserved* instance, costs $\$0.034/hour$ with an upfront payment of $\$61/year$, which is almost half price. Therefore, financially, clients are more attracted to RIs. Faced with such dilemma, RIs pose the concern of less future utilization as far as it is not used by either the current or other on-demand clients. This motivates providers to take the opportunity to achieve more resource utilization by keeping all instances in use. Providers may reallocate unused RIs of current owner to other on-demand clients to keep all resources utilized. This approach makes the RIs unavailable for the current owner. Obviously, it is an obligation for providers to assure the availability of RIs associated with owners, otherwise, lack of resources leads to unmet demands and, while reflecting the SLA violations, leads to financial consequences and penalties. To assure asset availability when lacking resources, providers can seek for more affordable and cost-efficient Cloud open markets to outsource their clients demands. In addition to the fact that Cloud open marketplaces (e.g., Zimory, SpotCloud) and federation offerings (e.g., CloudKick, ScaleUp) offer more resource utilization mechanisms, they also enable further cost reduction due to the market competitive advantage among providers.

The decision to outsource the request to the federation parties is relatively dependent to the asset's price. In a similar model, the Amazon Web Services (AWS) also offer a spot instance pricing model, where the price fluctuates as the market supply and demand changes, and the spot instances will be provisioned to the bidders who won the competition. As soon as the asset's spot price goes above the winning bid, resources will be released. In open Cloud markets, the providers hardly can rely on such mechanism since there is no guarantee as they might lose the resources when the asset price crosses their bid. In order to en-

[1]http://aws.amazon.com/ec2/pricing/

courage providers to benefit from the Cloud market, we need a dynamic economic model that keeps resource and financial elasticity sustainably balanced by controlling the asset price oscillation while demand and supply fluctuate. To this end, our contribution is twofold: (i) Analyzing the financial options and pricing elasticity concepts in Cloud federation market. (ii) The flexible pricing model that calculates the optimal premium price of the federation options efficiently and accurately.

The paper continues with a motivation scenario in support of an elastic economic model for pricing Cloud federation assets at section 2. Section 3 presents the basic concepts and preliminaries where the conceptual basis and mathematical models are detailed. Based on this, CAPT pricing model is derived in section 4. We simulate and evaluate our CAPT model and numerical results will be given in section 5 to support the efficiency of our model. Subsequently, section 6 surveys related works. Finally, Section 7 concludes the paper and presents an outlook on future research directions.

2 MOTIVATION

Along with elastic resource provisioning, providers may face the limitations and insufficiency of their own resource pool supply. In effect, they can transfer the risk of lacking resources to the federation markets. Federation markets can be of interest for providers as well as for consumers. Clients may profit from lower costs and better performance, while providers may offer more sophisticated services (Kurze et al., 2011). However, hereinafter we focus on the provider perspective. Thus providers can benefit from the increasing capacity and diversity of federated resources. In our model, we employ financial option theory as an interface to elastically allocate an extra pool of federated resources. In finance, an option[2] is a contract which gives the buyer (the owner) the right, but not the obligation, to buy or sell an underlying asset or instrument at a specified strike price on or before a specified date.

Pricing elasticity and resource trading among federation members lead to competitive contracting process, which aims at finding reasonable and fair price of the asset. The contracting process is to write an option that contains future aspects of trading. For instance, whenever the provider lacks the required resources, then can take advantage of exercising such options to allocate corresponding resources respectively. Using options, providers take the rights to provision seller's resources which match their demands among parties at a price equal or above to their expectation of the asset payoff. Now, the concern is, how to price an option to be reasonable for both parties? Obviously, option pricing is an elastic process (Dustdar et al., 2011), sensitive to the fluctuation of the asset price determined by supply and demand between federation parties in spot market. As a consequence, pricing elasticity that comes in two types of *Demand* and *Provisioning* may drive a wedge between the buying and selling price of an asset. Thus controlling the pricing elasticity of the demand and provisioning with respect to their effects on revenue stream by fair pricing of such options appears to be vital. This paper aims at addressing the pricing elasticity of the asset in federation market by fair pricing of the *option*. The option price is determined by a broker acting on behalf of the Cloud federation and therefore standardised across the federation. This option gives the right to obtain an instance at a given price, established at the agreement's stipulation time.

In this scenario, at stage 1 as shown in Fig. 1, the clients request for on-demand and RIs and keep using them. At stage 2 another client benefits from the existing RI. As soon as the RI is suspended, *Provider A* can utilize this instance by reallocating it to unmet on-demand request. Therefore, upon lacking resources, any incoming on-demand request at stage 3 will be responded by reallocation of the RI at 4 to a new client. At this moment, stage 5, *Provider A* buys an option from federation broker as a supporting mechanism for future resource capacity planning. The provider avoids buying resources at a price that is higher than the one charged to its own customers. As soon as the previous client claims for the RI at 6 which is now allocated to the request 3, the provider will take advantage of the option signed with *Provider B* by exercising it at 7 and the *Provider B* has an obligation to provision the promised resources at 8. Our focus lies on stages 5 and 7 where the provider is looking for a well priced option to be exercised later to achieve more utilization. In our federation model, *Provider A* is the demander and *Providers B & C* are the resource suppliers in the federated environment.

The fact that future valuation of federated assets depends on the correlated elasticity between provisioning and demand, suggests that the optimal utilization of an asset is primarily driven by its price volatility in open Cloud markets. This influences the trend of providers to be more concentrated on controlling this pricing elasticity. Although the elasticity of a demand is an initial impetus in asset valuation, the pricing elasticity of the demand might lead to ineffi-

[2]http://en.wikipedia.org/wiki/Option_(finance)

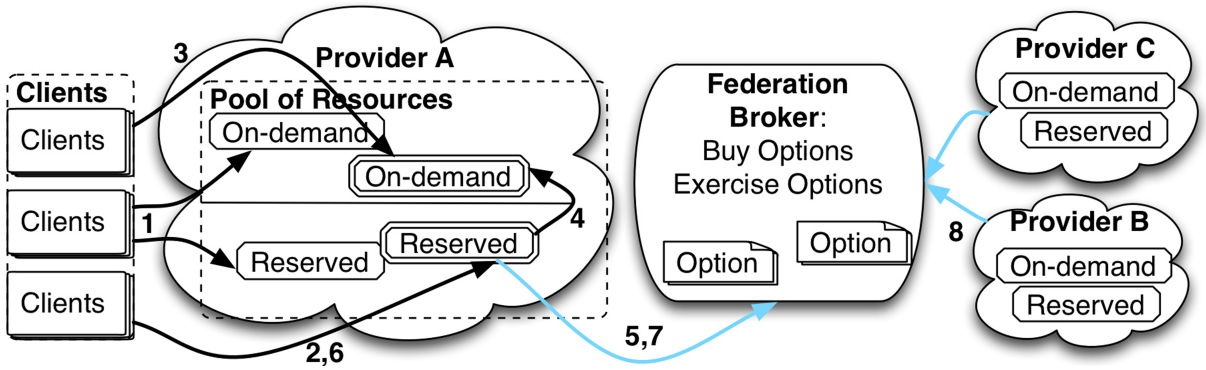


Figure 1: Resource utilization in Cloud federation using options.

cient revenue generation. For instance, the resource demand can be affected to a greater degree by minor changes in asset price. This leads to a question, how can volatility in price cause so much sensitivity in future demands? The reason is amplification. Blame is usually laid on asset price fluctuation and dynamic valuation. The price changes in federation market will be propagated across providers (such as domino effects), causing more sensitivity and concerns on provider's demand. The next question is, how can we control the pricing elasticity and decrease Cloud market sensitivity to future asset price changes? In this study we employ financial option theory which takes care of future valuation of the asset. Then by using the Binomial-Trinomial Tree (BTT) option pricing (shyr Dai and dauh Lyuu, 2010) methodology, we control the Cloud asset price changes and its propagation through the market. As the option price rises or falls, our CAPT model will adjust its structure to the price volatility to come up with an option price that is predictable and fair for both option holder and writer.

3 TERMS AND PRELIMINARIES

In this section we present basic concepts, economic terms and numerical methods and their interpretations considered in the study.

3.1 Cloud Federation Contracts

In finance, an option is a contract but the major difference arise from the rights and obligations of an option's buyer and seller. A *Call* option gives the buyer the right, but not the obligation, to purchase the underlying asset at a specified price (the strike or exercise price) during the life of the option. The cost of obtaining this right is known as the option's "*premium*" which is the price that is offered in the exchange. We use the term *premium* for an *option premium* in this study. The option buyer's loss is limited to the premium paid. When you own a *Call*, what you do by exercising your right is to *Call for Resource Provisioning* from provider that offered the *Call* to you. The buyer's right becomes the seller's obligation when the option is exercised. An American option can be exercised at any time during the life of the contract while European option can only be exercised at maturity date. The CAPT is modeled with American call options.

American options are provided by a pool of providers and purchased by other providers as a hedge to cover potential excess demand. Using this method, providers are able to re-sell on-demand VMs that have previously been sold as RI. If the RI owner decides to use the instance then rather than violate an SLA, the excess demand can be covered by exercising previously purchased options to enable Cloud-bursting using the federated pool of resources. The option pricing model determines an option price that is inelastic (such that supply and demand are not highly sensitive to price), thereby reducing self-reinforcing oscillations in supply and demand. The paper demonstrates that the option pricing model converges to a more stable price over time and the simulated provider increases profit from outsourcing provisioning using options.

3.2 Cloud Asset Pricing Elasticity

In Cloud systems, elasticity is the ability to automatically increase or decrease resource allocation to asset instances as demand fluctuates. Cloud financial elasticity is a measure of how much resource buyers and sellers respond to changes in market conditions. It's a measure of the responsiveness of quantity demanded or provisioned to a change in one of its determinants like price or quality. In this paper we address the Cloud federation asset pricing elasticity. The law of demand states that a fall in the price of a resource

raises the quantity demanded. To be more specific, the price elasticity of demand measures how willing providers are to buy less or more options as its price rises or falls. To sum up, the concept of *Price Elasticity of Demand* (PEoD) measures of how much the resource quantity demanded due to a price change. And the *Price Elasticity of Provisioning* (PEoP) measures how much the resource quantity provisioned due to a price change (Mankiw, 2012). The PEoD and PEoP formulas are:

$$PEoD = \frac{(\%Change\ in\ Quantity\ Demanded)}{(\%Change\ in\ Price)} \quad (1)$$

$$PEoP = \frac{(\%Change\ in\ Quantity\ Provisioned)}{(\%Change\ in\ Price)} \quad (2)$$

3.3 Pricing Elasticity Interpretation

Regarding interpretation, we analyze the Cloud asset price elasticity only with their absolute values. The PEoD variable values, denote how sensitive the demand for an asset is to a price change. In financial markets, the rule is if a provider's asset has a high elasticity of demand, the more the price goes up, the fewer consumers will buy and try to economise their needs. Correspondingly, in Cloud federation markets, a very high price elasticity suggests that when the price of a resource goes up, our provider will be more sensitive and demand for less assets or buy less call options. Conversely, when the price of that resource goes down, then the provider will demand for more assets or buy more call options. A very low price elasticity implies just the opposite, that changes in price have little influence on demand or exercising the call. To sum up, when demand is price inelastic, total revenue moves in the direction of a price change. When demand is price unit elastic, total revenue does not change in response to a price change. When demand is price elastic, total revenue moves in the direction of a quantity change. In order to see whether the price is elastic or inelastic we use the following rule of thumb:

$$\text{VM PEoD} = \begin{cases} >1 & \text{Demand is price elastic.} \\ =1 & \text{Demand is unit elastic.} \\ <1 & \text{Demand is price inelastic.} \end{cases} \quad (3)$$

Next is price elasticity of provisioning in federation resource supply pool. The law of supply states that higher prices raise the quantity supplied. The price elasticity of supply measures how much the quantity supplied responds to changes in the price. Supply of a good is said to be elastic if the quantity supplied responds substantially to changes in the price. Supply is said to be inelastic if the quantity supplied responds only slightly to changes in the price. PEoP denotes how sensitive the provisioning of an asset is to a price change. In Cloud federation markets, a very high price elasticity of provisioning suggests that when the price of a resource goes up, Cloud federation members will be more sensitive to price changes and provision more assets or sell more call options to make more profit. Thus, the resource quantity supplied can respond substantially to price changes. Same as PEoD, in order to see whether the price is elastic or inelastic in PEoP, we use the following rule of thumb:

$$\text{VM PEoP} = \begin{cases} >1 & \text{Provisioning is price elastic.} \\ =1 & \text{Provisioning is unit elastic.} \\ <1 & \text{Provisioning is price inelastic.} \end{cases} \quad (4)$$

Finding the right balance between these two polar approaches of PEoD and PEoP to come to a new equilibrium is a challenge as we address it using our CAPT model. In equilibrium, asset aggregate demand has to equal the asset supply. To be more specific, in our evaluation, we will show that our pricing model, calculates the fair price of the option that makes the demand, price inelastic and provisioning, price elastic. This leads to increasing demand, regardless of the asset price oscillation.

3.4 Assumptions

It is an indication that the following assumptions underlying our model has been considered for the proper positioning of this study. In a Cloud market, resources are virtualized to abstract concepts like virtual machines (VMs) and assumed as intangible assets. They are also seen as assets as long as associated with a contract that can be exercised by an option. Federation formation pose some concerns like contract management, data policies, SLA violations and etc. We believe these concerns should be addressed in the business models agreed among parties.

4 CAPT MODEL

The option pricing can be represented by numerical methods like trees. This section shows how to generate the CAPT tree for pricing options. The model benefits from the Binomial and Trinomial tree methods as detailed below. This section shows how to generate the CAPT tree for pricing options.

4.1 Binomial Tree

Binomial tree model is a numerical pricing method that approximates option price. Let a derivative on

$S_{(t)}$ initiates at time 0 and matures at time T. A lattice partitions this time span into n equal-distanced time steps and specifies the value of $S_{(t)}$ at each time step which denotes the Cloud asset price. Let the length between two adjacent time steps be $\Delta t \equiv T/n$. The established Cox-Ross-Rubinstein (CRR) binomial tree (Cox et al., 1979) is shown in Fig. 2. As we move forward in time, each asset price S can either move upward to become S_u with probability P_u, or move downward to become S_d with probability $P_d \equiv 1 - P_u$. The CRR lattice adopts the following solution:

$$u = e^{\sigma\sqrt{\Delta t}}, d = e^{-\sigma\sqrt{\Delta t}}, P_u = \frac{e^{r\Delta t} - d}{u - d}, P_d = \frac{e^{r\Delta t} - u}{d - u} \quad (5)$$

Where σ is price volatility, Δt is duration of a step and r denotes the interest rate.

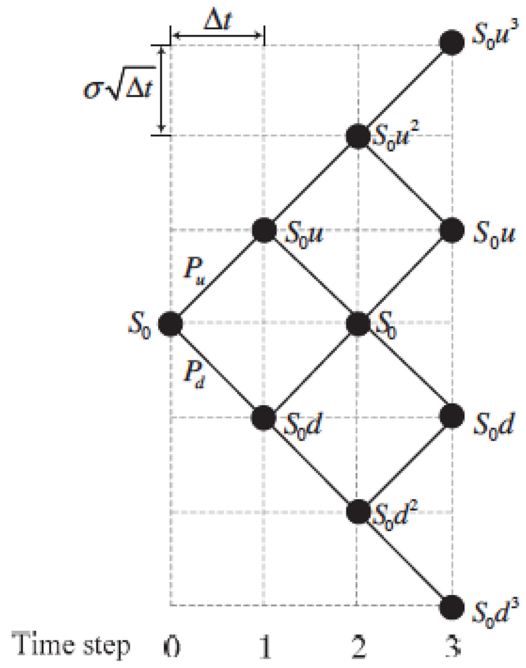


Figure 2: CRR Binomial Pricing Tree.

4.2 Trinomial Tree

A trinomial tree can be built in a similar way to the binomial tree but has three possible paths (up, down, and stable) per node leading to more efficient pricing. The jump sizes (u, d) can be calculated in a similar way with doubled time spacing. The transition probabilities are given as:

$$P_u = \left(\frac{e^{\frac{r\Delta t}{2}} - e^{-\sigma\sqrt{\frac{\Delta t}{2}}}}{e^{\sigma\sqrt{\frac{\Delta t}{2}}} - e^{-\sigma\sqrt{\frac{\Delta t}{2}}}} \right)^2 P_d = \left(\frac{e^{\sigma\sqrt{\frac{\Delta t}{2}}} - e^{\frac{r\Delta t}{2}}}{e^{\sigma\sqrt{\frac{\Delta t}{2}}} - e^{-\sigma\sqrt{\frac{\Delta t}{2}}}} \right)^2 \quad (6)$$

$$P_m = 1 - P_u - P_d \quad (7)$$

Now it is possible to find the value of the underlying asset, S for any sequence of price movements. It will generate a directed graph with nodes labeled as asset prices and edges connecting nodes separated by one time step and a single price up, down and middle jumps as N_u, N_d, N_m , where the price after i period at node j (or after i ups and j downs) is given by: $S_{(i,j)} = u^{N_u} d^{N_d} m^{N_m} S(t_0)$ where $N_u + N_d + N_m = i$. Finally, in both binomial and trinomial tree methods, the option value can be computed by standard backward induction method.

4.3 Growing the CAPT Tree

This section visualizes how the BTT tree is constructed for pricing the options briefly. In this model as illustrated in Fig. 3 the root of the tree is the node S which is formed by a trinomial tree and the rest of the tree is constructed using binomial method with the first two time steps truncated. The barriers (the black nodes) are H_0 and L_0 at time T_0 and H_1 and L_1 at time T_0+T_1. These barriers define the allowable range for the price fluctuation of the underlying asset serving to limit both, profits and losses, for federation parties. The tree adjusts and adapts its structure to the price volatility and the moving barriers to come up with an option price that is predictable and fair for both option holder and writer.

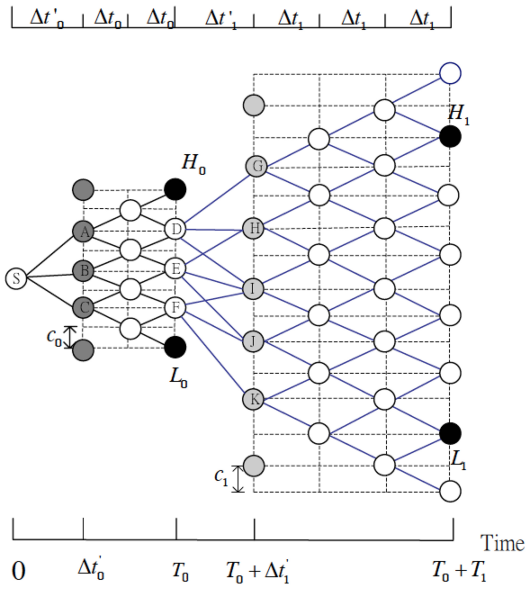


Figure 3: Cloud Option Pricing using Bino-Trinomial Tree.

The combinatorial pricing algorithm (Dai and Lyuu, 2007) is used to evaluate the option values on the three CRR trees as shown in Fig. 3, with root

nodes A, B, and C. The option price of the CAPT at node S is also evaluated by the backward induction method.

5 MODEL EVALUATION

Now, we present results from our simulation observation that show the efficiency of our model. We have implemented a Cloud federation environment using Cloud simulation platform, CloudSim (Calheiros et al., 2011). The simulated Cloud federation uses our option pricing model for trading assets and VM provisioning. The unit of resource being observed is an Amazon EC2 Standard Small Instance (US East). At the date of simulation (Sept 2013), resources advertised at a price of $\$0.085/hour$ for an on-demand instance. For RIs, the same type instance for 12 months costs $\$0.034/hour$. For evaluation purposes, (i) the reserved capacity of the data center is considered as steady constant value during simulation. (ii) to economize the equations, we do not take into account the operational costs (i.e., hardware and software acquisition, staff salary, power consumption, cooling costs, physical space, etc.) of the data center. It imposes a constant value within the model.

5.1 Simulation Setup

The simulation environment is developed to capture the behavior of our CAPT model in Cloud federation where supply and demand fluctuate in daily patterns directly inspired by real-world market. For a provider, who benefits from this market, simulation was implemented with a resource pool capable of 400 simultaneous running VMs capacity including reserved and on-demand. We have implemented the following three components on top of *CloudSim* simulator. Further details of our option-based federation simulator entities and settings are as follows.

5.1.1 CAPT Request Generator (ReqG)

The workload pattern generation was needed in order to mimic the real world IaaS Cloud requests. We implemented *CAPT-ReqG* agent to create jobs by using the *Cloudlet* class in *CloudSim*. In our model, each job has an arrival time as we scheduled the workload on a daily-basis pattern and a duration time which is the holding time of the instance by the job and metered to charge the consumer respectively. Given that our workload follows daily pattern based on normal Gaussian distribution for the 24 hours of a day and considering standard business hours (from 9 to 17) as peak hours, we generate a randomly distributed arrival time for requests in each specific hour. The load decreased 60% on weekends.

5.1.2 CAPT Resource Allocator (ResA)

We have developed the *CAPT-ResA* agent to determine the association between jobs and federated resources. Our allocation policy finds a mapping between the batch of jobs outsourced to the federation and VMs associated to options. In the simulation, the VM provisioning policy is extended to best fit with respect to the option status. The providers are implemented using *DataCenter* class in CloudSim, as it behaves like an IaaS provider. The *CAPT-ResA* receives requests from *CAPT-ReqG*, allocates resources and binds the jobs to the VMs accordingly. For resource allocation, we have used shared pool strategy. In case of arriving a new on-demand job, the agent checks if the number of currently running on-demand jobs exceeds the capacity of on-demand pool and if so, it will allocate VMs from its reserved pool while buying an option from federated Cloud. As soon as it receives requests which can not be met in-house, the agent will exercise the options that were bought before and outsource the new jobs to federated pool.

5.1.3 CAPT Option Handler (OptH)

Our *CAPT-OptH* agent implements the option pricing model as detailed in section 4. It also routes the option exercising request to the *CAPT-ResA* agent to have the requested resource provisioned. The pricing policy is set to resource/hour consumed for each instance, for the duration an instance is launched till it is terminated. Each partial resource/hour consumed will be charged as a full hour. There are six metrics that affect the CAPT option pricing, (i) the current stock price, S_0 set to $\$0.034/hour$ (ii) the exercise price (spot price), K is generated based on Amazon spot price observed pattern. (iii) the time to option expiration T set to 1 month (iv) the volatility σ which is the range and speed in which a price moves, set to 31.40%per annum. It is observed by the *cloudexchange.org*[3] which is real-time monitoring of Amazon EC2 spot prices. (v) the interest rate, r is set to 19.56% per annum since the Amazon EC2 SLA[4] interest rate is 1.5% per month and (vi) the dividend expected during the life of the option is set to $5.17. Both *High* and *Low* barriers are set to $0.039 and $0.030. The simulation was run 50 times. The experiment duration is set to 6 months and the mean value

[3] https://github.com/tlossen/cloudexchange.org

[4] http://aws.amazon.com/agreement/

of the results is evaluated to mimic the real-world environment.

5.2 Evaluation Measure

There are four empirical measures as we care to specify and observe their behavior during the simulation: (i) *Provider's profit* (P_{pr}), which our model claims to ensure the optimal utilization of the resources for providers. The profit measurement equation is:

$$P_{pr} = R_{on} + R_{res} + R_{(op,exe)} - C_{(op,pre)} - C_{(gen)} \quad (8)$$

where R_{on} and R_{res} are the providers' total revenue received over their own on-demand and RIs. $R_{(op,exe)}$ is the revenue of exercising the options since the option exercise price is less than their own instance price sold to their clients before. $C_{(op,pre)}$ denotes the premium to be paid for the purchase of the option which our model calculates accurately. Finally, $C_{(gen)}$ covers general costs of provider as we assumed a constant value. (ii) Second measure is *QoS Violations* (QoS_v), that holds the number of rejected or unmet reserved and on-demand instances reflecting the SLA violations. Third and forth measures are (iii) *Price Elasticity of Demand* (ε_D) and (iv) *Price Elasticity of Provisioning* (ε_P) where their absolute values are highly correlated with the asset price changes. Their computation is done with these equations:

$$\varepsilon_D(vm) = \frac{\%\Delta_{Qd}}{\%\Delta_{P_{vm}}} \quad and \quad \varepsilon_P(vm) = \frac{\%\Delta_{Qp}}{\%\Delta_{P_{vm}}} \quad (9)$$

The $\varepsilon_D(vm)$ and $\varepsilon_P(vm)$ denote the price elasticity of demand and provisioning of an asset, and measures the percentage change in the quantity of VM demanded and provisioned per 1% change in the price of its option premium. Our economic model should make $\varepsilon_D(vm)$ "*price inelastic*" and $\varepsilon_P(vm)$ "*price elastic*" as interpreted in section 3.2.

5.3 Results and Debate

The aggregate results imply utility and are reported as summary in Table 1. Results show that those providers are able to reach an utilization rate of 99% and achieve gains both from the in-house instances and from those obtained by exercising the option's rights from other providers of the federation. Taking these results together, four points stand out in this simulation. First, is the profit made from exercising options. To interpret this, note that in our approach, providers buy the options that its exercise price are less than their own VM provisioning price. As observed, providers were able to meet 86% of an incoming requests by inhouse provisioning and outsourced 14% of their demands to the federation, in which are fully provisioned to celebrating 8.7% more profit. Second, is the achievement over the QoS agreed with the client for resource delivery. For both reserved and on-demand, no QoS violation (no unmet request) is detected. Third, as our results indicate, the value of pricing elasticity of demand (*PEoD*) is kept less than 1 denoting that the demand became "*price inelastic*" serving to increasing demand, regardless of the asset price oscillation. From the federation perspective (resource suppliers side), the value of pricing elasticity of provisioning (*PEoP*) is more than 1 denoting that the provisioning became "*price elastic*" indicating the providers are flexible enough to adapt the amount of resources they provision. These values are consistent with the number of options purchased and exercised, leading to more economies of scale. Finally, is the utilization value, which is considerable. This indicates that optimal utilization of resources is achieved to exploit the efficiency and accuracy of our model.

To form a basis for comparison, our next two figures depict the dependencies between option pricing elasticity and its demand and provisioning. Fig. 4, shows how CAPT controls the option pricing elasticity and converges to a more stable price smoothly. Our approach finds the optimal option price of the federated resource in the Cloud to come to an equilibrium between PEoD and PEoP. The asset equilibrium price occurs when the supply resource pool matches the aggregate demand indicating an optimal resource utilization. From the provisioning perspective, as shown in Fig. 5, it can be seen, that a 21% increase (using midpoint method) in the asset price leads to a 35% increase in quantity provisioned. This indicates an elastic supply. Since the asset pricing elasticity is controlled, we see a synchronous correlation between price and supply changes. As a result, the total revenue moves in the direction of price change.

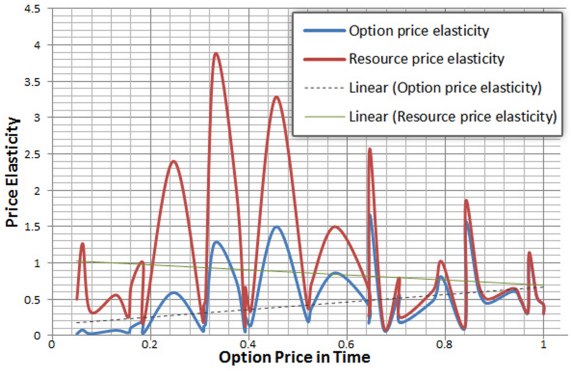


Figure 4: Resource price elasticity controlled via options.

Table 1: Cloud federation market simulation summary (6 months).

			QoS Violations		Profit		Price Elasticity		Options	
Market↓ Measures →	Workload	Utilization	Reserved	On-demand	In-house	Option	PEoD	PEoP	Bought	Exercised
Cloud Federation Market	98455	99%	0	0	35293.63	3339.97	0.095	1.28	25676	14020

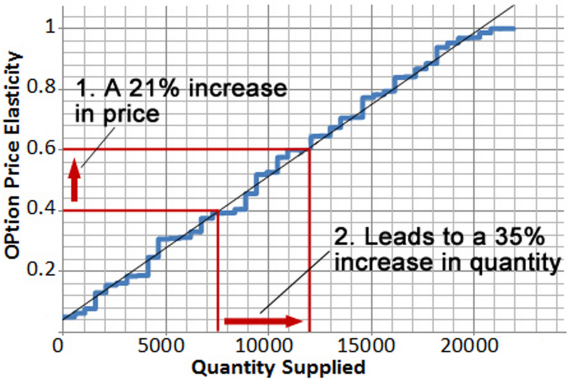


Figure 5: Price elasticity of resources provisioned.

6 RELATED WORK

In relation to our approach, there are some alternatives that propose federation economic model more focused on the provider's perspective. A broker-based federation approach has been proposed by (Villegas et al., 2012), (Rogers and Cliff, 2012) and (Raj, 2011). These studies decouple the brokerage strategies and federation economic valuation. Zhang proposes an economic model for the evaluation of the economic value of Cloud Computing Federation in providing one computing unit such as the power and human resources (Zhang and Zhang, 2012). Just as Clouds enable users to cope with unexpected demand loads, a Federated Cloud will enable individual Clouds to cope with unforeseen variations of demand. Authors in (Gomes et al., 2012) investigate the application of market-oriented mechanisms based on the General Equilibrium Theory of Microeconomics to coordinate the sharing of resources between the Clouds in the federated environment. In (Zhao et al., 2012), authors present an online resource marketplace for open Clouds by adopting an eBay style transaction model based on auction theory. Here (Samaan, 2013) establishes a novel economic sharing model to regulate capacity sharing in a federation of hybrid Cloud providers. The idea of financial options is used by (Sharma et al., 2012) as a financial model for pricing Cloud compute commodities by using Moore's law on depreciation of asset values, to show the effect of depreciation of Cloud resource on QoS. In (Toosi et al., 2012) authors incorporate financial options as a market model for federated Cloud environments. In contrast to existing approaches, we use financial option theory for asset trading and propose a dynamic and adaptive option pricing model which enhance profit by controlling the pricing elasticity of demand and provisioning in the Cloud federation.

7 CONCLUSION AND OUTLOOK

Providers consider federations as an alternative pool of resources to their expected consumption growth. Their demand to use the federated asset is dependent to the pricing elasticity of demand, as if the elasticity is high, then they will be more careful on buying options. In this paper, we proposed a financial option pricing model to address the pricing elasticity concerns in above situation. Our economical model is for implementing a future market of virtualized resources in a system where a federation of Cloud providers is used to reduce risks and costs associated with the capacity planning of Cloud providers. Providers will benefit by this model to make decisions when to buy options in advance and when to exercise them to achieve more economies of scale.

So far, we have proposed an economic model that considers future aspects of trading like capacity planning or resource allocation over upcoming demands. The CAPT model empowers vendors to get additional resources as and when required. This economic model aims for the leverage of demand and supply form the IaaS provider and third party providers point of view, finding suboptimal price policies between resources owned by the provider and options to external providers using Cloud bursting when needed. This study covers two aspects of resource elasticity: *Resource Quantity* and *Price*. As an outlook, our future work includes further extension to the model that can also support the *Quality of Service (QoS)* aspect in federation environment.

REFERENCES

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, A. F., and Buyya, R. (2011). Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exper.*, 41(1):23–50.

Cox, J. C., Ross, S. A., and Rubinstein, M. (1979). Option pricing: A simplified approach. *Journal of Financial Economics*, 7(3):229–263.

Dai, T.-S. and Lyuu, Y.-D. (2007). An efficient, and fast convergent algorithm for barrier options. In Kao, M.-Y. and Li, X.-Y., editors, *Algorithmic Aspects in Information and Management*, volume 4508 of *Lecture Notes in Computer Science*, pages 251–261. Springer Berlin Heidelberg.

Dustdar, S., Guo, Y., Satzger, B., and Truong, H.-L. (2011). Principles of elastic processes. *Internet Computing, IEEE*, 15(5):66–71.

Gomes, E. R., Vo, Q. B., and Kowalczyk, R. (2012). Pure exchange markets for resource sharing in federated clouds. *Concurr. Comput. : Pract. Exper.*, 24(9):977–991.

Kurze, T., Klems, M., Bermbach, D., Lenk, A., Tai, S., and Kunze, M. (2011). Cloud federation. *Computing*, (c):32–38.

Mankiw, N. G. (2012). Elasticity and its applications. In *Principles of Microeconomics, 6th Edition*, pages 89–109. Harvard University.

Raj, G. (2011). An efficient broker cloud management system. In *Proceedings of the International Conference on Advances in Computing and Artificial Intelligence*, ACAI '11, pages 72–76, New York, NY, USA. ACM.

Rogers, O. and Cliff, D. (2012). A financial brokerage model for cloud computing. *Journal of Cloud Computing*, 1(1):1–12.

Samaan, N. (2013). A novel economic sharing model in a federation of selfish cloud providers.

Sharma, B., Thulasiram, R., Thulasiraman, P., Garg, S., and Buyya, R. (2012). Pricing cloud compute commodities: A novel financial economic model. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 451–457.

shyr Dai, T. and dauh Lyuu, Y. (2010). The bino-trinomial tree: A simple model for efficient and accurate option pricing. *Journal of Derivatives*, 17(4):7–24.

Toosi, A., Thulasiram, R., and Buyya, R. (2012). Financial option market model for federated cloud environments. In *Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on*, pages 3–12.

Villegas, D., Bobroff, N., Rodero, I., Delgado, J., Liu, Y., Devarakonda, A., Fong, L., Masoud Sadjadi, S., and Parashar, M. (2012). Cloud federation in a layered service model. *J. Comput. Syst. Sci.*, 78(5):1330–1344.

Zhang, Z. and Zhang, X. (2012). An economic model for the evaluation of the economic value of cloud computing federation. In Zhang, Y., editor, *Future Communication, Computing, Control and Management*, volume 141 of *Lecture Notes in Electrical Engineering*, pages 571–577. Springer Berlin Heidelberg.

Zhao, H., Yu, Z., Tiwari, S., Mao, X., Lee, K., Wolinsky, D., Li, X., and Figueiredo, R. (2012). Cloudbay: Enabling an online resource market place for open clouds. In *Utility and Cloud Computing (UCC), 2012 IEEE Fifth International Conference on*, pages 135–142.

Creo: Reduced Complexity Service Development

Per-Olov Östberg and Niclas Lockner
Dept. of Computing Science, Umeå University, Umeå, Sweden
{*p-o, lockner*}*@cs.umu.se*

Keywords: Service-orientated Architecture, Service Development Tools.

Abstract: In this work we address service-oriented software development in distributed computing environments, and investigate an approach to software development and integration based on code generation. The approach is illustrated in a toolkit for multi-language software generation built on three building blocks; a service description language, a serialization and transport protocol, and a set of code generation techniques. The approach is intended for use in the eScience domain and aims to reduce the complexity of development and integration of distributed software systems through a low-knowledge-requirements model for construction of network-accessible services. The toolkit is presented along with a discussion of use cases and a performance evaluation quantifying the performance of the toolkit against selected alternative techniques for code generation and service communication. In tests of communication overhead and response time, toolkit performance is found to be comparable to or improve upon the evaluated techniques.

1 INTRODUCTION

Cloud computing has in recent years evolved to an established paradigm for provisioning of IT capacity. While this approach can offer several benefits compared to traditional static provisioning, e.g., facilitation of more flexible service types (Armbrust et al., 2010) and improvements in cost and energy efficiency of large-scale computing (Walker, 2009; Berl et al., 2010), it also places focus on a current problem in distributed computing: the increasing complexity of development and management of systems in distributed computing environments (Kephart and Chess, 2003).

Service-Oriented Computing (SOC) is a popular approach to software development and integration in large-scale distributed systems. SOC is argued to be well suited for cloud environments as it places focus on representation of logic components as network-accessible services, and aims to facilitate development and integration of systems through coordination of service interactions. At architecture level, Service-Oriented Architectures (SOAs) define service interfaces as integration points and address system composition at interface or protocol level. While a number of SOA techniques have emerged, service development and integration are still complex issues and there exists a need for development tools that provide non-complex and low-learning-requirement environments for efficient development of service-based systems.

To illustrate these issues, we here take the perspective of eScience application development. In eScience[1], distributed computing techniques are used to create collaborative environments for large-scale scientific computing. In comparison to commercial software stacks, scientific computing tools are typically prototype-oriented, developed in projects with limited software development budgets, and often composed of heterogeneous components developed in multiple languages and environments. In addition, eScience applications often use distributed or parallel programming techniques to exploit the inherent parallelism of computational problems. As many current eScience efforts are approaching construction of virtual infrastructures using cloud technology, they here serve as illustrative examples of the difficulties of developing multi-language software stacks in heterogeneous distributed computing environments.

In this work we address reduction of complexity in service-based software development, and present an easy-to-use toolkit for efficient cross-language integration of software services. The toolkit is based on three core components: a simplified syntax service description language, a transparent data serialization and transmission protocol, and a set of code generation tools designed to abstract complexity in service and service client development.

[1] Computationally intensive science carried out in highly distributed network environments.

The remainder of the paper is structured as follows: Section 2 presents project background and a brief survey of related work, Section 3 outlines the proposed approach and toolkit, and Section 4 discusses use cases for the approach. In the second half of the paper, Section 5 contains a performance evaluation quantifying toolkit performance against selected alternative techniques for code generation and service communication, followed by conclusions and acknowledgements in sections 6 and 7.

2 RELATED WORK

A number of tools for service development and middleware construction exist, ranging in complexity and abstraction levels from very simple fine-grained interprocess communication tools to advanced middleware construction tools featuring advanced data marshalling, call translation, and remote reference counting techniques. In general there exists trade-offs between complexity and efficiency that make service technologies more or less suitable for certain situations, and many technologies have been developed for specific application scenarios.

For example, direct interprocess communication technologies such as traditional remote procedure calls (RPC) (Birrell and Nelson, 1984) and Java Object Serialization (JOS) (Oracle, 2005) (over sockets) provide transparent development models but offer little in ways of complexity abstraction. Other approaches such as Java Remote Method Invocation (RMI) (Wollrath et al., 1996) and the Microsoft Windows Communication Framework (WCF) (Mackey, 2010) offer development models tightly integrated into mature commercial software development environments, but lose some applicability in multi-platform application scenarios. There exists also standardized approaches to multi-language and multi-platform service development, e.g., the Common Object Request Broker Architecture (CORBA) (Vinoski, 1993), but while such standardized approaches typically are very expressive and capable of application in multiple programming styles, e.g., object-orientation and component-oriented development, this general applicability often comes at the price of very steep learning curves and high development complexity.

In service-oriented computing and architectures, programming models such as SOAP and REST-style web services are widely used due to features such as platform independence, high abstraction levels, and interoperability. The SOAP approach to web services favors use of standardization of XML-based service description and message formats to facilitate automated generation of service interconnection code stubs, dynamic service discovery and invocation techniques, and service coordination and orchestration models. SOAP-style web services are however often criticized for having overly complex development models, inefficiencies in service communication, and low load tolerances in servers (although developments in pull-based parser models have alleviated some of the performance issues (Govindaraju et al., 2004)).

The REpresentational State Transfer (REST) (Fielding, 2000) web service model is often seen as a light-weight alternative to the complexity of SOAP-style web service development. The REST approach discourages standardization (of message formats), promotes (re)use of existing wide-spread technology, and aims to give service developers more freedom in, e.g., choice of data representation formats and API structures. While this approach facilitates a development model well suited for smaller projects, it is sometimes argued to lead to more tightly coupled service models (that require service client developers to have knowledge of service-side data structures) and introduce technology heterogeneity in large systems.

Although service models are considered suitable for large-scale system integration, and some understanding of the applicability of web services has been gained (Pautasso et al., 2008), neither approach fully addresses the requirements of service-oriented software development and a number of technologies for hybrid service-RPC mechanisms have emerged. These include, e.g., interface definition language (IDL) based technologies such as Apache Thrift (Slee et al., 2007), an RPC framework for scalable cross-language service development, Apache Avro (Apache, 2009), a data serialization system featuring dynamic typing, and Google protocol buffers (Google, 2008), a method for serializing structured data for interprocess communication. For high performance serialization and transmission, there also exists a number of non-IDL based serialization formats and tools such as Jackson JSON (Jackson, 2009), BSON (MongoDB Inc., 2007), Kryo (Kryo, 2009), and MessagePack (Furuhashi, 2011).

In addition to trade-offs for technical performance and applicability, tools and development models often impose high learning requirements in dimensions orthogonal to the task of building distributed systems. For example, the Distributed Component Object Model (DCOM) requires developers to understand data marshalling and memory models, Java RMI distributed garbage collection, CORBA portable object adapters (type wrappers), and SOAP web services XML Schema (for type definition and valida-

tion). As distributed systems are by themselves complex to develop, debug, and efficiently analyze, there exists a need for software development tools that provide transparent and intuitive development models, and impose low learning requirements.

In this work we build on the service development model of the Service Development Abstraction Toolkit (Östberg and Elmroth, 2011), and investigate an approach to construction of development tools focused on reducing complexity of service-based software development. The aim of this approach is to combine the high abstraction levels of SOAP-style web services (using a simplified service description syntax) with the communication efficiency of more direct RPC-style communication techniques, and produce tools with low learning requirements that efficiently facilitate service development. As the work is based on code generation, the approach can be seen akin to development of a domain-specific language (Van Deursen et al., 2000) for service description, but the main focus of the work is to reduce overhead for exposing component logic as network-accessible services. The work is done in eScience settings, and presented results are primarily intended to be applied in scientific environments, e.g., in production of tools, applications, and middlewares for scientific simulation, experimentation, and analysis.

3 CREO

Service-oriented architectures typically expose components and systems as platform independent, network-accessible services. While this approach gracefully abstracts low-level integration issues and provides for high-level architecture design models, it can often lead to practical integration issues stemming from, e.g., complexity in service development models, steep learning curves of service development tools, and lack of distributed systems development experience in service client developers.

In this paper we build on earlier efforts presented in (Östberg and Elmroth, 2011) and (Östberg et al., 2012), and propose an approach to service development that places the responsibility of service client development on service developers. As this shift in responsibility introduces noticeable additional complexity in service development, e.g., in requirements for multi-language service client development, we note a need for tools to support the approach and present *Creo* - a service development toolkit based on automated code generation.

The Creo toolkit is aimed to reduce complexity in construction of network-accessible services by providing a development model that lowers learning requirements and increases automation in service development. While the toolkit is designed to be simple to use and targeted towards developers with limited distributed systems development experience, it also strives to provide service communication performance high enough to motivate use of the toolkit in mature service development scenarios.

To limit the scope of the work, we have initially designed the toolkit to support development of services in a single language (Java), and service client development in four languages common in eScience environments: C, C#, Java, and Python. The toolkit implementation patterns are however transparent and modularized, and all modules are designed to be extensible to code generation in additional languages. The intent of the toolkit is to provide robust service communication stubs in general purpose programming languages that can later be used to build integration bridges into special purpose environments such as R and Matlab. The choice of Java as service language is motivated by the language's rich development APIs, robustness in performance, platform independence, and wide-spread adoptance in operating systems and server platforms. The design philosophy of the toolkit can be summarized as supporting advanced implementation of services while keeping generated code for clients as transparent, lightweight, and free of external dependencies as possible.

To combine the ease-of-use of high abstraction level tools with the communication performance of more fine-grained approaches, the toolkit development model is based on the service description approach of SOAP-style web services combined with a customized version of the RASP protocol presented in (Östberg et al., 2012). The toolkit service development process can be summarized in three steps:

1. Service description. Service type sets and interfaces are defined in a custom service description (interface definition) language
2. Communication code generation. Service and service client communication stubs are generated from service descriptions.
3. Service integration. Logic components are exposed as services through implementation of generated service interfaces, and service clients are implemented based on the generated communication stubs for service interconnection.

In all steps of this process, the toolkit aims to reduce the complexity of service development by providing intuitive tools and formats for service description, data representation, and code generation.

3.1 Service Description

Program 1 A sample Creo service description.

```
// annotations
@PACKAGE("packagename")

// type definitions
struct MetaData
{
  String description;
  long timestamp;
}

struct Data
{
  MetaData metadata;
  double[] samples;
}

// interface definitions
interface DataService
{
  void storeData (Data[] data);
  Data retrieveData (String description);
}
```

For data type and service interface definition, the toolkit employs a service description language comprised of three parts:

- Annotations. Define code generation parameters, e.g., service package names.
- Types. Specifies a basic set of primitive types and a struct mechanism for type aggregation.
- Interfaces. Define service interfaces in terms of methods and method parameters.

The service description language format is based on the block syntax of the C/C++ family of languages. In the interest of simplicity, the primitive type set is restricted to a basic type set commonly occurring in most programming languages: `byte`, `char`, `int`, `long`, `float`, `double`, and `String`. The language supports direct aggregation of primitive types in structs and arrays as well as construction of compound types via aggregation of structs. This allows construction of hierarchical data types such as trees, but not cyclic data types such as graphs. Program 1 contains a sample service description demonstrating the aggregation mechanisms of the Creo service description language.

While alternative representation formats with more advanced features exist, e.g., schema-based type set and data validation in XML and WSDL, the design philosophy of this work is to reduce complexity rather that offer advanced features. The goal of the description language is to provide a convenient format that has great expressive power, is as unambiguous as possible, and introduces as few learning requirements as possible. The primitive type set defined, as well as the concept of aggregation of fields in records and arrays, are prevalent in programming languages and should prove intuitive to developers regardless of background. To minimize the learning requirements of the tool, the type interpretations and syntax of the description language are based on a subset of the well-known Java programming language.

3.2 Data Representation

To promote transparency, the representation format specified in service description also directly outlines the data structures used in data serialization and transmission. For language and platform independence, all values are transformed to and from network byte order in transmission and support code is generated for programming languages not supporting description language features (e.g., byte order transformation, string classes, or array types). For aggregated types, types are serialized in the order declared (and stored in memory), with size counters prefixing data for array types and strings. As data are declared and stored in hierarchical structures (trees), data serialization is essentially a left-wise depth-first traversal of data trees, where individual node values are stored sequentially. In terms of invocation semantics, Creo defines call-by-value semantics for invocation of remote service methods. As data are serialized by value, the use of reference and pointer types inside data blocks passed to services is not supported. In particular, use of circular references (e.g., cyclic graphs) may lead to inefficient transmission performance or non-terminating loops.

For efficiency in transmission (i.e. minimization of system calls and alignment of network package sizes to maximum transfer units), all data are serialized and deserialized via transmission buffers located in the generated code stubs. The protocol used for transmission of data between clients and services (illustrated in Figure 1) is a customized version of the Resource Access and Serialization Protocol (RASP) of the StratUm framework (Östberg et al., 2012). The description language does not support encoding of explicit exception messages for propagating error information across process boundaries.

3.3 Code Generation

Service integration code is typically provided in one of two forms: APIs or service communication stubs. To reduce complexity in service client development,

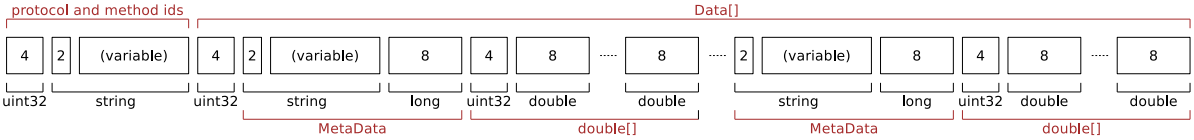


Figure 1: Byte layout of the Creo protocol request message for the `sendData()` method of Program 1. Data encoded in the order defined in service descriptions, arrays and strings prefixed with item counts. Byte block sizes and primitive types in black, protocol preamble (protocol and method ids) and aggregated (struct and array) types in red.

and increase the transparency of the service communication mechanisms, the Creo toolkit uses a code generation approach centered around immutable wrapper types and call-by-value interfaces. The rationale of this design is to make use of generated client code as intuitive as possible, and to facilitate a service client development model that doesn't require prior distributed systems development experience.

Use of code generation techniques rather than APIs fundamentally assumes that service descriptions rarely change (as service-oriented architectures tend to be designed in terms of service interfaces), and have the added benefits of allowing typed languages to catch type errors earlier while keeping service client implementations loosely coupled to services.

3.3.1 Code Generator

From a high level, the Creo toolkit can be seen to be composed of three components: a service description parser, a framework generator, and a custom package generator. To promote flexibility and facilitate adaptation to new requirements, e.g., support for new programming languages or representation formats, the architecture of the toolkit is designed to be modular and extensible. The separation of code generation for frameworks and custom packages (i.e. code specific to data types and services defined in service descriptions) serves to facilitate third party implementation of code generator plug-ins. With this separation it is possible to contribute plug-in modules to support alternative implementations of, e.g., data serialization routines and client implementations, without having to deal with generation of framework code.

The service description parser is constructed using a combination of in-memory compilation of the service description types (after replacing selected keywords to make service descriptions Java compliant), use of the Java reflection API (to validate description structures), and a custom language parser (that extracts parameter information). To isolate code generators from document parsing, the parser provides a full internal API that completely describes the type sets and document structures of service descriptions.

3.3.2 Generated Code - Framework

To establish a uniform model for client-service communication, all service client code implements a framework model for connection establishment, data serialization, and transmission capabilities. This framework is structured around an identified core feature set that includes, e.g., primitive type representation and serialization (including network byte order transformations), array and string type wrapper types (for languages not providing such types), and socket-level read and write transmission buffers.

The purpose of the framework is to separate service and client logic independent of the types and services defined in service descriptions, and reduce the complexity of generating code for service-dependent logic. Implementation of this framework pattern keeps all service client implementations lightweight and compatible with the service implementation, which facilitates development of client implementations in additional languages. On the service side, the framework code is connected to the service-dependent code through a provider-pattern implementation for service data type serializer factories.

3.3.3 Generated Code - Service Side

On the service side, the generated framework is extended with a lightweight service hosting environment containing basic server functionality such as thread and service management. The architecture of the service framework is based on the principle of abstracting as much as possible of the service boilerplate code required to expose components as services. It is the intent of the toolkit that service implementation should consist only of two steps - generation of the service framework from a service description file and implementation of a service (Java) interface.

The basic structure of the generated services is designed around the information flow in the system; a server hosts services, parses incoming requests, and passes request messages onto an incoming message queue for the requested service. The service implementation processes requests, generates and pushes response messages onto the outgoing message queue for the service. The server continuously monitors

all service message queues and sends response messages when available. The core of the generated service framework is message-oriented and defined around the concept of asynchronous message queues, and does not restrict service implementations to use of only synchronous request-response communication patterns. However, while service implementations are free to define their own communication patterns in terms of the messages exchanged between clients and services, use of asynchronous communication patterns requires modifications of the generated service clients to fully support such exchanges. For reference, an asynchronous client (in Java) is provided with the generated service framework.

3.3.4 Generated Code - Client Side

The architecture of the generated service clients follows the same pattern in all implementing service client languages (C, C#, Java, and Python), and is designed to abstract fine-grained service communication tasks. A service API is generated exposing the methods defined in service descriptions, and all data are managed in immutable wrapper types based on the types defined in service descriptions. Service communication details, such as connection establishment and data marshalling, are abstracted by clients stubs.

The underlying philosophy of the toolkit is that it should be the responsibility of the service developer to provide integration code (service clients) and APIs for services, and the toolkit aims to abstract as much as possible of that process. To promote transparency, all client code generated is designed to follow the same design pattern and all generated service client code is designed to be as homogeneous as possible in architecture, code structure, and API functionality support. When applicable, all code is generated along with sample build environment data files (e.g., makefiles for C and ant build files for Java). In-memory compilation and generation of Java Archive (JAR) files are supported for Java.

4 USE CASES

To illustrate toolkit use, we here briefly discuss example application scenarios in the eScience domain. Envisioned use cases for the Creo toolkit include:

- Coordinated multi-language logging and configuration. Scientific applications in the eScience domain often consist of multiple components and systems developed in multiple programming languages. Coordinated logging of application state information can be very useful for visualization and management of application processes, which can be achieved by, e.g., developing a database accessor component in Java and exposing it as a service using the Creo toolkit. Client stubs generated by the toolkit can then be used to coordinate system logs from multiple sources without introducing external dependencies in systems. Similarly, multi-component systems can also use this technique to coordinate system configuration, allowing dynamic reconfiguration of systems (use cases from the StratUm (Östberg et al., 2012) project).
- Multi-component system integration. The Aequus system (Östberg et al., 2013)) system is designed for use in high performance and grid computing infrastructures. While the core of the system is developed in Java, the system also contains specialized components and tools developed in other languages, e.g., scheduler integration plug-ins in C and visualization and statistics tools in Python and Matlab. Use of the Creo toolkit allows smooth integration of different parts of the system without extensive distributed systems development effort.
- System evaluation experiments. Distributed computing infrastructure systems constructed as service-oriented architectures often require simulation experiments for testing and validation of functionality. The previously mentioned Aequus system is developed and evaluated using emulated system environments for system tests and scalability simulations. In these settings the Creo toolkit allows easy integration of multiple simulation components for surrounding systems (e.g., batch schedulers and accounting systems), and construction of large-scale emulation systems for system evaluation.
- Application cloud migration. Many eScience applications are initially developed for use on a single machine and later (for performance and scalability reasons) transformed into multi-component systems using parallel and distributed computing techniques. As part of this process, staging of applications into cloud environments often requires some form of reformulation of computational algorithms to better adapt to horizontal cloud elasticity models. The Creo toolkit can here be used to, e.g., build staging and monitoring tools or to facilitate remote communication with applications running in cloud data centers.

Use cases such as these illustrate not only the expressive power of tools for service development and component integration, but also the importance of keeping such tools simple and reducing the complexity of building distributed systems. Use of develop-

Table 1: A brief overview of the feature sets of the evaluated service technologies.

	Creo	Thrift	SOAP	REST	RMI
Interface type	IDL	IDL	IDL	protocol	API / stubs
Integration style	stubs	stubs	API / stubs	API / protocol	stubs
Data representation format	binary	text / binary	text	text / binary	binary

ment tools with steep learning curves or advanced knowledge requirements for, e.g., serialization formats, marshalling techniques, and transmission formats, can greatly add to the complexity of building distributed systems. For many purposes, and prototype development in particular, reduction of complexity and ease-of-use often outweigh the additional features of more advanced approaches.

5 EVALUATION

Service-based software design is an area with many competing approaches to service development and integration, making objective evaluation of new tools non-trivial.

In this work we identify three abstraction levels for development toolkits; low (fine-grained message level integration), intermediary (remote procedure call communication abstraction), and high (service-oriented component integration); and evaluate the proposed toolkit against selected tools from each abstraction level in the dimensions of serialization overhead, transmission overhead, and service response time. To facilitate future comparison against third party tools, we select well-established and easily accessible tools for the evaluation.

For low level abstractions we compare the performance of the toolkit against that of Apache Thrift (Apache, 2010), a software framework for scalable cross-language service development. As the toolkit primarily targets service development in Java, we have for high and intermediary levels selected Java-based tools. For intermediary level we evaluate two related technologies: Java Remote Method Invocation (RMI) (Wollrath et al., 1996), an object-oriented remote procedure call mechanism that supports transfer of serialized Java objects and distributed garbage collection, and Java Object Serialization (JOS) (Oracle, 2005), the object serialization technology used by Java RMI. For high level, we evaluate the toolkit against two popular web service technologies: REST web services (using the RESTlet framework version 2.0.15 (Restlet, 2013)) and SOAP web services (using the Apache Axis 2 SOAP framework version 1.6.2 (Apache, 2005)). Table 1 provides a brief comparison of the feature sets of the evaluated service technologies.

5.1 Testbed and Experimental Setup

To evaluate the technical performance of the toolkit we measure three facets of service communication performance; serialization overhead, transmission overhead, and response time; and quantify these against corresponding measurements of selected alternative tools. Serialization overhead is here defined in terms of the computational capacity used for generation and parsing of service messages, and is included in tests as it can heavily impact the execution footprint of service-based tools. Transmission overhead is here defined to be the additional bandwidth requirements introduced by service data representation formats, and is measured by quantitative comparison of total message sizes and message payload (raw data) sizes. To isolate the communication overhead components introduced by service tools in response time measurements, thin service implementations (minimal request processing times) are used.

Tests are performed using three types of request data; coarse-grained data (byte chunks), fine-grained number-resolved data (integer and float values), and fine-grained string-resolved data (text segments). For each test and request type, tests are performed with request sizes grown by orders of magnitude (blocks of 100, 1k, 10k, 100k, 1M, 10M and 100M bytes). Coarse-grained requests consist of large chunks of bytes without structured format. For clients based on Creo, Thrift, RMI, and JOS coarse-grained data are sent as raw byte arrays. For REST-based clients, requests are sent in HTTP POST requests as raw bytes with the MIME type ”application/octet-stream”. In SOAP-based clients, request data are encoded as Base64-encoded strings.

Data for fine-grained requests are created by grouping data in blocks of 10 bytes, grown by aggregating data blocks in groups of 10, and padded using smaller data blocks to align sizes with even exponentials of 2. For example, a 1k (1024 bytes) data block consists of 10 groups of 10 blocks of 10 bytes plus padding in the form of two 10 byte blocks and a 4-byte pad value (a 32-bit integer or a 4-byte string depending on type). Larger data blocks are grown using the same scheme, e.g., by aggregating ten 1k data blocks to form a 10k data block. Numbers-based data

blocks contain pairs of 64-bit double-precision floating point and 16-bit integer values. String-based data blocks contain 10-character strings.

For serialization overhead and service response time tests, all tests are done by measuring the client-side makespans of full operations, starting at the point of client invocation and ending when the client receives a uniform size 4 byte server response message. To isolate overhead components, all measurements are performed in closed loop system settings using sequential invocation patterns on dedicated machines with no competing load and isolated network segments. Experiments are repeated multiple (at least ten) times to minimize the impact of external factors on measurements. Parallel invocation tests are used to evaluate the load sensitivity and scalability of service tools. All services used in measurements are implemented in Java and service clients are implemented in C, C#, Java and Python. For tests of the service response time of REST and SOAP tools, request serialization is done in JSON (using the reference library of json.org) and XML (using JAXB).

All tests are run on a dedicated symmetric cluster where nodes are equipped with dual 16 core 2.1 GHz AMD Opteron 6272 processors and 54 GB RAM. Nodes are interconnected with 1 Gbps links and networks are configured using MTU sizes of 1500 bytes. All nodes run Ubuntu Linux 12.04 kernel version 3.2, OpenJDK 1.6, Python 2.7, Mono 2.10, and GLib 2.32. All software are accessible from Ubuntu repositories.

5.2 Serialization Overhead

To isolate measurements of data serialization overhead, it is necessary to exclude all artefacts from transmission of data between clients and the servers in tests. Additionally, as tools employ transmission (read and write) buffers that consume computational power and are orthogonal to data serialization, transmission buffers need to be bypassed in tests. To quantify the serialization overhead of Creo and Thrift service clients, both generated code and runtime libraries are modified so that no data are placed in transmission buffers or sent to servers after serialization. Furthermore, both tool's service clients are modified so that they do not read data from servers after invocations.

To avoid modifications of Java RMI stacks, we here include measurements of the underlying serialization technology used (JOS) and assume measurements are representative of the serialization overhead of RMI. To quantify the serialization overhead of JOS, `ObjectOutputStream` instances are wrapped around non-buffered dummy output streams (no data transferred to underlying sockets). After modifications, serialization overhead tests are performed in the same way as service response time tests.

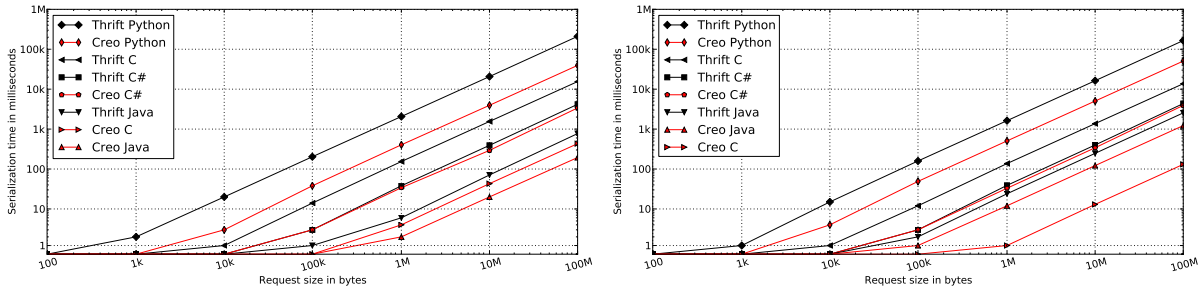
Results from data serialization overhead tests are visualized in figures 2 and 3. For ease of comparison, test results for multi-language tests (comparing Creo to Thrift) using fine-grained data tests are presented individually, separating tests using number-resolved and string-resolved data. As can be seen in Figure 2, Creo improves upon the the performance of Thrift for fine-grained data on average of factors 1.16 to 5.23 for C#, Java, and Python clients. Compared to the less mature Thrift C clients, Creo shows improvements of factors 36.84 to 115.69. When comparing the performance of Creo against that of other Java-based tools (illustrated in Figure 3), Creo exhibits performance improvements of on average of factors 5.66 to 388.56, which is attributed to use of more complex serialization techniques and text-resolved data representation formats in other tools.

These tests illustrate the magnitude of serialization overhead for complex serialization techniques, as well as the impact serialization overhead can have on service execution footprint and performance. For example, the JAXB serialization engine used in SOAP tests is unable to process messages of sizes 100 MB in standalone settings, indicating a potential source for load issues when used inside service engines.

5.3 Transmission Overhead

To evaluate transmission overhead for service communication a simple server component that counts and returns the number of bytes in requests is used. Service invocation makespan is measured on the client side and used to quantify transmission overhead for service invocations with known request payload sizes. Apache Thrift supports transmission of data using three protocols: text-resolved JSON and two binary protocols: TBinaryProtocol and TCompactProtocol, where the former sends data as-is and the latter uses variable-length encoding of integers. The purpose of this encoding scheme; which for example encodes 16-bit integers as 1-3 bytes, 32-bit integers as 1-5 bytes, and 64-bit integer as 1-10 bytes; is to reduce the size of payload and commonly occurring metadata such as the length of strings, arrays, and collections. In tests we primarily use TBinaryProtocol as it is supported in all languages, and evaluate the efficiency of TCompactProtocol in the languages supported (and quantify it against that of Creo and the binary protocol) in separate tests.

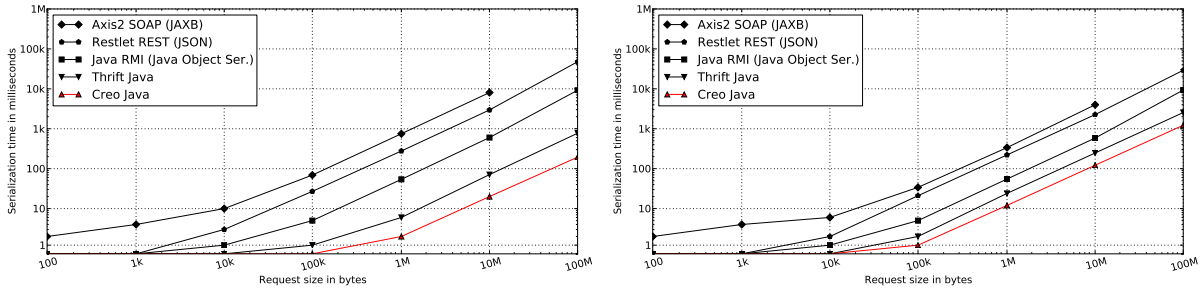
For ease of comparison, test results for compact binary representation formats (Creo, Thrift, and JOS) and text-resolved formats (JSON REST and XML



(a) Number-resolved data. On average Creo shows improvements of factors 36.84 (C), 1.23 (C#), 3.51 (Java), and 5.23 (Python) in serialization time.

(b) String-resolved data. On average Creo shows improvements of factors 115.69 (C), 1.16 (C#), 2.03 (Java), and 3.24 (Python) in serialization time.

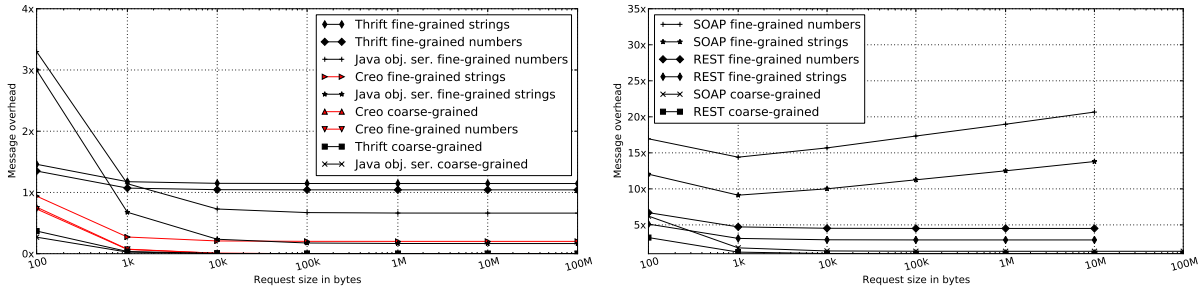
Figure 2: Creo and Thrift serialization time (in milliseconds) for fine-grained messages. Axes logarithmic.



(a) Number-resolved data. On average Creo shows improvements of factors 388.56 (SOAP), 177.18 (REST), and 34.84 (RMI) in serialization time.

(b) String-resolved data. On average Creo shows improvements of factors 30.37 (SOAP), 20.25 (REST), and 5.66 (RMI) in serialization time.

Figure 3: Serialization time (in milliseconds) of Java-based tools for fine-grained messages. Axes logarithmic.



(a) Creo, Thrift, and JOS.

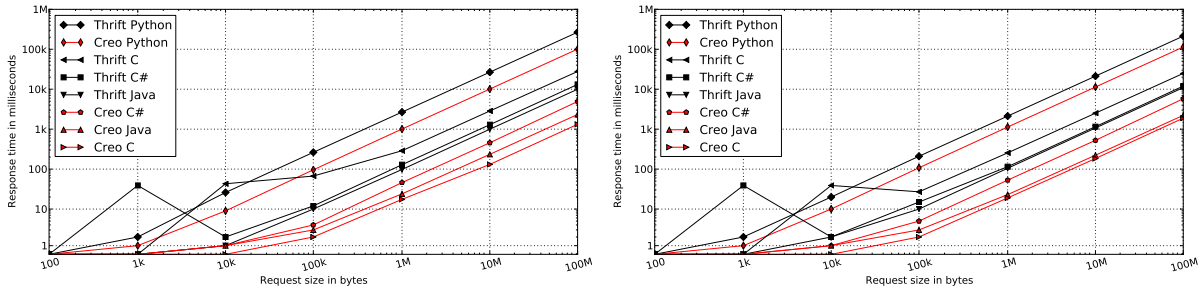
(b) REST and SOAP.

Figure 4: Transmission (message size) overhead for service invocation requests. Horizontal axis logarithmic.

SOAP) are presented separately. As can be seen in Figure 4a, compact coarse-grained (binary) data are represented with little overhead and fine-grained data are represented with overhead within a factor of 2 in size for Creo, Thrift and JOS. As can be seen in Figure 4b, the use of text-resolved representation formats can introduce significant overhead for fine-grained data, ranging in tests up to a factor of 20 for JSON REST and XML SOAP (both of which are unable to process messages larger than 10MB in tests).

5.4 Service Response Time

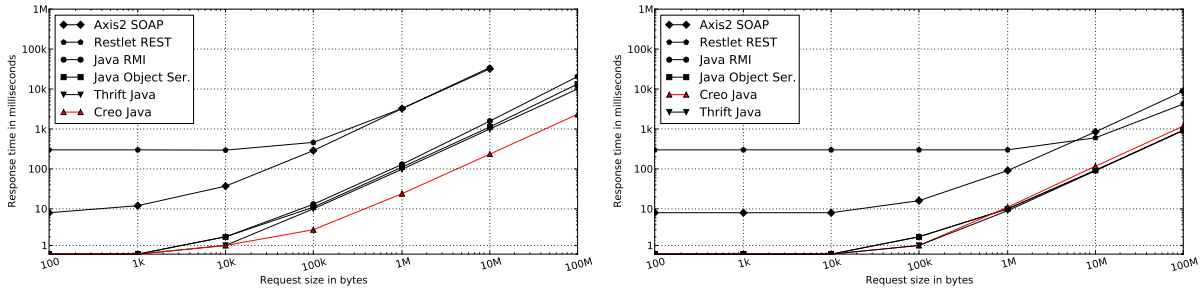
Having roughly quantified the impact of potential overhead sources for data serialization and transmission, we analyze the communication performance of the evaluated tools in terms of service request response times. Using closed system loop settings (sequential invocations of services deployed in isolated systems), we measure invocation makespan from the client perspective and use it as a measurement of service response time. To verify the transfer of results from sequential tests to (more realistic) parallel invocation scenarios, we also validate results using paral-



(a) Number-resolved data. On average Creo shows improvements of factors 19.76 (C), 2.77 (C#), 4.21 (Java), and 2.65 (Python) in service response time.

(b) String-resolved data. On average Creo shows improvements of factors 13.42 (C), 2.15 (C#), 4.77 (Java), and 1.87 (Python) in service response time.

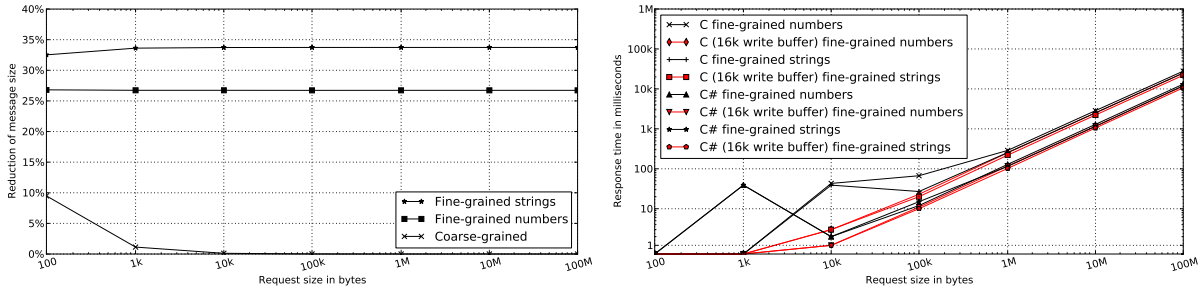
Figure 5: Creo and Thrift service response time (in milliseconds) for fine-grained messages. Axes logarithmic.



(a) Fine-grained number-resolved data. On average Creo shows improvements of factors 135.69 (SOAP), 140.93 (REST), 6.97 (RMI), and 5.04 (JOS) in service response time.

(b) Coarse-grained data. On average Creo shows improvements of factors 7.66 (SOAP), 11.99 (REST), 0.83 (RMI), and 0.83 (JOS) in service response time.

Figure 6: Service response time (in milliseconds) for Java-based tools. Axes logarithmic.



(a) Reduction of service invocation request size for TCompactProtocol compared to TBinaryProtocol.

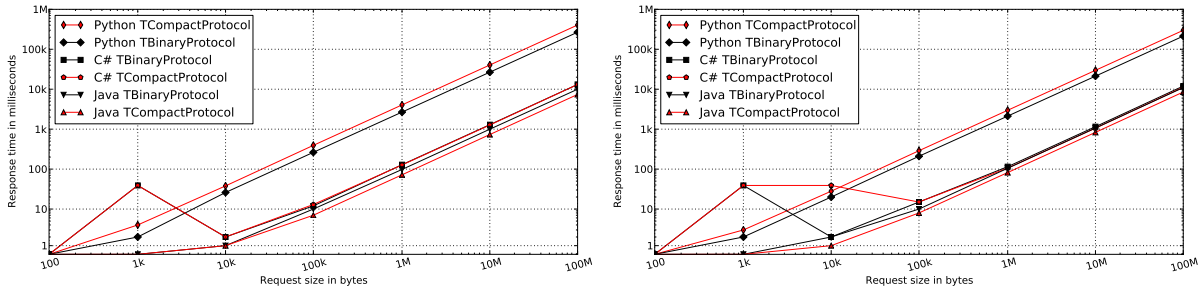
(b) Response time of Thrift's C and C# clients when using 16kB write buffers compared to using the default write buffers.

Figure 7: Transmission overhead in Thrift protocols and buffer alignment issues. Axes logarithmic.

lel invocation tests.

Results from response time tests are visualized in figures 5 and 6. Figure 5 illustrates comparison of the response time of Creo and Thrift services. On average, Creo improves on the response time performance of Thrift for fine-grained data on average of factors 1.87 to 4.77 for C#, Java, and Python clients. Compared to Thrift C clients, Creo shows improvements of factors 13.42 to 19.76. However, for coarse-grained data (unstructured binary data, not illustrated in graphs), Thrift service response times are on average 16% (C), 26% (C#), 27% (Java), and 32% (Python) lower than that of Creo (performance averages calculated for request sizes of 1MB, 10MB, and 100MB). The higher response times of Creo for coarse-grained data are attributed to the use of asynchronous message queues and immutable data structures on the service side, which cause redundant data replications in message transmission.

When comparing the response time of Creo to that of other Java-based tools (illustrated in Figure 6), we note performance improvements of at least factor 4.91 for fine-grained data, and comparative performance for coarse-grained data. As expected from analy-



(a) Number-resolved data. On average TCompactProtocol shows improvements of factors 1.2% (C#), 26% (Java), and -51% (Python) in service response time.

(b) String-resolved data. On average TCompactProtocol shows improvements of factors 7.4% (C#), 22% (Java), and -40% (Python) in service response time.

Figure 8: Response time performance of Thrift protocols. Axes logarithmic.

sis of serialization and transmission overhead, REST and SOAP web services exhibit response time performance degradations from the use of text-based representation formats and associated serializations.

5.5 Thrift Protocols

As mentioned, we use Thrift's TBinaryProtocol in tests as it is supported in all client languages. However, for selected languages, Thrift also supports the TCompactProtocol that in theory provides more efficient representation of data. To ensure fair comparison in tests, we here evaluate the use of this variable-length encoding scheme protocol. As can be seen in Figure 7a, Thrift's TCompactProtocol reduces Thrift transmission overhead of ca 27% (number-resolved data) and 34% (string-resolved data) in tests using fine-grained data. The greater reduction for string-resolved data stems from all test data blocks containing short strings (4 or 10 characters), causing string lengths to be serializable in a single byte. The variable-length encoding scheme has little effect on unstructured (coarse-grained) binary data, but shows an improvement for small messages as the protocol contains less metadata.

In tests, we note oscillations in the performance of Thrift's C and C# clients for data sizes of 1kB and 10kB (see Figure 5). After analysis we speculate that these effects arise due to buffer (size) alignment issues in tests. To investigate this, we evaluate the performance of the same clients with altered buffer sizes, and note (as illustrated in Figure 7b) that the effects can be alleviated using larger (16kB) message transmission buffers.

Finally we evaluate the service response time of Thrift's two binary protocols to investigate the potential impact of Thrift's variable-length encoding scheme on our tests. As illustrated in Figure 8, the TCompactProtocol results in response time improvements of 1.2% to 26% for C and C# clients, and performance degradations of 40% to 51% for Python clients. From these measurements we conclude that use of the TCompactProtocol would not significantly impact the findings of the evaluation.

6 CONCLUSIONS

In this work we investigate an approach to service-based software development and present a toolkit for reduction of complexity in service development and distributed component integration. The architecture of the toolkit is designed to be modular and extensible, and places focus on transparency and reduction of complexity. To reduce learning requirements, the toolkit employs a service description language based on the syntax and type interpretations of the well-known Java language. The service description language defines a set of primitive types and mechanisms for aggregation of types in arrays and structs.

The toolkit supports generation of code for construction of Java-based services as well as service clients in Java, C, C#, and Python. The toolkit uses the same code generation pattern for all languages, which defines immutable types that directly wrap the aggregation patterns defined in service descriptions. For transparency, the service communication protocol serializes data in the order and types defined in the service description language. A performance evaluation quantifying toolkit performance (in terms of overhead and response time) against Java Object Serialization, Java RMI, SOAP web services, REST web services, and Apache Thrift is presented. Toolkit performance is found to be comparable to or improve upon the performance of the alternative techniques.

ACKNOWLEDGEMENTS

The authors acknowledge Mikael Öhman, Sebastian Gröhn, and Anders Häggström for work related to the project. This work is done in collaboration with the High Performance Computing Center North (HPC2N) and is funded by the Swedish Government's strategic research project eSSENCE and the Swedish Research Council (VR) under contract number C0590801 for the project Cloud Control.

REFERENCES

Apache (2005). Apache Web Services Project - Axis2, http://ws.apache.org/axis2, February 2014.

Apache (2009). Apache Avro, http://avro.apache.org/, February 2014.

Apache (2010). Apache Thrift, http://thrift.apache.org/, February 2014.

Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al. (2010). A view of cloud computing. *Communications of the ACM*, 53(4):50–58.

Berl, A., Gelenbe, E., Di Girolamo, M., Giuliani, G., De Meer, H., Dang, M. Q., and Pentikousis, K. (2010). Energy-efficient cloud computing. *The Computer Journal*, 53(7):1045–1051.

Birrell, A. D. and Nelson, B. J. (1984). Implementing remote procedure calls. *ACM Transactions on Computer Systems (TOCS)*, 2(1):39–59.

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California.

Furuhashi, S. (2011). MessagePack, https://github.com/ msgpack/msgpack/blob/master/spec.md, February 2014.

Google (2008). https://developers.google.com/protocol-buffers/, February 2014.

Govindaraju, M., Slominski, A., Chiu, K., Liu, P., Van Engelen, R., and Lewis, M. J. (2004). Toward characterizing the performance of soap toolkits. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 365–372. IEEE.

Jackson (2009). https://github.com/FasterXML/jackson, February 2014.

Kephart, J. O. and Chess, D. M. (2003). The Vision of Autonomic Computing. *Computer*, 36:41–50.

Kryo (2009). https://github.com/EsotericSoftware/kryo, February 2014.

Mackey, A. (2010). Windows communication foundation. In *Introducing. NET 4.0*, pages 159–173. Springer.

MongoDB Inc. (2007). BSON, http://http://bsonspec.org, February 2014.

Oracle (2005). Java Object Serialization, http://docs.oracle.com/javase/7/docs/platform/ serialization/spec/serialTOC.html, February 2014.

Östberg, P.-O. and Elmroth, E. (2011). Increasing Flexibility and Abstracting Complexity in Service-Based Grid and Cloud Software. In F. Leymann, I. I., van Sinderen, M., and Shishkov, B., editors, *Proceedings of CLOSER 2011 - International Conference on Cloud Computing and Services Science*, pages 240–249. SciTePress.

Östberg, P.-O., Espling, D., and Elmroth, E. (2013). Decentralized scalable fairshare scheduling. *Future Generation Computer Systems - The International Journal of Grid Computing and eScience*, 29:130–143.

Östberg, P.-O., Hellander, A., Drawert, B., Elmroth, E., Holmgren, S., and Petzold, L. (2012). Reducing complexity in management of escience computations. In *Proceedings of CCGrid 2012 - The 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 845–852.

Pautasso, C., Zimmermann, O., and Leymann, F. (2008). Restful web services vs. big web services: making the right architectural decision. In *Proceedings of the 17th international conference on World Wide Web*, pages 805–814. ACM.

Restlet (2013). Restlet Framework, http://restlet.org, February 2014.

Slee, M., Agarwal, A., and Kwiatkowski, M. (2007). Thrift: Scalable cross-language services implementation. *Facebook White Paper*, 5.

Van Deursen, A., Klint, P., and Visser, J. (2000). Domain-specific languages: An annotated bibliography. *Sigplan Notices*, 35(6):26–36.

Vinoski, S. (1993). Distributed object computing with corba. *C++ Report*, 5(6):32–38.

Walker, E. (2009). The real cost of a cpu hour. *Computer*, 42(4):35–41.

Wollrath, A., Riggs, R., and Waldo, J. (1996). A distributed object model for the java system. *Computing Systems*, 9:265–290.

Discovering Secure Service Compositions

Luca Pino[1], George Spanoudakis[1], Andreas Fuchs[2] and Sigrid Gürgens[2]
[1]*School of Informatics, City University London, London, U.K.*
[2]*Fraunhofer Institute for Secure Information Technology, Darmstadt, Germany*
{luca.pino.1, g.e.spanoudakis}@city.ac.uk, {andreas.fuchs, sigrid.guergens}@sit.fraunhofer.de

Keywords: Software Services, Secure Service Compositions, Security Certificates.

Abstract: Security is an important concern for service based systems, i.e., systems that are composed of autonomous and distributed software services. This is because the overall security of such systems depends on the security of the individual services they deploy and, hence, it is difficult to assess especially in cases where the latter services must be discovered and composed dynamically. This paper presents a novel approach for discovering secure compositions of software services. This approach is based on secure service orchestration patterns, which have been proven to provide certain security properties and can, therefore, be used to generate service compositions that are guaranteed to satisfy these properties by construction. The paper lays the foundations of the secure service orchestration patterns, and presents an algorithm that uses the patterns to generate secure service compositions and a tool realising our entire approach.

1 INTRODUCTION

The security of service based systems (SBS), i.e., systems that make use of distributed and possibly dynamically assembled software services, has been a critical concern for both the users and providers of such systems (Raman et al., 2002; Majithia et al., 2004; Anisetti et al., 2013). This is because the security of an SBS depends on the security of the individual services that it deploys, in complex ways that depend not only on the particular security properties of concern but also on the exact way in which these services are composed to form the SBS.

Consider, for example, the case where the property required of an SBS is that the integrity of any data D, which are passed to it by an external client, will not be compromised by any of its constituent services that receive D. The assessment of this property requires knowledge of the exact services that constitute the SBS, the exact form of the composition of these services and the data flows between them, and a guarantee that each of the constituent services of SBS that receives D will preserve its integrity. Such assessments of security are required both during the design of an SBS and at runtime in cases where one of its constituent services S needs to be replaced and, due to the absence of any individual service matching it, a composition of services must be built to replace S. Whilst the construction of service compositions that satisfy functional and quality properties has received considerable attention in the literature (e.g., Aggarwal et al., 2004; Dustdar et al., 2005; Tan et al. 2009; Alrifai et al., 2012)., the construction of secure service compositions is not adequately supported by existing research.

In this paper, we present an approach for discovering compositions of services, which are guaranteed to satisfy certain security properties. Our approach is based on the application of *SEcure Service Orchestration patterns* (SESO patterns). SESO patterns specify primitive service orchestrations, which are proven to have particular security properties, if the constituent services of the orchestration satisfy other security properties. A SESO pattern specifies the order of the execution of its constituent services (e.g., sequential, parallel execution) and the data flows between them. It also specifies rules, which dictate the security properties that the constituent services of the orchestration must have for the orchestration to satisfy another security property as a whole. These rules express security property relations of the form *IF P THEN* $\wedge_{i=1,\ldots,n} P_i$ where P is a security property that is required of the service orchestration as a whole and P_i are security properties of the constituent services, which must be satisfied for P to be guaranteed. The security property relations expressed by the rules are formally proven. The constituent services of a SESO

pattern are abstract "placeholder" services that need to be instantiated by concrete services when the pattern is instantiated.

When a constituent service S of an SBS needs to be replaced at runtime and no single alternative service S' satisfying exactly the same security properties as S can be found, SESO patterns can be applied to discover compositions of other services that have exactly the same security properties as S and could replace it within SBS. SESO patterns determine the criteria (security, interface and functional) that should be satisfied by the services that could instantiate the orchestration specified by them. These criteria are used to drive a discovery process whose goal is to instantiate the pattern. If this discovery/pattern instantiation process is successful, i.e., different combinations of services that satisfy the required criteria and fit with the orchestration structure of the pattern can be discovered, any composition of services which is built from the pattern is guaranteed to have the required overall security property by-construction.

An earlier account of our approach has been given in (Pino and Spanoudakis 2012a; Pino and Spanoudakis 2012b). In this paper, we present the method that underpins the proof of security properties in SESO patterns, show examples of concrete proofs of security properties for specific SESO patterns, and present an amended version of the original composition algorithm that makes use of coarse-grained service workflows in the composition process in order to find service compositions that are not only secure but also functionally relevant to the service that is needed. In addition, we describe a tool that implements our approach.

The rest of this paper is organized as follows. Section 2 presents an overview of our approach. Section 3 discusses the validation of the security of primitive service orchestration patterns and provides examples of proofs of security properties for some of these patterns. Section 4 discusses the encoding of secure service orchestration patterns. Section 5 presents the new pattern driven secure service composition algorithm. Section 6 provides an overview of the tool that we have developed to implement our approach. Finally, Section 7 overviews related work and Section 8 provides conclusions and directions for future work.

2 OVERVIEW

The service composition approach that we present in this paper is part of a general framework developed at City University to support runtime service discovery (Zisman et al., 2013). This framework supports service discovery driven by queries expressed in an XML based query language, called *SerDiQueL*, which supports the specification of interface, behavioural and quality discovery criteria. The execution of queries can be reactive or proactive. In reactive execution, the SBS submits a query to the framework and gets back any services matching the query that the latter can find. In proactive execution, the SBS submits to the framework queries that are executed in parallel, to find potential replacement services that could be used if needed, without the need to initiate and wait for the results of the discovery process at this point (Zisman et al., 2013).

To take into account service security requirements as part of the service discovery process, we have extended the above framework in two ways: (i) we have extended SerDiQueL to enable the specification of the security properties that are required of individual services, as querying conditions, and (ii) we have developed a composition module supporting the construction of possible compositions of services that could replace a given service in an SBS in cases where a query cannot find any single replacement service, based on the approach that we present in this paper. A detailed description of the extended version of SerDiQueL (called *A-SerDiQueL*) that is used for (i) and (ii) is beyond the scope of this paper and can be found in (Spanoudakis et al., 2011). In this paper, we focus on the process of searching for and constructing secure service compositions. The key problems during the composition process are to ensure that the constructed composition of services: (a) provides the functionality of the service that it should replace, and (b) satisfies the security properties required of this service.

To address (a), our approach uses abstract service workflows. These workflows express service coordination processes that realize known business processes through the use of software services with fixed interfaces. Such workflows are available for specific application domains such as telecom services (IBM BPM Industry Packs), logistics (RosettaNet), and are often available as part of SOA architecting and realization platforms (e.g., IBM WebSphere). Service workflows are encoded in an XML based language that represents the interfaces, and the control and data flow between the workflow's composing activities.

To address (b), we are using the SESO patterns. These patterns are based on primitive service

orchestrations that have been proposed in the literature (e.g., sequential and parallel service execution) but augment them by specifying concrete security properties P_1, ..., P_n that must be provided by the individual services that instantiate the pattern for the overall orchestration to satisfy a required security property P_0. The derivation of these security properties is based on rules that encode formally proven relations between the security properties of the individual placeholder services of the pattern and the security property required of the entire service orchestration represented by the pattern. Once derived through the application of rules, the security properties required of the individual partner services of the orchestration are expressed as queries in A-SerDiQueL. These queries are then executed to identify concrete services with the required security properties, which could instantiate the placeholder services of the pattern. If such services are found the pattern is instantiated. The pattern instantiation process is gradual and, if it is completed successfully, a new concrete and executable service composition that satisfies the overall security property guaranteed by the pattern is generated.

A key element of our approach is the formal validation of the relations between the security properties of the individual placeholder services of a SESO pattern and the security property of the entire composition expressed by the pattern. The validation of such relations is discussed in the next section.

3 VALIDATING SECURE SERVICE COMPOSITIONS

The task of formally validating the security of a service composition requires a three-step approach. It starts with a formal model of the service to be replaced and the formal models of the services to be composed. Firstly, the service composition is represented in terms of a formal model derived from the models of the individual services by applying a set of formal construction rules. These rules project the respective security properties of each of the composed services as well as the targeted property of the service to be replaced into the composed system. Secondly, additional properties are added to the composed system regarding the behaviour of the orchestration engine, i.e., the primitive service orchestration pattern. Finally, the desired property is verified using the properties of the composed services and the orchestrator.

For the formal system representation and validation of security properties we utilize the Security Modeling Framework SeMF developed by Fraunhofer SIT (Gürgens et al., 2005b). In SeMF, a system specification is composed of a set $\mathbb{P}$ of agents and a set $\sum$ of actions, $\sum/P$ denoting the actions of agent P, and other system specifics that are not needed in this paper and are thus omitted. The behaviour B of a discrete system Sys can then be formally described by the set of its possible sequences of actions. Security properties are defined in terms of such a system specification. Relations between different formal models of systems are partially ordered with respect to different levels of abstraction. Formally, abstractions are described by so called alphabetic language homomorphisms that map action sequences of a finer abstraction level to action sequences of a more abstract level while respecting concatenation of actions. Language homomorphisms satisfying specific conditions are proven to preserve specific security properties, the conditions depending on the respective security property. A detailed account of SeMF is beyond the scope of this paper can be found in (Fuchs and Gürgens, 2011; Fuchs et al., 2011; Gürgens et al., 2005a; Gürgens et al., 2002) for.

Based on the representations of each of the service systems in the composition, we present a general construction rule using homomorphisms that map the service composition onto the individual services by preserving the individual services' security properties. This allows us to deduce the respective security properties to be satisfied by the composition. The different SESO patterns are translated into behaviour of the orchestrator regarding the invocation of the respective services. This includes functional and security related property statements. Based on this information it is possible to deduce the overall security properties of the composition system and validate whether they meet the expected results. In the next three sections, we illustrate our approach by exemplarily proving a specific data integrity property. The formal representation of services, composition and security properties is given in terms of generic agents and actions that are later used by the SESO patterns for instantiation towards concrete services and security properties. While our example is a very simple one, our approach can handle more complex service models, e.g. involving global agents (unique to all services), or service specific agents (e.g. backend storage) as well as various different orchestrations patterns, proving different instantiations of various security properties regarding integrity and confidentiality (Pino et al., 2012).

3.1 Formal Representation of Generic Service Composition

In the following, we denote the system model of the service S^0 to be replaced by a composition by Sys^0, the system models of the services S^1 and S^2 to be composed by Sys^1 and Sys^2, respectively, and the composition system by Sys^c. The sets of agents and actions are denoted analogously (i.e. by $\mathbb{P}^i$, $\sum^i$, for i=0, 1, 2). We then view the systems Sys^0, Sys^1 and Sys^2 as homomorphic images of the composed system Sys^c.

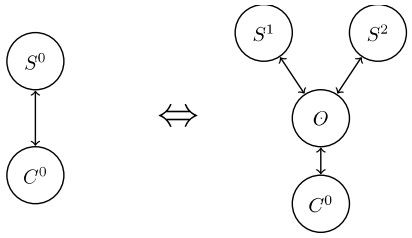


Figure 1: Service Composition.

The principal idea of substituting a service by a service composition is depicted in Figure 1: we assume services S^1 and S^2 to act independently of (i.e., not to invoke) each other. Thus we utilize an orchestration engine O for their composition that takes the role of both the clients C^1 and C^2 of Sys^1 and Sys^2 respectively, as well as the role of the service S^0 in Sys^0 to be replaced. We formalize this by using a generic renaming function $r_{P\to Q}: \sum \to \sum_{r_{P\to Q}}$ that replaces all occurrences of agent P in an action by Q. Based on this function, we define functions $r^i : \sum^i \to \sum^c$ (i = 0, 1, 2) as follows:

$$r^0(a) := r_{S^0\to O}(a) \quad \text{if } a \in \sum^0_{/C^0} \cup \sum^0_{/S^0}$$

$$r^j(a) := r_{C^j\to O}(a) \quad \text{if } a \in \sum^j_{/S^j} \cup \sum^j_{/C^j}$$

(j = 1, 2). The resulting set $\sum^c$ of actions of the composed system is then as follows:

$$\sum^c = r^0(\sum^0_{/C^0} \cup \sum^0_{/S^0}) \cup r^1(\sum^1) \cup r^2(\sum^2) \cup \sum^c_{/O}$$

$\sum^c_{/O}$ represents additional actions taken by the orchestration engine beyond the communication with client and services. These actions depend on the specific orchestration pattern used and will be discussed in the next section. Since the functions r^i are injective we can now use their inverse image in order to define the homomorphisms that map the composition system onto the abstract systems: each homomorphism h^i abstracts $\sum^c$ to $\sum^i$. Regarding the actions corresponding to those in $\sum^i$, h^i is simply the inverse of r^i, and all other actions are mapped onto the empty word. Hence for i = 0, 1, 2, we define $h^i : \sum^c \to \sum^i$ as follows:

$$h^i(a) = \begin{cases} a' & \text{if } \exists a' \in \sum^i : r^i(a') = a \\ \varepsilon & \text{else} \end{cases}$$

These homomorphisms serve as a means to relate not only the models of the individual systems to the composition model but also to relate - under certain conditions - their security properties. A homomorphism that fulfils certain conditions "transports" a security property from an abstract system to the concrete one, i.e. if the conditions are satisfied and the property holds in the abstract system, the corresponding property will also hold in the concrete system. Thus, the homomorphism *preserves* the property. The conditions that must be satisfied depend on the property in question; see (Gürgens et al., 2005a; Gürgens et al., 2002) for example. We use this approach to prove specific security properties for a composition of services based on the security properties of these services.

3.2 Formally Representing Sequential Composition

The actions of the systems are constructed from the service operations op_0, op_1, and op_2 as prefix, followed by one of the suffixes *IS*, *IR*, *OS*, *OR* to represent *InputSend*, *InputReceive*, *OutputSend*, *OutputReceive*, respectively. This results in the following agent and action sets:

$$\mathbb{P}^i \supseteq \{C^i, S^i\}, \sum^i \supseteq \left\{ \begin{array}{c} op_i\text{-}IS(C^i, S^i, data_i), \\ op_i\text{-}IR(S^i, C^i, data_i), \\ op_i\text{-}OS(S^i, C^i, f_i(data_i)), \\ op_i\text{-}OR(C^i, S^i, f_i(data_i)) \end{array} \right\}$$

In our simple example of a sequential composition pattern, the orchestrator forwards $data_0$ received from C^0 to S^1 which returns $f_1(data_0)$. These data are then forwarded by the orchestrator to S^2 who returns $f_2(f_1(data_0))$ which the orchestrator finally returns to the client. In a more complex scenario the orchestrator can for example alter (e.g., split) the client data and combine the output of S^1 with some data resulting from the client's input and send this to S^2. A proof for this more complex construction is achievable analogously to the one presented below.

The agent and action sets of the composition are constructed as specified in the previous section, using the functions r^0, r^1 and r^2. Function r^0 for example maps action op_0-$IS(C^0, S^0, data_0)$ onto op_0-$IS(C^0, O, data_0)$, hence $h^0(op_0\text{-}IS(C^0, O, data_0)) = op_0\text{-}IS(C^0, S^0, data_0)$, while $h^0(op_2\text{-}OR(O, S^2,$

$f_2(data_2))) = h^0(r^2(op_2\text{-}OR(C^2, S^2, f_2(data_2)))) = \varepsilon$., with $data_1 := data_0$ and $data_2 := f_1(data_1)$.

3.3 Validation of Integrity Preserving Compositions

Exemplarily, we will now prove that a specific data integrity property of S^0 is provided by the orchestration specified above. The definition of (data) integrity that we assume in our example is taken from RFC4949: "The property that data has not been changed, destroyed, or lost in an unauthorized or accidental manner." (Shirey, 2007). In SeMF, this property is expressed by the concept of *precedence*: *pre(a,b)* holds if all sequences of actions $\omega \in B$ that contain action b also contain action a. Obviously, precedence is transitive (we omit the trivial proof). Further, precedence is preserved by any homomorphism (Fuchs and Gürgens, 2011).

Let us now assume that service S^0 provides the integrity property that whenever the client receives $f_0(data_0)$ from the service, the client has sent $data_0$ to this service before:

P1' $pre(op_0\text{-}IS(C^0, S^0, data_0), op_0\text{-}OR(C^0, S^0, f_0(data_0)))$

As explained above, precedence is preserved by h^0 (as constructed in Section 3.1). Hence the corresponding property of the composition is (assuming $f_0 = f_2 \circ f_1$):

P1 $pre(op_0\text{-}IS(C^0, O, data_0), op_0\text{-}OR(C^0, O, f_2(f_1(data_0))))$

For our proof, we assume that the services Sys^1 and Sys^2 provide the properties:

P2' $pre(op_1\text{-}IS(C^1, S^1, data_1), op_1\text{-}OR(C^1, S^1, f_1(data_1)))$
P3' $pre(op_2\text{-}IS(C^2, S^2, data_2), op_2\text{-}OR(C^2, S^2, f_2(data_2)))$

The homomorphisms h^1 and h^2 as constructed in Section 3.1 preserve these precedence properties. Accordingly, the corresponding properties in Sys^c are:

P2 $pre(op_1\text{-}IS(O, S^1, data_0), op_1\text{-}OR(O, S^1, f_1(data_0)))$
P3 $pre(op_2\text{-}IS(O, S^2, f_1(data_0)), op_2\text{-}OR(O, S^2, f_2(f_1(data_0))))$

In addition, the orchestrator must act according to the pattern (as specified in Section 3.2), i.e., satisfy the following properties:

P4 $pre(op_0\text{-}IS(C^0, O, data), op_1\text{-}IS(O, S^1, data))$
P5 $pre(op_1\text{-}OR(O, S^1, data), op_2\text{-}IS(O, S^2, data))$
P6 $pre(op_2\text{-}OS(O, C^0, f_2(f_1(data_0))), op_2\text{-}OR(C^0, O, f_2(f_1(data_0))))$

Proof. By transitivity of precedence, from properties P2 to P6 we can conclude that property P1 holds.

The above proof is almost trivial but shows the principle of our approach. In (Pino et al., 2012) we have proven more complex integrity properties involving actions of global agents being invoked by either S^1 or S^2, as well as several confidentiality properties. All proofs use the approach presented in this paper: (i) deriving the formal model of the service composition from the formal models of the individual services, (ii) relating these models by using property preserving homomorphisms and thus representing the individual services' security properties in terms of the composition model, and (iii) using appropriate security properties to be satisfied by the orchestrator. Whilst we assume the orchestrator to behave correctly and hence to satisfy these additional properties, the security properties we assume for the individual services of the composition are translated into inference rules, which are then used in order to construct a service composition. It should also be noted that the proofs of security properties for specific SESO patterns need to be constructed offline and encoded in the patterns as rules, as we discuss in Sect. 4 below. At runtime, the rules encoded in specific pattern are used to deduce the security properties that need to be satisfied by the candidate services that can instantiate the pattern.

4 SECURE SERVICE ORCHESTRATION PATTERNS

Proofs of security properties, like the one that we discussed in Section 3, form the basis of SESO patterns in our approach. More specifically, an SESO pattern encodes: (a) a primitive orchestration describing the order of the execution and the data flow between placeholder services, and (b) the implications between the security properties of these services and the security property of the whole orchestration. The placeholder services within a primitive orchestration can be atomic activities (i.e., abstract partner services) or other patterns. The implications in (b) are of the form:

"IF P is a primitive orchestration with placeholders $S_1, \ldots, S_n$ and ρ^P is a security property required for P THEN ρ^P

can be guaranteed if each S_i in P satisfies a set of security properties ρ_j (j =1, …, m_i)".

These implications reflect proofs of security properties, developed based on the approach discussed in Sect. 3. They are encoded as inference rules and used during the composition process to infer the security properties that would be required of the placeholders of a pattern P for it to satisfy ρ^P. The benefit of encoding proven implications as inference rules is that there is no need to reason from first-principles when attempting to construct compositions of services, based on SESO patterns.

To be more specific, SESO patterns and implications of the above form are encoded as Drools production rules (Drools). Drools is a rule-based reasoning system supporting reasoning driven by production rules. Production rules in Drools are used to derive information from data facts stored in a Knowledge Base (KB). A production rule in Drools has the general form: *when* <*conditions*> *then* <*actions*>. When a rule is applied, the rule engine of Drools checks, through pattern matching, whether the conditions of the rule match with the facts in the KB and, if they do, it executes actions of the rule. This execution can update the contents of the KB by adding or deleting facts in it. The reasoning process of Drools is based on the Rete algorithm a pattern-matching algorithm that is known to scale well for large sets of data facts and rules (Forgy, 1982);. The latter property of Drools is the main reason for selecting it to represent and reason with SESO patterns in our approach.

Table 1 shows the encoding of integrity in the sequential orchestration pattern that was presented in Section 3.3 as a Drools rule. In particular our rule uses the following definition of integrity:

Definition 2. *Integrity(S, x, y) = pre*(op_0-*IS*(C^0, *S*, *x*), op_0-*OR*(0^0, *S*, y))

Using such more abstract security properties in the rules avoids the need to encode in the rule the formalism that the proof is based on. This makes it also possible to use SESO patterns proven through different formalisms in our approach.

Returning to the rule in Table 1, Lines 3-5 describe the primitive orchestration that the security property refers to. More specifically, the rule can be applied when a sequential pattern (`$P`) with two placeholders, i.e., activity `$S1` followed by activity `$S2`, is encountered. The rule defines the parameters of these activities: `$S1` has an input parameter `$d` and an output parameter `$f1d`, and `$S2` has an input parameter `$f1d` and an output parameter `$f2f1d`, as shown in Table 1. Line 6 describes the original security requirement requested on the composition pattern `$rhoP`, i.e. integrity on the pattern `$P` of its data `$d` and `$f2f1d`. This requirement is equivalent to the precedence property P1 presented in Section 3.3. Lines 8-9 (i.e., the `then` part of the rule) specify the security properties that are required of the activities of the pattern in order to guarantee `$rhoP`, namely: (i) integrity on the input (`$d`) and output (`$f1d`) of `$S1`, as stated by the precedence property P2, and (ii) integrity on the input (`$f1d`) and output (`$f2f1d`) of `$S2`, as required from P3. Additionally, we assume the framework executing the orchestration to satisfy properties P4–P6, hence these need not be mentioned in the rule. Finally, according to the rule, once the original requirement `$rhoP` is guaranteed by the new ones, it can be removed from the KB.

Similar encodings of other SESO patterns have been expressed using this approach but cannot be discussed due to space limitations. SESO pattern encoding rules, like the one presented above, are used during the composition process to infer the security properties that are required of the concrete services that may instantiate the placeholder services in a workflow. This process is discussed next.

Table 1: Integrity Rule for Sequential SESO Pattern.

```
1:  rule "Integrity - Sequential Orchestration"
2:   when
3:     $S1 := Activity($d := inputs, $f1d := outputs)
4:     $S2 := Activity($f1d := inputs, $f2f1d := outputs)
5:     $P := Sequential($S1 := activ1, $S2 := activ2)

6:     $rhoP : Integrity($P := subject, $d := inputs, $f2f1d := outputs)
7:   then
8:     insert(new Integrity($S1, $d, $f1d));
9:     insert(new Integrity($S2, $f1d, $f2f1d));
10:    retract($rhoP);
11: end
```

5 SESO PATTERN DRIVEN SERVICE COMPOSITION

The service composition process is carried out according to the algorithm shown in Table 2. This algorithm is invoked when an SBS service needs to be replaced but the service discovery query specified for it cannot identify any single service matching its conditions.

In such cases, the structural part of the query, which defines the operations that a service should have and the data types of the parameters of these operations, is used to retrieve from the repository of the discovery framework abstract workflows that can provide the required service functionality. An abstract workflow represents a coarse grained orchestration of activities, which collectively offer a specific functionality, and is exposed as a composite service. Such workflows are fairly common (Carminati et al., 2006; Medjahed et al., 2003) and result from the generation of reference process models in specific domains as in (RosettaNet; IBM BPM Industry Packs). The activities of an abstract workflow are orchestrated through a process consisting of the primitive orchestrations that underpin the security patterns, as discussed in Section 4. If such workflows are found the generation of a service composition is attempted by trying to instantiate each abstract workflow.

As shown in Table 2, initially, the algorithm identifies the abstract workflows that could be potentially used to generate a composition that can provide the operations of the required service (see STRUCTURALMATCH function in line 3). This is based on the execution of the query associated with the service to be replaced (Q_S). If such workflows are found, the algorithm continues by starting a process of instantiating the activities of each of the found workflows with services.

The activities of the workflows are instantiated progressively, by investigating each workflow W in a depth-first manner. More specifically, the algorithm takes the first unassigned activity A in W (in the control flow order) and builds a query Q_A based on the workflow specification and the discovery query Q_S. In particular, the structural part of Q_A is taken from the description of A in the abstract workflow. The security conditions in Q_A are generated through the procedure SECURITYCONDITIONS(Q_S,W). This procedure infers the security conditions for A based on the Drools rules that encode the SESO patterns detected within the current workflow. More specifically, all the information about the workflow, its patterns, activities, security properties and requirements are put into the KB. Then the rules that

Table 2: Service Composition Algorithm.

```
Require: Q_S - query for the required service
Ensure: ResultSet - set of instantiated workflows
1:  procedure SERVICECOMPOSITION(Q_S)
2:    for all abstract workflows AW in the repository do
3:      if STRUCTURALMATCH(Q_S, AW) == true then
4:        Put a copy of AW in WStack
5:      end if
6:    end for
7:    while there are more workflows in WStack do
8:      Get the first workflow W in the WStack
9:      Pop the first unassigned activity A from W
10:     Extract the structural query Q_A for A from W
11:     SecCond := SECURITYCONDITIONS(Q_S, W)
12:     Add to Q_A the security conditions SecCond
13:     Res := SERVICEDISCOVERY(Q_A)
14:     for all services S* in Res do
15:       W_S* := W[A/S*]                    //i.e. substitute S* for A in W
16:       if exists an unassigned activity in W_S* then
17:         Push W_S* in WStack
18:       else
19:         Add W_S* to ResultSet
20:       end if
21:     end for
22:   end while
23:   return ResultSet
24: end procedure
```

represent the detected SESO patterns are fired (i.e. applied), propagating the requirements through the workflow. The generated requirements for the unassigned activity are then retrieved and converted to query conditions. The propagation of security requirements is possible thanks to the fact that each workflow can be seen as a recursive application of primitive orchestrations.

Figure 2 shows the order of propagation through the use of the rules, on a workflow shown in (c). A security requirement ρ^S is initially given for a service *S* (Figure 2 (a)). The first rule that will be fired by Drools is the one for the outermost pattern of the workflow: a choice pattern (i.e., the *if-then-else* primitive orchestration in Figure 2 (b)). The security requirement is then propagated by the relevant rule (if such a rule exists) to the placeholders *A* and *B* returning the requirements ρ^{A1}, …, ρ^{An} and ρ^{B1}, …, ρ^{Bm} (with $n, m \geq 0$ and $n+m \geq 1$). For each security requirement ρ^{Ai} (with $i=1, ..., n$), a rule is fired to propagate the requirement to the sequential pattern that instantiates *A* (Figure 2 (c)). This process generates the security requirements for placeholders *C* and *D*.

If a security requirement cannot be propagated to the atomic activity level (e.g., no rules are defined for the given pattern or security property) then Drools returns an error state to point out that a security requirement cannot be guaranteed by the existing set of rules. This ensures that no security requirements are ignored.

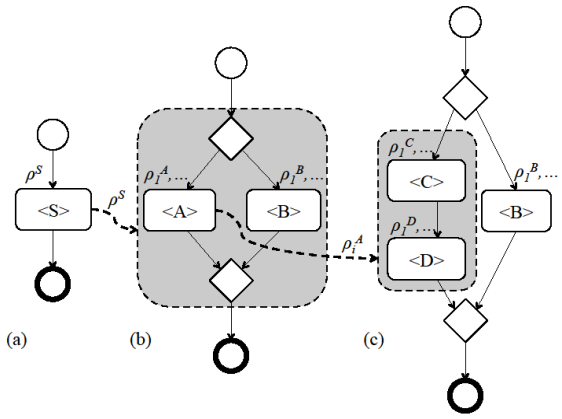


Figure 2: Recursive application of secure service orchestration patterns.

After constructing Q_A, the query is executed by the runtime discovery framework in (Zisman et al., 2012) to identify a list of candidate services for Q_A. The candidate services in this list (if any) are then used to instantiate the activity A in W. Note that the composition algorithm implements a depth-first search in the composition process in order to explore fully the instantiation of a particular activity within a pattern before considering other activities, as this is expected to spot dead-ends sooner than a breadth-first search.

5.1 Example

As an example of applying the algorithm in Table 2, consider a *Stock Broker* SBS that uses an operation *GetStockQuote* from a service *StockQuote* to obtain price quotations for given stocks. *GetStockQuote* takes as input a string *Symbol* identifying a stock and returns the current value of that stock in USD.

Suppose that the Stock Broker SBS has a security requirement regarding integrity of the input and output data of this operation, and would consider replacement services that can offer the same operation only if they have certificates confirming the satisfaction of this particular security requirement by the service. To deal with potential problems with *StockQuote* at runtime (e.g., unavailability), *Stock Broker* can subscribe a service discovery query Q_{SQ} for replacing *StockQuote* to the discovery framework and request its execution of proactive mode. Q_{SQ} should specify the functional and security properties that the potential replacement services of *StockQuote* must have. If the execution of Q_{SQ} results in discovering no single service matching it (i.e., when single service discovery fails), the service composition process is carried out. At this stage, according to the algorithm of Table 2, the framework will query the abstract workflow repository to locate workflows matching Q_{SQ}.

Suppose that this identifies an abstract workflow W_{SQ} shown in Figure 3 that matches the query. W_{SQ} contains three activities connected by two sequential patterns (see two dashed areas of workflow). The first placeholder of the outer sequence contains the activity *GetISIN*, which converts the *Symbol* identifying the Stock into the *ISIN* (another unique stock identifier). The second placeholder corresponds to the inner sequence. Within this inner sequence, the first placeholder is the activity *GetEURQuote* that returns the current stock value in EUR given the Stock *ISIN*. The second placeholder is the activity *EURtoUSD*, which converts a given amount from EUR to USD.

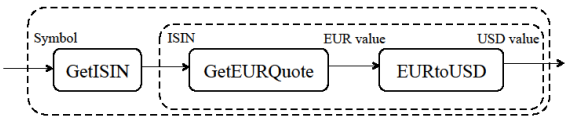


Figure 3: Abstract Workflow W_{SQ}.

The framework then infers the security properties required for each of the services that could instantiate the activities and uses them to query for such services. Initially, the rule shown in Table 1 is fired given the property required for the external sequential pattern, i.e. integrity on inputs and outputs of the workflow (i.e. *Symbol* and *USD value*). From the required security property, the rule derives two more properties: (1) integrity on inputs and outputs of *GetISIN* (i.e. *Symbol* and *ISIN*), and (2) integrity on inputs and outputs of the sequential inner pattern representing the second activity (i.e. *ISIN* and *USD value*). The second property fires again the rule and this propagates the requirement for integrity of the *ISIN* and *USD value*, resulting in the two properties: integrity on *GetEURQuote* of *ISIN* and *EUR value*, and integrity on *EURtoUSD* of *EUR value* and *USD value*.

After the application of the rules, we derive the required property for the first unassigned activity *GetISIN*, namely integrity of the input *Symbol* and the output *ISIN*. A query consisting of the interface and the security property required for *GetISIN* is then executed and the discovered services are used to instantiate the workflow. Note that in the discovery process, services are considered to satisfy the required security properties only if they have appropriate certificates asserting these properties. In a similar way, a query specifying the required interface and security property of integrity is created for the second (*GetEURQuote*) and the last activity (*EURtoUSD*). Each query is executed, and the workflow gets instantiated by the results. After the replacement service is fully composed, the service composition is published in a BPEL execution engine and its WSDL is sent to the Stock Broker SBS in order to update its bindings.

6 TOOL SUPPORT & EXPERIMENTS

To implement and test our approach, we have developed a prototype realizing the composition process and integrated it with the runtime service discovery tool described in Section 2. The prototype gives the possibility to select a service discovery query and execute it to find potential candidate services and service compositions. If alternative service compositions can be built, the alternatives are presented to the user who can select and explore the services in each of them. Figure 4 shows the results of an execution in the case of the example in Section 5.1. These include two alternative service compositions; see *GetUSDStockQuote-Wf1-0* and *GetUSDStockQuote-Wf1-1* in the *Ranking-1* panel (the appearance of the two compositions in the same line in the panel indicates that there is no ranking between these two compositions). If one of these compositions is selected, details about the service operations that have instantiated the abstract workflow activities are shown in the *Composition Details* panel. In this case, the abstract workflow with the two nested sequences of activities has been instantiated by *sequential(GetISIN, sequential(GetEURQuote, EURtoUSD))*.

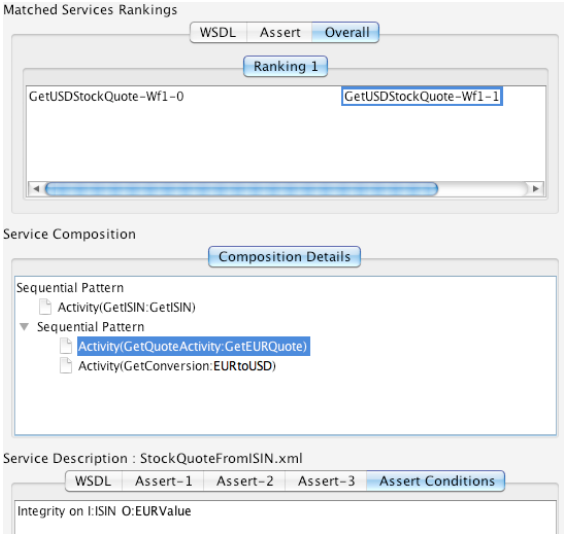


Figure 4: Screenshot of Composition tool.

Then, by selecting an activity in the workflow, the details of the service instantiating the selected activity are shown. These can be the WSDL description, the required security properties that the patterns generated for the query that was used to identify the service, and the certificates that demonstrated the satisfaction of these properties during the composition process. The bottom part of Figure 4 shows the required security properties that were used in the query for the service *GetEURQuote*.

Early performance tests of our approach have been carried out using service registries of different sizes. Table 3 shows average execution times for single service and service composition discovery obtained from using our tool on an Intel Core i3 CPU (3.06 GHz) with 4 GB RAM. The reported times are average times taken over 30 executions of a discovery query. In the experiments, we used service registries of four sizes (150, 300, 600 and 1200), 25 abstract workflows and 3 patterns.

Table 3: Execution times (in milliseconds) w.r.t. service registry size and number of generated compositions.

Registry size	150	300	600	1200
Single Service Discovery Time	194	275	355	642
Composition Discovery Time	777	2214	4943	12660
No. of generated Compositions	4	12	24	40

As shown in the table, the time required for building service compositions is considerably higher than the time required for single service discovery. The main part of this cost comes from the process of discovering the individual services to instantiate the partner links of the composition.

Although the overall composition time is high, its impact is not as significant, since as we discussed in Sect. 2 our framework can apply discovery and service composition in a proactive manner, i.e., discover possible service compositions in parallel with the operation of an SBS and use them when a service needs to be replaced. Furthermore, the cost of compositions can be reduced or kept under a given threshold by controlling the number of alternative compositions that the algorithm in Table 2 builds.

Whilst the benefits of the proactive approach have been shown in (Zisman et al., 2013) for the case of single service discovery, further experimentation is required to explore the same for the composition and assess the effect on performance of controls over the number of generated compositions.

7 RELATED WORK

The main focus of existing work in service composition is to address the problem of creating compositions that have certain functional and quality of service (QoS) property (Raman et al., 2002; Ponnekanti et al., 2002; Fujii et al., 2004; Majithia et al., 2004; Jaeger et al., 2004; Aggarwal et al., 2004; Dustdar et al., 2005; Tan et al. 2009; Alrifai et al., 2012). This work provides a foundation for functional and QoS properties but provides only basic support for addressing security properties in service composition, which is the main focus of our approach.

The problem of supporting security requirements (properties) in service composition has been a focus of work in the area of model based service composition. In this area, service compositions are modeled using formal languages and their required properties are expressed as properties on the model (Deubler et al., 2004; Dong et al., 2010; Bartoletti et al., 2005). Our approach to composition is also model based but uses model based property proofs to identify how overall security properties of compositions can be guaranteed through propagation to properties on the individual components (services) of the composition. Works in this field, however, provide proofs of additional security properties that could be used to extend the patterns used in our approach, even if they use different formalisms. An example of such proofs is given in (Mantel, 2002), which presents compositionality results related to information flows (e.g. non-interference) and that can be easily converted into SESO patterns and inference rules in our framework.

Another strand of work on automatic service composition focuses on discovering services that can guarantee given security properties (Carminati et al., 2006; Medjahed et al., 2003; Lelarge et al., 2006; Anisetti et al., 2013; Khan et al., 2012). Some of these approaches focus on specific types of security properties (Medjahed et al., 2003; Lelarge et al., 2006), whilst others (Carminati et al., 2006; Anisetti et al., 2013; Khan et al., 2012) focus on how to express and check security properties only for single partner services of a composition. In contrast, our approach can support arbitrary security properties and properties of entire service compositions.

The approaches in (Medjahed et al., 2003) and in (Khan et al., 2012) describe two ontology-based frameworks for automatic composition. The former work defines a set of metrics for selecting amongst different compositions but provides limited support for security. The latter work introduces hierarchies of security properties and mentions the possibility of using rules to reason about them but does not support the construction of secure service compositions. Lelarge et al., (2006) use planning techniques to build sequential compositions that guarantee the adoption of access control models. Carminati et al., (2006) introduce an approach to security aware service composition that matches security requirements with the external service properties. The approach presented in (Anisetti et al., 2013) focuses on the generation of test-based virtual security certificates for service compositions derived from the test-based security certificates of the external services part of the composition. The service compositions are based on templates that allow expressing security requirements on the

external services. The ideas underlining this approach can be used to extend the one presented in this paper to support the generation of virtual certificates for compositions.

The secure orchestration patterns that we use in our framework are similar to the workflow patterns in (Van Der Aalst et al., 2003), as they specify elementary workflows used to build compositions. Our patterns, however, include information not only about the control flow within the pattern but also about the data flow. They also extend these patterns with information regarding security properties to hold for the individual services in order to guarantee that their composition satisfies a required security property.

8 CONCLUSIONS

In this paper, we have presented an approach supporting the discovery of secure service compositions. Our approach is based on secure service orchestration (SESO) patterns. These patterns comprise specifications of primitive orchestrations describing the order of the execution and the data flow between placeholder services, and rules reflecting formally proven implications between the security properties of the individual placeholders and the security property of the orchestration as a whole. The formal proofs (and patterns) achieved so far cover different integrity and confidentiality properties for various forms of primitive orchestrations. The extension of our approach to cover other security properties (e.g., availability) is subject of ongoing work. During the composition process, the proven implications are used to deduce the actual properties that should be required of the individual services that may instantiate an orchestration for the orchestration as a whole to satisfy specific security properties.

In order to facilitate reasoning, SESO patterns are encoded as Drools rules. This enables the use of the Drools rule based system for inferring the required service security properties when trying to generate a service composition.

Our approach has been implemented and integrated with a generic framework supporting runtime service discovery that has been described in (Zisman et al., 2012). We are currently investigating the validity of our approach through a series of focus group evaluations. We are also conducting further performance and scalability analysis of our prototype, focusing on exploring the effect of a proactive composition generation approach and setting heuristic controls over the number of compositions generated by the algorithm.

ACKNOWLEDGEMENTS

The work reported in this paper has been partially funded by the EU F7 project ASSERT4SOA (grant no.257351).

REFERENCES

Aggarwal, R., Verma, K., Miller, J., and Milnor, W., 2004. Constraint driven web service composition in METEOR-S. In *Proc. of the IEEE International Conference on Services Computing, (SCC 2004)*, pp. 23-30.

Alrifai, M., Risse, T., and Nejdl, W., 2012. A hybrid approach for efficient Web service composition with end-to-end QoS constraints. In *ACM Transactions on the Web (TWEB)*, vol. 6, no. 2, Article 7.

Anisetti, M., Ardagna, C., and Damiani, E., 2013. Security Certification of Composite Services: A Test-Based Approach. In *Proc. of the IEEE 20th International Conference on Web Services (ICWS)*, pp. 475-482.

Bartoletti, M., Degano, P. and Ferrari, G. L., 2005. Enforcing secure service composition. In *Proc. 18th Comp. Sec. Found. Workshop (CSFW)*. IEEE Comp. Soc., pp. 211-223.

Carminati, B., Ferrari, E. and Hung, P. C. K., 2006. Security conscious web service composition. In *Proc. of the Int. Conf. on Web Serv. (ICWS)*. IEEE Comp. Soc., 489-496.

Deubler, M., Grünbauer, J., Jürjens, J. and Wimmel, G., 2004. Sound development of secure service-based systems. In *Proc. of 2nd International Conference on Service Oriented Computing (ICSOC)*. ACM, pp. 115-124.

Dong, J., Peng, T. and Zhao, Y., 2010. Automated verification of security pattern compositions. *Inf. Softw. Technol.*, vol. 52, no. 3, pp. 274-295.

Drools. [Online]. Available: http://www.jboss.org/drools/

Dustdar, S., and Schreiner, W., 2005. A survey on web services composition. *International Journal of Web and Grid Services*, vol. 1, no. 1, pp. 1-30.

Forgy, C., 1982. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligences*, vol. 19, no. 1, pp. 17-37.

Fuchs, A. and Gürgens, S., 2011. D05.1 Formal Models and Model Composition. ASSERT4SOA Project, Tech. Rep. [Online]. Available: http:// assert4soa.eu/public-deliverables/

Fuchs, A., Gürgens, S. and Rudolph, C., 2011. Formal Notions of Trust and Confidentiality - Enabling Reasoning about System Security. *Journal of Information Processing*, vol. 19, pp. 274-291.

Fujii, K., and Suda, T., 2004. Dynamic service composition using semantic information. In *Proc. of the 2nd international conference on Service oriented computing (ICSOC)*, pp. 39-48. ACM.

Gürgens, S., Ochsenschläger, P. and Rudolph, C., 2002. Authenticity and provability - a formal framework. In *Infrastr. Sec. Conf. (InfraSec)*. LNCS, vol. 2437, SV, pp. 227–245.

Gürgens, S., Ochsenschläger, P. and Rudolph, C., 2005a. Abstractions preserving parameter confidentiality. In *Europ. Symp. On Research in Computer Security (ESORICS)*. 418–437.

Gürgens, S., Ochsenschläger, P. and Rudolph, C., 2005b. On a formal framework for security properties. *International Comp. Standards & Interface Journal (CSI), Special issue on formal methods, techniques and tools for secure and reliable app.* 27(5) 457–466.

IBM BPM industry packs. [Online]. Available: http:// www-03.ibm.com/software/products/us/en/ business-process-manager-industry-packs/

Jaeger, M. C., Rojec-Goldmann, G., and Muhl, G., 2004. QoS aggregation for web service composition using workflow patterns. In *Proc. of the 8th IEEE International Enterprise distributed object computing conference, (EDOC 2004)*, pp. 149-159.

Khan, K. M., Erradi, A., Alhazbi, S. and Han, J., 2012. Security oriented service composition: A framework. In *Proc. of International Conference on Innovations in Information Technology (IIT)*, pp. 48-53.

Lelarge, M., Liu, Z. and Riabov, A.V., 2006. Automatic composition of secure workflows. In *Proc. of the Third international conference on Autonomic and Trusted Computing, (ATC)*. Berlin, SV, pp. 322-331.

Majithia, S., Walker, D. W., and Gray, W. A., 2004. A framework for automated service composition in service-oriented architectures. In *Proc. of the 1st European Semantic Web Symposium*, Lecture Notes in Computer Science, vol. 3053, pp. 269-283.

Mantel, H., 2002. On the Composition of Secure Systems. In *Proc. of the 2002 IEEE Symposium on Security and Privacy (SP2002)*. IEEE Computer Society, Washington, DC, USA, 88-.

Medjahed, B., Bouguettaya, A. and Elmagarmid, A.K., 2003. Composing web services on the semantic web. *The VLDB Journal*, vol. 12, no. 4, pp. 333-351.

Pino, L. and Spanoudakis, G., 2012a. Constructing secure service compositions with patterns. In *Services (SERVICES), 2012 IEEE Eighth World Congress on*. IEEE, pp. 184-191.

Pino, L. and Spanoudakis, G., 2012b. Finding secure compositions of software services: Towards a pattern based approach. In *5th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. IEEE, pp. 1-5.

Pino, L., Spanoudakis, G., Gürgens, S., Fuchs, A. and Mahbub, K., 2012. D02.2 ASSERT aware service orchestration patterns. ASSERT4SOA Project, Tech. Rep. [Online]. Available: http://assert4soa.eu/public-deliverables/

Ponnekanti, S. R., and Fox, A., 2002. Sword: A developer toolkit for web service composition. In *Proc. of the 11th World Wide Web Conference (Web Engineering Track)*, pp. 7-11.

Raman, B., Agarwal, S., Chen, Y., Caesar, M., Cui, W., Johansson, P., ... and Stoica, I., 2002. The SAHARA model for service composition across multiple providers. In *Proceedings of the First International Conference on Pervasive Computing*, Lecture Notes in Computer Science, vol. 2414, pp. 1-14.

RosettaNet. [Online]. Available: http:// www.rosettanet.org/

Shirey, R., 2007. Internet Security Glossary, Version 2. RFC 4949 (Informational), IETF. [Online]. Available: http://www.ietf.org/rfc/rfc4949.txt.

Spanoudakis, G., Mahbub, K., Pino, L., Foster, H., Maña, A. and Pujol, G., 2011. D02.1 ASSERTs aware service query language and discovery engine. ASSERT4SOA Project, Tech. Rep. [Online]. Available: http://assert4soa.eu/public-deliverables/

Tan, W., Fan, Y., and Zhou, M., 2009. A Petri Net-Based Method for Compatibility Analysis and Composition of Web Services in Business Process Execution Language. In *IEEE Transactions on Automation Science and Engineering*, vol.6, no.1, pp.94-106.

Van Der Aalst, W. M. P., Ter Hofstede, A. H. M., Kiepuszewski, B. and Barros, A.P., 2003. Workflow patterns. *Distrib. Parallel Databases*, vol. 14, no. 1, pp. 5-51.

Zisman, A., Spanoudakis, G., Dooley, J. and Siveroni, I., 2013. Proactive and reactive runtime service discovery: A framework and its evaluation. *IEEE Transactions on Software Engineering*, http:// doi.ieeecomputersociety.org/10.1109/TSE.2012.84, Dec 2012.

SHORT PAPERS

Proactive Adaptation in Service Composition using a Fuzzy Logic Based Optimization Mechanism

Silvana de Gyvés Avila and Karim Djemame
School of Computing, University of Leeds, LS2 9JT, Leeds, U.K.
{scsdga, scskd}@leeds.ac.uk

Keywords: Web Service Composition, Proactive Adaptation, Fuzzy Logic, Optimization, Quality of Service.

Abstract: The importance of Quality of Service management in service oriented environments has brought the need of QoS aware solutions. Proactive adaptation approaches enable composite services to detect in advance, according to their QoS values, the need for a change in order to prevent upcoming problems, and maintain the functional and quality levels of the composition. This paper presents a proactive adaptation mechanism that implements self-optimization based on fuzzy logic. The optimization model uses two fuzzy inference systems that evaluate the QoS values of composite services, based on historical and freshly collected data, and decide if adaptation is needed or not. Experimental results show significant improvements in the global QoS of the use case scenarios, providing reductions of up to 8.9% in response time and 14.7% in energy consumption, and an improvement of 41% in availability; this is achieved with an average increment in cost of 11.75 %.

1 INTRODUCTION

A composite service is a software solution with specific functionalities that can be seen as an atomic component in other service compositions, or as a final solution to be used by a consumer. The process of developing a composite service is called service composition, which consists in combining, in a structured way, the features provided by different services (Dustdar and Schreiner, 2005).

The nature of service composition, dynamicity offered by the environments where services are executed and growing amount of available services (that may provide the same functionality), have brought the need of mechanisms focused in ensuring that the consumer will obtain the expected results when invoking a composite service. To achieve this goal, it is important to consider the QoS aspects of the services involved in the composition, as their drawbacks will be inherited by the composite service. However, knowing the QoS of the components is not enough to warranty the behaviour of the composition, as unexpected events may occur at runtime, for example, services becoming unavailable or showing discrepancies in their QoS (Châtel et al., 2010). As a result, various adaptive mechanisms have been proposed in order to restore and maintain the functional and quality aspects of the composition. The aim of adaptive mechanisms is to provide composite services with capabilities that enable them to morph and function in spite of internal and external changes, searching to maximize the composition potential and reducing as much as possible human involvement (Zeginis and Plexousakis, 2010). Based on the moment when adjustments take place, adaptation approaches are classified as either *reactive* or *proactive*. The former corresponds to adaptation actions performed in response to an incident, while the later is related to actions taken in advance, before an incident impacts the system (Metzger, 2011).

This paper introduces a proactive adaptation approach for service composition that implements a self-optimization solution based on fuzzy logic. Fuzzy logic is an approximate reasoning technique suitable to deal with uncertainty (Zadeh, 1994), which can be used to support decision making in software systems. Current work in proactive adaptation for service composition is mainly focused on dealing with the decrease of the QoS values and service failures (Aschoff and Zisman, 2012, Yuelong et al., 2012, Leitner et al., 2010). On the other hand, approaches related to self-optimization are focused on the selection of services that provide the most appropriate QoS levels for the composition

(Ardagna et al., 2011, Calinescu et al., 2011, Cardellini et al., 2012).

The proposed optimization model combines the analysis of historical QoS data and fresh data (collected at runtime from the different stages of the composite service execution) in order to identify the need of adaptation, which can be related to prevent a decrease in the global QoS of the composition, but also, the possibility of improving the global QoS levels. Composite services are considered to be workflows formed by tasks, and tasks to be Web service invocations. The use of the fuzzy support systems enables the evaluation of the QoS parameters and helps deciding whether adaptation is needed or not. If adaptation is needed, the fuzzy systems provide the parameters to be used during the service selection process.

The approach has been implemented in a service composition framework and evaluated through the execution of two test cases. Results were compared with a non-adaptive approach. The major contributions for this paper are:

- The optimization approach for service composition that evaluates the benefit of adaptation.
- The use of fuzzy logic as a decision making tool to determine the need of adaptation in the context of proactive adaptation in service composition.

The remainder of the paper is structured as follows: background is briefly described in Section 2. The proposed framework, service selection and optimization models are described in Section 3. Section 4 presents the experimental description and results. Section 5 discusses some related work. Conclusion and future work are given in Section 6.

2 BACKGROUND

2.1 Adaptation in Service Composition

Adaptive mechanisms provide software systems with capabilities to: self-heal, self-configure, self-optimize, self-protect, etc., which are implemented considering the objectives the system should achieve, the causes of adaptation, the system reaction towards change and the impact of adaptation upon the system (Cheng et al., 2009). Adaptation in service composition aims to mitigate the impact of unexpected events that take place during the execution of composite services, maintaining functional and quality of service levels. Important aspects that can be considered as part of adaptation solutions in service composition are listed as follows (Cardellini et al., 2012):

- Adaptation goal is the purpose of adaptation, functional and/or non-functional (QoS).
- Adaptation level defines those elements that will change in order to achieve the adaptation goal.
- Adaptation actions are those used to solve the adaptation problem.
- Adaptive mechanisms correspond to the approaches applied to implement the adaptation actions (e.g. agent-based, policy-based, rule-based, etc.).
- Stage of adaptation is the time when adaptation is performed (development time, compile/link time, load time and runtime).
- Awareness levels describe the scope of information that will be available in order to adapt (Dustdar et al., 2010).

2.2 Reactive vs Proactive Adaptation

In service-based applications, reactive adaptation is triggered after problems have occurred, when situations like the use of faulty services or services that present undesirable QoS have already affected the application (Hielscher et al., 2008). The use of reactive mechanisms may cause increases in the execution time and financial loss, which can lead to user and business dissatisfaction (Aschoff and Zisman, 2012). Proactive approaches aim to deal with some of these drawbacks by detecting the need for a change, before reaching a point where a problem may occur.

Situations that can be predicted in proactive adaptation approaches for service composition include: the impact of a new requirement, misbehaviour of a service and the existence of new services (Aschoff and Zisman, 2012). Techniques such as data mining, online testing, statistical analysis, runtime verification and simulation are applied during the prediction stage of the process with the aim of accurately predict the future behaviour of the system (Metzger, 2011).

2.3 Fuzzy Logic

Fuzzy logic is a method based on multi-valued logic which aims to formalize approximate reasoning (Zadeh, 1994). It is used to deal with different types of uncertainty in knowledge-based systems. Some of the relevant characteristics of fuzzy logic are fuzzy sets, linguistic variables and fuzzy rules. A fuzzy set is a collection of objects characterized by a membership function with a continuous grade of

membership which can be ranged between zero and one (Zadeh, 1965). A linguistic variable is a type of variable that uses words instead of numbers to represent its values (e.g. slow, medium, fast) (Zadeh, 1994). The values used to define linguistic variables are called terms and the collection of terms is called the term set. A fuzzy rule is used to represent human knowledge using the form of IF-THEN within a fuzzy system (Li-Xin Wang, 1997).

During the execution of a fuzzy system, crisp inputs are converted to linguistic variables, this process is known as fuzzification. The variables' values are then evaluated using fuzzy rules, generating the linguistic values for the outputs. Finally, the defuzzification method uses these values to obtain crisp outputs values.

3 SYSTEM MODEL

An overview of the system model considered in this work is illustrated in Figure 1, which shows its core components: composition engine, adaptation manager, service binder, service selector, predictor and the sensors; and their interactions. This model was implemented with the aim of evaluating the proposed approach, enabling the execution of QoS aware service composition in an environment with proactive capabilities.

The composition engine is the software platform responsible for executing the composite services (processes' definitions) and hosting the components in charge of the adaptation process. Composite services are considered to consist of a series of abstract tasks that will be linked to executable services at runtime.

The adaptation manager works semi-independent of the rest of the components and is constantly monitoring and analyzing not only information collected by the sensors, but also the historical data. The use of historical data helps understanding the service behaviour and enables the detection of any possible deviation in the values of the QoS parameters. During the execution of a composite service, sensors collect fresh data, looking at activity and service levels, and send this information to the monitor. The monitor queries the historical database to obtain information about previous executions and states of the current service, then, sends this information to the analyzer, which evaluates both, fresh and historical data, in order to determine the need of adaptation. If adaptation is needed, the analyzer sends a request of adaptation to the planner, which obtains the adaptation values that will be sent

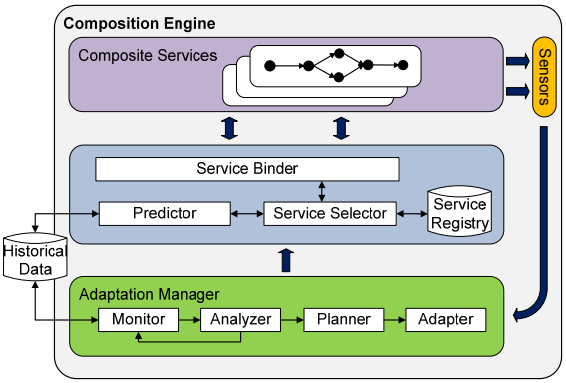


Figure 1: System model.

to the adapter. This information is forwarded to the service binder, in order to maintain/improve the QoS of the composition.

For each task in the composite service, the service binder invokes the service selector with the desired characteristics that the component service should provide. The service selector performs a search in the service registry based on the provided functional requirements. For each of the pre-selected services (candidates), the service selector invokes the predictor to obtain its estimated QoS. This information is sent to the service binder, which compares the candidates and selects the service that suits the request. If the need of a change was identified by the adaptation manager, the binder uses the adaptation values to perform the ranking and selection tasks.

It is considered that at the time of invoking a composite service, the system has available data from previous executions of the different possible components, in order to obtain accurate predictions about these components' quality characteristics. Also, for each task of the composite service, there exist at least two concrete services to invoke.

3.1 QoS Model

Services that offer the same functionality may be associated with several QoS attributes (Cardoso et al., 2004, Zeng et al., 2004), providing different QoS levels. By evaluating these attributes within a set of services that share the same goals, consumers can search/select components to be used in their applications.

The QoS attributes of a service can be evaluated during design and execution time. At design time, these attributes help in order to build a composite service based on the QoS requirements of the user. While at execution time, they can be monitored to maintain the desired QoS level. Information about

these attributes can be obtained from the service's profile (Hwang et al., 2007), nevertheless, when this information is not available, it can be obtained by analyzing data collected from past invocations (Cardoso et al., 2004).

In this work, the quality attributes that will be considered for each service are response time, cost, energy consumption and availability.

- Response time: time consumed between the invocation and completion of the service operation (Dai et al., 2009);
- Cost: fee charged to the consumer when invoking a service (Cardellini et al., 2012);
- Energy consumption: amount of power consumed by a server over a period of time (Buyya et al., 2010);
- Availability: probability that the service is up and ready for immediate consumption (W3C Working Group, 2003).

Considering response time and cost enables the selection of faster and cheaper services, providing a competitive advantage (Cardoso et al., 2004). Both parameters have been used in other approaches, like those presented in (Dai et al., 2009, Ying et al., 2009, Ardagna and Mirandola, 2010, Cardellini et al., 2012).

The amount of energy used by data centres has not only economical but also environmental impacts. Energy efficiency is becoming a key topic due to high energy costs and governments' pressure to reduce carbon footprints (Kaplan et al., 2009). Energy consumption has been selected as the third parameter because of the importance of energy efficiency when managing computing infrastructure and services.

The last parameter that has been selected is availability. By knowing the availability values of the different services, it is possible to select a subset of components that will provide a composition with higher probabilities to be fulfilled. Work that considers availability has been presented in (Huang et al., 2009, Canfora et al., 2008, Zeng et al., 2004).

To compute the values of these parameters at execution time, three situations have been considered within the composite service structure: single, sequential and concurrent service invocations. When computing the QoS parameters of a single service invocation, the QoS values of the activity that performs the invocation corresponds to the QoS values of the invoked service. For activities in a sequential structure, the values of response time, cost and energy consumption are summed for the different activities with service invocations, while availability is obtained by multiplying them.

$$Rt(P) = \sum_{i=1}^{n} Rt(s_i) \quad (1)$$

$$C(P) = \sum_{i=1}^{n} C(s_i) \quad (2)$$

$$Ec(P) = \sum_{i=1}^{n} Ec(s_i) \quad (3)$$

$$Av(P) = \prod_{i=1}^{n} Av(s_i) \quad (4)$$

For activities in a concurrent/parallel structure, the value of response time is considered as the maximum response time of the completed activities; values of cost and energy consumption are summed; and availability is the minimum availability value among the service invocations within the structure.

$$Rt(P) = max_{i=1,..,n}\, Rt(s_i) \quad (5)$$

$$C(P) = \sum_{i=1}^{n} C(s_i) \quad (6)$$

$$Ec(P) = \sum_{i=1}^{n} Ec(s_i) \quad (7)$$

$$Av(P) = min_{i=1,..,n}\, Av(s_i) \quad (8)$$

For equations (1) to (8), s_i corresponds to an activity with a service invocation within the composite service P.

3.2 Service Selection Model

Estimation of QoS values is a key step during service selection process. Estimated values are calculated using historical QoS data recorded from previous executions. This data is filtered, discarding values considered as outliers and the average of the last N executions of the remaining subset is obtained.

Concrete services are searched in the registry by name, assuming that this parameter includes/describes the service's functionality. The resulting set of candidate services is sorted according to the relationship between their estimated QoS values. Due to these attributes having different units of measure, their raw values are normalized before being processed and ranked. The following formula is used to normalize response time, cost and energy consumption, which are negative parameters (lower the value, higher the quality).

$$V_i = \frac{max_i - q_i}{max_i - min_i} \quad (9)$$

A different formula is used for availability, as it is a positive parameter (higher the value, higher the quality).

$$V_i = \frac{q_i - min_i}{max_i - min_i} \quad (10)$$

In both equations, max_i and min_i correspond to the maximum and minimum values of the evaluated

QoS parameter, respectively; and q_i correspond to the estimated value for the next execution. When $max_i = min_i$, then $V_i = 1$. After normalizing the corresponding values, results are computed using the Simple Additive Weighting formula:

$$W_i = rt_i(w_1) + c_i(w_2) + ec_i(w_3) + av_i(w_4) \quad (11)$$

Where rt_i is the service estimated response time; c_i is the service estimated cost; ec_i is the service estimated energy consumption; av_i is the service estimated availability; and w_1, w_2, w_3 and w_4 correspond to assigned weights where $w_1, w_2, w_3, w_4 \leq 1$ and $w_1 + w_2 + w_3 + w_4 = 1$.

3.3 Optimization Model

The proposed optimization model works as part of a proactive adaptation mechanism. It combines the analysis of historical and fresh data. QoS information of the different services and states of the composition is collected from service, task and process perspectives, where service corresponds to concrete Web services; task to elements within the composite service that invoke services; and process to the entire composition (service workflow). Based on this information, it is possible to take decisions about future actions.

The QoS parameters are obtained when the service invocation is performed. Response time is measured during the service's execution; the values of cost and energy consumption are retrieved from the service's WSDL file; while the value of availability is obtained based on historical data. According to the structures of the composite service, the QoS values of each task are computed using equations (1) to (8) and stored in the historical QoS data base, considering both individual values and accumulated. These values are used in order to obtain the global QoS of the composite service.

The service selection model previously described uses as weights for equation (11) the results of the optimization model evaluation. This model is based on two fuzzy support systems, which assess the QoS of the composition, determine the need of adaptation and, when adaptation is needed, obtain the weight values to be used during service selection. The optimization mechanism identifies when the QoS of the composition is decreasing. It also considers situations where a number of the accumulated QoS values of the previous activity in the process are better than expected, which provides the possibility of improving other QoS parameters.

The idea of using fuzzy logic is to understand the relationship between the QoS values of the composite service and the need of adaptation. In this context, QoS parameters can be expressed using linguistic variables. Two inference engines have been defined to 1) obtain the benefit of adaptation, 2) obtain the weights to be used during service selection. Each of these systems uses its own linguistic variables and rules.

The first fuzzy support system evaluates the QoS of the composite service every N milliseconds, in order to identify as soon as possible the need of adaptation. It uses as inputs the measured QoS values collected from the composite service execution. The defined input variables are response time, cost, energy consumption and availability, which are expressed with three terms low, medium and high. To establish these terms for each of the linguistic variables, an interval is defined at runtime using data collected from previous executions. Historical data is analyzed, obtaining maximum/minimum values and standard deviations from each of the QoS parameters. Sigmoidal functions (open to the left and right) are used to define the low and high terms, while Gauss function is used to define the medium term. The system takes the inputs and based on the corresponding fuzzy rules, provides the estimated benefit of adaptation. Four different levels of benefit of adaptation (low, medium, high and very high) were established, falling in the interval [0, 1], and defined with Gauss functions.

The second fuzzy support system uses the value of the benefit of adaptation (output of the first system) and the errors between the estimated and the measured QoS as inputs. The error value is computed per each parameter using the following formula:

$$e\,(p_i) = \frac{x\,(p_i) - x_0\,(p_i)}{x_0(p_i)} \quad (12)$$

Where $x(p_i)$ is the estimated data; and $x_0(p_i)$ is the real measured data.

Input variables corresponding to the QoS errors are expressed with three terms, low, medium and high, falling in the interval [-1, +1]. Benefit of adaptation is expressed with four terms, as defined in the first fuzzy system.

By evaluating the different errors and the benefit of adaptation, the system provides the values to be used as weights during the service selection process. Output variables (response time weight, cost weight, energy consumption weight and availability weight) are expressed with five terms, very low, low, medium, high and very high, falling in the interval [0,1] and are defined using Gauss functions.

The algorithm presented in Table 1 describes the QoS evaluation method applied during the optimization process, which involves the use of the fuzzy systems previously described. Once the execution of a composite service starts, the adaptation manager constantly evaluates its QoS and obtains the errors between estimated and measured values (steps 1 to 12). The measured QoS values are used as inputs for the first fuzzy system. The benefit of adaptation is obtained (step 13) and evaluated (step 14); if it is medium or higher then there is a need of adaptation. When adaptation is needed, the system determines the adaptation weights. This action is performed by the second fuzzy system (steps 15 to 18). Weights are then adjusted, to fulfil the restriction $\alpha + \beta + \gamma + \phi = 1$ (step 19). Finally, the algorithm returns the weight values α, β, γ and ϕ (step 20). These values are sent to the service binder to be used at the moment of selecting the next service. When adaptation is not needed, the service binder ranks the services using fixed weight values.

Table 1: QoS evaluation algorithm.

Input:
rt → response time
cost→ cost
ec → energy consumption
av →availability
eRt → response time error
eCost→ cost error
eEc→ energy consumption error
eAv→ availability error

Output:
ω → benefit of adaptation
α → response time weight
β →cost weight
γ →energy consumption weight
ϕ →availability weight

(1) Sort by response time
(2) rt ← **Obtain** measured response time
(3) eRt ← **Obtain** response time error
(4) Sort by cost
(5) cost ← **Obtain** measured cost
(6) eCost ← **Obtain** cost error
(7) Sort by energy consumption
(8) ec ← **Obtain** measured energy consumption
(9) eEc ← **Obtain** energy consumption error
(10) Sort by availability
(11) av ← **Obtain** measured availability
(12) eAv ← **Obtain** availability error
//fuzzy system 1
(13) ω ← **Obtain** benefit of adaptation
(14) **if** ω >= medium **then**
//fuzzy system 2
(15) α ← **Obtain** response time weight
(16) β ← **Obtain** cost weight
(17) γ ← **Obtain** energy consumption weight
(18) ϕ ← **Obtain** availability weight
(19) **Adjust** weights
(20) **return** α, β, γ and ϕ

4 EVALUATION

To evaluate the proposed optimization approach, an experimental environment was setup and two composite services were developed as test cases. Experiments were carried out to address the following question:

- Does the use of a proactive adaptation approach based on self-optimization helps improving the global QoS of a composition?

4.1 Experimental Environment

The experimental environment consists of 4 nodes configured on a WAN, distributed between United Kingdom and Germany, with estimated values for bandwidth and latency around 32Mbit/s and 29ms, respectively. Node 1 is a computer with Windows Vista, 4GB RAM and one Intel core2 duo 2.1GHz processor (located in United Kingdom). This node hosts the BPEL engine (Apache ODE 1.3.4), service registry (jUDDI 3.0.4) and historical data base (MySQL 5.1.51). It is in charge of coordinate the execution of the compositions and record all the gathered information. Nodes 2 to 4 are virtual machines setup on remote servers (located in Germany), each of the VM's uses Debian Squeeze x86 and 1GB RAM. These nodes host one application server (Tomcat 6.0.35.0) each, which contains 3 sets of Web services. In total there are 9 Web services deployed per composition's activity.

The initial values of the QoS parameters were established based on the node where the service is running and the corresponding set. Delays are inserted on some of the service sets, to obtain different response times, not only based on the network latency, but the Web services performance. This information is shown in Table 2.

Values of the cost and energy consumption change over time, or between services' executions. This adds dynamicity to the test environment and helps obtaining sensible results; also avoiding the invocation of only one service per each of the tasks in the composite services. To turn cost into a dynamic QoS value, the number of times a service

has been invoked within a period of time is evaluated continuously. Based on this information, it is possible to establish a new cost based on the demand (number of times the service is invoked), assuming that higher the demand, higher the cost. Cost value is updated on the WSDL file of each service.

Regarding energy consumption, each of the servers where the Web services are executed is assumed to have different hardware and software configurations. Servers information and their characteristics were selected from the Energy Star report (Energy Star, 2012). Using the model proposed in (Buyya et al., 2010), which is based on the percentage of CPU usage, it is possible to determine an approximate value to the server energy consumption.

$$P(u) = P_{max} \cdot k + (1 - k) \cdot P_{max} \cdot u \quad (13)$$

$$E = \int_t P(u(t)) \quad (14)$$

Where $P(u)$ is the power consumed in an instance of time; P_{max} is the power consumed when the server is fully utilized; u is the utilization level; and k is the fraction of power consumed by the idle server. E is the total energy consumed by a node over a period of time t.

Servers' utilization is considered to be variable over time. The power consumed by a server is obtained periodically and exposed on the WSDL files of the corresponding services. It is computed using equations (13) and (14) and the data presented in Table 2.

Table 2: QoS parameters initial setup.

Server	Set	Time delays (ms)	Cost	Energy Consumption (W/sec)		Availability
				Idle	Load	
Node 2	S1	0	120	50.75	129.5	0.9
	S2	350	80			0.9
	S3	200	100			0.9
Node 3	S1	0	150	45.27	81.9	0.64
	S2	350	100			0.62
	S3	200	120			0.63
Node 4	S1	0	100	210.85	388.3	0.5
	S2	350	60			0.46
	S3	200	80			0.48

4.2 Experiment Description

Two test cases have been modelled in order to asses the proposed approach. These models are BPEL (OASIS, 2007) processes that represent typical examples for service composition scenarios. Test

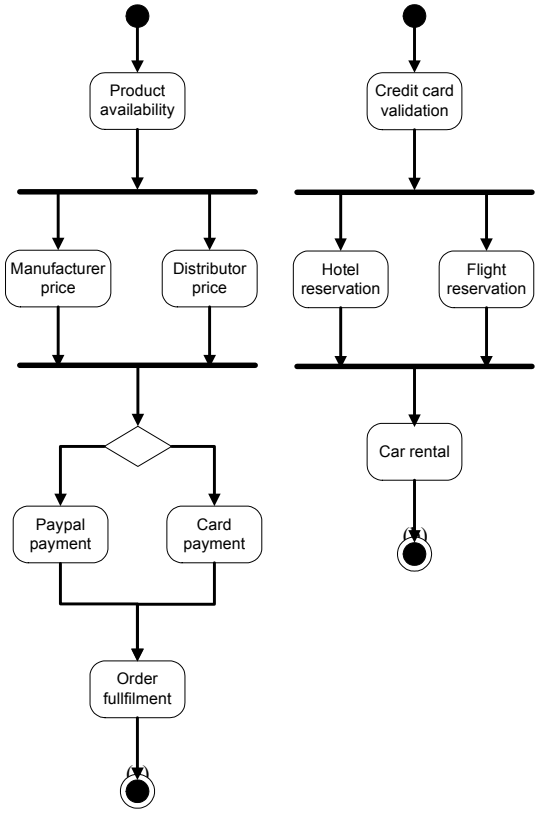


Figure 2: Test cases. (a) Order booking process. (b) Travel planning process.

case 1 is illustrated in Figure 2(a), it implements an order booking process that validates the product availability, obtains the best price of the product from two different providers, selects the best provider, performs the payment, and finally completes the order. Test case 2 implements a travel planning process, as shown in Figure 2(b). It validates a credit card, performs flight and hotel reservations in parallel, and finally invokes a car rental operation. For matter of simplicity, both diagrams only depict those activities that involve service invocations.

Per each of the tasks in the processes, there are 9 candidate services, distributed among the servers (nodes) that fulfil the required functionality and offer different QoS, giving a total of 45 candidate services to be used in test case 1 and 36 for test case 2. These services were previously registered into the service registry (UDDI), and executed several times to populate the historical data base and enable the estimation of their QoS attributes. Both processes are hosted and invoked from Node1.

In order to evaluate the proposed approach, both test cases where executed 100 times. These executions were performed from two different perspectives: 1) using the proactive optimization

mechanism; 2) using a non-adaptive method. The experiment was repeated 5 times to assess the consistency of the results based on statistical analysis.

4.3 Evaluation Results

The behaviour of the proactive optimization mechanism was compared with a non-adaptive approach, where service selection was performed using fixed weights set to 0.25. Initial results show improvements in the global QoS values of the composition when using the proposed approach. Global QoS refers to the final values of the different QoS properties (response time, cost, energy consumption and availability).

The plots shown in Figure 3 depict the behaviour of the order booking process, showing the mean values of the different QoS parameters after performing 5 sets of 100 runs. For the proposed approach, the values of cost and energy consumption change over time, as previously described, while for the non-adaptive approach, remain constant. For both cases, the value of availability changes according to the behaviour of the component services.

After analyzing the value of each of the QoS parameters, in both processes, it was identified that, in order to improve response time, energy consumption and availability, there was an increment in the composition's cost.

In test case 1, results show that the proposed approach provides a mean reduction of 2% with a standard deviation of 6.7% in the measured response time values. Also, it can be noticed from Figure 3(a), that it presents a more stable behaviour, without showing high peaks, as compared to the non-adaptive approach. This is due to the constant evaluation of the QoS parameters during execution.

In terms of energy consumption, it is important to notice that this value is not only based on power consumption, but also influenced by time. As a result, a small response time may produce a small energy consumption value. Figure 3(b) shows the values corresponding to energy consumption, which have a similar behaviour to response time, and provide a mean reduction of 14.7% with a standard deviation of 18.9%. Results also indicate that there is a significant improvement in the processes' availability, presenting a mean increase of 41% with a standard deviation of 35%. The availability values corresponding to the order booking process are illustrated in Figure 3(c). Regarding cost, it can be

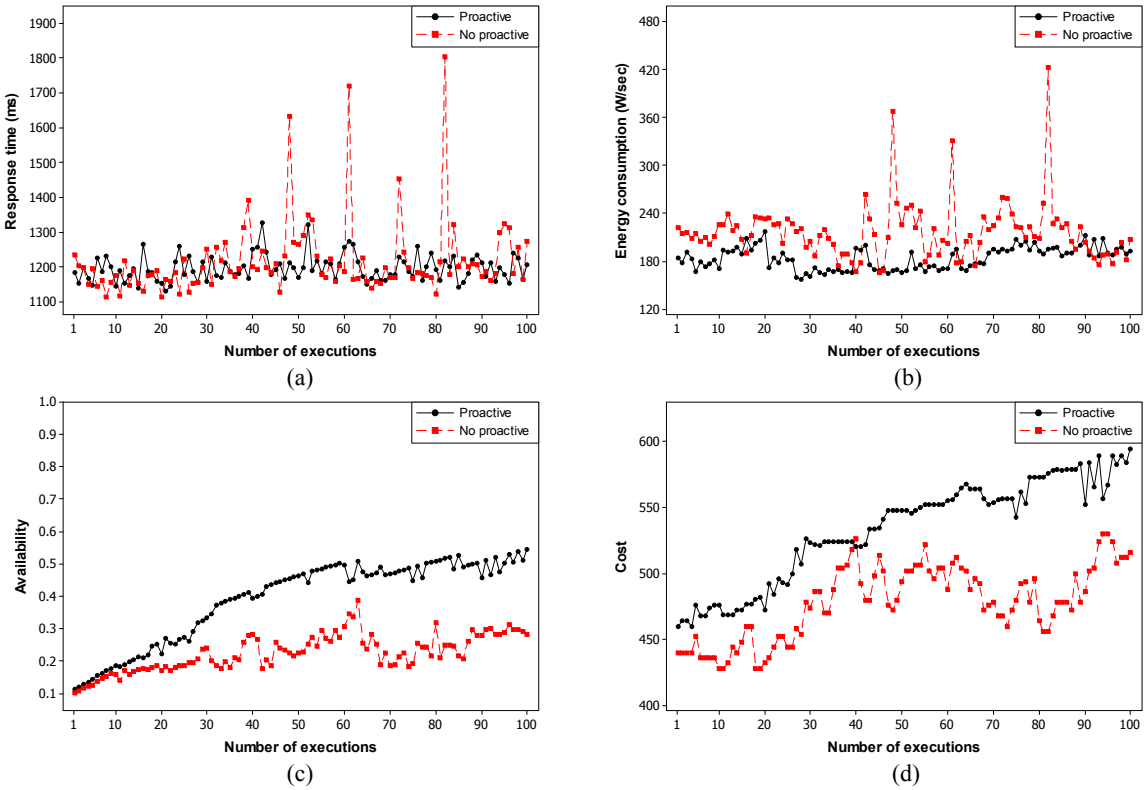


Figure 3: Order booking process results. (a) Response time. (b) Energy consumption. (c) Availability. (d) Cost.

noticed from Figure 3(d) that the use of the proposed approach turns into more expensive composite services. It shows a mean increase of 11% with a standard deviation of 8.4%.

Results obtained from test case 2 show a similar behaviour; where response time, energy consumption and availability values are improved, while cost increases. In terms of response time, it shows a mean reduction of 8.9% with a standard deviation of 16%. For energy consumption, the obtained mean reduction is 4.6% with a standard deviation of 29%. Regarding availability, it provides an improvement of 18% with a standard deviation of 25%. Finally, in terms of cost, there is an increment of 12.5% with standard deviation of 6.8%.

Based on the analysis of the weight values obtained by the optimization model and sent to the service binder, the parameter that had the higher impact within the adaptation process was energy consumption, followed by response time. Because of this, at the moment of selecting new services to be invoked, priority would be given to those that are being executed on servers with lower energy consumption, and lower response time. Which, based on the QoS configuration, are the services that also involve higher costs. Different QoS configurations may give different results; however, because of the use of multiple QoS criteria, it is likely to find that not all the parameters can be improved.

5 RELATED WORK

Several works have been proposed to mitigate the impacts of unexpected events during the execution of services, ensuring/maintaining the functional and quality levels. These approaches can be classified based on the moment when adaptation takes place into the categories: reactive and proactive. Reactive adaptation occurs after the appearance of an undesired event, while proactive adaptation aims to predict and prevent the occurrence of the problem (Aschoff and Zisman, 2012).

Some of the approaches that support reactive adaptation implement self-* properties. Self-healing mechanisms aim to prevent composite services from failing, from functional and non-functional perspectives. Projects like those presented in (Canfora et al., 2008, Erradi and Maheshwari, 2008, Dai et al., 2009, Wu et al., 2009, Ying et al., 2009, Ardagna et al., 2011, Wenjuan et al., 2010, Baker et al., 2013) apply self-healing approaches, where new services are selected and invoked after a functional failure or a QoS constraint violation. Self-optimization mechanisms are closely related to the selection of services at runtime, in order to maintain the expected QoS of the entire composition. Examples of works belonging to this category are described in (Ardagna et al., 2011, Calinescu et al., 2011, Cardellini et al., 2012).

Approaches that support proactive adaptation are presented in (Hielscher et al., 2008, Tosi et al., 2009, Leitner et al., 2010, Aschoff and Zisman, 2012, Metzger, 2011, Yuelong et al., 2012, Sammodi et al., 2011). The work presented in (Aschoff and Zisman, 2012) introduces a proactive adaptation approach that enables service replacement (1-1, 1-n, n-1, n-m) when it detects situations that may cause the composition to stop its execution (unavailable or malfunctioning services); or that allow the composition to continue its execution, but not in its best way. Also it considers the emergence of better services and new requirements. The approach uses a composition template as start point and selects a set of candidate services to be used in the composition and their replacements. The approach introduced in (Yuelong et al., 2012) combines runtime information with design-time specifications (of each component service within a composition), in order to construct a k-step model of the current service states. The resulted model can be used to be compared with the desired behaviour of the composition. The work in (Leitner et al., 2010) aims to minimize Service Level Agreement (SLA) violations. It uses predictions of SLA violations generated with regressions of monitored and estimated data. These predictions are evaluated at defined checkpoints. In (Hielscher et al., 2008), a framework that uses online testing to trigger proactive adaptation in service-based applications is described. Test objects can be single or composite services. While performing online testing, if an online test fails or deviates from its expected behaviour, the framework will trigger adaptation to avoid undesirable consequences. One of the application scenarios for this approach is composite services. The work presented in (Tosi et al., 2009) proposes a self-adaptive mechanism based on the use of test cases to obtain possible mismatches between requested and provided services. When the diagnosis mechanism reveals mismatches, it triggers adaptation strategies that update the structure and the behaviour of the client application to solve the identified problems. Even though this approach is not mainly focus in service composition, it presents a proactive mechanism that works in service-based applications. In (Sammodi et al., 2011) it is presented a proactive approach for adaptation in service-based applications that

combines monitoring, online testing and quality prediction. When a service is likely to be used with a high frequency, it is selected to be tested. The use of pre-defined test cases (concrete data inputs) enables the system to collect information about the behaviour of the services and complement the data gathered during monitoring. Finally, the work described in (Metzger, 2011) discusses two main directions than can be followed in order to perform proactive adaptation in service oriented systems. The first direction is to improve the failure predictions techniques. Some prediction techniques identified by the authors include data mining, online testing, runtime verification, statics analysis and simulation. The second direction is by dynamically estimating the accuracy of the predicted failures during the operation of the service-oriented system.

Although these approaches are closely related with the work described in this paper, there are significant differences:

- The QoS parameters considered in this study are response time, cost, energy consumption and availability, while related approaches mainly focus on response time.
- Adaptation is not limited to failure prevention, but also considers the possibility of improvement in the QoS levels.
- The use of the benefit of adaptation, obtained from the measured QoS values, to determine whether adaptation is needed or not.
- The use of fuzzy logic as a decision making tool to determine the need of adaptation in the context of proactive adaptation in service composition.

6 CONCLUSION AND FUTURE WORK

This paper presents a proactive adaptation approach for service composition that implements a self-optimization mechanism, which aims to improve/maintain the quality levels of the composite services. At runtime, this mechanism performs a continuous evaluation of the composite services' behaviour and triggers adaptation when the benefit of performing this action is considered to be significant. In order to perform this decision making process, it uses fuzzy support systems and carries out an analysis on historical and fresh QoS data.

Evaluation results show that the proposed mechanism improves the global QoS values of the compositions, showing significant improvements regarding response time, energy consumption and availability. However, it was not possible to improve all the considered quality parameters, as there was an increment in the cost of the composite services. This behaviour is shown due to the relationship between the values of quality parameters exhibit by the services, as those services with lower energy consumption and higher availability, also display higher costs.

Future work includes the use of Service Level Agreements on top of the composite services, in order to provide a contract between the provider and consumer. Also, it is considered the extension of the adaptation mechanism, by adding features that enable service replacements considering different structures with the forms 1-n, n-1 and n-m. Finally, with the aim of performing a comparison, it is considered the implementation and evaluation of mechanisms proposed in the related work, using the same composition environment.

REFERENCES

Ardagna, D., L. Baresi, et al., 2011. A Service-Based Framework for Flexible Business Processes. *IEEE Software* 28(2): 61-67.

Ardagna, D. and R. Mirandola, 2010. Per-Flow Optimal Service Selection for Web Services Based Processes. *Journal of Systems and Software* 83(8): 1512-1523.

Aschoff, R. and A. Zisman, 2012. Proactive Adaptation of Service Composition. *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'12)*. Zürich, Switzerland: 1-10.

Baker, T., O. F. Rana, et al., 2013. Towards Autonomic Cloud Services Engineering via Intention Workflow Model. *Proceedings of the 10th International Conference on Economics of Grids, Clouds, Systems, and Services (GECON'13)*, Zaragoza, Spain.

Buyya, R., A. Beloglazov, et al., 2010. Energy-Efficient Management of Data Center Resources for Cloud Computing: A Vision, Architectural Elements, and Open Challenges. *Proceedings of the 2010 International Conference on Parallel and Distributed Processing Techniques and Applications*, las Vegas, USA.

Calinescu, R., L. Grunske, et al., 2011. Dynamic QoS Management and Optimization in Service-Based Systems. *IEEE Transactions on Software Engineering* 37(3): 387-409.

Canfora, G., M. D. Penta, et al., 2008. A Framework for QoS-Aware Binding and Re-Binding of Composite Web Services. *Journal of Systems and Software* 81(10): 1754-1769.

Cardellini, V., E. Casalicchio, et al., 2012. MOSES: A Framework for QoS Driven Runtime Adaptation of

Service-Oriented Systems. *IEEE Transactions on Software Engineering* 38(5): 1138-1159.

Cardoso, J., A. Sheth, et al., 2004. Quality of Service for Workflows and Web Service Processes. *Journal of Web Semantics* 1(3): 281-308.

Châtel, P., J. Malenfant, et al., 2010. QoS-based Late-Binding of Service Invocations in Adaptive Business Processes. *Proceedings of the International Conference on Web Services (ICWS'10)*, Miami, USA, IEEE Computer Society.

Cheng, B., R. Lemos, et al., 2009. Software Engineering for Self-Adaptive Systems: A Research Roadmap. *Software Engineering for Self-Adaptive Systems* 5525: 1-26

Dai, Y., L. Yang, et al., 2009. QoS-Driven Self-Healing Web Service Composition Based on Performance Prediction. *Journal of Computer Science and Technology* 24(2): 250-261.

Dustdar, S., C. Dorn, et al., 2010. A Roadmap Towards Sustainable Self-Aware Service Systems. *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, Cape Town, South Africa, ACM.

Dustdar, S. and W. Schreiner, 2005. A Survey on Web Services Composition. *International Journal on Web and Grid Services* 1(1): 1–30.

Energy Star, 2012. Computer Servers Product List - Families.

Erradi, A. and P. Maheshwari, 2008. Dynamic Binding Framework for Adaptive Web Services. *Proceedings of the 2008 Third International Conference on Internet and Web Applications and Services*, Athens, Greece, IEEE Computer Society.

Hielscher, J., R. Kazhamiakin, et al., 2008. A Framework for Proactive Self-adaptation of Service-Based Applications Based on Online Testing. *Proceedings of the 1st European Conference Service Wave*, Madrid, Spain, Springer-Verlag.

Huang, A., C.-W. Lan, et al., 2009. An Optimal QoS-based Web Service Selection Scheme. *Information Sciences* 179(19): 3309-3322.

Hwang, S.-Y., H. Wang, et al., 2007. A Probabilistic Approach to Modeling and Estimating the QoS of Web-Services-Based Workflows. *International Journal of Information Sciences* 177(23): 5484-5503.

Kaplan, J., W. Forrest, et al., 2009. Revolutionizing Data Center Energy Efficiency, McKinsey.

Leitner, P., A. Michlmayr, et al., 2010. Monitoring, Prediction and Prevention of SLA Violations in Composite Services. *Proceedings of the IEEE International Conference on Web Services (ICWS'10)*, Miami, USA.

Li-Xin Wang, 1997. *A Course in Fuzzy Systems and Control*, Prentice Hall.

Metzger, A., 2011. Towards Accurate Failure Prediction for the Proactive Adaptation of Service-Oriented Systems. *Proceedings of the 8th Workshop on Assurances for Self-adaptive Systems*, Szeged, Hungary, ACM.

OASIS, 2007. Web Services Business Process Execution Language Version 2.0. Retrieved Dec. 2013, from http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

Sammodi, O., A. Metzger, et al., 2011. Usage-Based Online Testing for Proactive Adaptation of Service-Based Applications. *Proceedings of the IEEE 35th Annual Computer Software and Applications Conference (COMPSAC'11)*, Munich, Germany.

Tosi, D., G. Denaro, et al., 2009. Towards Autonomic Service-Oriented Applications. *International Journal of Autonomic Computing* 1(1): 58-80.

W3C Working Group, 2003. QoS for Web Services: Requirements and Possible Approaches. Retrieved Dec. 2013, from http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/.

Wenjuan, L., Z. Qingtian, et al., 2010. A Framework to Improve Adaptability in Web Service Composition. *Proceedings of the 2nd International Conference on Computer Engineering and Technology (ICCET'10)*, Chengdu, China.

Wu, G., J. Wei, et al., 2009. Towards Self-Healing Web Services Composition. *Proceedings of the First Asia-Pacific Symposium on Internetware*, Beijing, China, ACM.

Ying, Y., Z. Bin, et al., 2009. A Self-Healing Composite Web Service Model. *Proceedings of the IEEE Asia-Pacific Services Computing Conference (APSCC'09)*, Biopolis, Singapore.

Yuelong, Z., W. Xiaobin, et al., 2012. Predicting Failures in Dynamic Composite Services with Proactive Monitoring Technique. *Proceedings of the IEEE Eighth World Congress on Services (SERVICES'12)*, Honolulu, USA.

Zadeh, L. A., 1965. Fuzzy Sets. *Information and Control* 8(3): 338-353.

Zadeh, L. A., 1994. The Role of Fuzzy Logic in Modeling, Identification and Control. *Modeling, Identification and Control* 15(3): 191-203.

Zeginis, C. and D. Plexousakis, 2010. Web Service Adaptation: State of the Art and Research Challenges. Heraklion, Crete, Greece, Institute of Computer Science.

Zeng, L., B. Benatallah, et al., 2004. QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering* 30(5): 311-327.

Preference-driven Refinement of Service Compositions*

Cheikh Ba[1,3], Umberto Costa[2], Mirian Halfeld-Ferrari[3], Rémy Ferre[3], Martin A. Musicante[2], Veronika Peralta[4] and Sophie Robert[3]

[1]*Université Gaston Berger, Saint Louis, Sénégal*

[2]*DIMAp, Universidade Federal do Rio Grande do Norte, Natal, Brazil*

[3]*Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, FR-45067 Orléans, France*

[4]*Université François Rabelais Tours, LI, Blois, France*

cheikh.ba.sn@gmail.com, {mirian, sophie.robert}@univ-orleans.fr, remy-ferre@laposte.net, {mam, umberto}@dimap.ufrn.br, veronika.peralta@univ-tours.fr

Keywords: Service Compositions, Abstract Services, Refinement, User Preferences.

Abstract: The Service Oriented Computing Paradigm proposes the construction of applications by integrating pre-existent services. Since a large number of services may be available in the Cloud, the selection of services is a crucial task in the definition of a composition. The selected services should attend the requirements of the compound application, by considering both functional and non-functional requirements (including quality and preference constraints). As the number of available services increases, the automation of the selection task becomes desirable. We propose a method for the refinement of service compositions that takes the abstract specification of a composition, the definition of concrete services and user preferences. Our algorithm produces a list of refinements in preference order. Experiments show that our method can be used in practice.

1 INTRODUCTION

Cloud computing provides a cheaper, reliable infrastructure in which service providers may deploy applications or store data to be used by their customers. Access to the resources is granted to users by offering a client-server infrastructure that can be used to perform tasks at different levels of abstraction. Cloud infrastructure includes models at three levels of abstraction : Cloud Infrastructure as a Service (*IaaS*), Cloud Platform as a Service (*PaaS*) and Cloud Software as a Service (*SaaS*). The three models of Cloud computing propose the use of *services* to access resources. Services may be used to query data or to perform computations.

The long-term goal of our work is the construction of a platform (*PaaS*) to provide a high-level, abstract way of specifying service compositions. The compositions would be obtained according to the software development process composed by *Specification*, *Refinement based on user preferences* and *Coding*.

*This work was partly supported by the National Institute of Science and Technology for Software Engineering (INES), funded by CNPq (Brazil) 573964/2008-4 and 305619/2012-8; CAPES/UdelaR (Brazil) 021/10; CAPES/STIC-AmSud (Brazil) 052/14.

In this paper, we only deal with the refinement step: The specification of a compound service is defined in terms of *abstract* services (*i.e.*, specifications of basic service capabilities). Abstract services may be combined to specify the functional and non-functional behaviour of the composition. This abstract composition will be refined into a composition of concrete, available services. The refinement algorithm presented here builds new compositions by combining concrete services, in order to match the specification. Concrete services are available in the Cloud. We suppose that the capabilities of each concrete service are also specified in terms of abstract building blocks, similar to those used in the abstract composition. The choice among different possible compositions is guided by user preferences.

In (Costa et al., 2013), we have outlined a refinement technique that obtains all the possible combinations of concrete services to match the specification. This leads to an algorithm with NP time complexity. In this paper we use the notion of *user preferences* to set conditions to favour the use of certain services. The search space (combinations of actual services in the Cloud) may be ordered in accordance to these preferences. This order may be used to produce the solutions (concrete compositions) in an incremental

way, so that the refinement process can stop when the desired number of compositions is reached.
Paper Organisation: Section 2 presents related work. Section 3 discusses how to use preferences to select services. Section 4 presents our method. A prototype is shown in Section 5. Section 6 concludes the paper.

2 RELATED WORK

In (Costa et al., 2013), we presented a method (referred here as Refinement1) for the automatic refinement of web service compositions, based on MiniCon algorithm (Pottinger and Halevy, 2001). This method takes into account both functional and non-functional requirements, as well as semantic information. User preferences are not taken into account and no ranking is provided for the produced refinements.

Several works (Zhao et al., 2011; Mesmoudi et al., 2011; Paiva Tizzo et al., 2011) are close to Refinement1. In (Zhao et al., 2011) the authors use the original MiniCon algorithm to rewrite *information-providing* compositions in terms of available services. The main difference between their approach and ours is that we have substantially modified the original MiniCon algorithm to support ontology alignment, optional parameters, quality constraints and conversion functions. In (Mesmoudi et al., 2011) the authors propose a method of service discovery that builds buckets of relevant services for each sub-goal in a composition, by taking into account the semantics description of services. Like us, they distinguish between the phase of refinement (or discovery) and a latter step of orchestration. In (Paiva Tizzo et al., 2011) the authors propose a model to improve the dynamic selection of services in the context of WS-BPEL processes. Service compositions are described as SPARQL queries; those queries are evaluated at runtime by a SPARQL server to produce service endpoints; those endpoints are used to invoke the selected services. The proposed approach is semi-automatic and does not take user profiles or QoS into account.

Recently, the QoS-based web service selection and composition in service oriented applications has gained the attention of many researchers (Alrifai et al., 2008; Alrifai et al., 2010; Sandionigi et al., 2013; Zeng et al., 2003). These papers focus on *how to select services* on the basis of QoS, *i.e.*, in solving a multi-criteria decision making problem which corresponds to the identification of the best candidate web services from a set of functionally equivalent services. For instance, (Alrifai et al., 2008) models the problem as the Multi-Choice Multidimensional Knapsack problem and proposes a scalable QoS computation approach based on a heuristic algorithm, which decomposes the optimization problem into small subproblems. Their heuristic is a hybrid approach that combines global optimization with local selection in order to find a close-to-optimal selection more efficiently. Improvements of this method are presented in (Alrifai et al., 2010). Some approaches are very specific to the targeted application. For example, in (Sandionigi et al., 2013) the authors are capable to express the service selection in situational computing applications as a Mixed Integer Linear Programming problem from decision variables expressing binding and *ad hoc* QoS constraints. Contrary to them, our goal is to show how *to use* the result of a QoS-based service selection *to decrease the search space* while looking for a concrete instantiation of an abstract composition. Moreover, several QoS aspects may be summarized in a given user preference.

Qualitative preference models, including conditional and temporal clauses, are used in (Lin et al., 2008; Sohrabi and McIlraith, 2010) in order to model expressive service goals that should be satisfied as much as possible. However, most works rely on quantitative preference models, expressing a utility function (generally a weighted sum that combines multi-criteria preferences). For example, given a set of concrete compositions, (Haddad et al., 2010) recommends the compositions that maximize a utility function. As another approach, (Lamparter et al., 2007) uses utility policies and linear programming for selecting compositions. These proposals use user preferences for choosing the best compositions among a set of pre-generated compositions. Our goal is to use user preferences for guiding and optimizing the generation process. The work of (Lemos et al., 2012) use PreferenceSQL constructors for expressing hard and soft preferences ; the former are used to prune the set of candidate services while the latter are used to order them (they do not address service composition). However, preferences are used after performing a structural matching between a requested service and each candidate service, while we aim at using preferences for optimizing the process.

3 BACKGROUND

Abstract compositions are specified by equations, in accordance to the following:

Definition 1 (Abstract Composition) . *Let*

$$C(\bar{t}) \equiv_{def} A_1(\bar{t_1}),\ldots,A_n(\bar{t_n}),Q_1(\bar{t'_1}),\ldots,Q_m(\bar{t'_m}). \quad (1)$$

be the definition of an abstract composition. The left-hand side of the specification defines the interface of

the composition. The elements of the tuple $\bar{t}$ are formal parameters and represent input and output data. We use the ? and ! markings to indicate input and output parameters, respectively.

The right-hand side of the definition consists of abstract services ($A_1,\ldots,A_n$), and constraints ($Q_1,\ldots,Q_m$), expressing capabilities of the composition. Abstract services correspond to semantic descriptions of service functionalities and specifies the relationship between their required inputs and expected outputs. Ontologies may be used to align the representation of such concepts and to describe the relationships between services.

Abstract services are the building blocks used to specify the composition to be refined. Constraints are relational expressions built on variables and constants. They can express static conditions (to be verified during the refinement process) and dynamic conditions (to be verified dynamically by the refined composition). The composition designer specifies an order of importance between abstract services.

Example 1 . *Let us suppose the specification of a composite service for buying books over the Internet:*

$$\begin{aligned} &LookForBooks(Uid?,Pwd?,Isbn?,Loc?,Price!,Addr!,Ack!)\\ &\equiv Authentication(Uid?,Pwd?,Tkn!,Prot!,Form!),\\ &\quad BookStore(Tkn?,Isbn?,Loc?,Addr!,Price!,Invoice!),\\ &\quad Payment(Tkn?,Invoice?,Ack!),\\ &\quad Prot = \text{“}REST\text{”}, Ack = \text{“}OK\text{”}. \end{aligned}$$

The client supplies an identification, password, ISBN and location and expects to receive the book's price, a pickup address and the transaction acknowledgement. The composition begins with the authentication of the client. This step returns a token to identify the client in the store, as well as the exchange protocol and format. The BookStore service also uses the client's location to return the nearest pickup address for the book, as well as the price and the invoice, to process the payment. After looking up the book's price, the bill is paid by using the credit card information.There are two constraints in the composition. The first one is a static constraint that specifies the protocol to be used for authentication. The second condition establishes that the whole process was successfully finished (to be verified dynamically).

Concrete services are also specified by using abstract services and constraints:

Definition 2 (Concrete Services) . *Let*

$$S(\bar{t}) \equiv_{def} A_1(\bar{t_1}),\ldots,A_k(\bar{t_k}),Q_1(\bar{t'_1}),\ldots,Q_r(\bar{t'_r}). \quad (2)$$

be the definition of a concrete service. As in the case of Definition 1, the left-hand side of the specification defines the interface of the service. The elements of the tuple $\bar{t}$ are formal parameters and represent input and output data. We use the ? and ! markings to indicate input and output parameters, respectively.

The right-hand side of the definition consists of abstract services ($A_1,\ldots,A_k$), and constraints ($Q_1,\ldots,Q_r$), expressing capabilities of the service.

In this case, the left-hand side of the definition gives the name and interface of the concrete service. The right-hand side uses abstract services and constraints to express the capabilities of the service. The semantic information embedded in the abstract services can help to broaden the number of available services for the refinement process. We suppose that the specification of each concrete service is given by the service supplier/publisher.

Our refinement method matches the specification of concrete services with that of the abstract composition. The algorithm considers the definition of each available concrete service to cover parts of the abstract composition. These matchings will be used later by the algorithm, to build a concrete solution (refinement) of the specification.

The refinement process uses preference information to guide the traversal of the search space of concrete services. This traversal should be done in an order determined by the user preferences. In this way, the refinement algorithm should be capable of proposing concrete compositions that comply with the composition specification and that are presented in decreasing order of preferences. We associate a score to each concrete service available to the composition developer. Services having a greater score are preferred to the ones having lower scores.

Definition 3 (User Preferences) . *Given a set $\mathcal{S}$ of concrete services and a real-valued scoring function function $u : \mathcal{S} \mapsto [0,1]$, a preference $P = (\mathcal{S}, <_P)$ is derived from the scoring function, where for two concrete services $x,y \in \mathcal{S}$, $x <_P y$ iff $u(x) < u(y)$.*

The above definition is interpreted as “*I like y better than x if it has a better score*”. Notice that scoring functions can represent very different types of preferences, ranging from simple scores (e.g.based on response time) to complex, multi-criteria expressions that combine various quality perspectives. These scores may be manually defined by users or communities, may be deduced or estimated from user activity, or may be automatically assessed. Our approach is independent of how the scoring function is defined.

The notion of preference classifies the available concrete services in accordance to the user's point of view. However, this notion is orthogonal to the semantics of each concrete service, since the definition of preferences does not need to take into account the functionality of concrete services.

We aim to use preference information to produce compositions that maximize the combined weight of its component services. In order to adapt this information to the context of our refinement procedure, we introduce the notion of *Coverage Domain*.

Definition 4 (Coverage Domain) . *For each abstract service A_i of an abstract composition we define the Coverage Domain $\mathcal{A}_i$, such that: (1) $\forall S \in \mathcal{S}$ s.t. $S \in \mathcal{A}_i$, S contains the abstract service A_i in its definition. (2) For each $\mathcal{A}_i$, there is a threshold ξ_i, defined at the same time as the abstract composition. Only services S such that $u(S) \geq \xi_i$ will be included in $\mathcal{A}_i$.*

Notice that, since each abstract service A_i is a semantic annotation that denotes that S provides a defined functionality, the coverage domain $\mathcal{A}_i$ represents the set of services that can perform such a provision. Also, according to Definition 4, an empty coverage domain $\mathcal{A}_i$ indicates that no concrete service in $\mathcal{S}$ will provide the functionality described by A_i. In this case, no refinement can be produced over $\mathcal{S}$.

4 REFINEMENT PROCESS

High-level compositions are specified in terms of abstract services, constraints and a total order over the coverage domains. Given this high-level composition and the specification of the available concrete services, our approach generates refined compositions, built upon concrete services. The refinement process incrementally generates concrete compositions, in accordance to the user's preferences.

Algorithm 1 traverses a search space formed by combinations of concrete services, in order to build a list of concrete compositions that refine the abstract composition specification.

Algorithm 1 . (Refinement Algorithm)

```
procedure refinement (C, A)                                   1
  I:= BuildServiceOrganiser(C, A);                            2
  BuildStorePCD(C, A, I);                                     3
  while number of desired solutions is not reached do         4
    for each P := GetNextSetOfPCD(I); do                      5
        R := ProduceRewriting(C, P);                          6
        Publish(R); //R may be empty.                         7
    end do                                                    8
  end while                                                   9
end procedure                                                 10
```

Given the abstract composition C, and the coverage domains $\mathcal{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_k\}$, built from user's preferences as defined in Section 3, we suppose that the order between coverage domains is defined by the indices ($1 \leq i \leq n$) in the right-hand side of the specification of C. The procedure *BuildServiceOrganiser* initialises an index-like data structure to store the concrete services in the lexicographical order (based on the score of concrete services and the coverage domain order). For each concrete service, the procedure *BuildStorePCD* produces mappings (called PCD - Partial Coverage Descriptors) to cover parts of the abstract specification. A PCD describes *(i)* which part of the abstract composition is covered by a concrete service and *(ii)* how to relate the data processed by the composition with the parameters of this concrete service.

The procedure *BuildStorePCD* associates each PCD to a concrete service and stores them for future use. Algorithm 1 implements an iterator which generates sets of PCD by traversing the index I, in decreasing lexicographical order, according to the user's preferences. The preference of a PCD is that of the concrete service to which it is associated. Notice that the search space is segmented to be processed stepwise. At each step, the iterator computes sets $\mathcal{P}$ of PCD such that: (i) they have the same rank on the lexicographical order and (ii) for each $\mathcal{P}$ there is only one PCD from each coverage domain. In line 5, function GetNextSetOfPCD returns one of these sets $\mathcal{P}$, which, in turn will be passed to the last phase of the refinement method. In line 6 the combination of the PCD in $\mathcal{P}$ is verified and possibly published.

The rationale for the search space segmentation is to reduce the number of produced combinations. Indeed, Refinement1 generates all possible rewritings by testing all possible PCD's combinations. In this new approach the algorithm may exit before exhausting all the possibilities, emphasizing our goal of avoiding the cost of a combinatorial explosion.

The following sub-sections detail our proposal.

4.1 Partial Coverage Descriptors

Given the abstract composition defined as
$C(\ldots) \equiv_{def} A_1(\ldots), \ldots, A_n(\ldots), Q_1(\ldots), \ldots, Q_m(\ldots).$
and each concrete service S, defined as
$S(\ldots) \equiv_{def} A_i(\ldots), \ldots, A_j(\ldots), Q_k(\ldots), \ldots, Q_l(\ldots).$
our algorithm tries to match some services on the right-hand side of the definition of each concrete service with the same abstract services appearing on the right-hand side of C. This matching consists in producing a (semantic) mapping to make their parameters compatible.

For each possible matching, a tuple containing the mapping information will be produced. Each of these tuples is a PCD. The definition of PCD is based on the MiniCon Descriptions in (Pottinger and Halevy, 2001). It was adapted in (Costa et al., 2013) to the context of web services. Intuitively, a PCD

$\langle S, h, \varphi, G, Def, has\_opt \rangle$ defines how a service can be part of a concrete composition that meets the abstract specification. S is the name of the concrete service involved in the matching. A mapping φ defines the correspondence between the terms appearing on the abstract composition and terms that appear on the concrete service definition. For example, given $C(\ldots) \equiv_{def} A_1(\ldots), \ldots, A_i(x,y), \ldots, A_n(\ldots), Q_1(\ldots), \ldots, Q_m(\ldots)$ and $S(\ldots) \equiv_{def} \ldots, A_i(u,v), \ldots$ we have that $\varphi(x) = u$ and $\varphi(y) = v$. The mapping h corresponds to the head homomorphism in (Pottinger and Halevy, 2001). For example, given $C(\ldots) \equiv_{def} A_1(\ldots), \ldots, A_i(x,x), \ldots, A_n(\ldots)$ and $S(u,v) \equiv_{def} \ldots, A_i(u,v), \ldots$ we need to equate u and v (*i.e.*, $h(u) = u$, $h(v) = u$), so that φ may be defined as a function (*i.e.*, $\varphi(x) = u$). G is a set of abstract service names and quality constraints covered by S. The set Def contains conditions that cannot be guaranteed by S alone. The flag has_opt indicates the use of an optional parameter.

Given an abstract composition C, the algorithm below builds PCD and stores each one in the index I.

Algorithm 2 . (Building and storing PCD)

```
procedure BuildStorePCD(C, 𝒜, I)                                   1
  for each A_i in C do                                             2
    for each S ∈ 𝒜_i do                                            3
      // where 𝒜_i is the coverage domain of A_i                   4
      if there are mappings h and φ for A_i in the                 5
                    definitions of C and S then                    6
        G := {A_i}; Def := ∅;                                      7
        PCD := ⟨ S, h, φ, G, Def, has_opt ⟩;                       8
        AS := {A' | A' is an abstract service or                   9
          quality constraints in C sharing parameters              10
          with A_i or with other elements of AS}                   11
        PCD_OK := true;                                            12
        while AS≠ ∅ and PCD_OK do                                  13
          A' := choose an abstract service from AS;                14
          if ( h, φ can be extended to cover A') then              15
            Update PCD w.r.t. h, φ, G, Def, has_opt                16
            AS := AS − A';                                         17
          else PCD_OK := false;                                    18
          end if                                                   19
        end while;                                                 20
        if PCD_OK then StoreInIndex(PCD,S,𝒜_i)                     21
end if
      end if                                                       22
    end for                                                        23
  end for                                                          24
end procedure                                                      25
```

In line 5, parameters appearing on the left-hand side of a composition should only be mapped to parameters appearing on the left-hand side of concrete service specifications or optional ones. Then, Algorithm 2 looks for other abstract services or quality constraints connected to A. Thus, in line 9, the set AS contains all abstract services or quality constraints of C that *(i)* have a data dependency to A and *(ii)* are not mapped by φ to the parameters of S.

Example 2 . *Let us consider the composition LookForBooks specified in Example 1. Assume the following concrete service specifications:*

PickAndPayBooks(T?, I?, M?, A!, P!, K!) $\equiv_{def}$
BookStore (T?, I?, M?, A!, P!, V!) ,
Payment (T?, V?, M?, K!).
Amazon(T?, I?, M?, A!, P!, V!) $\equiv_{def}$
BookStore(T?, I?, M?, A!, P!, V!)
Visa(T?, V?, M?, K!) $\equiv_{def}$ *Payment(T?, V?, M?, K!).*

The PCD $D_1 = \langle PickAndPayBooks, h_1, \varphi_1, \{BookStore, Payment\}, \emptyset, \textbf{false} \rangle$, *where* h_1 *is the identity function and* φ_1 *maps terms Tkn?, Isbn?, Loc?, Addr!, Price!, Invoice!, Ack! to terms T?, I?, M?, A!, P!, V!, K!, respectively, is built by Algorithm 2 since:* (i) *the parameters appearing on the left-hand side of the composition are mapped by* φ_1*, and* (ii) *Invoice is a term that appears only on the right-hand side of the definition of LookForBooks and is mapped by* φ_1 *to V in PickAndPayBooks.*
The inclusion of Payment in D_1 *is explained by the fact that* $\varphi_1(Invoice)$ *is a term which does not appear on the left-hand side of PickAndPayBooks and acts as output parameter of BookStore and input parameter of Payment. Thus,* D_1 *can be used in the rewriting of LookForBooks to cover BookStore and Payment.*

Two other PCD can be built: $D_2 = \langle Amazon, h_2, \varphi_2, \{BookStore\}, \emptyset, \textbf{false} \rangle$ *and* $D_3 = \langle Visa, h_3, \varphi_3, \{Payment\}, \emptyset, \textbf{false} \rangle$ *where* φ_2 *and* φ_3 *are the intuitively expected mappings. Since these two PCD are feasible, then our approach can propose a rewriting of LookForBooks with service Amazon for BookStore and Visa for Payment.*

Algorithm 2 detects two situations where the construction of PCD is not possible (*i.e.*, when nothing is stored in I): (i) In line 5, if no match between services S and A exists and (ii) in line 15, if no match between services S and A' exists. In the second situation, as $A' \in AS$, S must cover the abstract service A'.

When the construction of a PCD is possible, it is stored in I. For each coverage domain $\mathcal{A}$ of an abstract service $A \in G$, the computed PCD is inserted in I, associated to the concrete service $S \in \mathcal{A}$. After the execution of *Algorithm 2*, the index I contains all possible PCD, computed on the basis of a composition C and organised according to an user's preferences.

4.2 An Iterator

Our method proposes an organiser, an index-like structure, which is built for each abstract composition, according to the user's preferences and the cov-

erage domain order. An iterator traverses this organiser in the decreasing lexicographical order and analyses, progressively, each concrete service combination susceptible to be a composition rewriting. The iterator avoids building useless compositions by respecting the following rules: (1) A concrete service can take part in a composition iff it does not cover a domain already covered by previous services. (2) A coverage domain is not visited if it has already been covered by a service in the composition.

The iterator generates each set $\mathcal{P}$ of PCD involved in a generated composition. At each iteration, we obtain all the sets $\mathcal{P}$ corresponding to the same preference. The function GetNextSetOfPCD in Algorithm 1 returns one of these sets to be possibly completed by the last step of our method.

Figure 1 illustrates the three-level structure used to implement our index. The first level sets the order of coverage domains. The second level is organised according to the concrete service scores. For instance, in Figure 1, concrete services in $\mathcal{A}_0$ having the highest weight are found in the slot labelled $rank_{0.0}$ while those having the lowest weight are found in the slot labelled $rank_{0.m_0}$. In this figure, we suppose that for domain $\mathcal{A}_0$ we have m_0 ranks, while for $\mathcal{A}_1$ we have m_1 and so on. In the third level we use a hash table to store concrete services. More precisely, a hash function associates a slot to a concrete service and its PCDs are stored in the corresponding linked list.

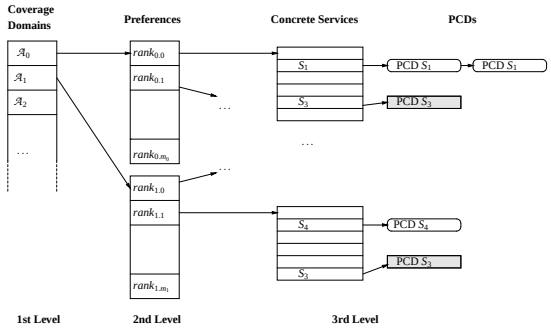


Figure 1: Organiser: to store PCD for concrete services according to the user's preferences.

4.3 Producing Refinements

Algorithm 3 is the last step of our approach. Its role is to verify whether a given set $\mathcal{P}$ of PCD can be correctly combined in order to produce a composition refinement. Algorithm 3 completes the composition according to imposed quality constraints and optional parameters, when they apply.

The input of Algorithm 3 is a set $\mathcal{P}$ containing k partial coverages for n abstract services. The number of PCD in $\mathcal{P}$ does not necessarily equal the number of abstract services, since a concrete service may cover more that one abstract service. The iterator guarantees that $\mathcal{P}$ respects the following properties: (i) PCD in $\mathcal{P}$ cover all the abstract services $A_1, \ldots, A_n$ of C and (ii) there are no overlapping between the abstract services covered by different PCD. The only allowable intersection between them is given by deferred quality constraints.

Algorithm 3 . (PCD Combination and Rewriting)

```
function ProduceRewriting(C, P)                                          1
  if P = {PCD_1, ..., PCD_k} is such that                                2
   (a) All deferred constraints in Def_1 ... Def_k hold;                 3
   (b) Input and output optional parameters should match.                4
  then                                                                   5
    Pre := ∅; Pos := ∅;                                                  6
    for each var. x ∈ Q_i such that Q_i ∉ G_1 ∪ ··· ∪ G_k do              7
      if x is an input parameter of C then Pre := Pre ∪ Q_i               8
      end if;                                                            9
      if x is an output parameter of C then Pos := Pos ∪ Q_i              10
      end if;                                                            11
    end for                                                              12
    return ⟨Pre⟩ C'(EC(t̄)) ≡_def S_1(t̄_1), ..., S_k(t̄_k) ⟨ Pos ⟩;          13
  end if                                                                 14
end function.                                                            15
```

Algorithm 3 tries to cover the definition of the abstract composition C, by verifying whether, for partial coverages in $\mathcal{P}$, the following conditions hold: (a) The deferred quality constraints appearing in the PCDs must hold when their variables are instantiated using the mappings of the PCDs (line 3). (b) Each term in C mapped to an optional output parameter (inside the definition of S_i) can only be mapped to optional input parameters (inside the definition of any concrete service) (line 4).

If the combination of PCD in $\mathcal{P}$ satisfies the conditions above, one concrete composition is produced. The refined composition is returned (line 13) with its pre- and post-conditions. These conditions cannot be statically verified and need to be checked at runtime by the generated concrete composition. They represent quality constraints of the abstract composition that are not ensured by the concrete services (since they have a dynamic nature). An example is the condition $Ack =$ "*OK*" of Example 1.

The following example covers all the aspects of our refinement algorithm.

Example 3 . *Given the specification of the composition of Example 1, suppose that we have defined the following coverage domains obtained from the available concrete services and user preferences:*

***Bookstore**: Amazon (0.9), BarnesAndNoble (0.85).*

***Payment**: Visa (0.4) and MasterCard (0.35).*

***Authentication**: OrangeAuth (0.8), YahooAuth (0.7), Twitter (0.65), Facebook (0.6).*

According to Algorithm 1, the first call to the function in line 5, will return a set of PCD built on the concrete services Amazon, Visa and YahooAuth. Notice that, although OrangeAuth is preferred over YahooAuth, the service cannot be used in the composition since it uses the SAML protocol, instead of REST, which is required by the specification. In this way, no PCD is generated on this service. Thus, our procedure generates the following refinement:

$$LookForBooks'(Uid?, Pwd?, Isbn?, Loc?, Price!, Addr!, Ack!)$$
$$\equiv_{def} \quad YahooAuth(Uid?, Pwd?, Tkn!, Prot!, Form!),$$
$$Amazon(Tkn?, Isbn?, Loc?, Addr!, Price!, Invoice!),$$
$$Visa(Tkn?, Invoice?, Ack!), Ack = \text{“}OK\text{”}.$$

In the case in which further solutions are sought, our algorithm will continue to produce refinements with less preferred services.

Notice that Refinement1 modifies MiniCon by imposing more restrictions (and thus avoiding some rewritings). We can see that the solutions obtained by our algorithm are functionally correct. This claim is supported by the results in (Pottinger and Halevy, 2001) and the following facts: (1) Algorithm 2 builds PCD for each concrete service in $S \in \mathcal{A}_i$. Each PCD is designed to cover part of the abstract composition. It matches the parameters of the abstract composition with those of the concrete service and defines substitution functions to maintain that matching during the (future) execution of the service. (2) Algorithm 3 selects those combinations of PCDs that covers the entire abstract composition. The parameter substitutions are combined to keep track of the data exchanged by the concrete services.

5 OUR METHOD IN PRACTICE

Our iterator proposes concrete service combinations, which implies an exponential time complexity (in the number of PCDs generated by BuildStorePCD in Algorithm 1). This is due to the combinatorial nature of the problem, also faced by the original MiniCon and our refinement method. As, in the worst case, one can have $(n.M.m)$ PCDs, the complexity is $O(n.M.m)^n$ where n is the number of abstract services in the composition, m is the maximal number of abstract services in the specification of a concrete service and M is the number of concrete services. The worst case of our algorithm occurs when all services have the same preference. In this case, the complexity of our method if the same as that of MiniCon. Our solution prunes the search space by introducing the notion of coverage domains and by ordering solutions in accordance to the user preferences. As coverage domains offer a semantic classification of services, in general, a PCD does not belong to all the coverage domains. Our index structure organises PCDs according to coverage domains, so that there is no need to build all PCD combinations. By building solutions (concrete compositions) in a decreasing order of user preferences the refinement process can stop when the desired number of compositions is reached.

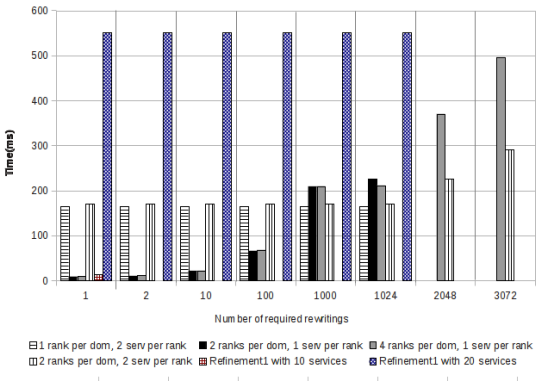


Figure 2: Execution time of our method *w.r.t.* nb of required (desired) rewritings. Varying nb of ranks and services: 10 or 20 (1 or 2 ranks) or 40 (2 or 4 ranks) services. Execution time of Refinement1 method computing all possible solutions for 10 and 20 services.

We have implemented a prototype of our method in Java. Figure 2 shows the average time from 60 executions on a Intel Quad-Core 2.70GHz, 2GB over synthesized services. Its goal is to show the result of synthesized compositions, done in order to evaluate the behaviour of our method in some bad situations.

Consider a composition with 10 abstract services and a varying number of concrete services defined by just one abstract service. Each concrete service covers at most one abstract service of our composition. In this case, a great number of rewritings is possible and our new proposal offers a considerable time saving when compared to our previous work.

We have run tests with 20 and 40 concrete services equally distributed over the 10 coverage domains (Figure 2). These tests have been chosen in order to illustrate the worst cases in terms of complexity. With only one preference rank all compositions have the same priority. For instance, with 40 services, 4 per rank, our algorithm behaves like Refinement1 and we have 4^{10} compositions with the same priority. Improvements are reported in situations where different priorities are assigned to services. For instance, Figure 2 shows that for 20 services (2 ranks per do-

main) we need 8.53*ms* and 21.5*ms* to offer 1 and 10 required rewritings, respectively. A similar execution time is needed for 40 services, 4 ranks per domain, since in both cases we have 1 service per rank. Notice that before executing ProduceRewriting (Algorithm 1), the iterator generates all the combinations having the same priority. In this case, even if one solution is required, all of them are published. In Figure 2, for 40 services, 2 ranks per domain, we have 2^{10} rewritings to propose.

Figure 2 also shows the running time for Refinement1 when dealing with 10 and 20 services. For 10 concrete services only one solution exists: both methods have the same performance ($\approx 9ms$). For 20 services, since there are 2 services per domain, Refinement1 computes all the 2^{10} rewritings in 550*ms*.

With our method we can compute the 2^{10} rewritings in about 170*ms* when we have one rank per domain and 2 services per rank. In this case all the solutions are computed in one shot as they have the same priority. In the case with 2 ranks per domain and one service per rank, our method needs about 5*ms*, 70*ms* and 210*ms* to compute one solution, 100 solutions and all the solutions, respectively. Notice that the time generation of the 1024 solutions in the first case is slightly smaller compared to the latter case as the iterator computes at each step only one rewriting (no two rewritings have the same priority). This difference illustrates the small price introduced by our priority-oriented method.

6 CONCLUSIONS

Given the abstract specification of a composition, our method produces combinations of services available in the Cloud, in order to refine the specification. The algorithm presented here extends improves our previous work in (Costa et al., 2013) by classifying solutions according to an user's profile. This new proposal generates solutions in a preference order and avoids the production of a combinatorial number of rewritings. Our experiments show that efficiency has been greatly improved *w.r.t.* (Costa et al., 2013). An important perspective is to consider web intelligence techniques to better explore user's preferences.

REFERENCES

Alrifai, M., Risse, T., Dolog, P., and Nejdl, W. (2008). A scalable approach for qos-based web service selection. In *ICSOC Workshops*, pages 190–199.

Alrifai, M., Skoutas, D., and Risse, T. (2010). Selecting skyline services for qos-based web service composition. In *WWW*, pages 11–20.

Costa, U., Halfeld Ferrari, M., Musicante, M., and Robert, S. (2013). Automatic refinement of service compositions. In *Proceedings of the International Conference on Web Engineering (ICWE)*, pages 400–407.

Haddad, S., Mokdad, L., and Youcef, S. (2010). Selection of the best composite web service based on quality of service. In *ISSS/BPSC*, volume 177 of *LNI*, pages 255–266.

Lamparter, S., Ankolekar, A., Studer, R., and Grimm, S. (2007). Preference-based selection of highly configurable web services. In *Proceedings of the 16th international conference on World Wide Web*.

Lemos, F., Grigori, D., and Bouzeghoub, M. (2012). Adding non-functional preferences to service discovery. In Brambilla, M., Tokuda, T., and Tolksdorf, R., editors, *ICWE*, volume 7387 of *Lecture Notes in Computer Science*, pages 299–306. Springer.

Lin, N., Kuter, U., and Sirin, E. (2008). Web service composition with user preferences. In *ESWC*, volume 5021 of *Lecture Notes in Computer Science*, pages 629–643. Springer.

Mesmoudi, A., Mrissa, M., and Hacid, M.-S. (2011). Combining configuration and query rewriting for web service composition. In *ICWS*, pages 113–120.

Paiva Tizzo, N., Coello, J., and Cardozo, E. (2011). Improving dynamic Web service selection in WS-BPEL using SPARQL. In *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*, pages 864–871.

Pottinger, R. and Halevy, A. Y. (2001). Minicon: A scalable algorithm for answering queries using views. *VLDB J.*, 10(2-3):182–198.

Sandionigi, C., Ardagna, D., Cugola, G., and Ghezzi, C. (2013). Optimizing service selection and allocation in situational computing applications. *IEEE T. Services Computing*, 6(3):414–428.

Sohrabi, S. and McIlraith, S. A. (2010). Preference-based web service composition: A middle ground between execution and search. In *International Semantic Web Conference (1) - ISWC 2010*, volume 6496 of *Lecture Notes in Computer Science*, pages 713–729. Springer.

Zeng, L., Benatallah, B., Lei, H., Ngu, A. H. H., Flaxer, D., and Chang, H. (2003). Flexible composition of enterprise web services. *Electronic Markets*, 13(2).

Zhao, W., Liu, C., and Chen, J. (2011). Automatic composition of information-providing web services based on query rewriting. *Science China Information Sciences*, pages 1–17.

Collaborative, Dynamic & Complex Systems
Modeling, Provision & Execution

Vasilios Andrikopoulos, Santiago Gómez Sáez, Dimka Karastoyanova and Andreas Weiß
Institute of Architecture of Application Systems, University of Stuttgart, Stuttgart, Germany
{*vasilios.andrikopoulos, santiago.gomez-saez, dimka.karastoyanova, andreas.weiss*}*@iaas.uni-stuttgart.de*

Keywords: Collaborative, Dynamic & Complex Systems, Service Orchestration & Choreography, Pervasive Computing, Service Networks, Context-awareness.

Abstract: Service orientation has significantly facilitated the development of complex distributed systems spanning multiple organizations. However, different application areas approach such systems in domain-specific ways, focusing only on particular aspects relevant for their application types. As a result, we observe a very fragmented landscape of service-oriented systems, which does not enable collaboration across organizations. To address this concern, in this work we introduce the notion of Collaborative, Dynamic and Complex (CDC) systems and position them with respect to existing technologies. In addition, we present how CDC systems are modeled and the steps to provision and execute them. Furthermore, we contribute an architecture and prototypical implementation, which we evaluate by means of a case study in a Cloud-enabled context-aware pervasive application.

1 INTRODUCTION

Complex software systems involving multiple, independent partners/software components collaborating in order to achieve one or more goals find predominant application in the current IT landscape. Cases of such systems from different domains are for instance business applications targeting enactment of complex business transactions and service networks, scientific workflows providing one approach for scientific experimenting in eScience, and pervasive systems representing one flavor of ubiquitous computing. Based on our research work towards building support systems for the development and execution of such applications in these domains, we conclude that while all the above-mentioned application areas concentrate on creating complex systems with very specific features critical for the corresponding domain, there are requirements valid across all domains. Our experience also shows that synergies between these domains can be exploited and potential benefits realized through reuse of research results and available software systems.

In this respect, in this work we investigate the requirements towards software systems in the above mentioned application areas with the purpose of identifying overlaps and differences. As we are going to show, the overlaps are significant and the differences are mainly due to the special focus on critical aspects in each domain, and not because the solutions are not relevant in the other domains. Based on these findings we introduce the innovative notion of *Collaborative, Dynamic and Complex (CDC)* systems aiming to cover all identified requirements and allowing to apply already existing technologies and software systems. CDC systems exhibit the aspects of modeling, provision and execution.

The contributions of this work target enabling the modeling, provision and execution of CDC systems and can be summarized by:

- The synthesis of existing technologies and approaches from the service-oriented computing paradigm and beyond, into a new, unified type of Collaborative, Dynamic and Complex (CDC) systems.
- The specification of the architecture for a framework that supports the various aspects (modeling, provision and execution) of CDC systems.
- The realization of this framework into a prototype called CoDyCo, and its use for the evaluation of our approach based on a case study.

The remaining paper is structured as follows. Section 2 looks into different application areas that deal with relevant for this work systems in order to highlight their similarities and establish the minimum set

of requirements for our work. Section 3 presents our proposal for CDC systems and positions them with respect to existing approaches. Section 4 introduces the architecture of a CDC-supporting framework required for implementing CDC systems. Section 5 reports on the overview and current state of CoDyCo. Section 6 summarizes a case study we performed using CoDyCo for purposes of evaluation. Finally, Section 7 presents related works, and Section 8 concludes the paper.

2 MOTIVATION

In the following, we look into the areas of pervasive systems, service networks and scientific workflows. Our experience in research projects[1] shows, that despite the differences, the available approaches from these areas have many commonalities.

Pervasive Systems. Pervasive systems strive towards enabling the paradigm of ubiquitous computing and have been a subject of interdisciplinary research. Advances in pervasive systems have focused on the aspect of context-awareness, i.e. taking into account the context of physical and virtual entities, which is in fact a view of the physical environment, and the influence of the context on the applications the entities are using or participating in (Baldauf et al., 2007). A major requirement in these systems is the ability to adapt their behavior with respect to the context. Another major challenge is the optimization of the distribution of applications based on context data and resource consumption. The distribution of pervasive applications across multiple software system and hardware devices require their integration and coordination towards enabling a collaboration among participating devices and systems. Due to the dynamic characteristics of the environment of pervasive applications, with participants and devices appearing and disappearing constantly, supporting context sharing, adaptation, and scalability are particularly challenging. Since recently, Cloud Platforms in the scope of the Internet of Things and Smart Systems initiatives have been investigated from the point of view of enabling scalability, multi-tenancy and adaptability (Distefano et al., 2012). Later in this paper, we present a detailed description of a pervasive context-aware application for booking a taxi nearest to the location of a user based on the information provided by mobile devices available to the taxi drivers. We also use the taxi application for purposes of evaluation as a case study.

[1]For example SimTech http://www.simtech.uni-stuttgart.de/, S-Cube http://www.s-cube-network.eu/, 4CaaSt http://www.4caast.eu/, ALLOW Ensembles http://www.allow-ensembles.eu/.

Service Networks. Service Networks (SNs) (Caswell et al., 2008) are considered a specialized view on business processes, which focus on assisting business experts to evaluate the value of participating in a collaborative business activity. SNs are modeled as a network of business services exchanging offerings; basically, the composite perceived value of the exchanged offerings with the other services determines the value of participating in the network to one participant. Typical examples of SNs are supply chains; the taxi application described later can also be viewed as an SN. There is a significant gap between the meta-models used by business experts when designing the SNs, and the technological realization that needs to be bridged by means of software engineering techniques like model-driven development and code generation, whereas both top-down and bottom-up approaches are required. In addition, service networks are inherently collaborative activities and therefore imply efforts towards integration of applications across organizations.

A SOA-based realization of service networks, as well as a meta-model and graphical notation are presented in (Danylevych et al., 2010) and (Bitsaki et al., 2008). The interoperability of service implementations in an SN have been addressed by means of Web services and the high-level meta-model has been mapped on choreographies of composite service (i.e. organization-specific business processes). Additionally, choreographies take over the role of coordinating the services in a network, which addresses another important requirement. Changes in the perceived value of a network to a participant may initiate changes in the individual partners or in the network as a whole, which have to be propagated to their technological realization. We therefore identify the need for adaptation of service networks; some preliminary attempts to support only some types of service networks adaptation (Wagner et al., 2012) are already available. Monitoring the value of an SN for a participant is not directly measurable, but can only be derived based on monitoring data provided by the execution environment for choreographies, orchestrations and services. Approaches based on business activity monitoring, like (Guinea et al., 2011) and (Wetzstein et al., 2012) are only first steps towards the necessary technological support.

Scientific Workflows. Scientific workflows enable the modeling and execution of scientific experiments

and are part of the technology landscape in eScience (Sonntag and Karastoyanova, 2010). A major requirement in this field is first and foremost the user friendliness of the approach, so that scientists do not face a high learning burden when using the experiment modeling tools. The division between the way scientists model an experiment and the meta-models used in the supporting IT systems is significant and there are different approaches towards eliminating it (Sonntag and Karastoyanova, 2010). Both top-down and bottom-up approaches are required to enable the use of existing software and the development of experiments from scratch. The distributed nature of complex scientific experiments requires integration and composition of scientific computing software, which presents an additional challenge due to the lack of clearly defined software engineering principles for building scientific applications, including scientific workflows. Reusability is hampered by the heterogeneous landscape of applications and integration is of high complexity, because of the large number of available techniques for composition. Since scientific discovery is based on exploring physical phenomena, huge amounts of data are collected via numerous types of mobile devices and sensors (e.g. simulations of the distribution of CO_2 in the soil, weather forecasts, biological system simulations, simulations of manufacturing systems, etc.), and need to be processed. Computations in scientific workflows are time-consuming and also distributed, and most often they do not exhibit characteristics of pervasive applications. Adaptation during the modeling and execution of scientific workflows is a must, as evidenced by existing work (Sonntag and Karastoyanova, 2010), (Sonntag and Karastoyanova, 2012).

Despite the different focus of the application systems described above, they all exhibit overlapping characteristics that can be leveraged in a unified manner across the various areas. The following section presents our proposal toward this goal.

3 CDC SYSTEMS

We define *Collaborative, Dynamic and Complex* (CDC) systems as distributed systems enabling collaboration among participants across different organizations. Participants of CDC systems are services, representing software systems of different granularity, virtual and physical devices, and individuals. CDC participants join and leave the system at will in order to fulfill their individual goals. CDC systems are capable of adapting with respect to different triggers in the system and/or in their environment. CDC systems consist potentially of a large amount of participants dealing with large amounts of data as part of multiple interactions between them, following one or more coordination protocols. CDC systems have three fundamental aspects: *Modeling*, *Provision* and *Execution*.

With respect to *modeling*, we use choreographies to define the high-level, domain-specific models of CDC systems. *Choreographies* describe the interaction protocol of the involved participants and the participant roles' definitions. In SOA environments, individual participant roles are implemented by service *orchestrations* exposed as services, whereas their service interfaces are compliant with the participant role definitions modeled in the choreography. The *services* composed by the orchestrations are either available in the software landscape of the participating organizations, or are discoverable in global service registries. Utilizing these SOA-based approaches provides a flexible way of composing applications in complex systems and facilitates application integration. To enable *context-awareness*, choreographies and orchestrations, as well as involved services, have to incorporate in their models context information and define its use and reaction to potential changes. Since context information may be part of correlation data of orchestrations belonging to an enacted choreography, a mapping between context and *correlation* mechanisms has to be in place.

Performance indicators, like KPIs, utility, value, etc. are an inseparable part of the CDC system models. On the one hand, they are used to define the indicators according to which users will measure and evaluate whether they achieve their goals in a collaboration. On the other hand, this is the information needed to derive the data to be monitored during the execution of the CDC system. Therefore choreographies, orchestrations and services models have to contain elements defining the necessary *monitoring* information. In order to enable the *dynamic features* of CDC systems, constructs accommodating *adaptation* mechanisms in the choreographies and orchestrations have to be incorporated. Available approaches from the fields of workflow adaptation, flexible scientific workflows and pervasive dynamic flows, e.g. (Wetzstein et al., 2012) or (Sonntag and Karastoyanova, 2010) can be applied individually or in combination. Change propagation across all levels of the CDC systems and thus adaptation of choreographies can be identified as a major research challenge.

As identified in Section 2, two types of approach in modeling are required: top-down and bottom-up. *Top-down* CDC system modeling entails starting the development of the system with a choreography representing a realization of a high-level (domain-

specific model), like a SN, scientific workflow, or pervasive application. Techniques required to map the choreography into orchestrations and services, like code generation and transformations, are available from software engineering and various existing SOA-enabling systems. The *bottom-up* approach involves deriving a meaningful choreography model based on existing orchestrations and/or services. In this case, deriving fault handling, monitoring and adaptation information is based on the corresponding capabilities of the involved services and correctness of the derived choreography.

The *provisioning* aspect of CDC systems entails the provision of the choreography, which also requires the deployment of orchestrations onto execution engines, their provisioning as services, populating the system with the corresponding context and correlation data, and configuring the monitoring infrastructure with the requirements from the CDC model. Mechanisms in service composition systems and scientific workflows for the provisioning of orchestrations and services are already available. Solutions for mapping monitoring requirements to monitoring probes are available in pervasive systems and service-based applications. The provision of a choreography results into adaptive and context-aware orchestrations available as a service. The choreography can be initiated multiple times for multiple interactions and can be started by any of the participating orchestrations or services, if they are allowed to do so by the choreography definition. Any underlying infrastructure should therefore enable sharing of resources across different CDC systems while correlating interactions to tenants and their users.

Running a choreography is therefore realized as a distributed *execution* of the collaboration among participating orchestrations and services. Since context-awareness is inherent to the CDC system model, the execution environment has to be able to support this property. Adaptation mechanisms, predefined in the system model (like abstract activities, binding strategies for services, reactions to context change, etc.) and such that are orthogonal to the model (like manual adaptation, forced termination, substituting a service endpoint, etc.) need to be realized by the execution environment. Furthermore, the execution environment of CDC systems must scale with their number of participants and their interactions, as well as the volumes of data exchanged. Monitoring information is necessary in order to enable such scaling.

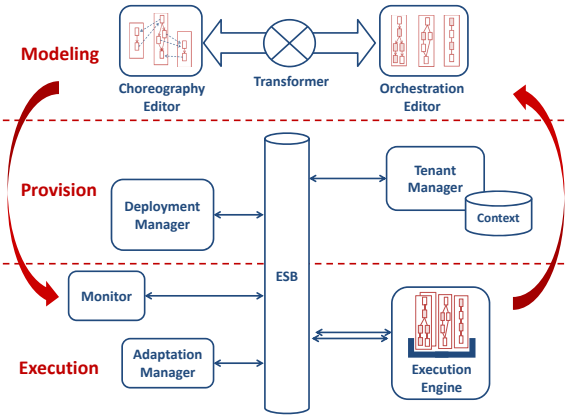


Figure 1: CDC-supporting framework — Architectural view.

4 CDC FRAMEWORK ARCHITECTURE

Figure 1 provides an overview of our proposal for a framework supporting the modeling, provision and execution of CDC systems. Starting from the modeling aspect, a *Choreography Editor* is required to create, visualize and manage the choreography models of the CDC systems. A *Transformer* component can then either generate orchestration templates that the CDC participants are meant to implement (in the top-down approach in the previous section), or derive possible choreographies from existing orchestrations (in the bottom-up approach). In either case, an *Orchestration Editor* (not necessarily but preferably in the same environment as the Choreography Editor) should be available for orchestration visualization and manipulation. The transformer components also requires as input the service descriptions of the used orchestrations in the bottom-up approach or generates (abstract) service descriptions for derived orchestrations. Moving to the provisioning aspect, the *Deployment Manager* allows the assignment of the necessary for operation computational resources to the orchestrations involved in the modeled choreographies. Beyond physically deploying the necessary artifacts on an *Execution Engine*, this additionally entails the creation of all service endpoints necessary for accessing the orchestration logic by the system participants. The Deployment Manager also handles the information needed for late and dynamic binding to concrete service endpoints and provides it to the ESB during the execution of orchestrations.

In principle, multiple organizational domains may be using the same instantiation of this framework for different CDC systems. It is therefore necessary to

offer *multi-tenancy capabilities* out of the box for all components *in the provision and execution aspect* of the framework. A *Tenant Manager* is responsible for this role, and implements administration and management capabilities for existing and new *tenants* (organizational domains) and their *users* (individuals or sub-systems in the same domain). The Tenant Manager is also meant to implement access control to both choreography and orchestration models, and to the computational resources corresponding to them during the execution of CDC systems, as assigned to them by the Deployment Manager. Only authorized parties should be allowed, for example, to participate in a given choreography. Furthermore, any collected *contextual information* relevant for tenants and users in terms of a representation of their environment, e.g. their physical location or the quality of observed data, is stored and accessed through the Tenant Manager.

While the Deployment and Tenant Managers play prominent roles in the provision aspect of CDC systems, they are also heavily involved during CDC system execution, since both of them need to interact with the actual *Execution Engine* that runs the orchestrations defined during modeling. Furthermore, the Execution Engine has to provide fault handling capabilities, both for pre-defined fault and compensation handlers in the orchestration models, and for failures during execution like service failures and unavailability of other components in the framework (e.g. access to the Deployment Manager).

The *Adaptation Manager* is responsible for triggering and managing the adaptivity features of CDC systems by providing mechanisms for different types of adaptations across the levels of the systems. It implements and/or coordinates the actions necessary to enable the adaptation constructs from the CDC system model and the ones implemented only on the level of the execution environment. The Adaptation Manager collaborates also with the Deployment Manager when necessary, e.g. for re-binding service endpoints, and with the Execution Engine, e.g. for injecting a new activity and control connectors into an existing orchestration or deploying a new orchestration in case a choreography has been changed. The Adaptation Manager acts on information provided by the *Monitor* component which monitors and analyses the behavior and performance of the executed orchestrations, of the enacted choreographies, and also of the execution components in the framework. The Monitor must be configurable based on the monitoring information required for the CDC system and is responsible for providing to the users of choreographies and orchestrations personalized views of the relevant monitoring information on their devices.

Leveraging the SOA paradigm, all components in the framework relevant to execution should be provided as services and communicate through an *Enterprise Service Bus (ESB)* solution to facilitate their integration. Furthermore, each component should be designed and implemented allowing for both types of scalability: *horizontal* (modify number of available instances as required) and *vertical* (adjustment of available computational resources for each component) (Vaquero et al., 2011).

5 IMPLEMENTATION

In this section we present *CoDyCo*, a realization approach of the CDC-supporting framework presented in the previous section. As shown in Fig. 2, two separate environments are distinguished in the CoDyCo architecture: a *Modeling and Monitoring Toolset*, and a *Runtime Environment*.

Starting from the Modeling and Monitoring Toolset, CDC choreographies are specified in the BPEL4Chor language using our *BPEL4Chor Designer*. BPEL4Chor was first introduced by (Decker et al., 2008) as an extension of the WS-BPEL language (OASIS, 2007) and stemming from the business transactions field. However, BPEL4Chor choreographies are by definition not executable and therefore the transformation of BPEL4Chor definitions to WS-BPEL process (orchestration) skeletons (Reimann, 2007) is supported in CoDyCo by a *BPEL4Chor Transformer* (Fig. 2). Based on the choreography topology, the participants' grounding definitions, i.e. their WSDL interfaces, and their message links, this transformation generates the executable BPEL orchestrations for each participant in the choreography. The skeletons only model the interactions between partners so that together they can enact the choreography. Manual refinements can be performed on the created orchestrations, using our Mayflower BPEL Designer (Sonntag et al., 2012) developed in the context of the SimTech project[2] as an Eclipse-based BPEL designer. These refinements allow defining specific process logic for each participant, for example by reusing predefined process fragments from a process fragment library, as demonstrated in (Sonntag et al., 2012) and (Schumm et al., 2010). Both choreography definition and transformation functionalities are wrapped as an Eclipse Graphical Editor, and provide a palette with the graphical elements of the choreography language (Weiß et al., 2013). Only the top-down modeling approach (see

[2]The SimTech project: http://www.simtech.uni-stuttgart.de

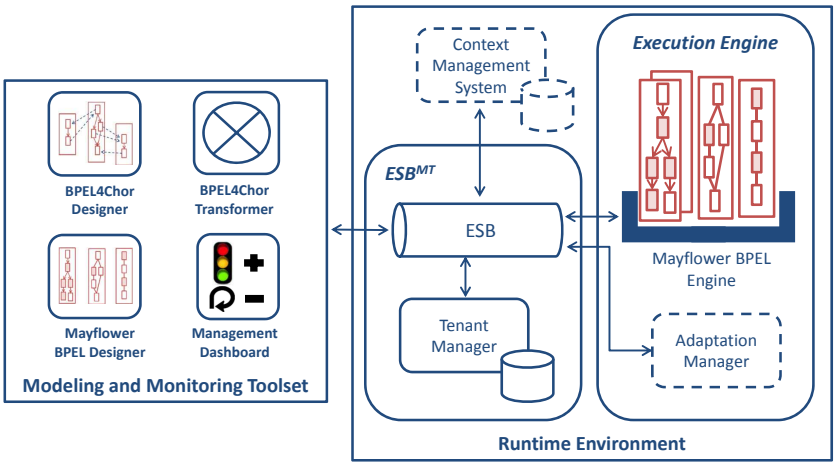


Figure 2: The CoDyCo System: Overview and Current State.

Section 3) is currently supported by the Modeling and Monitoring Toolset; supporting the bottom-up approach is part of our future work. For more information on the status of the tools the interested reader is referred to (Weiß et al., 2013).

The deployment and instantiation of the BPEL processes generated by the Modeling and Monitoring Toolset is done on our Mayflower BPEL Engine. This engine is an extended version of the open-source Apache ODE Engine. It provides an interface for event publishing and configurable filtering and a BPEL event model[3] (Khalaf et al., 2007), which have been specialized for the purposes of monitoring and triggering dynamic adaptation of process instances(Sonntag et al., 2012). Functionalities provided by the Mayflower BPEL Engine address the requirements of the CDC execution environment in terms of orchestration execution, storing of historical information, failure and compensation handling, and in combination with the Adaptation Manager also enable some dynamic adaptation patterns. For instance, dynamic adaptation of process instances triggered by humans are supported in CoDyCo through the *Modeling and Monitoring Toolset*. More specifically, the Mayflower BPEL Designer interacts with the Mayflower BPEL Engine and the Adaptation Manager and so allow the users to view the status of process instances and trigger their adaptation. Based on this users can adapt the process instance by changing its graphical representation.

The adaptation operations that can be performed on a process instance in our current implementation are, e.g. re-execution or forced iteration of activities (Sonntag and Karastoyanova, 2012), insertion and deletion of process elements, or their substitution. The changes made on the viewed process instance are then propagated to the Adaptation Manager Component, which is responsible for performing the actual adaptation on the concrete process instance in the engine. This is also made possible by auxiliary functions in the Mayflower BPEL Designer, mirrored in the Mayflower BPEL Engine, like enabling user subscriptions to monitoring events published by the engine, and hence retrieval of real-time information about a concrete process instance, and built-in actions per process instance like stop, suspend, resume, etc. (Sonntag et al., 2012). Automatic adaptation of running process instances, like injection of process fragments is currently not supported by the Adaptation Manager in CoDyCo [4].

Communication between participants collaborating in the scope of a specific choreography instance, and between the different components comprising the runtime environment is established through the multi-tenant open source ESB solution *ESB^{MT}* (Strauch et al., 2012a), (Strauch et al., 2012b). ESB^{MT} enhances an existing ESB Solution, Apache ServiceMix [5] with multi-tenant awareness both at the administration and management, and messaging levels. Management and administration functionalities implemented by a *Tenant Manager* enable the dynamic deployment and configuration of service endpoints with tenant- and user-specific information, while tenant-aware messaging capabilities isolate tenants' messages routed to the service endpoints (Strauch et al., 2012b). As tenants represent organizational units, e.g. Taxi Company A, which has N taxi drivers,

[3]ODE PGF: http://www.iaas.uni-stuttgart.de/forschung/projects/ODE-PGF/

[4]Note that this is possible only manually, as described above.

[5]Apache ServiceMix: http://servicemix.apache.org/

and M potential taxi customers, their communication in the scope of a choreography is supported in the ESB^{MT} through tenant-aware service endpoints. At the moment we are working on the integration of our execution engine with the ESB^{MT} as a JBI Service Engine, to enable multi-tenancy support for orchestrations and choreographies, too.

The Management Dashboard is the component in the Modeling and Monitoring Toolset responsible for providing analyzed and personalized monitoring information to users and tenants about the execution state of choreographies and orchestrations. Our prototype visualizes the execution status of orchestrations, the data they produce and consume, and the adaptations that have been performed (Sonntag et al., 2012). This is possible due to the interaction of the Management Dashboard with the Mayflower BPEL Engine via its event publishing interface, as described above. Our approaches for monitoring choreographies (Wagner et al., 2012) and KPIs (Wetzstein et al., 2012) are not yet integrated into CoDyCo.

The *Context Management System* stores the system context (i.e. a set of context properties of all active tenants) and constantly synchronizes its current configuration with the tenant specific applications by retrieving context data from each of the tenant users created in the system, e.g. from a location provider embedded system in a specific taxi cab. Context information is used in the execution of choreographies by the corresponding orchestrations through *Context Integration Processes* (CIPs) (Wieland et al., 2007) and may trigger their adaptation. Context-aware adaptations will be handled by a collaboration of the *Adaptation Manager*, *the Mayflower Engine*, the *Context Management System (CMS)*, and the *Tenant Manager* and is part of our future work on the implementation. The modeling tool is currently also missing features supporting modeling of context in the choreographies.

6 CASE STUDY

In the scope of project 4CaaSt[6], the *Taxi Scenario* use case has been defined, where a service provider offers a taxi management software as a service to different taxi companies, i.e. tenants. Taxi company customers, who are the *users* of the tenant, submit their taxi transportation requests to the company they are registered with. The taxi company uses the taxi management software to contact nearby taxi drivers. Once one of the contacted taxi drivers has confirmed the transportation request, the taxi management software sends a transport notification containing the estimated arrival time to the customer. As discussed in Section 2, the Taxi Scenario constitutes a pervasive context-aware application and therefore, in the scope of this work, an ideal candidate for the evaluation of our proposal.

Figure 3 shows the processes the Taxi Scenario application realizes. The simplified BPMN diagram has three lanes depicting the three participants of the application choreography: Customer, Taxi Company, and Taxi Service Provider. If a Customer wishes to book a Taxi, he sends an initial request to the Taxi Company call center (usually through a Web GUI), which forwards it to the Taxi Service Provider. The Taxi Service Provider process determines the nearby available taxis and the contact information of the taxi drivers using CIPs (Wieland et al., 2007) (not shown here for brevity). Subsequently, the transport request is sent to each available taxi driver, and their responses are collected for a specified duration. The gathered transport information is sent back to the Customer. Implementing the Taxi Scenario required the manual design of all involved processes as orchestrations for each participant and their interactions, as well as the implementation of most services involved in them (except from those that already existed, e.g. a context provisioning framework exposed as a service (Knappmeyer et al., 2010)).

For the evaluation of our approach we started with the BPMN diagram in Fig. 3 which we translated into BPEL4Chor. Figure 4 shows the processes depicted in Fig. 3 as participants in our BPEL4Chor Designer. The rectangular shapes in the editor view in Fig. 4 stand for the choreography participants, whereas the message links and their directions are depicted by labeled arrows. The set of taxi drivers are represented by the Taxi Transmitters participant, standing for the devices carried by the drivers. Inside each participant, the control flow regarding its communication behavior such as receive or send activities is visible. The resulting choreography model was then transformed into a series of BPEL4Chor artifacts by the BPEL4Chor Designer: a participant topology specifying the involved participants in the choreography, the participant types, and the message links between participants.

The BPEL4Chor artifacts are used by the BPEL4Chor Transformer to generate Abstract BPEL processes and WSDL files. These WSDL files contain the technical information about the interfaces between the participants, i.e. the port types, operations, messages, and partner links. Each previously modeled participant was transformed into exactly one Abstract BPEL process. Basic executable completion of

[6]The 4CaaSt project: http://www.4caast.eu

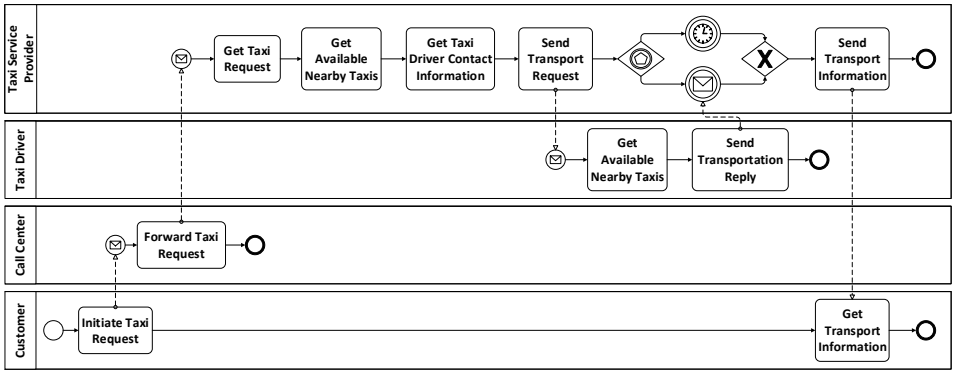


Figure 3: The BPMN process diagram of the Taxi Application.

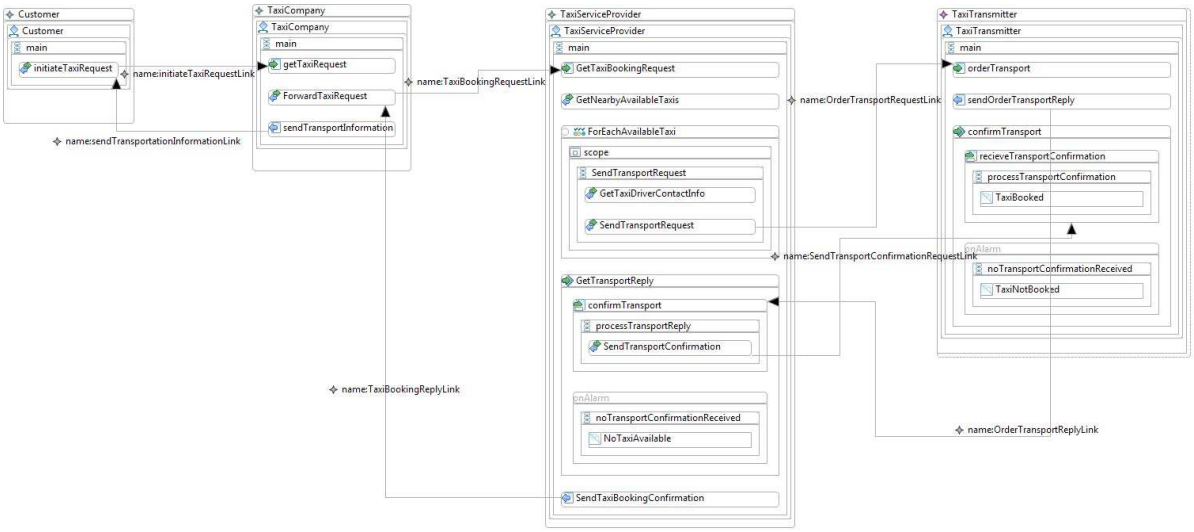


Figure 4: Excerpt of the Taxi Application choreography modeled with the Chor Designer.

the Abstract BPEL processes, i.e. their transformation into executable ones, is supported by the BPEL4Chor Designer, as well as the manual refinement of the process logic in each participant that is not part of the choreography. The resulting (executable) processes were then deployed and executed successfully in our Mayflower BPEL Engine. Through this process we have also identified a number of technical issues with the current implementation of the BPEL4Chor Designer and Transformer (Weiß et al., 2013). For example, not all BPEL activities are currently supported, and the generated WSDL files need manual definition of the involved message types. We are already working on addressing these deficiencies.

7 RELATED WORK

As discussed in (Barker et al., 2009), the interaction between participants in a choreography can be modeled following the interaction, or interconnection modeling approaches. The former approach models atomic interactions between participants through *interaction activities*, while the latter interconnects the communication activities of each participant of the choreography. The WS-CDL[7] language standard supports the interaction approach. Using the WS-CDL language as the basis, the Savara[8] project aims to provide tooling support for a top-down choreography modeling approach. Interconnection modeling approaches are supported in the CHOReOS In-

[7]WS-CDL:http://www.w3.org/TR/ws-cdl-10/
[8]http://www.jboss.org/savara

tegrated Development and Runtime Environment[9], in the Open Knowledge European project[10], and in BPEL4Chor (Decker et al., 2007). The CHOReOS environment supports the choreography specification using BPMN 2.0 collaborations (CHOReOS Consortium, 2011), and encompasses choreography adaptation based on service availability and QoS assurance.

The Open Knowledge framework employs a multiagent protocol to control the interactions between participants in the choreography. Therefore, participants must be specified and deployed prior to the choreography enactment, and adaptation based on context modifications is not considered. As discussed in the previous sections, BPEL4Chor wraps the choreography specification in a layer atop of WS-BPEL which contains the choreography control flow, its participants description and message links between them, and the mapping support to their concrete communication descriptions (WSDL). BPEL4Chor does not support the explicit specification of rules for context-aware adaptation purposes, but decouples the choreography specification from communication specific details, enabling extensibility for dynamic context-aware choreography adaptation.

Context-aware systems have been widely studied in the scope of Ubiquitous Computing. In (Baldauf et al., 2007) a set of context-aware systems are presented, and a comparison focusing on the architectural principles of context-aware middleware and framework to ease the development of context-aware applications is provided. The CoWSAMI middleware infrastructure utilizes Web services for managing location context in open ambient intelligence environments (Athanasopoulos et al., 2008). The utilization of an ESB as the central piece for communication support in context-aware systems is discussed in (Chanda et al., 2011), where a Context-aware ESB (CA-ESB) is proposed to discover and orchestrate services based on the users' location and available services in specific regions.

Concerning different context views in pervasive environments, in (Abdulrazak et al., 2010) micro and macro context-awareness modeling approaches are presented. The former describes the users' surroundings and aims to provide access to local context data, while the latter aggregates local context data to provide a global perspective of different spaces. Self-configuration operations in micro context-awareness models involve coordination of peers in a decentralized manner, making choreographies suitable for modeling the coordination between peers. Furthermore, in (Roy et al., 2008) high system availability is achieved by decentralizing the coordination of entities collaborating in context construction and decision making activities in open intelligence spaces ensures.

Context-aware workflows as an approach for easing the development of context-aware applications are presented in (Wieland et al., 2007). Thus, they propose Context4BPEL, a WS-BPEL[11] extension for explicitly modeling the influence of context on workflows. However, WS-BPEL supports orchestration of services within a business process, while choreography modeling approaches demand a further semantic support for specifying process interactions from a global view. Further research on workflow flexibility has been conducted by integrating support of human interactions during the execution of scientific workflows in (Karastoyanova et al., 2012). This approach triggers human interactions for non-automated activities via a framework supporting a multi-protocol communication between a scientific workflow management system and pluggable communication devices. All these approaches are focusing on only one particular aspect of CDC systems.

8 CONCLUSIONS AND FUTURE WORK

Our investigation into different application areas like pervasive systems, service networks and scientific workflow systems that have been influenced by service-orientation illustrated a series of overlapping characteristics that have not been leveraged so far. Toward this purpose, in this work we introduced the notion of Collaborative, Dynamic and Complex (CDC) Systems as dynamic distributed systems that allow participants from different organizations to collaborate to fulfill their goals. We discussed three fundamental aspects of CDC systems: modeling, provision and execution, and presented the architecture of a framework that supports these aspects. We then showed the current status of the prototypical implementation of CoDyCo, a system that realizes this framework. A case study on a context-aware pervasive application was then presented for purposes of evaluating our proposal.

Currently, we are working on improving the state of the CoDyCo prototype by addressing the deficiencies identified during the case study. Future work is aimed at finalizing the different aspects of our proposal. Concerning modeling and provision CDC systems, the bottom-up modeling approach has to be realized, as well as enabling context-awareness in

[9]EU Project CHOReOS: http://www.choreos.eu/

[10]Open Knowledge: http://www.openk.org/

[11]WS-BPEL 2.0: http://docs.oasis-open.org/wsbpel/2.0/

choreographies and orchestrations for both type of approaches. This also entails the realization of the context management system. In addition, multi-tenancy awareness has to be enabled for choreographies and orchestrations, and reflected in the Execution Engine. The Management Dashboard has to be integrated with approaches to monitoring KPIs and business transactions which is a step towards enabling the monitoring of choreographies. In terms of adaptation, available approaches context-aware adaptation and automatic adaptation of orchestration has to be integrated in CoDyCo. Finally, the scalability features of the CoDyCo components have to be investigated further in the scope of our Cloud computing research.

ACKNOWLEDGEMENTS

This work is funded by the projects FP7 EU-FET 600792 ALLOW Ensembles and the German DFG within the Cluster of Excellence (EXC310) in Simulation Technology.

REFERENCES

Abdulrazak, B., Roy, P., Gouin-Vallerand, C., Giroux, S., and Belala, Y. (2010). Macro and micro context-awareness for autonomic pervasive computing. In *Proceedings of iiWAS'2010*, pages 427–434. ACM.

Athanasopoulos, D., Zarras, A. V., Issarny, V., Pitoura, E., and Vassiliadis, P. (2008). Cowsami: Interface-aware context gathering in ambient intelligence environments. *Pervasive and Mobile Computing*, 4(3):360–389.

Baldauf, M., Dustdar, S., and Rosenberg, F. (2007). A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277.

Barker, A., Walton, C. D., and Robertson, D. (2009). Choreographing web services. *IEEE Transactions on Services Computing*, 2(2):152–166.

Bitsaki, M., Danylevych, O., van den Heuvel, W.-J., Koutras, G., Leymann, F., Mancioppi, M., Nikolaou, C., and Papazoglou, M. (2008). An Architecture for Managing the Lifecycle of Business Goals for Partners in a Service Network. In *Proceedings of ServiceWave'08*, Lecture Notes in Computer Science, pages 196–207. Springer Berlin Heidelberg.

Caswell, N. S., Nikolaou, C., Sairamesh, J., Bitsaki, M., Koutras, G. D., and Iacovidis, G. (2008). Estimating value in service systems: A case study of a repair service system. *IBM Systems Journal*, 47(1):87–100.

Chanda, J., Sengupta, S., Kanjilal, A., and Bhattacharya, S. (2011). CA-ESB: Context Aware Enterprise Service Bus. *International Journal of Computer Applications*, 30(3):1–8.

CHOReOS Consortium (2011). CHOReOS Dynamic Development Model Definition (D2.1).

Danylevych, O., Karastoyanova, D., and Leymann, F. (2010). Service Networks Modelling: An SOA & BPM Standpoint. *J.UCS*, 16(13):1668–1693.

Decker, G., Kopp, O., Leymann, F., Pfitzner, K., and Weske, M. (2008). Modeling Service Choreographies Using BPMN and BPEL4Chor. In *Proceedings of CAiSE'08*, pages 79–93. Springer.

Decker, G., Kopp, O., Leymann, F., and Weske, M. (2007). BPEL4Chor: Extending BPEL for Modeling Choreographies. In *Proceedings of ICWS'07*, pages 296–303. IEEE Computer Society.

Distefano, S., Merlino, G., and Puliafito, A. (2012). Enabling the cloud of things. In *Proceedings of IMIS'12*, pages 858–863.

Guinea, S., Kecskemeti, G., Marconi, A., and Wetzstein, B. (2011). Multi-layered monitoring and adaptation. In *Proceedings of ICSOC'11*, pages 359–373. Springer.

Karastoyanova, D., Dentsas, D., Schumm, D., Sonntag, M., Sun, L., and Vukojevic, K. (2012). Service-based Integration of Human Users in Workflow-driven Scientific Experiments. In *Proceedings of eScience'12*, pages 1–8. IEEE Computer Society Press.

Khalaf, R., Karastoyanova, D., and Leymann, F. (2007). Pluggable framework for enabling the execution of extended bpel behavior. In *Proceedings of WESOA'07*, pages 376–387. Springer.

Knappmeyer, M., Kiani, S. L., Frà, C., Moltchanov, B., and Baker, N. (2010). ContextML: a light-weight context representation and context management schema. In *Proceedings of ISWPC'10*, pages 367–372. IEEE.

OASIS (2007). Web services business process execution language version 2.0.

Reimann, P. (2007). Generating BPEL Processes from a BPEL4Chor Description. Studienarbeit 2100, Universität Stuttgart, Institut für Architektur von Anwendungssystemen.

Roy, P., Abdulrazak, B., and Belala, Y. (2008). Approaching context-awareness for open intelligent space. In *Proceedings of MoMM'08*, pages 422–426. ACM.

Schumm, D., Karastoyanova, D., Leymann, F., and Strauch, S. (2010). Fragmento: Advanced Process Fragment Library. In *Proceedings of ISD'10*, pages 659–670. Springer.

Sonntag, M., Hahn, M., and Karastoyanova, D. (2012). Mayflower - Explorative Modeling of Scientific Workflows with BPEL. In *Proceedings of BPM'12*, pages 1–5. CEUR Workshop Proceedings.

Sonntag, M. and Karastoyanova, D. (2010). Next Generation Interactive Scientific Experimenting Based On The Workflow Technology. In *Proceedings of MS'10*. ACTA Press.

Sonntag, M. and Karastoyanova, D. (2012). Ad hoc Iteration and Re-execution of Activities in Workflows. *International Journal On Advances in Software*, 5(1 & 2):91–109.

Strauch, S., Andrikopoulos, V., Leymann, F., and Muhler, D. (2012a). ESB^{MT}: Enabling Multi-Tenancy in Enterprise Service Buses. In *Proceedings of CloudCom'12*, pages 456–463. IEEE Computer Society.

Strauch, S., Andrikopoulos, V., Sáez, S. G., Leymann, F., and Muhler, D. (2012b). Enabling Tenant-Aware Ad-

ministration and Management for JBI Environments. In *Proceedings of SOCA'12*, pages 206–213. IEEE Computer Society.

Vaquero, L., Rodero-Merino, L., and Buyya, R. (2011). Dynamically scaling applications in the cloud. *ACM SIGCOMM Computer Communication Review*, 41(1):45–52.

Wagner, S., Fehling, C., Karastoyanova, D., and Schumm, D. (2012). State Propagation-based Monitoring of Business Transactions. In *Proceedings of SOCA'12*, pages 1–8. IEEE.

Weiß A., Andrikopoulos, V., Gómez Sáez, S., Karastoyanova, D., and Vukojevic-Haupt, K. (2013). Modeling Choreographies using the BPEL4Chor Designer: an Evaluation Based on Case Studies. Technical Report 2013/03, IAAS, University of Stuttgart.

Wetzstein, B., Zengin, A., Kazhamiakin, R., Marconi, A., Pistore, M., Karastoyanova, D., and Leymann, F. (2012). Preventing KPI Violations in Business Processes based on Decision Tree Learning and Proactive Runtime Adaptation. *Journal of Systems Integration*, 3(1):3–18.

Wieland, M., Kopp, O., Nicklas, D., and Leymann, F. (2007). Towards context-aware workflows. In *Proceedings of CAiSE07*, pages 11–15. Springer.

All links were last followed on January 29, 2014.

Choreography-based Consolidation of Multi-instance BPEL Processes

Sebastian Wagner[1], Oliver Kopp[1,2] and Frank Leymann[1]
[1]*IAAS, University of Stuttgart, Universitaetsstr. 38, Stuttgart, Germany*
[2]*IPVS, University of Stuttgart, Universitaetsstr. 38, Stuttgart, Germany*
{*wagnerse, kopp, leymann*}*@iaas.uni-stuttgart.de*

Keywords: BPEL, Choreography, Process Consolidation, Multi-instance Interactions.

Abstract: Interaction behavior between processes of different organizational units such as an enterprise and its suppliers can be modeled by choreographies. When organizations decide, for instance, to gain more control about their suppliers to minimize transaction costs, they may decide to insource these companies. This especially includes the integration of the partner processes into the organization's processes. Existing works are able to merge single-instance BPEL process interactions where each process model is only instantiated once during choreography execution. However, there exist different interaction scenarios where one process interacts with *several* instances of another process and where the number of instances involved is not known at design time but determined during runtime of the choreography. In this work we investigate these interaction scenarios and extend the process consolidation approach in a way that we can emulate the multi-instance interaction scenarios in the merged process model.

1 INTRODUCTION

To enable collaboration between companies, their processes have to interact with each other. The required interaction behavior can be specified with interconnection choreographies, where the communicating activities of the interacting business processes are connected by message links (Decker et al., 2008). In Figure 1 an interconnection choreography is shown where a travel agency queries a set of different airlines to check flight availability and price for the dates specified by a traveler (the traveler is not depicted in Figure 1). Then it selects the cheapest airline that has a flight available and orders a flight for the traveler.

In previous work (Wagner et al., 2011), a process consolidation approach has been proposed that merges all or a subset of complementing BPEL processes (OASIS, 2007) belonging to the same choreography into a new single process model. These processes can be either abstract or executable.

One motivation for the consolidation of interacting processes is that we want to reverse the fragmentation of BPEL processes proposed by Khalaf and Leymann (Khalaf and Leymann, 2006; Khalaf and Leymann, 2010). This fragmentation approach splits single BPEL processes into several interacting BPEL process fragments that keep the operational semantics of the original process. The consolidation of interacting process models can also lead to significant performance gains as the message transfers between processes are avoided and since the number of instances is decreased. Performance measurements conducted by Wagner et al. (Wagner et al., 2013b) have shown that the CPU load, required for executing a consolidated process that was created from a choreography consisting of 5 interacting processes, is reduced by 80%

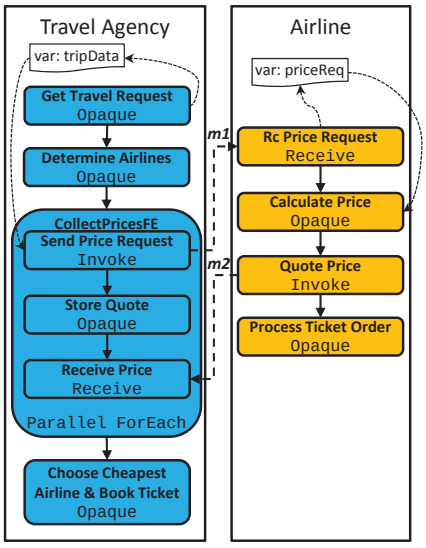


Figure 1: Example Choreography (adapted from (Decker et al., 2008)).

(compared to the CPU load required for executing the original choreography).

As cloud providers charge for the resources used by their customers (pay-per-use) these performance gains do also lead to reduced process execution costs when the processes are enacted in cloud environments. A more technical reason for consolidating interacting BPEL processes is that available BPEL workflow engines are just capable to deploy and execute business processes but cannot enact choreographies. After a choreography was consolidated into a single process, it can be executed on these workflow engines. So far, process consolidation has been investigated just for one-to-one interactions (Barros et al., 2005) (Wagner et al., 2013a), where for each process participating at the choreography just one instance is created during choreography execution. However, the travel agency interacts with a set of multiple instances of the airline process (one-to-many interaction). The set of airlines involved is unknown during the design time of the choreography as it is determined by the travel agency during runtime.

In this work, we will extend the process consolidation approach to emulate these one-to-many multi-instance interaction scenarios in the merged process model P_{Merged} without introducing new BPEL language constructs or additional middleware. We will also show how multi-instance interactions where the number of instances is not known at design time can be emulated by the consolidated process P_{Merged}. Thereby, the sequential relations between the business activities A_{BA} in P_{Merged} must be the same as the sequential relations between business activities of the original choreography $\mathcal{C}$. The set of business activities A_{BA} includes all activities that implement a certain business function. In Figure 1 business activities are marked with the label `opaque`. Preserving the sequential relation between the business activities ensures that the originally modeled control flow of activities performing the actual business logic in $\mathcal{C}$ is approximated as far as possible in P_{Merged} and that at least all control flow constraints between these activities are kept in P_{Merged}.

The consolidation approach described in this and previous work focuses on BPEL as workflow language due to the following reasons: BPEL is still the de facto language for executable workflows (Leymann, 2010) and widely used in industry. BPEL provides a well-defined operational semantics and a variety of theoretical models are available for this language (van Breugel and Koshkina, 2006). In contrast to BPEL, BPMN (Object Management Group (OMG), 2011) is still underspecified and contains ambiguities (Kossak et al., 2012; Börger, 2012; Wohed et al., 2006). The well-defined operational semantics of BPEL is required to create a consolidated and portable process model that can be executed on any BPEL workflow engine. Another reason why we are focusing on BPEL is that, in the long run, we want to be able to reverse the fragmentation of processes described by Khalaf et al. (Khalaf and Leymann, 2006; Khalaf and Leymann, 2010), where they focus on the fragmentation of executable BPEL processes.

The remainder of this paper is structured as follows. section 2 provides an overview on BPEL, the choreography language BPEL4Chor, and the consolidation of one-to-one interactions. The control flow properties of multi-instance interactions are discussed in section 3. We suggest the consolidation approach for multi-instance interactions in section 4. The results of the consolidation approach are discussed in section 5 and the prototype implementing the approach is presented in section 6. Related work is presented in section 7 before section 8 concludes this work and gives an outlook about future work.

2 PRELIMINARIES

Firstly, this section provides an overview about the choreography language BPEL4Chor and the language elements of BPEL relevant for realizing multi-instance interactions and their consolidation (subsection 2.1). Then an overview about the process consolidation approach for one-to-one interactions is given (subsection 2.2).

2.1 BPEL and BPEL4Chor

BPEL is a workflow language designed for enabling programming in the large (DeRemer and Kron, 1976) based on Web services. BPEL itself does not offer a capturing of multiple processes interacting with each other (Decker et al., 2009). This is enabled by BPEL4Chor (Decker et al., 2009), which provides a participant topology listing of all participants of a choreography and a list of message links linking communication activities of the participant behavior descriptions. As language for a participant behavior description, BPEL is used. BPEL enables describing the behavior of a participant without revealing the processes internal behavior. Thereby, the `opaque` activity allows to hide such behavior. It is replaced by actual business logic when the BPEL process is deployed and running (Aalst et al., 2008). A partner process is addressed by using correlation tokens or by an endpoint reference (Barros et al., 2007). In BPEL4Chor, a `forEach` activity may iterate on a set

of participant references to enable a parallel interaction with multiple process instances of the same type. The `forEach` activity iterates its child activity exactly N+1 times, where N equals the `finalCounterValue` minus the `startCounterValue` (OASIS, 2007). The flag `parallel` can be used to specify whether this iteration should happen in parallel or sequentially. A `completionCondition` may be used within the `forEach` to allow the `forEach` activity to complete without executing or finishing all the branches specified: Remaining branches are terminated when the `completionCondition` evaluates to true. Although BPEL allows graph-based modeling with links connecting the activities (Kopp et al., 2009), it has the constraint, that links may not cross the boundary of a `forEach` activity (OASIS, 2007, SA00071). This constraint will get important when translating message links into control flow links.

We define as business activities those activities that perform the actual business logic. These activities interact with services outside the choreography, perform data manipulations, implement user interactions etc. Business activities do not contribute directly on intra-choreography communication (i. e., they are neither source nor target of a message link). Moreover, a business activity must not have any visible child activities (e. g., loops are no business activities). If a set of activities is indexed, it denotes the set the business activities of the indexed participant. For instance, $A_{BA,i}$ denotes the set of all business activities of participant i.

2.2 Consolidation Overview

In this section, we summarize the idea of the process consolidation by Wagner et al. (Wagner et al., 2011). The consolidation operation merges a set of n interacting processes of a choreography C into a single process P_{Merged}.

The consolidation has to keep the explicitly modeled control flow constraints between business activities of the same process, i. e., $\{(a,b) \mid a,b \in A_{BA,i}\}$ and the implicit control flow constraints between activities that originate from different processes, i. e., $\{(a,b) \mid a \in A_{BA,i}, b \in A_{BA,j}$, where $i \neq j\}$. Explicit control flow constraints are imposed by the BPEL control flow constructs such as control links between activities and structured activities (e. g., loops). Implicit control flow constraints are imposed by the interaction patterns between the processes that have to be merged. The initial asynchronous interaction between the "Travel Agency" and "Airline" implies for instance that the `receive` activity "Rcv Price Request" and its successor activities have to be started after the `invoke` activity "Send Price Request" completed. There exists, for instance, no implicit control flow constraint between the activity "Store Quote" and "Calculate Price" as "Calculate Price" may be started or completed even before "Store Quote" was started or completed. This may happen if the activity "Store Quote" is long running. The synchronous interaction, in turn, additionally implies that the successor activities of a sending synchronous `invoke` activity are not started until it received a response from the partner where it has sent a message to before (synchronous interactions are not depicted in Figure 1). Hence, if "Quote Price" would send a synchronous response to the "Send Price Request" activity, "Store Quote" would be executed after "Calculate Price" completed. To capture these implicit control flow constraints, the consolidation operation *materializes* them into explicit control flow relations. An example for control flow materialization for an asynchronous interaction is given in Figure 2. To derive the control flow from asynchronous interactions, the sending activity is replaced by a synchronization activity "syn_s" and the receiving activity is replaced by a second synchronization activity "syn_{rc}". The synchronization activity "syn_s" is an `assign` activity that emulates the former message transfer between the send/receive activity, i. e., it copies the former message payload data to the variable where the message payload was written to before. Between "syn_s" and "syn_{rc}" a control link is created that ensures that "syn_{rc}" and its successors are not started before "syn_s" has been completed. The incoming and outgoing links of the sending and receiving activity are mapped to the synchronization activities "syn_s" and "syn_{rc}" respectively. In our scenario, "Send Price Request" is for

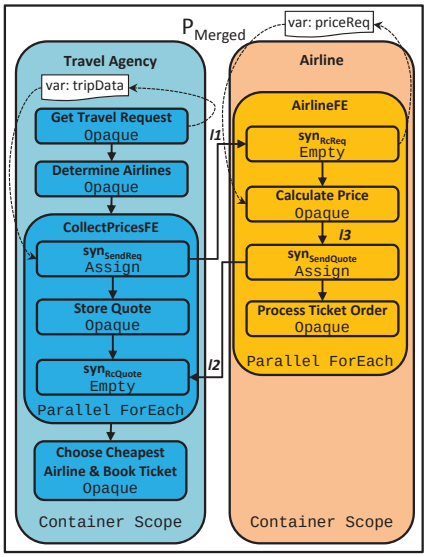


Figure 2: Consolidated Process Model Created from Example Choreography.

instance replaced by "$\text{syn}_{\text{SendReq}}$" and "Rc Price Request" by "$\text{syn}_{\text{RcReq}}$". The latter activity could be also removed as it has no incoming links and the control link could point directly from "$\text{syn}_{\text{SendReq}}$" to "Calculate Price". A more formal description of deriving the control flow from interaction patterns is described by Wagner et al. (Wagner et al., 2012b).

After the basic concepts of control flow materialization were explained, we summarize the different steps of the consolidation for one-to-one interactions. Deviations from these steps in multi-instance interactions will be discussed in section 4.

(i) *Creation of Merged Process Model P_{Merged}:* The new process P_{Merged} is created that acts as container for the set of activities. All activities of all processes to be merged are put into P_{Merged}.

(ii) *Creating Participant Containers:* The activities $A_1, \ldots, A_n$ have to be added to P_{Merged} in an isolated fashion as they were also isolated from each other in $\mathcal{C}$. This preserves the control flow of $\mathcal{C}$ since uncaught faults in A_i do not cause $A_j (i \neq j)$ to fail. Assume, for instance, that the "Travel Agency" and "Airline" process were merged into a single process. If activity "Store Quote" is running and "Calculate Price" causes an uncaught fault, P_{Merged} is terminated. Thus, also the activity of "Store Quote" fails. This would cause an invalid execution trace during the execution of $\mathcal{C}$ as the activity "Store Quote" can still be performed even though "Travel Agency" might have crashed. Hence, containers have to be created to isolate the activities of the former processes from each other. In BPEL, the activities can be isolated from each other by placing each activity set A_i into a separate container `scope` CS_i that catches all faults.

(iii) *Control Flow Materialization:* Based on the interaction patterns, the control flow between all pairs of activity sets A_i and A_j is materialized.

(iv) *Resolving Language Violations:* The materialization may lead to control flow or data flow constructs in P_{Merged} violating certain language constraints. For instance, as a result of the materialization of the synchronous interaction between the "Travel Agency" and the "Airline", control links are created that cross the boundaries of the `forEach`. However, crossing loop boundaries is forbidden in BPEL. In this step, these violations have to be resolved.

(v) *Data Flow Adjustments:* To share data between activities within different container scopes CS_i and CS_j, the variables used by "syn_s" and "syn_{rc}" to emulate the message transfer between the scopes have to be globalized, i. e., they have to be lifted to the process `scope` P_{Merged}.

3 MULTI-INSTANCE INTERACTIONS

To describe the consolidation operation for multi-instance interactions in section 4, this section discusses the control flow constraints implied by multi-instance interactions (subsection 3.1) and the different approaches to initiate multi-instance interactions (subsection 3.2).

3.1 Control Flow Constraints

On process instance level, an instance of the processes $P_1, \ldots, P_n$ can create another instance of $P_1, \ldots, P_n$ (see subsection 3.2) and synchronize itself with this or other instances via message exchanges. Apart from that, the instances run isolated from each other. If, for instance, one instance crashes, the other instances continue their operation. We refer to this as "instance autonomy". As a result, the different instances of one multi-instance process run completely isolated from each other if they do not exchange messages (Kopp et al., 2010). This also implies that there is no control flow relation between an instance i and an instance j of the same activity where instance i belongs to another process instance than instance j. In our example in Figure 1 an instance i of "Calculate Price" may be executed before, during, after etc. an instance j of "Calculate Price" during choreography execution. In this work, we will discuss how far this instance autonomy can be preserved in P_{Merged}.

3.2 Multi-instance Process Instantiation

The only explicit influence a process P_A has on the lifecycle of another process P_B is when P_A creates an instance of process P_B. In BPEL, a process instance is created implicitly when an instance creating activity receives a message. In this paper, as instance creating activities, we treat the `receive` activity (flagged with `createInstance`) only and do not regard the 1:m choice activity `pick`. The example choreography contains the one instance-creating activity "Rc Price Request" that creates an instance of the "Airline" process after it received a message via message link $m1$. When a message link connects an invoke with an instance creating activity, we call the invoke "starting invoke" (a_{start}) and the instance creating activity a_{init}. We distinguish between *static*, *dynamic* and *hybrid* instance creation scenarios that base on the multiple instance

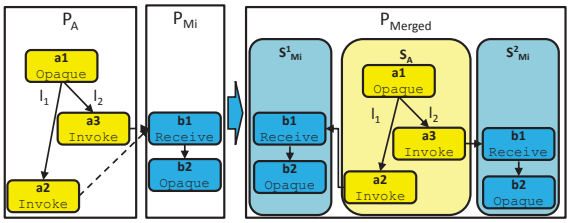


Figure 3: Static Multi-instance Partner Instantiation (left) and Container Generation (right).

patterns described by Aalst et al. (van der Aalst et al., 2003).

In *static multi-instance creation scenarios*, the number of instances of multi-instance process to be executed can be determined at design time. Static instantiation can be either implemented by performing exactly one instance of one or more possible starting invokes a_{start} or by performing a fixed number of instances of the same activity a_{start}, e. g., within a repeatable construct whose number of iterations is defined at design time. In Figure 3 on the left two instances of multi-instance process P_{Mi} are created when $a2$ and $a3$ are performed.

In *dynamic multi-instance creation scenarios*, the number of instances of a_{start} is unknown at design time but determined at runtime. Dynamic scenarios can be implemented by repeatable constructs whose number of iterations is only known at runtime such as loops or event handlers. In this work, we focus on `forEach` loops. The number of instances of the activity "Send Price Request" that create an instance of the "Airline" process is, for instance, determined by the set of available airlines that is created during runtime of activity "Determine Airlines". Another example of dynamic instance-creation is shown at the left side of Figure 4. There, each activity instance of $a2$ and $a3$ creates a process instance of P_{Mi}. The number of instances of P_{Mi} depends on the number of executed activity instances of $a2$ and $a3$. That, in turn, depends on the number of iterations of their surrounding `forEach` loop whose upper bound N is determined at runtime.

Also a combination of both instantiation approaches can be implemented, i. e., some instances of multi-instance process are created statically while others are created dynamically. An example for this *hybrid instantiation approach* is shown in Figure 5. Each execution of activities $a2$, $a3$ creates an instance of P_{Mi}, but the number of instances of $a4$ to be executed is not known at design time.

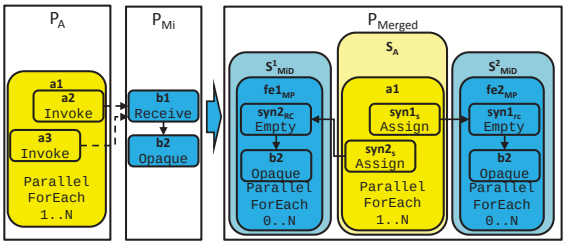


Figure 4: Dynamic Multi-instance Partner Instantiation (left) and Container Generation (right).

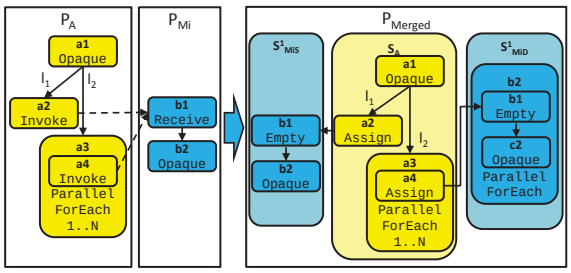


Figure 5: Hybrid Multi-instance Partner Instantiation (left) and Container Generation (right).

4 MULTI-INSTANCE PROCESS CONSOLIDATION

Based on the consolidation steps for single instance interactions described in subsection 2.2, the control-flow properties and the different approaches to initiate multi-instance interactions described in section 3, the consolidation approach for multi-instance interactions is proposed in this section.

4.1 Container Generation

As described in subsection 2.2, in the second step of the consolidation, for each process model $P_1, \ldots, P_n$ to be merged a container $S_1, \ldots, S_n$ (`scope`) is created in P_{Merged}, where each container contains the respective activities of $P_1, \ldots, P_n$. This approach cannot be directly applied for multi-instance processes as a (potentially unknown) number of instances of activities have to be executed during the execution of P_{Merged} that run simultaneously and isolated from each other. We have to distinguish between the container generation for static multi-instance creation scenarios and for dynamic multi-instance creation scenarios as in the former scenario the number of activity instances to be isolated is known while in the other case it is unknown.

During the execution of C, one static starting invoke activity a_{start} can potentially create one instance of a multi-instance process by sending a message to an activity a_{init}. To emulate this behavior, "multi-instance

process unrolling" is performed. That means, for each static starting invoke a^i_{start}, a separate container S^i_{MiS} is created as immediate child activity of P_{Merged}. Even though the invokes may be mutually exclusive, we generate a container for each invoke. BPEL's execution semantics (the dead path elimination) ensures that the container is only executed if the invoke itself had been executed. This also implies that for each activity a_{start} residing within a handler (fault handler, event handler, ...), a container CS_{MP} is generated as they may potentially be executed during the execution of $\mathcal{C}$. Thus, the number of static instantiation activities for multi-instance process determines the number of containers CS_{MiS} that are generated within P_{Merged}. The execution of the two static starting invoke activities for P_{Mi} $a2$ and $a3$ in Figure 3 could be mutually exclusive if the transition conditions of control links l_1 and l_2 always evaluate to different Boolean values or if both links evaluate to *false*. Nevertheless, due to the over-approximation the two containers S^1_{MiS} and S^2_{MiS} are created in P_{Merged}. Creating a separate container S_{MiS} for each multi-instance process instantiation also ensures the same instance-independence of the activities as in $\mathcal{C}$. To perform the control flow materialization correctly for each generated container CS_{MiS}, a new set of message links is created.

As the number of instances of a_{start} cannot be determined in dynamic multi-instance creation scenarios, the number of process instances of multi-instance process is not known at design time. Hence, the multi-instance process cannot be "unrolled" into different containers as in static multi-instance creation scenarios. To emulate these scenarios in P_{Merged}, a container for the activities of the multi-instance process is needed. This container has to realize the simultaneous and isolated execution of an at design time unknown number of instances of the activities of the multi-instance process. The only construct in BPEL supporting a simultaneous execution of a number of instances (branches) of its root activities (along with its children) in an isolated fashion is the parallel `forEach` activity. Therefore, for each dynamic a_{start}, a dynamic container `scope` S_{MiD} is created that contains a parallel `forEach` activity fe_{MP} that contains the root activity of the multi-instance process within its `scope`. The scope is required as BPEL enforces the immediate child of a `forEach` loop being a `scope`. This results in two levels of isolation: (i) the scope within the `forEach` isolates the root activity instances of the multi-instance process from each other by catching all faults that may be thrown by them; (ii) the scope S_{MiD} isolates fe_{MP} from the other containers.

Note that for each dynamic a_{start}, a separate dynamic container S_{MiD} is created. Even if n distinct dynamic a_{start} activities that send a message to the same instance-creating activity a_{init} are located within the same parent loop as shown in Figure 4. The reason for creating a separate dynamic container for each of the n activities $a^1_{init}, \ldots, a^n_{init}$ is that all or a subset of the a_{init} activities may be performed simultaneously.

The attribute values for fe_{MP} are defined as follows: It gets the start and the final counter value of the parent forEach of a_{start} and a unique id to distinguish it from other forEach loops. The counter values are required to assign data between the instances of the forEach loop as described in subsection 4.4. It may happen that a_{start} is not executed during each iteration due to certain control flow conditions. Consequently, more instances of fe_{MP} may be created than necessary. Due to the synchronization activities that are created during the control flow materialization, it is ensured that the business activities in the "unused" instances are not activated. The parent forEach of an a_{start} may have a completion condition defined that specifies that the processing of the forEach maybe ended as soon as a subset of its branches completed successfully even though there are still running branches. This condition is not applied to fe_{MP}, because during the execution of $\mathcal{C}$ all instances of a multi-instance process that were created are allowed to complete.

In the hybrid instantiation scenarios of multi-instance processes, static and dynamic multi-instance containers are created in P_{Merged} that contain the activities of the multi-instance process. In Figure 5, the static container S^1_{MiS} is created and the dynamic container S^1_{MiD} for the instantiation activities $a2$ and $a4$ respectively.

Static and dynamic multi-instance containers are started when P_{Merged} becomes active. However, the business activities within the containers are not executed until the incoming links are activated.

4.2 Control Flow Materialization

The basic concepts to materialize the control flow from the message flow for single instance scenarios have been described in subsection 2.2. These concepts can be also applied to multi-instance scenarios since on the instance level also one-to-one communication takes place between the interacting processes. For instance, in the example choreography in Figure 1, an instance of the "Send Price Request" activity communicates with exactly one instance of the "Rc Price Request" activity. Therefore, we do not need to modify the control flow materialization for multi-instance scenarios, neither for static, dynamic, nor hybrid instantiation scenarios. Activities that were used in $\mathcal{C}$ to assign the endpoint references (EPR) of the processes to be called from the communication activities can be removed.

For instance, the activity "Determine Airlines" returns a set of EPRs. In each iteration of "Collect Prices FE" another airline EPR is assigned to "Send Price Request". This assignment is not shown in Figure 1. The consolidated process model P_{Merged} created from the example choreography is depicted in Figure 2.

4.3 Resolving Link Violations

If dynamic multi-instance containers are created, the control flow materialization causes cross-boundary link violations as the consolidation always generates control links between synchronization activities that cross boundaries of forEach loops. For instance, the link $l1$ from activity "$\text{syn}_{\text{SendReq}}$" to activity "$\text{syn}_{\text{RcReq}}$" in Figure 2. To resolve these violations, the link status value ("true" if the link is enabled, "false" if it is disabled and "undef." if undetermined) of "syn_{s}" to "syn_{rc}" could be written to a new variable v_{ls} that can be also accessed from within the loop to check if "syn_{rc}" can be started, i. e., if the link is enabled. However, as from within the loop it is unknown when v_{ls} is set, another loop preceding "syn_{rc}" is needed that constantly polls the value of v_{ls} until it is set from "undef." to "true" or to "false". As the permanent polling stresses the workflow engine and the underlying resources, this solution is not an option. Of course, one could adjust the polling interval but a useful polling interval would have to be determined for each business scenario to ensure the overall process execution time is not negatively affected. Instead, we propose a *forEach loop fragmentation approach* to resolve the cross-boundary link violations. We also describe how link status values can be propagated between the created forEach loop fragments.

4.3.1 ForEach Loop Fragmentation

Each pair of activities "syn_{s}" and "syn_{rc}" violating the cross-boundary link constraint is moved from its original forEach loop into a new forEach loop fe_{syn}. By convention, fe_{syn} is always placed into the container scope CS of "syn_{s}" to ensure that no data are copied by "syn_{s}" if another activity fails within CS. All opaque activities directly or indirectly preceding "syn_{s}" (but no other synchronization activity) are moved from their original forEach loop to a new FE-fragment fe_{pred} that is also created in the container scope of "syn_{s}". The direct and indirect predecessor activities of "syn_{rc}" (if any) that do not precede another synchronization activity and that were contained in the same original forEach as "syn_{rc}", are moved to a new FE-fragment fe_{rc}. This FE-fragment must be created in the container scope of "syn_{rc}" as the opaque activities preceding "syn_{rc}" originate from this container. The FE-fragments fe_{syn} and fe_{rc} are connected via a control link $l(fe_{rc}, fe_{syn})$. This maintains the original execution order that defined that "syn_{rc}" is not started until its predecessors completed. Accordingly, FE-fragment fe_{pred} is connected to fe_{syn} via control link $l(fe_{pred}, fe_{syn})$ to ensure that "syn_{s}" is only started after its predecessor activities completed. If "syn_{s}" had no predecessor activities in its original forEach, fe_{syn} is connected to the predecessor activities of the original forEach of "syn_{s}". FE-fragment fe_{syn} is either connected to the FE-fragment that contains the direct successor activities of "syn_{s}" or directly to its successor activities in case it does not reside within a forEach. Additionally, fe_{syn} is connected to the forEach hosting the direct successor activities of "syn_{rc}" or directly to the successor activities of "syn_{rc}" that do not reside within a forEach. All opaque activities that do not precede any synchronization activity are left in their corresponding original forEach loops and connected to the fe_{syn} fragment that contains their preceding "syn_{s}" and "syn_{rc}" respectively. Note, that all FE-fragments that host opaque activities inherit the start and end counter values of the forEach loop where the opaque activities originate from. Also the fault handlers and termination handlers are adapted from the original forEach. FE-fragments fe_{syn} inherit the attributes and handlers from `forEach` syn_s. The control links between the activities within a FE-fragment are kept to maintain the execution order between these activities. All incoming links of activities whose predecessors reside within another FE-fragment are removed. When the successor activities of an activity were moved to another FE-fragment its outgoing links are removed. Figure 6 shows the fragmented version of P_{Merged} of our example scenario. In the "Travel Agency" container the two FE-fragments "CollectPricesFE_1" and "CollectPricesFE_2" are created. The "Airline" container hosts the activities "AirlineFE_1" to "AirlineFE_3". The control links between the FE-fragments and also their preceding and succeeding activities are created as described above.

4.3.2 Link Status Propagation

The split of the original forEach loops of P_{Merged} into several FE-fragments causes control links to break if activities a_{src} and a_{trg} that were connected via link $l(a_{src}, a_{trg}, tc)$ (tc represents the transition condition of the link) were placed into different FE-fragments fe_m and fe_n. Khalaf and Leymann (Khalaf and Leymann, 2006) describe an approach to split a single process into several individual process fragments. As this also causes control links to break if the source and target activity of a link reside within different process fragments, they described a technique to prop-

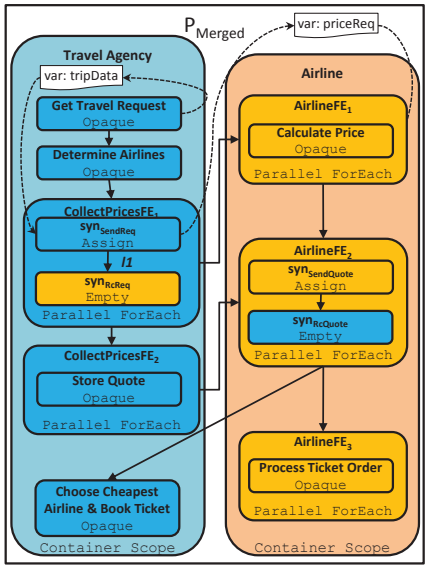


Figure 6: Fragmented Version of Consolidated Process Model P_{Merged}.

agate the link status from one process fragment to another via message exchanges. We adapt this approach to propagate the link status from one FE-fragment to another by using variables instead of message exchanges. In the FE-fragment hosting a_{src} a new scope S_{ls} with a fault handler FH is added. To S_{ls} the assign activity $assign_{true}$ and to FH the assign activity $assign_{false}$ is added. $assign_{true}$ writes the link status value "true" to a newly introduced variable var_{ls} that resides in the parent scope of fe_m and fe_n. $assign_{false}$ writes the link status value "false" to var_{ls}. A new link $l_{src}(a_{src}, assign_{true}, tc)$ is created. The attribute *suppressJoinFailure* within S_{ls} is set to "no", i. e., when the transition condition tc evaluates to "false" a *bpel:joinFailure* is thrown. This failure is caught by FH and $assign_{false}$ is executed that writes the link status value "false" to a newly created variable var_{ls}. If tc evaluates to "true" $assign_{true}$ writes "true" to var_{ls}. In the FE-fragment that hosts the link target a_{trg} a new `empty` activity em is created that is connected to a_{trg} via the new link $l_{trg}(em, a_{trg}, read(var_{ls}))$. Thereby, the function $read(var_{ls})$ reads the link status of var_{ls}. An instance of the `empty` activity em is started as soon as the corresponding `forEach` branch becomes active. As the execution order between activities a_{src} and a_{trg} is preserved by the execution order between their hosting FE-fragments, the value of var_{ls} is always set before em is started. Figure 7 shows exemplary the FE-fragments "AirlineFE$_1$" and "AirlineFE$_2$" with the link propagation logic. To propagate the status of link $l3$ from "Calculate Price" to "syn$_{\text{SendQuote}}$" as in the original unfragmented "AirlineFE" the status of the outgoing link of "Calculate Price" is written to var_{l3}

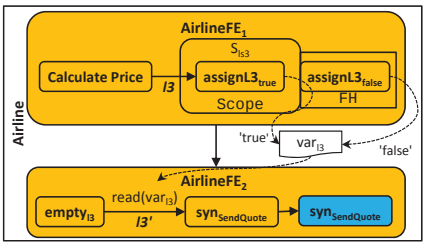


Figure 7: Passing Link Status Values Between FE-Fragments.

by "assignL3$_{\text{true}}$" and "assignL3$_{\text{false}}$" respectively, depending on the status of $l3$ in "AirlineFE$_1$". This status is then applied to the new link $l3'$ in $AirlineFE2$ by reading var_{l3}. The link propagation approach does also ensure that death-path elimination can be performed. If, for instance, the incoming link of "Calculate Price" evaluates to "false" and "syn$_{\text{SendQuote}}$" had any outgoing links (which is not the case in the example), the outgoing links of "syn$_{\text{SendQuote}}$" are automatically set to "false".

So far, we regarded one variable var_{ls}, where the link status is written to and read from. However, since multiple instances of the same link are created, also multiple instances of link status values must be hold. Therefore, var_{ls} has to be extended to hold multiple status values of the same link and also assign activities have to be modified accordingly to access the correct instance of a link status. This is described in subsection 4.4.

4.4 Data Flow

As described in subsection 2.2, the variables have to be globalized to the process scope of P_{Merged} for sharing data between different container scopes CS. To share data and link status values between different FE-fragments that originate from the same forEach, these variables have to be lifted to the parent container scope CS_{MiD} of the FE-fragments. In contrast to the one-to-one interaction scenarios during choreography runtime, several instances of the same variable are created, e. g., for each "Airline" partner, one instance of the variable "priceReq". We refer to these variable as "multi-instance variables" V_{MI}. To store multiple instances of a variable v_{MI}, it is replaced by a map mp_{MI}. Each entry of mp_{MI} represents an instance of v_{MI}, i. e., the entries of mp_{MI} inherit the data type of the original v_{MI} and can be uniquely identified by a key key_{IID} (instance id) of type `<xsd:id/>` that represents an instance of v_{MI}.

This raises the question how the key_{IID} is composed, how a branch (instance) of a FE-fragment can be related to a certain entry in mp_{MI} via key_{IID} and

how to ensure that data can be shared between different FE-fragments. Intuitively, EPRs could be used to compose keys as they are also employed in $\mathcal{C}$ to identify an instance of a multi-instance process. However, there is technically no way to inject a key such as an EPR into a forEach branch from outside to a specific instance when it starts. Thus, a forEach branch is not aware of what key it is related to. Therefore, a convention is needed to enable the branch to determine its associated key by itself. The only information that uniquely identifies a forEach branch among the other branches and that does not change during its lifetime is the value of the instance counter variable. As different forEach loops are contained within P_{Merged}, their instance counter values are not globally unique but only within the respective forEach. To ensure global uniqueness and to share data between the same branches (i. e., branches with the same counter value) of FE-fragments that originate from the same forEach loop, the static id *feid* (defined at design time) and the dynamic instance counter value *iid* (defined at runtime) are concatenated. Note, that all FE-fragments that originate from the same forEach share the same *feid*. Assume, for instance, that the execution of three instances of the "Airline" process has to be emulated. The first instance would get the instance key "AirlineFE_1", the second "AirlineFE_2" and so on. Similarly, three instances of the forEach "CollectPricesFE" would get the instance keys "CollectPricesFE_1" to "CollectPricesFE_3". As described in subsection 4.1, fe_{MI} inherits the start and end counter value of the parent forEach of a_{start} (e. g., "AirlineFE" has the same start and end counter values as "CollectPricesFE"). Also the different FE-fragments inherit the values of the forEach loops they were created from. Since all of these FE-fragments share the same start and end counter variables, their branches can be logically related to each other by using their instance id *iid*, e. g., branch "CollectPricesFE_1" can be related to "AirlineFE_1", "CollectPricesFE_2" to "AirlineFE_2" etc. Data can be passed between all logical related FE-fragments by rewriting assign statements and transition conditions accessing multi-instance variables v_{MI}. As these variables are always located within FE-fragments, data accesses to v_{MI} can be rewritten during the consolidation to data accesses to those entries of mp_{MI} that share the same *iid*.

5 DISCUSSION

In this section the properties of a process P_{Merged} created by applying the multi-instance process consolidation approach proposed in section 4 are discussed.

The consolidation maintains the control flow constraints defined in $\mathcal{C}$. On the one hand, the original control links between activities moved to the same FE-fragment (e. g., "$\text{syn}_{\text{SendQuote}}$" and "$\text{syn}_{\text{RcQuote}}$" in FE-fragment "$\text{AirlineFE}_2$") are maintained. One the other hand, the execution order between consecutive activities a_i and a_j moved to different FE-fragments is kept by connecting the FE-fragments based on the control relation between a_i and a_j. The proposed propagation technique for link status values guarantees that activities are only executed if their incoming links evaluate to "true" even if link source activity resides within another FE-fragment. Since the fault handler and termination handler from the original `forEach` are attached to the FE-fragments and since the FE-fragments are executed sequentially the fault handling and semantics of BPEL is also kept, i. e., if within a certain FE-fragment branch an activity fails, all running activities within this branch are terminated and activities within the succeeding FE-fragment branches that represent the same instance (i. e., branches with the same instance counter value *iid*) are not started anymore.

The consolidation keeps the control flow constraints modeled in $\mathcal{C}$. However, P_{Merged} cannot generate all execution traces between business activities that could be generated by $\mathcal{C}$. The reason is that the multi-partner instances lose their instance autonomy from each other. The instance autonomy is lost, because of the FE-fragmentation. Assume, for instance, the execution time of activity "Calculate Price" in the choreography in Figure 1 depends on the input data, i. e., the price request. Then, depending on the request, the instance "Calculate Price^{i}" is long running while in another instance "Calculate Price^{j}" is short running. Therefore, the succeeding activity instances of "Calculate Price^{i}" would be started earlier as those of "Calculate Price^{j}". If all succeeding activity instances i and j would need the same execution time, all activity instances i would be completed before activity instances j. The FE-fragmentation, however, imposes also control flow relations between different instances that are emulated. This is due to the fact, that all branches of a forEach, i. e., an FE-fragment, have to be completed until the succeeding FE-fragment can be started. Hence, no instance of "$\text{syn}_{\text{SendQuote}}$" is started until all instances of "Calculate Price" completed as the emulated instances have to "wait" for each other. This can significantly increase the time until a business goal is reached, especially, if the execution times of individual instances of the same activity are highly different and if only a subset of instances is required for fulfilling the business goal (e. g., the "Travel Agency" may only need the response from three out of 10 airlines). However, if $\mathcal{C}$ and P_{Merged} are started with the same input data, their overall execution time is the same, as

in both case all business activity instances (that are activated via a path) have to be executed. If we have just static multi-instance scenarios where a_{start} does not reside within a forEach loop, we do not encounter the problem of losing instance autonomy that was described above. That is the reason why we distinguish between these scenarios.

The multi-instance consolidation may lead to a complex process P_{Merged}. This is because the multi-instance process unrolling and the forEach fragmentation create many new activities and the corresponding control links in P_{Merged}. Hence, the maintainability of P_{Merged} is decreased, i. e., the process structure is difficult to understand. This, in turn, makes it very difficult to change the process. For that reason we recommend that the consolidation should be applied when the choreography is not changed anymore. For instance at deployment time the choreography can be consolidated to save resources (refer to section 1). To monitor the state of the original choreography state propagation rules can be used that derive the state of the choreography from from the runtime state of P_{Merged} (Wagner et al., 2012a).

6 IMPLEMENTATION

In previous work (Debicki, 2013) a Java prototype was developed that gets a BPEL4Chor choreography as input and returns the consolidated process model in BPEL. The consolidated process model can then be deployed on a workflow engine (Wagner et al., 2013b).

Initially the prototype implemented the consolidation steps for one-to-one interactions described in subsection 2.2. For evaluation of the multi-instance process consolidation approach described in section 4 it was extended (Dadashov, 2013) to create multi-instance containers in P_{Merged}, to perform the forEach loop fragmentation and to create multi-instance variables.

Internally, the prototype performs the consolidation steps on a BPEL4Chor EMF[1] object model[2] which is an extension of the Eclipse BPEL EMF model[3]. The EMF models provide Java object serializations of BPEL processes and BPEL4Chor choreographies respectively. To consolidate interacting processes the prototype reads a ZIP file that contains the XML representations of the participant topology and the participant behavior descriptions. These XML representations are transformed into a BPEL4Chor EMF model and a new BPEL EMF model for P_{Merged} is generated. Then the prototype inspects the BPEL4Chor EMF model to determine the container scopes to be generated in P_{Merged} and to perform the control flow materialization. After the consolidation operations completed P_{Merged} is transformed back into a BPEL schema-compliant XML. A process modeler has to replace the `opaque` activities in P_{Merged} (if any) by executable activities (executable completion (Aalst et al., 2008)). As deployment descriptors a workflow engine specific the process modeler also has to add them manually after the consolidation completed. Then P_{Merged} can be deployed.

[1]http://www.eclipse.org/modeling/emf/.

[2]https://github.com/IAAS/BPEL4Chor-model.

[3]http://www.eclipse.org/bpel/developers/model.php.

7 RELATED WORK

Mendling and Simon (Mendling and Simon, 2006) propose an approach where semantically equivalent events and functions of Event Driven Process Chains (Scheer et al., 2005) are merged. Küster et al. (Küster et al., 2008) describe how change logs can be employed to merge different process variants that were created from the same original process. These approaches merge processes that are semantically equivalent or that are different variants of the same original process. Our approach focuses on the consolidation of collaborating processes into a single process model. Moreover, none of these approaches deals with multi-instance interactions.

An alternative way to generate a BPEL orchestration of a BPEL4Chor choreography is using an intermediate format. However, there is currently no approach keeping the structure of the generated orchestration close to the structure of the original choreography. For instance, Lohmann and Kleine (Lohmann and Kleine, 2008) do not generate BPEL scopes out of Petri nets, even if the formal model of Lohmann (Lohmann, 2007) generates a Petri net representation of BPEL scopes.

There are other ways to describe inter-organizational collaboration. For instance, the lifecycle of a business entity can be put into the center of modeling (Hull et al., 2011). In this paper, we did not follow that approach, but used the interconnection model choreography approach.

8 CONCLUSION AND OUTLOOK

In this paper, we introduced an approach to consolidate BPEL processes interacting with multi-instance processes in a choreography into a single process model that emulates the multi-instance behavior of

the original choreography. To perform the consolidation, we distinguished three instantiation scenarios for the multi-instance processes. In static instantiation scenarios, the number of instances can be determined during the consolidation. To emulate the static multi-instance scenarios, for each possible instance, a separate container containing activities of the multi-instance process is created within the consolidated process. In dynamic instantiation scenarios, the number of instances cannot be determined during the consolidation. To emulate this scenario, the activities of multi-instance processes are copied into a parallel forEach loop whose number of iterations is determined during runtime. Each branch of this loop emulates one instance of the multi-instance process. The control flow materialization may create control links from the message links that cross the boundaries of the forEach loop what is not permitted in BPEL. To resolve these violations we proposed a technique to sequentially split the forEach loop into different fragments. We also discussed how the data flow within the consolidated process model has to be modified to store and access multiple instances of the same variable.

Currently, we do only support consolidation of multi-instance processes that are instantiated by an instance creating activity that resides in a forEach loop. In the future also the consolidation of multi-instance processes is supported that are instantiated from within nested and condition-controlled loops such as while loops.

Reference passing is an additional major aspect of multi-instance interactions, e. g., the travel agency could pass the endpoints of the three cheapest airlines to a traveler process who then decides by itself what airline is booked. Our approach has to be extended accordingly to support these interaction scenarios as they are very common in the context of multi-instance processes. We also have to discuss more in depth how BPEL's compensation handling mechanism is affected by the split of a single forEach into different fragments.

As shown by Wagner et al. (Wagner et al., 2013b), the execution time and performance overhead of a choreography execution can be significantly reduced if its processes are consolidated into a single process model, because message serialization, message transport, and message de-serialization is avoided. In future work, we have to analyze the performance differences between consolidated process models that emulate multi-instance processes and their original choreography.

As BPMN becomes more and more important and since it shares a lot of concepts and language constructs with BPEL and BPEL4Chor, we have to investigate how our consolidation approach can be applied to merge BPMN collaboration diagrams.

ACKNOWLEDGEMENTS

This work was partially funded by the BMWi project Migrate! (01ME11055) and the BMWi project CloudCycle (01MD11023).

REFERENCES

Aalst, W. M. P. v. d., Lohmann, N., Massuthe, P., Stahl, C., and Wolf, K. (2008). Multiparty contracts: Agreeing and implementing interorganizational processes. *Comput. J.*, 53(1):90–106.

Barros, A., Decker, G., Dumas, M., and Weber, F. (2007). Correlation patterns in service-oriented architectures. In *FASE*, volume 4422. Springer.

Barros, A., Dumas, M., and ter Hofstede, A. (2005). Service Interaction Patterns. In *BPM*. Springer.

Börger, E. (2012). Approaches to modeling business processes: a critical analysis of bpmn, workflow patterns and yawl. *Softw. Syst. Model.*, 11(3):305–318.

Dadashov, E. (2013). Choreography-based Business Process Consolidation in One-To-Many interactions. Master thesis, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany.

Debicki, P. (2013). Choreographie-basierte Konsoliedierung von BPEL Prozessmodellen. Diploma thesis, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany.

Decker, G., Kopp, O., and Barros, A. (2008). An Introduction to Service Choreographies. *Information Technology*, 50(2):122–127.

Decker, G., Kopp, O., Leymann, F., and Weske, M. (2009). Interacting services: From specification to execution. *Data & Knowledge Engineering*, 68(10):946–972.

DeRemer, F. and Kron, H. (1976). Programming-in-the-large versus programming-in-the-small. *Software Engineering, IEEE Transactions on*, SE-2(2):80 – 86.

Hull, R., Damaggio, E., Fournier, F., Gupta, M., Heath, F. T., Hobson, S., Linehan, M., Maradugu, S., Nigam, A., Sukaviriya, P., et al. (2011). Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In *7th International Workshop, WS-FM 2010*, LNCS. Springer-Verlag.

Khalaf, R. and Leymann, F. (2006). Role-based Decomposition of Business Processes using BPEL. In *ICWS 2006*. IEEE.

Khalaf, R. and Leymann, F. (2010). Coordination for Fragmented Loops and Scopes in a Distributed Business Process. In *BPM*. Springer.

Kopp, O., Eberle, H., Leymann, F., and Unger, T. (2010). The Subprocess Spectrum. In *Proceedings of the Business Process and Services Computing Conference: BPSC 2010*. GI e.V.

Kopp, O., Martin, D., Wutke, D., and Leymann, F. (2009). The Difference Between Graph-Based and Block-Structured Business Process Modelling Languages. *Enterprise Modelling and Information Systems*, 4(1):3–13.

Kossak, F., Illibauer, C., and Geist, V. (2012). Event-based gateways: Open questions and inconsistencies. In Mendling, J. and Weidlich, M., editors, *Business Process Model and Notation*, volume 125 of *Lecture Notes in Business Information Processing*, pages 53–67. Springer Berlin Heidelberg.

Küster, J., Gerth, C., Förster, A., and Engels, G. (2008). A tool for process merging in business-driven development. In *Proceedings of the Forum at the CAiSE*.

Leymann, F. (2010). Bpel vs. bpmn 2.0: Should you care? In *BPMN*, pages 8–13.

Lohmann, N. (2007). A Feature-Complete Petri Net Semantics for WS-BPEL 2.0. In *WS-FM*.

Lohmann, N. and Kleine, J. (2008). Fully-automatic Translation of Open Workflow Net Models into Simple Abstract BPEL Processes. In *Modellierung*. GI e.V.

Mendling, J. and Simon, C. (2006). Business process design by view integration. In *BPM Workshops*. Springer.

OASIS (2007). *Web Services Business Process Execution Language Version 2.0 – OASIS Standard*.

Object Management Group (OMG) (2011). *Business Process Model and Notation (BPMN) Version 2.0*. OMG Document Number: formal/2011-01-03.

Scheer, A.-W., Thomas, O., and Adam, O. (2005). *Process Aware Information Systems: Bridging People and Software Through Process Technology*, chapter Process Modeling Using Event-Driven Process Chains. Wiley-Interscience.

van Breugel, F. and Koshkina, M. (2006). Models and Verification of BPEL. http://www.cse.yorku.ca/ franck/research/drafts/tutorial.pdf.

van der Aalst, W. M. P., ter Hofstede, A. H. M., Kiepuszewski, B., and Barros, A. P. (2003). Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51.

Wagner, S., Fehling, C., Karastoyanova, D., and Schumm, D. (2012a). State Propagation-based Monitoring of Business Transactions. In *Proceedings of the 2012 IEEE International Conference on Service-Oriented Computing and Applications*. IEEE Xplore.

Wagner, S., Kopp, O., and Leymann, F. (2011). Towards Choreography-based Process Distribution In The Cloud. In *CCIS*.

Wagner, S., Kopp, O., and Leymann, F. (2012b). Towards Verification of Process Merge Patterns with Allen's Interval Algebra. In *ZEUS*. CEUR.

Wagner, S., Kopp, O., and Leymann, F. (2013a). Consolidation of Interacting BPEL Process Models with Fault Handlers. In *Proceedings of the 5th Central-European Workshop on Services and their Composition (ZEUS 2013)*, pages 1–7, Rostock. CEUR Workshop Proceedings.

Wagner, S., Roller, D., Kopp, O., Unger, T., and Leymann, F. (2013b). Performance optimizations for interacting business processes. In *IC2E*. IEEE.

Wohed, P., van der Aalst, W. M. P., Dumas, M., ter Hofstede, A. H. M., and Russell, N. (2006). On the suitability of bpmn for business process modelling. In *Business Process Management*, pages 161–176.

A Cloud Application for Security Service Level Agreement Evaluation

Valentina Casola[1], Massimiliano Rak[2] and Giuseppe Alfieri[2]
[1]*DIETI, University of Naples Federico II, Naples, Italy*
[2]*DIII, Second University of Naples, Aversa, Italy*
casolav@unina.it, massimiliano.rak@unina2.it, alfi_86@hotmail.it

Keywords: Cloud Computing, Security, Cloud Application, SPECS, Negotiation, SLA.

Abstract: Cloud security is today considered one of the main limits to the adoption of Cloud Computing. Academic works and the Cloud community (e.g., work-groups at the European Network and Information Security Agency, ENISA) have stated that specifying security parameters in Service Level Agreements actually enables the establishment of a common semantic in order to model security among users and Cloud Service providers (CSPs). However, despite the state of the art efforts aiming at building and representing Cloud SecLAs there is still a gap on the techniques to reason about them. Moreover a lot of activities are being carrying out to clearly state which are the parameters to be shared, their meanings and how they affect service provisioning. In this paper we propose to build up a cloud application that is able to offer Security level Evaluation based on SLA expressed in many different ways. Such application can be offered as a service by Third Parties in order to help customers to evaluate the offerings from providers. Furthermore it can be used to help customers to negotiate security parameters in a Multi-Cloud system and perform Cloud brokering on the basis of a quantitative evaluation of security parameters.

1 INTRODUCTION

Cloud security is today considered one of the main obstacle to the widespread adoption of Cloud Computing. In this paradigm, due to the self-service on-demand characteristic, all data and servers reside *on the cloud* and charged on a pay-per-use basis. Indeed, this model brings great advantages from a business point of view, as there are no maintenance and start-up costs for any infrastructures, but there is a strong negative perception of *loss of control* over the data and resources.

To face security requirements in the Cloud, early academic works like (Kandukuri B.R.,et. al., 2009) and the Cloud Community (e.g., work-groups at the European Network and Information Security Agency (ENISA) (Dekker M. and Hogben G.,2011)) stated that specifying security parameters in Service Level Agreements (referred as *Security Level Agreements or SecLAs* over this paper) actually enables the establishment of a common semantic in order to model security among users and CSPs.

However, despite the state of the art efforts aiming at building and representing Cloud SecLAs (e.g., the CSAs SLA and PLA working groups (Cloud Security Alliance, 2012)), there is still a gap on the techniques to reason about them.

Indeed, a lot of activities are carried out to clearly state which are the parameters to be included in the SecLAs, their meanings and how they affect service provisioning. One of the biggest problems that arises, even after Security parameters are clearly specified, is the needing to help comparing and choosing different providers on the basis of such parameters: security features cover an incredible amount of different aspects of the systems and the customers, often, are not security experts, even if they have specific security constraints. As an example, let us consider a health center that wants to use the Cloud to maintain a backup of its data. The information to be stored have privacy and availability constraints, moreover in many countries the law imposes constraints on data localization. The advantage of using the cloud, which can grant availability and in given conditions privacy constraints, is annihilated by the absence of clear grants on such security grants. Even if CSPs offer SLAs specifying the security grants, it will be hard for the customer to clearly compare the offerings from different providers that offer the same service in different ways, with different services and expressing the grants through different policies and SLAs.

The work proposed in this paper born in the con-

text of the SPECS (SPE,) FP7 project, which aims at building a platform able to offer security services to Customers for Multi-Cloud environments. We propose a cloud application that is able to offer Security Level Evaluation over SLAs expressed in many different ways. The evaluation founds on the REM methodology, which aims at enabling a quantitative evaluation of security policy. Such application is offered as-a-service by a Third Party in order to help customers to evaluate the offerings from providers. Moreover it can be used to help customers to negotiate security parameters in a Multi-Cloud system as proposed in (Liccardo et al., 2012; Amato et al., 2012).

In order to evaluate Cloud providers, we will focus on the security mechanisms provided to end users, demonstrating that it is possible to use a standard language to represent security parameters and enabling the evaluation of different providers.

The reminder of this paper is organized as follows, next sections describe the key concepts to evaluate and compare different cloud providers and the requirements for the development of a security evaluation application; we will outline the background related to this paper, i.e. the mOSAIC Platofrm with the API used to develop the Cloud Application and the REM evaluation methodology. Section 4 describes how to build policies from the security mechanisms exposed to end users, in order to make a security evaluation. The following section, 5 describes the application architecture and its reusable components. Paper ends with section 6 dedicated to conclusions and future works.

2 PROBLEM STATEMENT AND APPROACH

As outlined in the previous section, our goal is the development of an application able to help Customers to evaluate and compare different Cloud Service Providers (CSP) on the basis of the security they are able to grant. In a companion paper we proposed to evaluate providers on the basis of the STAR repository, maintained by CSA, which contains the answers that CSPs has given to a complex questionnaire dedicated to Cloud Security. In this paper, instead, we will focus on the real security mechanisms the CSPs expose to Customers and the implication they may have on the security effectively granted.

We analysed different cloud providers (Amazon EC2, Google Compute Engine, GoGrid), they offer similar features (from customers point of view they are mostly equivalent) but they use completely different technologies and techniques in order to enforce security mechanisms into the service invocation mechanism. As an example, Amazon Web Services (aws) uses SOAP messages to control Amazon Ec2 services, they are secured through a set of HMAC applied and evaluated to each message. GoGrid, instead, performs an authentication through a dedicated authentication server, which issues a security token to be attached to each Web Service invocation. Such different choices have impact both on performances and on the level of security offered.

Even if such features are clearly visible to end users, so they can be explicitly evaluated by a third party, the high number of different technologies offered by different providers and the lack of a clear description of the adopted approaches, make such comparison a very difficult task, usually performed by experts or, intuitively, by Customers.

We propose to follow a more systematic approach, adopting existing standards to represent the way in which security mechanisms are described and services are invoked: WS-Policy(Bajaj et al., 2006) and WS-Security-Policy(Della-Libera et al., 2002).

WS-Policy is a container language that allows to create a XML Policy based on Assertions. It mainly allows the use of a group of statements that are alternatives to each other (also the empty alternative) (*ExactlyOne* statement) or a group of statements that must be met (even 0 assertions) (*All* statement). Ws-Security-Policy is a language that defines a set of Security Assertions to be used within Ws-Policy. These Assertions are useful to specify both what is required and what should be guaranteed on the messages and communication protocols in terms of security when using a particular service.

We want to point out that the semantic of Security Assertions made available by Ws-Security-Policy is often directly linked to the way in which Security is managed in the SOAP WebServices (namely in the structure of the SOAP messages described in the standard Ws-Security). Since, de facto, a REST message does not exist, there is not yet a standard language to describe what is required or offered in terms of Security on REST communications.

Such standard representation offers a common basis to represent the security mechanisms adopted by the CSPs and this will facilitate the adoption of the REM methodology to evaluate and compare the offered solutions. In particular, in order to use the REM, we will build a common template, whose instances will correspond to the mechanisms adopted by the different providers.

In synthesis the technique we propose is based on the following steps:

1. **Description.** Collect WS-SecurityPolicy repre-

sentations of the security mechanisms offered by different CSPs based on a standard template we prepared. Such description can be offered directly by a provider (rarely) or defined by third parties.

2. **Comparison.** Using the REM methodology we quantitatively measure each policy, offering a clear basis to compare the different offerings of providers. Thanks to the flexibility of the REM methodology different customers can specify different weights and evaluation criteria, obtaining customized evaluations.

The main goal is the implementation of a cloud application to provide the security evaluation as-a-service. Before illustrating the details of such technique, in the next section, we will provide a technical background to describe the the mOSAIC framework to easily develop the cloud application and the REM methodology.

3 TECHNICAL BACKGROUND

The solution we propose to build up a cloud application to evaluate providers, founds on two existing solutions, these will be summarized in this section. In particular, in the following we will detail only the key ideas needed to fully understand the proposed approach: the mOSAIC Framework to easily implement cloud applications and the REM evaluation methodology to evaluate security. For further details the interested reader may refer to the related papers.

3.1 The mOSAIC Framework

The mOSAIC framework aims at providing a simple way to develop cloud applications (Petcu et al., 2011c; Petcu et al., 2011b; Petcu et al., 2011a); the target user for the mOSAIC solution is the application developer (*mOSAIC user*). In mOSAIC, a cloud application is structured as a set of components running on cloud resources (i.e., on resources leased by a cloud provider) and able to communicate with each other. Cloud applications are often provided in the form of Software-as-a-Service, and can also be accessed/used by users other than the mOSAIC developer (i.e., by *final users*). In this case, the mOSAIC user acts as service provider for final users.

The mOSAIC framework is composed of a few stand-alone components. Among them, the most important roles are played by the *Platform* and the *Cloud Agency*. The first one (mOSAIC Platform) enables the execution of applications developed using the mOSAIC API. The second one (Cloud Agency) acts as a provisioning system, brokering resources from a cloud provider, or even from a federation of cloud providers.

mOSAIC can be used in three different scenarios:

- when a developer wishes to develop an application not tied to a particular cloud provider;
- when an infrastructure provider aims at offering "enhanced" cloud services in the form of SaaS;
- when a final user (e.g., a scientist) wishes to execute his own application in the cloud because he needs processing power.

A mOSAIC application is built up as a collection of interconnected *mOSAIC components*. Components may be (i) core components, i.e., predefined helper tools offered by the mOSAIC platform for performing common tasks, (ii) COTS (commercial off-the-shelf) solutions embedded in a mOSAIC component, or (iii) *cloudlets* developed using the mOSAIC API and running in a *Cloudlet Container*. mOSAIC cloudlets are stateless, and developed following an event-driven asynchronous approach (Petcu et al., 2011c; Petcu et al., 2011a).

The mOSAIC platform offers ready-to-use components such as queuing systems (*rabbitmq* and *zeroMQ*), which are used for component communications, or an HTTP gateway, which accepts HTTP requests and forwards them to application queues, NO-SQL storage systems (as KV store and columnar databases). mOSAIC components run on a dedicated virtual machine, named mOS (mOSAIC Operating System), which is based on a minimal Linux distribution. The mOS is enriched with a special mOSAIC component, the *Platform Manager*, which makes it possible to manage set of virtual machines hosting the mOS as a virtual cluster, on which the mOSAIC components are independently managed. It is possible to increase or to decrease the number of virtual machines dedicated to the mOSAIC application, which will scale in and out automatically.

Defining a new application is very easy: a cloud application is described in a file named *Application Descriptor*, it lists all the components and the cloud resources needed to enable their communication. A mOSAIC developer has both the role of developing new components and of writing application descriptors that connect them together.

3.2 Evaluation of Security

The methodology implemented to evaluate security is the Reference Evaluation Model (REM) (Casola et al., 2007a; Casola et al., 2007b), it defines how to express in a rigorous way the security policy, how

to evaluate a formalized policy, and how to state the provided security level. Any policy is represented through a tree, which contains all the policy provisions (intermediate nodes and leaves).

In Figure 1 the three methodology phases are shown: **Policy Structuring**, **Policy Formalization** and **Policy Evaluation**:

The goal of the **Structuring** phase is to associate an enumerative and ordered data type K_i to the n leave-provisions of the policy. A policy space "P" is defined as P $=K_1 \times K_2 \times \ldots \times K_n$, i.e. the vectorial product of the n provisions K_i. The space is defined according to a policy template that strongly depends on the application context.
In the **Formalization** phase the policy space "P" is turned into an homogeneous space "PS". This transformation is accomplished by a normalization and clusterization process which allows to associate a Local Security Level (LSL) to each provision; after that the provisions may be compared by comparing their LSLs.

The goal of the **Evaluation** phase is to pre-process the "PS" vector of LSLs and evaluate the so called Global Security Level L_{Px} associated to the policy P_x. The Global Security Level has been defined on the basis of an Euclidean distance among matrices and some reference levels:

$$L_{Px} = \begin{cases} L_0 & iff d_{x0} \leq d_{10} \\ L_1 & iff d_{10} < d_{x0} < d_{20} \\ L_2 & iff d_{20} < d_{x0} < d_{30} \\ L_3 & iff d_{30} < d_{x0} < d_{40} \\ L_4 & iff d_{40} \leq d_{x0} \end{cases}$$

where $d_{i,0}$ are the distances among the references and the origin of the metric space (denoted as $\emptyset$). This function gives a numerical result to the security; indeed the idea is to evaluate the security associated to an infrastructure through the evaluation of its security policy.

The GSL is a measure of the security provided by an infrastructure according to its security policy; it is obtained by formalizing the process that is manually performed by security experts while trying to extend trust to other domains. The details of the methodology are out of the scope of this paper, and they can be found in (Casola et al., 2007a).

4 A PRACTICAL EXAMPLE

In order to show in practice how the proposed approach works, in this section we propose the WS Security Policy template and the instances for the three generic providers.

The apply the REM to three Cloud Service Providers, denoted by **ProviderA, ProviderB** and **ProviderC** we need to describe the security mechanisms according to the WS-* models that we created.

To better understand the policy, we first summarize the behaviour of the three providers.
ProviderA applies a redundant security mechanism, it supports HTTPS for the *Authentication of the Service* and the *Message Confidentiality*. At Message Layer it supports HMAC to assure the identity of the sender (with a Secret Key for Encryption) and it also signs the body of the message ensuring the message integrity (operation redundant because it is on HTTPS).
ProviderB supports HTTPS for the REST call invocations, offering:

- *Authentication of the Service* : The user knows that he is sending the message to the right destination.
- *Confidentiality of messages* : No one can be aware of content of a sent message, nor alter it.

At Message Layer it attaches an `AuthenticationToken` that is created on the basis of a shared secret (password), concatenates the secret string to a string already containing a username, a timestamp, the user agent (http field) and then applies the MD5 hash function. The receiver performs the same operations before accepting the request.

ProviderC basically offers the same mechanism of ProviderB : HTTPS and an `AuthenticationToken` for the User Authentication. The only positive difference is that it uses SHA-1 instead of MD5.

4.1 Building the Common Template

In order to represent such features, Ws-SecurityPolicy supports three main types of containers:

- `Transport Binding` the security mechanism is enforced at transport layer, it may contain different subfields as `HttpsToken` and `TrasportToken`;
- `Symmetric Binding` the security mechanism is enforced at SOAP level as specified in Ws-Security, by using symmetric keys; it also allows to express other types of tokens as `SignatureToken` and `EncryptionToken`;
- `AsymmetricBinding` the security mechanism is enforced at SOAP level as specified in WS-Security, by using asymmetric keys.

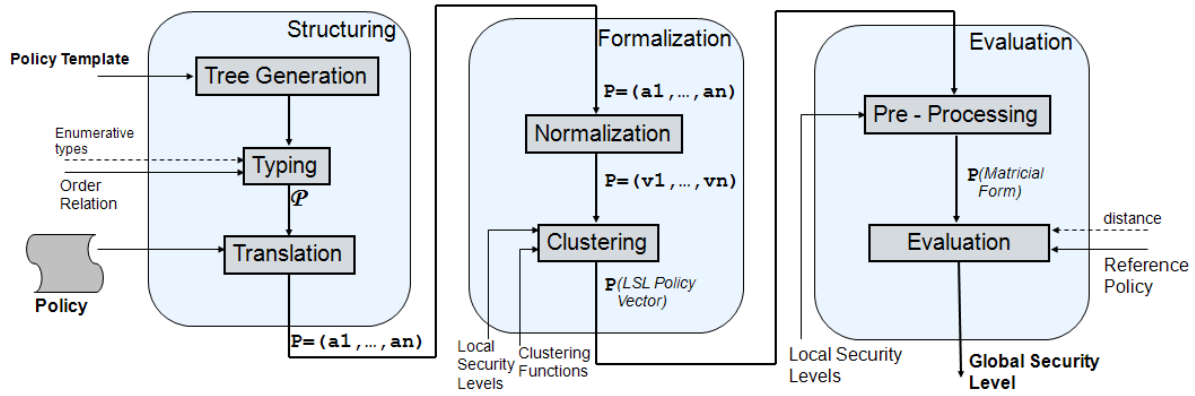


Figure 1: Phases of the evaluation methodology.

In our template we will focus only on the first two containers.

We defined a `TransportBinding` container, giving to the sequence `TransportBinding`/`TransportToken` the semantic meaning of Security at the Transport Layer, and to the `HttpsToken` subfield the semantic meaning of support of the HTTPS protocol.

We gave to the sequence `SymmetricBinding` / `SignatureToken` the semantic meaning of security at the Message Layer (Http message), where there may be some signed message parts. We did not add the sequence `SymmetricBinding`/`EncryptionToken`, because none of the analyzed Providers provides confidentiality on the message at the message layer. Furthermore, in `SignatureToken` we added predetermined fields including `AlgorithmSuite` and *SignedParts*.

In particular, among the possible `AlgorithmSuite` alternatives we used: `Md5`, `SHA1`, `HMAC`, `No` (from `XMLSignature` Specification).

Among the possible *SignedParts* subfields we added :

- `TimeStamp` against replay attacks (WS-Trust Specification);
- `Password` obtaining thus Authenticity of the Sender (Ws-Security Specification);
- `Body of Message` obtaining thus Integrity of the message;

By combining WS-Policy and WS-SecurityPolicy to define the template, we built the template presented in the following listing:

Listing 1: The Ws Security Policy Template.

```
<?xml version="1.0" encoding="utf-8"
    ?>
<wsp:Policy xmlns:wsp="http://www.w3
    .org/ns/ws-policy"
xmlns:sp="http://docs.oasis-open.org
    /ws-sx/ws-securitypolicy/200702"
xmlns:ds="http://www.w3.org/2000/09/
    xmldsig#"
xmlns:wsse="http://docs.oasis-open.
    org/wss/oasis-wss-wssecurity
secext-1.1.xsd"
xmlns:wsu="http://schemas.xmlsoap./
    ws/2005/trust
/Ws-Trust.xsd">
<wsp:ExactlyOne>
<wsp:All>
<sp:TransportBinding>
<wsp:Policy>
<wsp:ExactlyOne>
<wsp:All>
<sp:TransportToken>
<wsp:Policy>
<wsp:ExactlyOne>
<wsp:All>
<sp:HttpsToken sp:IncludeToken="
    http://docs.oasis
open.org/ws-sx/ws-securitypolicy
    /200702
    IncludeTokenAlwaysToRecipient"
wsp:Optional="true"/>
<!--NotPresent < Present-->
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
</sp:TransportToken>
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
</sp:TransportBinding>
<sp:SymmetricBinding>
<wsp:Policy>
<wsp:ExactlyOne>
<wsp:All>
<sp:SignatureToken sp:IncludeToken="
    http://docs.oasis-
```

```
open.org/ws-sx/ws-securitypolicy
    /200702
    IncludeTokenAlwaysToRecipient">
<wsp:Policy>
<wsp:ExactlyOne>
<wsp:All>
<sp:SignedParts>
<wsu:Timestamp wsp:optional="true"/>
<!--NotPresent < Present-->
<wsp:Body wsp:optional="true"/>
<!--NotPresent < Present-->
<wsse:Password wsp:optional="true"/>
<!--NotPresent < Present-->
</sp:SignedParts>
<sp:AlgorithmSuite>
<wsp:Policy>
<wsp:ExactlyOne>
<wsp:All>
<wsp:ExactlyOne>
<ds:SignatureMethod ds:Algorithm=
"http://www.w3.org/2000/09/xmldsig#
    hmac-sha1"/>
<ds:SignatureMethod ds:Algorithm=
"http://www.w3.org/2000/09/xmldsig#
    sha1"/>
<ds:SignatureMethod ds:Algorithm=
"http://www.w3.org/2000/09/xmldsig#
    md5"/>
<!--md5 < sha1 < hmac-sha1-->
</wsp:ExactlyOne>
</wsp:All>
<wsp:All>
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
</sp:AlgorithmSuite>
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
</sp:SignatureToken>
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
</sp:SymmetricBinding>
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
```

Note that a security expert can perform different kind of analysis on the policies. For example, being ProviderA the only one supporting HTTPS, one will expect it will perform better than the others. On the other hand, ProviderB and ProviderC do not ensure the integrity of the message, nor its confidentiality. If an attacker applies the man in the middle technique, he can replace the message with a different body. At that point even the identity of the sender loses meaning. If the attacker is not able to make the man in the middle technique but only sniff the messages, he could retrieve the md5 hash and crack it in seconds obtaining the user password. The SHA1 is stronger than md5 and he can only guess the password by a bruteforce attack. If the communication is on HTTPS, the attacker should perform a bruteforce attack and the longer is the password, more secure will be the mechanism.

About ProviderA, the attacker can not use the man in the middle technique, because the user has signed the content with HMAC, but he can observe it: the Security is in the difficulty with which the attacker can guess the key with a offline bruteforce attack. In particular, with HTTPS, the attacker should force with a brute force online attack which is impractical. From these considerations, it could be said that: the TransportBinding/TransportToken branch should leveling out differences however it makes appear the ProviderA the most secure. On the SymmetricBinding/SignatureToken branch the ProviderA should provide a Security Level quite high compared to the others.

Once made these preliminary actions, and applying the methodology, the results we obtain in the evaluation give three different Global Security Level. For brevity's sake we cannot report the whole evaluation process but the final result is:

- *P*roviderA GSL := 3;
- *P*roviderC GSL := 2.67;
- *P*roviderB GSL := 2.5;

Such results are coherent with an intuitive analysis of the behaviours described at start of the section.

Even if such analysis can be done by an independent expert, the approach we propose offers a clear step toward an automatization of the procedure, assuming a collection of Ws-Security-Policy descriptions of the services offered by different Cloud Providers, which can be done even by third parties.

5 THE SECURITY EVALUATION APPLICATION

In this section we will illustrate the functionalities that the cloud Evaluation Application can offer to a security evaluator.

The evaluation of the Security Level of a SecLA is an action that can be carried out by Security Administrators who will have the task of managing their Platform. They can be Developers who want to test an application or Methodology Researchers who want to use the application to run tests in order to study possible alternatives in the evaluation methodology field. The role of the security evaluator can be clearly hired

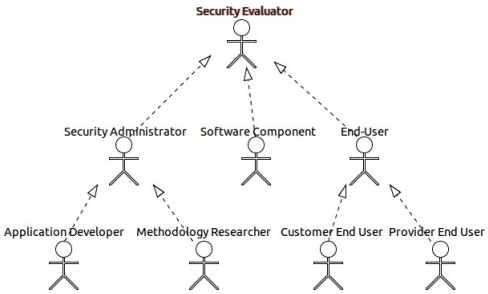


Figure 2: The Application Users.

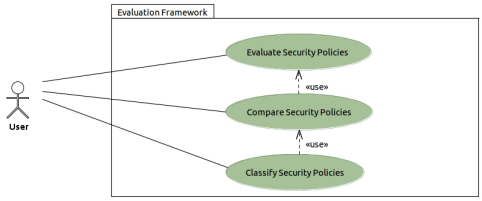


Figure 3: The Application Use Case diagram.

by End Users who use the application to choose a Provider. It in turn may be classified as a Customer End User, or as a Provider End User who wants to use the service to test the Security Level of their standards before entering the market. Figure 2 clarifies the relationships among all these actors.

The Application functionalities are reported in Figure 3 with the UML Use Case diagram.

The user can request the evaluation of the security requirements according to the REM Methodology (Casola et al., 2007a), compare them with other security policies (eventually supplied by cloud providers) and classify the results in order to choose the best solution.

In the next sections we will illustrate at first a Java library that enables the application of the REM Methodology, then we will present the mOSAIC components that enable its execution in a cloud environment.

5.1 The Java Library

The *Java Library* consists of a set of *Java Components* organized in packages.

The *Representation Package* supplies the data structures and all the mechanisms required to represent data. In particular, the *parsers package* contains a set of Java Classes to transform security policies expressed in an interoperable format (XML, for information tranfers) into a format suitable for the elaborations (Java Jung Tree) (Jun,) being the latter defined in the *data_structure package*. The ”parsers” and ”data_structures” packages can be extended with new Java Classes in case a new type of Security Policy (corresponding to a new template) should be processed by the system. The *distance_evaluator package* implements the specific evaluation technique. The *difference_evaluator* package provides classes able to compare two instances of Security Policies. These packages can be extended in order to implement new evaluation techniques which extend those already defined.

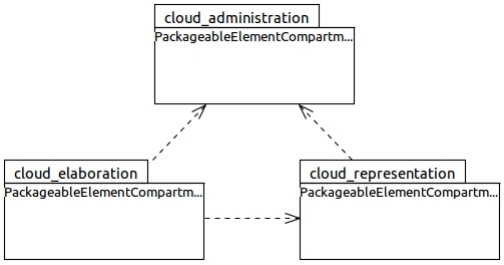


Figure 4: Cloudlets organization.

5.2 The Cloud Application Components

The Cloud application, based on the mOSAIC framework briefly described in section 3.1 is composed of a set of ready-to-use Cloudlets, that embeds the Java Components. The Cloudlets can be reused and composed by a developer in order to realize its custom evaluation application.

Figure 4, shows the implemented cloudlets organized by packages, depending on offered functionalities. The *cloud_representation* package contains the cloudlets offering to developers and final users a way to convert their Security Policies from an human readable format to a platform readable representation (and vice versa), ready to be elaborated. These cloudlets implement, using the mOSAIC API, the classes defined in the *representation package* of the Java Library; they are:

- *QSTRepresentation* (converts and stores the policy from readable format via user interface to internal representation);
- *WeightsRepresentation* (converts and stores informations about possible weights from user interface to internal representation);
- *ResultRepresentation* (converts and stores results internally represented to an human readable format from the interface);

The *cloud_elaboration* package contains the cloudlets offering to developers the Security Policy evaluation functionalities, with likelihood to choose between the various elaboration techniques. The cloudlets contained in this package therefore implement, using the mOSAIC API, the classes defined

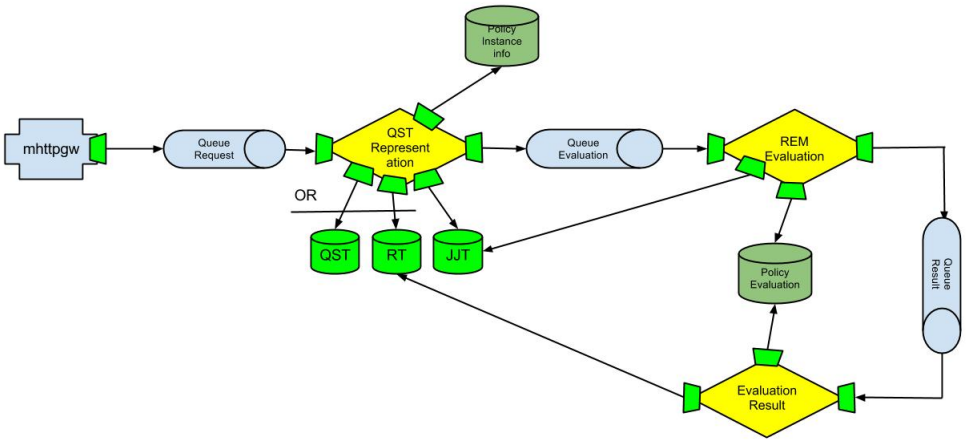


Figure 5: The mOSAIC Evaluation cloudlets.

in the *elaboration package* of the Java Evaluation Framework.

The cloudlets contained in the package are:

- *REMEvaluation* (offers the evaluation functionalities. It implements, using the mOSAIC API, the ”distance_evaluator” package of Java Evaluation Framework);
- *WeightsEvaluation* (applies the weights to security policies in accordance with defined techniques and the type of representation. It implements using the mOSAIC API the ”weights_evaluator” package of the Java Evaluation Framework);
- *RemCompare* (offers comparation functionalities between two instances of a Security Policy. It implements, using the mOSAIC API, the ”difference_evaluator” package of the Java Evaluation Framework);
- *Ranking* (rankes the Security Policies conveniently evaluated and compared);

The *cloud_administration* package contains essential cloudlets for the administration, the messages management, and the interoperability of cloudlets defined in the above packages.

Furthermore, in addition to defined packages, a set of *mangement kv-stores* is managed by the system. They keep the necessary informations for the correct execution of the mOSAIC Evaluation Applications.

Figure 5 shows the mOSAIC application that evaluates a Security Policy submitted by an user.

To describe in greater detail the modus-operandi of cloudlets, we proceed to illustrate in detail the operations of two cloudlets of the mosaic application.

The *QSTRepresentation* takes in input a message from an HTTP RESTful interface, extracts data, stores them (eventually) in the QST kv-store, and converts them in two different representations:

- RT (XML representation) (optional, for user retrieval);
- JJT (Java Jung Tree Data Structure representation);

The cloudlet sends the JJT representation of the Security Policy on an exit queue.

The *REMEvaluation* takes in input a message from a the representation cloudlet, containing the JJT representation of a Security Policy, evaluates it and stores the JJT evaluated. The cloudlet can send the JJT list of evaluated Security Policies on an exit queue.

Summarizing, the user defines the istance of his Security Policy (in the QST format) and submits it to the system through an HTTP interface (mhttpgw). This transforms the data into a suitable elaboration format through the QST Representation Cloudlet and sends it to REMEvaluation Cloudlet, which executes the evaluation according to one of techniques described in the early section. The chosen technique can be specified by the user through the same interface. The Evaluation Result Cloudlet stores the result in a manner that it can be referred later by the user.

6 CONCLUSIONS AND FUTURE WORKS

The *loss of control* over the resources is the first cause of the perception of low security in cloud computing. In this paper we made a fist step in the direction of offering to customers a new way to control the level of security offered by provider over the resources they control. We proposed a way to use standard languages

to build up comparable policies that enable customers to compare different provider on the basis of the technologies adopted using an automated and quantitative way. We proposed the adoption of WS Policy and WS Security Policy as standard languages to describe the mechanisms adopted by provider and, in future works, we aims at building a collection of policies describing the technological solutions adopted by different cloud providers. We have shown on real case studies the application of the proposed technique over real cloud providers, showing that the results obtained are coherent with a manual evaluation of the technology adopted. Moreover we proposed a Cloud application able to automatize the process and support the security evaluators to apply the technique. The results in this paper will be improved in the future, building a dedicated framework which suppot the development of custom evaluation application and able to fully exploit the flexibility of the underling methodology, which will be further extended.

REFERENCES

Jung - java universal network/graph framework toolkit.

Specs project.

Amato, A., Liccardo, L., Rak, M., and Venticinque, S. (2012). Sla negotiation and brokering for sky computing. In *CLOSER*, pages 611–620.

Bajaj, S., Box, D., Chappell, D., Curbera, F., Daniels, G., Hallam-Baker, P., Hondo, M., Kaler, C., Langworthy, D., Nadalin, A., et al. (2006). Web services policy 1.2-framework (ws-policy). *W3C Member Submission*, 25:12.

Casola, V., Mazzeo, A., Mazzocca, N., and Vittorini, V. (2007a). A policy-based methodology for security evaluation: A security metric for public key infrastructures. *Journal of Computer Security*, 15(2):197–229.

Casola, V., Mazzocca, N., Luna, J., Manso, O., and Medina, M. (2007b). Static evaluation of certificate policies for grid pkis interoperability. In *Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on*, pages 391–399. IEEE.

Della-Libera, G., Gudgin, M., Hallam-Baker, P., Hondo, M., Granqvist, H., Kaler, C., Maruyama, H., McIntosh, M., Nadalin, A., Nagaratnam, N., et al. (2002). Web services security policy language (ws-securitypolicy). *Public Draft Specification (Juli 2005)*.

Liccardo, L., Rak, M., Di Modica, G., and Tomarchio, O. (2012). Ontology-based negotiation of security requirements in cloud. In *Computational Aspects of Social Networks (CASoN), 2012 Fourth International Conference on*, pages 192–197.

Petcu, D., Craciun, C., Neagul, M., Lazcanotegui, I., and Rak, M. (2011a). Building an interoperability api for sky computing. In *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, pages 405–411. IEEE.

Petcu, D., Crăciun, C., Neagul, M., Panica, S., Di Martino, B., Venticinque, S., Rak, M., and Aversa, R. (2011b). Architecturing a sky computing platform. In *Towards a Service-Based Internet. ServiceWave 2010 Workshops*, pages 1–13. Springer.

Petcu, D., Craciun, C., and Rak, M. (2011c). Towards a cross platform cloud api. components for cloud federation. In *Procs. 1st International Conference on Cloud Computing and Services Science, SciTePress–Science and Technology Publications, Portugal*, pages 166–169.

Utility-based Decision Making in Collective Adaptive Systems

Vasilios Andrikopoulos[1], Marina Bitsaki[2], Santiago Gómez Sáez[1], Dimka Karastoyanova[1], Christos Nikolaou[2] and Alina Psycharaki[2]

[1]*Institute of Architecture of Application Systems, University of Stuttgart, Stuttgart, Germany*

[2]*Transformation Services Laboratory, University of Crete, Heraklion, Greece*

{*andrikopoulos, gomez-saez, karastoyanova*}*@iaas.uni-stuttgart.de,* {*bitsaki, nikolau, psyharak*}*@tsl.gr*

Keywords: Collective Adaptive Systems, Utility, Decision Making, Choreography.

Abstract: Large-scale systems comprising of multiple heterogeneous entities are directly influenced by the interactions of their participating entities. Such entities, both physical and virtual, attempt to satisfy their objectives by dynamically collaborating with each other, and thus forming collective adaptive systems. These systems are subject to the dynamicity of the entities' objectives, and to changes to the environment. In this work we focus on the latter, i.e. on providing the means for entities in such systems to model, monitor and evaluate their perceived utility by participating in the system. This allows for them to make informed decisions about their interactions with other entities in the system. For this purpose we propose a utility-based approach for decision making, as well as an architecture that allows for the support of this approach.

1 INTRODUCTION

Collective systems comprise heterogeneous entities collaborating towards the achievement of their own objectives, and the overall objective of the collective (Andrikopoulos et al., 2013). Such systems are usually large scale, typically consisting of both physical and virtual entities distributed both organizationally and geographically. Entities in these systems actively influence the operation of the system by interacting with other entities. Their behavior is guided by their individual objectives and the decisions they make in order to satisfy them. In this respect, support for decision making on the level of entities is an integral part of the realization of such systems.

For this purpose, in the EU ALLOW Ensembles project[1], we use the concept of Collective Adaptive System (CAS) (Kernbach et al., 2011), and define the underpinning concepts for modeling, execution and adaptation of CAS entities and their interactions. In particular, we propose to model and manage entities as collections of cells encapsulating their functionalities. Entities collaborate with each other to achieve their objectives in the context of ensembles describing the interactions among them. As a way to measure entity objectives' achievement, we use the concept of *utility* as a function over its preferences, context, state and interactions with other entities.

The contributions of this work can be summarized as follows:

1. We propose a utility-based approach for the purpose of decision making in CAS systems based on the conceptual model introduced in the project.
2. We extend an existing architecture in order to enable the modeling and execution of enabling mechanisms, and discuss their implementation based on well-established technologies.

The remaining of the paper is structured as follows: starting from a motivating scenario in Section 2, we then introduce our proposal on how to deal with decision making in CAS systems (Section 3). Consequently, in Section 4 we extend the architecture introduced in (Andrikopoulos et al., 2013) in order to enable the modeling and execution of the mechanisms supporting our proposal as distributed, large scale, pervasive systems; we also discuss the implementation of these concepts based on well-established technologies. The paper closes with a summary of related work (Section 5), and concludes with an outline of research challenges and future work (Section 6).

[1]ALLOW Ensembles: http://www.allow-ensembles.eu

2 MOTIVATION

Supporting citizens mobility within the urban environment is a priority for municipalities worldwide. As part of the effort to offer "smart services" to citizens, in the ALLOW Ensembles project the case of an urban mobility scenario is used as a CAS for demonstration and evaluation purposes (Andrikopoulos et al., 2013). The FlexiBus service is a special type of bus in this scenario that operates a flexible route set by passenger needs, allowing the advance booking of pickup points along a (dynamically) calculated route to a set of predefined destinations (i.e., city center).

Using the FlexiBus system, each *Passenger* can request a trip to one of the predefined destinations in the system by communicating with the *Route Planner*, asking to start at a certain time and from a preferred pickup point. Each passenger can pay their trip directly in the bus (cash, with a credit card or a monthly pass) or through the FlexiBus company web site. Furthermore, during the route execution, each passenger waiting for a bus can be notified for problems on a selected route (e.g. bus delays, accidents, etc.). Each *Bus Driver* is assigned by the *Route Manager* to a precise but dynamically calculated by the Route Planner route to execute, including the list of passengers assigned to it, and a unique final destination. Bus drivers communicate with an assigned Route Manager which monitors and ensures the correct execution of the route, in order to ask for the next pick-up point, and to communicate information like passengers' check-in. During the route realization, a bus on route can also accept passengers that have not previously booked if there are available seats and the passenger pick-up point is along the route.

The Route Planner attempts to define routes in a manner that reduces the wasteful overlap that occurs with many individual passengers traveling to the same destination. This means carrying more people in one route to radically reduce the number of FlexiBuses on the road, fuel usage, CO_2 emissions, traffic congestion, etc. The passengers on the other hand have concrete requirements on e.g. their arrival time, and they attempt to optimize across changing dimensions e.g. fare cost or time spent on the bus. The FlexiBus system is therefore required to support the interaction of different entities (software or physical) striving to achieve potentially conflicting and changing over time goals. In the following sections we present our proposal for using utility as the means for allowing these entities to co-exist and reconciliate their goals in the system.

3 APPROACH

3.1 The ALLOW Ensembles Model

In large-scale collective adaptive systems, entities which are heterogeneous actors of both human and software actors, interact in various ways to form a distributed and dynamic CAS system that evolves continuously to achieve a goal in a given context. For this purpose, in ALLOW Ensembles we model both types of actors as *entities* aggregating different functionalities, e.g. the check-in on a bus route, or the payment of the fare, as reusable *cells*. Cells encapsulate some functionality that the entity offers to, or requires from the system. Cells from different entities interact with each other as part of their predefined functionality.

The outcome of these interactions are *ensembles* that are created to fulfill specific goals initiated by the entities. Even though each entity has its own selfish interests, the cooperation with others entails an increase in each one's satisfaction expressed in utility terms. For this purpose, in the scope of the project we propose to model the economic perspective of systems of ensembles. In particular, we consider that entities take part in games according to various criteria in order to achieve a set of objectives. Entities have potentially conflicting interests within the system and strive to achieve individual objectives, thus formulating *non-cooperative games* (Osborne and Rubinstein, 1994). On the other hand, entities cooperate to achieve a common task and fulfill a set of group objectives, a situation that can modeled as *cooperative games* (Wiese, 2010). We incorporate both types of games to model entities' behavior according to the goals needed to be achieved.

3.2 Utility & Strategic Ensembles

Our position for dealing with such CAS systems as the ones discussed above is to analyze economic models that assign *utility* (Norstad, 1999) to entities according to their properties and impose constraints according to the desired goals. The utility of an entity is a measure of satisfaction experienced by the entity for using a service or consuming a good. Entities make choices so as to maximize their utility. A utility function is a way to quantify utility, assigning a number to every possible choice of the entity such that more preferred choices get assigned larger numbers than less preferred ones (Varian, 2010). We consider that the cells of an entity that are involved in the selected ensemble operate in order to satisfy the goal that is fulfilled by the ensemble and contribute to the improvement of the entity's utility. If the entity participates

in another ensemble (this is determined by the interactions that the entity decides to get involved to), the same cells operate in a new context that attributes a new level of utility to the entity. Game theory can be used to model the selection from a set of candidate ensembles based on the criterion of *utility maximization* (Varian, 2010): each entity chooses the ensemble that maximizes its utility taking into account the actions taken by the other entities too. Once an ensemble is being executed, it can be evaluated and compared to competitors or former ensembles through the *collective utility* that aggregates the individual utilities of all entities involved.

In order to model the economic behavior of our system, we define the concept of *meta-cells* that represent the economic characteristics of functional cells. We distinguish different activities that are performed by meta-cells according to the goals assigned to the respective functional cells. More specifically, they calculate the utility of an entity when participating in a given ensemble, as well as the collective utility of an ensemble derived by the various entities that interact to form the ensemble, they communicate with other cells and take decisions/compute strategies of entities, they collect data and perform measurements (resource consumption, satisfaction, costs, delays, prices, etc.), and they run optimization algorithms to improve the performance of ensembles.

Interactions of functional cells initiated by the entities result in the creation of functional ensembles (or simply, ensembles) in order to fulfill specific goals. Simultaneously, strategic utility-based interactions trigger the initialization of meta-cells that are managed by a new structure called the *strategic ensemble* in order to handle decision making on the level of interaction between entities. The objectives of a strategic ensemble include the following:

- to impose constraints according to entity goals in order to reduce the various choices of entities,
- to evaluate ensembles from the perspective of one entity according to the entity's preferences dynamically,
- to assign utility to each entity when participating in an ensemble in order to make the optimal choice and manage the negotiation between two entities.

Figure 1 illustrates one of the strategic ensembles in the FlexiBus scenario discussed in the previous sections. The meta-cells of the entities in the FlexiBus scenario are described as follows:

1. Route Evaluation (Route Planner): evaluates each route according to passenger preferences, passenger's previous choices and route's reliability based on historical data.
2. Routes Calculation (Route Planner): makes a list of routes according to passenger goals, examines whether the requirements of current passengers are violated (estimate new travel times) for all routes in the list and removes the routes that do not satisfy passengers' goals or negotiates with passengers, evaluates the routes according to passengers' preferences and FlexiBus' benefits.
3. Driver Allocation (Route Manager): negotiates with the FlexiBus drivers to find an appropriate Bus Driver entity to execute a route.
4. Utility Evaluation (Bus Driver): calculates the utility of the driver for the requested route according to his preferences and accepts or rejects the offer of the Route Manager to execute a route.
5. Utility Evaluation (Passenger): calculates a passenger's utility for each route in the list and announces the optimal choice. In Algorithm 1 we provide an example of passenger utility as a function of travel time t, trip cost C and a set of preferences such as payment method, smoking/non-smoking trips, seat preferences, delay tolerance and so on.

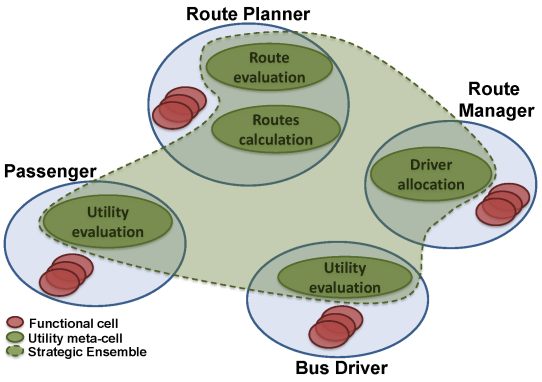


Figure 1: A Strategic Ensemble in the FlexiBus Scenario.

The structure of our methodology implies that information flows among the various components of the system. Meta-cells extract information from entities. Goals may impose constraints on meta-cells that solve optimization problems. Entities communicate through meta-cells and use them in order to select appropriate cells to form ensembles or reconfigure ensembles for increasing system's utility.

3.3 Strategic Ensembles Life-cycle

The life-cycle of the strategic ensemble is closely related to that of the execution ensembles. It is created before the execution ensemble, since the meta-cells

Algorithm 1 : Utility Evaluation Algorithm for Passenger Entities.

for all routes **do** calculate U:
$U = \sum_{i=1}^{k} w_i v_i + w_{k+1} e^{-at-bC}$, where

$$v_i = \begin{cases} \frac{(m+1)-j}{(m+1)} & \text{if choice } j \text{ of a total of } m \\ & \text{choices is satisfied} \\ 0 & \text{if preference } i \text{ is not satisfied,} \end{cases}$$

$w_i, i = 1,...,k$ are weights for each of k preferences and w_{k+1} the weight for the component that accounts for travel time and cost ($\sum_{i=1}^{k+1} w_i = 1$), and a and b are the impact factors for travel time t and cost C respectively.
end for
return $max\{U\}$

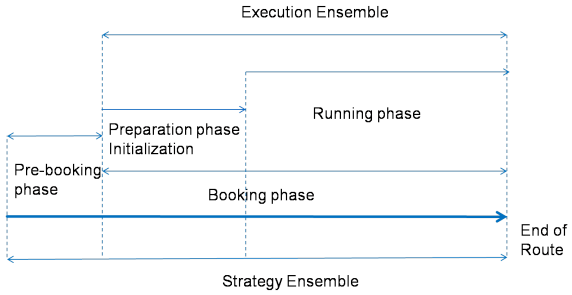


Figure 2: The life-cycle of the Strategic Ensemble.

are responsible for making decisions about the selection of the most beneficial ensembles. Moreover, it runs in parallel to the execution ensembles, since it affects the operations of the execution ensembles until their termination. We consider that each entity has its own set of meta-cells, which interact with other cells of this entity and with cells and meta-cells of other entities as well. Meta-cells of an entity join or leave a strategic ensemble and as a result the related execution ensembles, according to the decisions the entity makes based on her calculated utility.

In particular about the FlexiBus scenario, we consider two phases in the life-cycle of a route (execution ensemble): the *pre-booking phase*, where a route is going to be executed if a certain number of requests is reached until a certain deadline, and *the execution phase*, where the route is bound to start or it has already started. The timeline for the strategic ensemble that runs through this execution ensemble is shown in Figure 2. According to the scenario, the execution cell of the Passenger invokes her meta-cell by sending the trip request. Then, a series of meta-cell interactions start with the aim to make the optimal decision for each entity. The Utility Evaluation meta-cell of the Passenger invokes the Routes Calculation meta-cell of the Route Planner by sending the trip request and waits to receive a list of routes, in order to evaluate them. The Routes Calculation meta-cell interacts with the Route Evaluation meta-cell of the Route Planner, as well as with the active Route Managers in order to identify whether the Passenger can join an existing route. If this is possible, the Route Planner negotiates with the Passengers already in the route, since they may need to re-evaluate their utility from being on this route, if e.g. the arrival time changes due to the additional passenger. The list of routes returned to the Passenger are evaluated by the Passenger using Algorithm 1 and ranked accordingly. The Passenger then joins in an (execution) ensemble with the Route Manager, Bus Driver and other Passengers in the route that she evaluated as optimal.

4 ARCHITECTURE & IMPLEMENTATION

An architecture design capable of supporting the phases of the CAS lifecycle is presented in (Andrikopoulos et al., 2013). However, such architecture does not support the utility-based decision making capabilities proposed in the previous section; for this purpose in this work we enhance it with the necessary components as depicted in Figure 3.

The architecture for the modeling and the execution of CAS systems comprises two major component groups: a *Modeling Tool* and a *Runtime Environment*. The *Modeling Tool* consists of three main components: a *Choreography Modeler* to create choreography models which specify the interactions between cells of functional and strategic ensembles, a *Transformer* to generate process skeletons that can be completed to executable processes, which can be subsequently modified in the *Process Modeler* The *Modeling Tool* is developed as a pair of interoperable Eclipse Graphical Editors that support the specification of choreographies in the BPEL4Chor language (Decker et al., 2008). Both functional cells and meta-cells are specified using the WS-BPEL language, and therefore the process is developed as an extension of the BPEL Eclipse designer described in (Sonntag et al., 2012). The editor is also equipped with monitoring capabilities that provide the user real-time information related to the flow execution.

The *Runtime Environment* comprises the necessary components to enable the execution of both cells and meta-cells. The cell and meta-cell models are deployed on one or more *Execution Engines* and can be instantiated at any time. Cell models may contain abstract activities, and therefore the Execution Engine must be able to start the execution of incomplete pro-

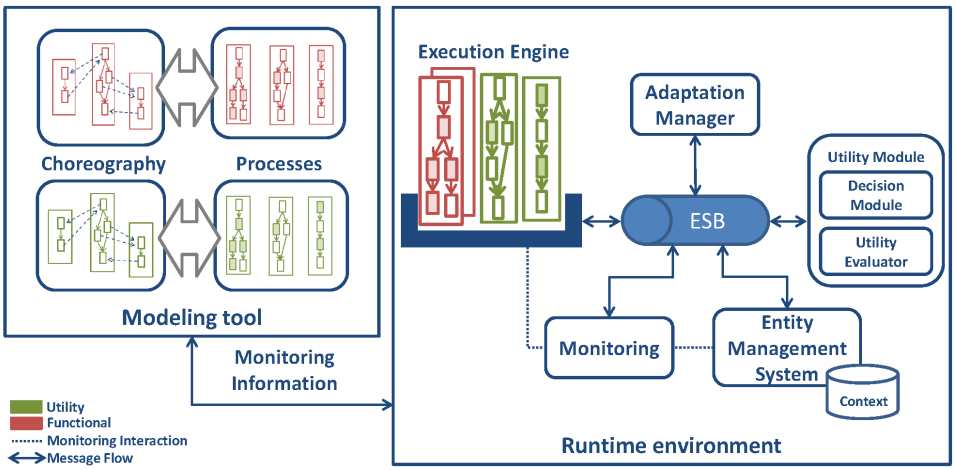


Figure 3: Architecture overview

cesses, allowing the dynamic injection of additional activities retrieved from the *Adaptation Manager* or triggered through the *Modeling Tool*. The realized *Execution Engine* is an extended Apache ODE Engine[2], an open source implementation of BPEL, and presented in (Sonntag et al., 2012). The *Entity Management System (EMS)* deals with all aspects of entity management. When a new entity is created, its available functional and meta-cells are deployed in the *Execution Engine*, and its corresponding properties to its context model. The EMS allows the Adaptation Manager to access the cell models, instances, context properties, and the necessary data for utility evaluation, e.g. user preferences, needed for the utility based planning and adaptation operations.

The interaction of the *Utility Module* with one or multiple meta-cells participating in a strategic ensemble provides the necessary functionality for utility evaluation and decision making support. The *Decision Module* and the *Utility Evaluator* deal with such operations in a generic and domain independent manner. Therefore, meta-cells are responsible for retrieving the necessary data, e.g. context information, user preferences, etc., and orchestrate the multiple services offered in the *Utility Module*. Monitoring information retrieval, e.g. KPIs or WS-BPEL Events (Kopp et al., 2011) functionalities must be wrapped in the *Monitoring* component.

All components should be provided as services and communicate through an *Enterprise Service Bus (ESB)* solution to facilitate their integration. As multiple organizational levels can be present in such systems, we use the ESB^{MT} multi-tenant aware ESB solution, as presented in (Strauch et al., 2013), for ensuring communication isolation between multiple organizations and its users. ESB^{MT} enhances the Apache ServiceMix solution[3] with multi-tenant communication support within service endpoints deployed in the ESB, and multi-tenant aware dynamic endpoint deployment and management capabilities.

5 RELATED WORK

Several studies related to the collective or adaptive aspects of complex systems have been driven in areas such as *Swarm Intelligence* (C. Pinciroli et al., 2011; Levi and Kernbach, 2010), *Autonomic computing* (Bruni et al., 2012; Lewis et al., 2012), and *Multi-agent based systems* (Cabri et al., 2011; Lavinal et al., 2006). These converge in the need for defining interaction rules between participants towards adapting the system under ordinary and extraordinary events. However, there exists a gap towards a general solution definition for all aspects in CAS. In this work we aim to target this problem by proposing the usage and extension of existing modeling and execution approaches towards providing a generic and evolutionary utility-based adaptation in CAS.

Interactions between multiple participants in a choreography can be modeled following the interaction or the interconnection modeling approaches (Barker et al., 2009). In the former, communication between participants is modeled using atomic *interaction activities*. The WS-CDL (Kavantzas et al., 2005) and the Savara[4] project are approaches that support the explicit specification of the *interaction activities*. On the other hand, the interconnection modeling approach consists on interconnecting the communication activities of each partic-

[2]Apache ODE: http://ode.apache.org/

[3]Apache ServiceMix: http://servicemix.apache.org

[4]http://www.jboss.org/savara

ipant of the choreography. This approach is supported in the CHOReOS Integrated Development and Runtime Environment[5], in the Open Knowledge European project[6], and in BPEL4Chor (Decker et al., 2007). As BPEL4Chor enables the choreography specification atop of WS-BPEL and decouples the choreography behavior specification from the technical communication details, it is considered as an appropriate extensible point in CAS modeling and specification.

Value is intimately linked to service systems: the latter exist only as long as they can create value for both the providers and the consumers; otherwise they dissolve and disappear. In (Bloch and Jackson, 2007), value is defined as "benefits of an agent accrued by his participation in the network minus any costs involved in setting up the network links directly or indirectly". A method is proposed in (Caswell et al., 2008), for computing values by taking into account service system partners' satisfaction and additional value accrued by the relationship levels of the various partners. In (Gordijn et al., 2001), the authors combine IT systems analysis with business modeling to build an e-business model that specifies e-business scenarios rather than defining values.

While various approaches have been proposed to measure service network performance, little experimental testing or theoretical investigation on competing networks has been performed. (Biem and Caswell, 2008) describe building block elements of a value network model and design a network-based strategy for a prescriptive value network analysis. (Allee, 2008) provides a systematic but qualitative way for approaching the dynamics of intangible value realization, inter-convertibility, and creation. Both of the above approaches use qualitative methods to describe value in a service network in contrast to (Caswell et al., 2008) that calculates value in a quantifiable manner. A service system's performance is studied in (Voskakis et al., 2011) by solving value optimization problems with respect to service prices.

(Van Dinther et al., 2009) studies the strategic behavior of service providers within service value networks and proposes an auction-based mechanism to efficiently match service offers and requests and determine prices, showing that incentive compatibility holds under certain conditions. (Katsamakas, 2005) analyzes industry structure in the presence of value networks and has developed a model dealing with a number of design aspects, such as the supplier number and the partner investments importance. It is shown that industry structure is more likely to shift to competition between value networks in which IT plays an important role. Systems encountering competition between service networks, analyzed with respect to their evolution through well-defined strategies, have received little attention. Initial work is provided in (Bitsaki et al., 2012) on competing service networks by defining and simulating strategies in games of incomplete information. The results show that Nash Equilibria exist when one player represents each service system.

[5]CHOReOS: Large Scale Choreographies for the Future Internet: http://www.choreos.eu/

[6]Open Knowledge: http://www.openk.org/

6 CONCLUSIONS AND FUTURE WORK

Collective Adaptive Systems (CAS) are formed by multiple heterogeneous virtual and physical entities that collaborate with each other towards fulfilling their individual objectives (Kernbach et al., 2011). However, such collaboration can change over time, as entities' objectives can dynamically adapt or change due to external or internal environmental events. Therefore, on the one hand entities should be provided with the necessary mechanisms that will guarantee a successful fulfillment of their objectives, while on the other hand such mechanisms must minimize the impact of entities' influences on the collective. For this purpose, the concepts of utility and decision making are brought into the cells and ensembles paradigm of CAS in the ALLOW Ensembles project.

In this work we focus on providing the necessary support to entities to facilitate the modeling, monitoring, and evaluation of their perceived utility by participating in the system. Therefore, we describe in a first step a conceptual view on meta-cells and strategic ensembles, and propose the strategic ensemble lifecycle as an emerging negotiation mechanism. A utility evaluation algorithm is then proposed for a concrete entity of the FlexiBus scenario. We plan to extend the approach to model the various interactions among entities by means of games in order to make decisions on which ensembles to choose to participate or which entities to add in an existing ensemble, taking into account future passenger demands or other environmental events.

In a second step, an architectural and a work in progress implementation approach presented in (Andrikopoulos et al., 2013) is enhanced towards supporting the multiple required phases for utility-based decision making in CAS. Future work also focuses on analyzing and utilizing multiple provisioning and management techniques for supporting entities joining and leaving the system, as well as investigating

on different distribution and deployment options of the runtime environment. For these purposes, the previously described components must be extended and integrated in the prototype implementation.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the EU's Seventh Framework Programme (FP7/2007-2013) project ALLOW Ensembles (grant agreement no. 600792), and from the German Research Foundation (DFG) within the Cluster of Excellence (EXC310) in Simulation Technology.

REFERENCES

Allee, V. (2008). Value network analysis and value conversion of tangible and intangible assets. *Journal of Intellectual Capital*, 9(1):5–24.

Andrikopoulos, V., Bucchiarone, A., Sáez, S. G., Karastoyanova, D., and Mezzina, C. A. (2013). Towards Modeling and Execution of Collective Adaptive Systems. In *Proceedings of WESOA'13*, pages 1–12. Springer.

Barker, A., Walton, C. D., and Robertson, D. (2009). Choreographing web services. *IEEE Transactions on Services Computing*, 2(2):152–166.

Biem, A. and Caswell, N. (2008). A value network model for strategic analysis. In *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*, pages 361–361. IEEE.

Bitsaki, M., Nikolaou, C., Voskakis, M., van den Heuvel, W.-J., and Tsikrikas, K. (2012). Performance analysis and strategic interactions in service networks. *International Journal On Advances in Networks and Services*, 4(3 and 4):375–385.

Bloch, F. and Jackson, M. O. (2007). The formation of networks with transfers among players. *Journal of Economic Theory*, 133(1):83–110.

Bruni, R., Corradini, A., Gadducci, F., Lluch-Lafuente, A., and Vandin, A. (2012). A conceptual framework for adaptation. In *FASE*, pages 240–254.

C. Pinciroli et al. (2011). Argos: A modular, multi-engine simulator for heterogeneous swarm robotics. In *IROS*, pages 5027–5034.

Cabri, G., Puviani, M., and Zambonelli, F. (2011). Towards a taxonomy of adaptive agent-based collaboration patterns for autonomic service ensembles. In *CTS*, pages 508–515.

Caswell, N. S., Nikolaou, C., Sairamesh, J., Bitsaki, M., Koutras, G., and Iacovidis, G. (2008). Estimating value in service systems: a case study of a repair service system. *IBM Systems Journal*, 47(1):87–100.

Decker, G., Kopp, O., Leymann, F., Pfitzner, K., and Weske, M. (2008). Modeling Service Choreographies Using BPMN and BPEL4Chor. In *Proceedings of CAiSE'08*, pages 79–93. Springer.

Decker, G., Kopp, O., Leymann, F., and Weske, M. (2007). BPEL4Chor: Extending BPEL for Modeling Choreographies. In *Proceedings of ICWS'07*, pages 296–303, Salt Lake City. IEEE Computer Society.

Gordijn, J., Akkermans, H., and Van Vliet, J. (2001). Designing and evaluating e-business models. *IEEE intelligent Systems*, 16(4):11–17.

Katsamakas, E. (2005). Competing value networks, incomplete contracts and it. In *Proceedings of HICSS'05*, pages 212b–212b. IEEE.

Kavantzas, N., Burdett, D., Ritzinger, G., Fletcher, T., Lafon, Y., and Barreto, C. (2005). Web services choreography description language version 1.0.

Kernbach, S., Schmickl, T., and Timmis, J. (2011). Collective adaptive systems: Challenges beyond evolvability. *ACM Computing Research Repository (CoRR)*.

Kopp, O., Henke, S., Karastoyanova, D., Khalaf, R., Leymann, F., Sonntag, M., Steinmetz, T., Unger, T., and Wetzstein, B. (2011). An Event Model for WS-BPEL 2.0. Technischer Bericht Informatik 2011/07, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany.

Lavinal, E., Desprats, T., and Raynaud, Y. (2006). A generic multi-agent conceptual framework towards self-management. In *NOMS*, pages 394–403.

Levi, P. and Kernbach, S. (2010). *Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution*, volume 7. Springer Verlag.

Lewis, P., Platzner, M., and Yao, X. (2012). An outlook for self-awareness in computing systems. *Awareness Magazine*.

Norstad, J. (1999). An introduction to utility theory. Unpublished manuscript at http://homepage. mac. com/j. norstad.

Osborne, M. J. and Rubinstein, A. (1994). *A Course in Game Theory*. MIT Press.

Sonntag, M., Hahn, M., and Karastoyanova, D. (2012). Mayflower - Explorative Modeling of Scientific Workflows with BPEL. In *Proceedings of BPM'12*, pages 1–5. CEUR Workshop Proceedings.

Strauch, S., Andrikopoulos, V., Sáez, S. G., and Leymann, F. (2013). ESB^{MT}: A Multi-tenant Aware Enterprise Service Bus. *International Journal of Next-Generation Computing*, 4(3):230–249.

Van Dinther, C., Blau, B., and Conte, T. (2009). Strategic behavior in service networks under price and service level competition. In *Wirtschaftsinformatik (1)*, pages 599–608. Citeseer.

Varian, H. (2010). *Intermediate microeconomics: a modern approach*. Norton & Company.

Voskakis, M., Nikolaou, C., van den Heuvel, W.-J., and Bitsaki, M. (2011). Service network modeling and performance analysis. In *Proceedings of ICIW'11*, pages 58–63.

Wiese, H. (2010). Applied cooperative game theory. http://www.uni-leipzig.de/ micro.

A Web Service Discovery Approach Based on Hybrid Negotiation

Raja Bellakhal[1], Walid Chainbi[2] and Khaled Ghédira[1]
[1]*Higher Institute of Business, University of Tunis, Tunis, Tunisia*
[2]*National Engineering School of Sousse, University of Sousse, Sousse, Tunisia*
{Raja.Belakhal, Walid.Chainbi}@gmail.com, khaled.ghedira@isg.rnu.tn

Keywords: Web Service Discovery, Negotiation, Quality of Service (QoS), Multi-Agent-Systems (MAS).

Abstract: An effective discovery system must be able to retrieve services responding to the users' specific preferences in a changing and dynamic environment. Actually, the existing discovery systems have three problems. Firstly, some of them fail to find Web services providing the same QoS as defined in their related description files, since the QoS data are considered as static. Secondly, the discovery systems based on negotiation lack the accuracy in simulating similarly the real humans' interactions. Thirdly, the negotiation based approaches implemented to discover services are static and do not consider contexts as well as characteristics of each provider. These shortcomings affect negatively the systems performance and usability. Consequently, the quality of the returned services as well as the systems' reputation will be deteriorated. In this paper, we propose an hybrid discovery approach based on negotiation that solve these drawbacks. We argue that our approach enhances the discovery system performance and usability by implementing a negotiation process that is closer to humans' interactions. Moreover, by considering the existing dependencies between the concurrent negotiations, the discovery process will be more efficient. Unlike previous work, the negotiation process is dynamic by taking into account the provider's context and reputation.

1 INTRODUCTION

The Web is no longer a simple way to access information, its evolution makes it also a provider of services. Web Services refers to applications exposed by providers over the Web. They provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks. Service discovery is the process of locating Web service, and retrieving Web services descriptions that have been previously published (Bromuri et al., 2009). Often Web services operate in a rapidly changing environment. Consequently, their QoS parameters change quite frequently, either due to the external (e.g., the variance on the service demand, server outages, the congestion at the router (Bannock Consulting, 2000)) or the internal (e.g., poor upstream connectivity (Kouki et al., 2013), the changes of the providers' preferences (Bannock Consulting, 2000)) environment changes. In most studies dealing with the discovery task, QoS parameters are considered as static (Ran, 2003), (Zhou and Chen, 2005), (Lakhal and Chainbi, 2012). Therefore, the non-functional features of the returned services will not be the actual ones that reflect the service functioning. Some studies have tried to find solutions for the static QoS parameters problem (Chen et al., 2010), (Zhou and Chen, 2009), (Platzer et al., 2007). These solutions use respectively statistics (e.g., the sampling method (Zhou and Chen, 2009)) and networks (e.g., sniffing, probing, and interceptors (Platzer et al., 2007)) based methods. These technologies need over resources utilization as well as huge computational and statistical efforts. Recently, other works adopting the negotiation approach in order to dynamize the QoS parameters appeared (Bentahar et al., 2008), (Napoli et al., 2013). The adoption of negotiation while discovering the best service has also several benefits including the discovery time optimization, the resolution of conflicts between the providers' and clients' preferences, and the improvement of the success rate. However, the main weakness of these solutions is related to the difference between humans' behaviors during their interactions and those of agents during the systems functioning. These systems do not simulate some common

negotiators' behaviors observed during their real life interactions. Simulating these details about negotiators' behaviors will positively impact the system's performance and outputs. During real life negotiation, each client uses his/her experience, competences, and knowledge to negotiate differently his/her goals. Also, negotiators make continuously new offers and give explanations about the reasons of choosing such offers. In our approach, negotiators will generate offers, give explanations and behave differently. Additionally, in these systems the negotiation strategies are static. These strategies are used by all providers without caring about the specific context and characteristics of each one of them. Also the consideration of dependencies between the different negotiation processes is missed. These dependencies are the cooperation relations existing between the different client's instances. These instances exchange information about the providers' offers and use them to propose more attractive offers. Considering these dependencies make more efficient the overall negotiation process. The objective of this paper is to present a discovery approach based on an hybrid and a dynamic negotiation process. Given that they are software components that have autonomous behavior and that try continuously to meet their own objectives, by taking into account their resources, skills and perceptions, agents are suitable entities to be adopted for the simulation of both users' and the providers' behaviors during their interactions (Chaib-draa et al., 2001).

The reminder of this paper is organized as follows. In the second section, we review the existing discovery approaches, and we outline their main characteristics as well as their drawbacks. In the third section, we highlight the main contributions brought by our approach. A buying-selling case study that illustrates our approach is presented in the last section. Finally, we summarize our work and we give hints concerning our future work.

2 RELATED WORK

Two types of discovery approaches have been proposed to solve the static QoS parameters problem, namely, the discovery approaches that are network as well as statistical based methods, and those that are adopting the negotiation. Compared to researches dealing with the first type of approaches, the negotiation based works are fewer.

2.1 Approaches based on Network and Statistical Methods

In (Al-Masri and Mahmoud, 2009) a WSB (Web Service Broker) is used to update the QoS parameters values. More especially, a WSCE (Web Service Crawler Engine) is used to collect dynamically the updated QoS data. In (Julie and Kumar, 2012), a collaborative filtering approach that predicts automatically the QoS parameters values is adopted. An important drawback related to this work is that there are no large-scale real-world Web service QoS data sets available for studying the prediction accuracy. In (Zhou and Chen, 2009) the data related to the users' and the providers' feedbacks are gathered and then processed according to a set of statistical methods. The drawback of approaches which consider the users' and the providers' feedbacks is related to the big statistical efforts computations. Other works use statistical methods to estimate values of the QoS parameters before the reception of feedbacks (Thio and Karunasekera, 2005). In these approaches, the most statistical jobs are executed when original data is received in the service registry. Consequently, this will take up many other hardware resources of the service registry (Zhou and Chen, 2009). In (Chen et al., 2010), the authors present a model for the management and the discovery of services based on two dynamic QoS parameters, namely the response time and the execution time. The monitor that acts as an interceptor during each service invocation is used. Most of the existing works enabling the dynamicity of the QoS data that are discussed above are suffering from general drawbacks. Many of these approaches are restricted in the sense that they can't handle the dynamicity of some QoS parameters types including the service's cost and the service's price. These parameters are of a paramount interest in the cloud computing field, and should be considered as dynamic during the discovery process. Additionally, some of the presented approaches using statistical methods require over resources utilization as well as huge computational efforts. Moreover, since some systems use the gathered information during the service invocation to keep up-to-date, then in the case where a service is not invoked periodically, its related non-functional parameters can't reflect the actual service's performance.

2.2 Approaches Based on Negotiation

Bentahar et al., adopt in the context of the

communities' construction, an argumentation based approach (Bentahar et al., 2008). The selection of the best service is done in terms of functionality and QoS (i.e., response time, availability, throughput, reliability, etc.). Agents use knowledge, beliefs, and argumentation capabilities to reason. A persuasive negotiation protocol composed of different dialogue game was used. A dialogue game is the set of moves that are made during the negotiation. This work adopts only one model of negotiation and the complexity of the used protocol depends on the size of the agents' knowledge bases. A negotiation approach that adopts a coordinator between the different Web services based on the MAIS (Multichannel Adaptive Information Systems) architecture is presented in (Comuzzi and Pernici, 2005). The negotiation capabilities related to each provider are presented as policies in the service specification. Services involved in the negotiation are selected, and then one auction based strategy is chosen from the different auction models and applied between these services. In this work, only one negotiation model is used and the implemented system is not based on the agent technology. The work presented by Napoli et al., is close to our work, since it deals with the static QoS parameters problem (Napoli et al., 2013). However, unlike our work, the generation of offer related to the QoS parameters is permitted only in the provider's side, the user has only the role of evaluating offers. In this work, the standard contract net protocol and a unique negotiation model are adopted. The paper presented by Siala and Ghédira use a variant of the contract-net protocol, called the directed contract-net protocol that has the effect of decreasing the number of the negotiation rounds comparing to the standard Contract-Net while composing services. A coordinator supervising the different negotiations is used. Dependencies between the different negotiations strategies are not considered in this work (Siala and Ghédira, 2011). Bromuri et al., present a decentralized multi-agent system that adopts an argumentation based strategy to find the best service. In this work, the best service is firstly selected among a set of the available services, then a negotiation session is started with the chosen service in order to achieve an agreement about the required QoS parameters. This strategy restricts the client's chances to find the best offer since he/she negotiates his/her preference with only one provider, while other providers can have more interesting offers (Bromuri et al., 2009).

3 THE HYBRID NEGOTIATION APPROACH CONTRIBUTIONS

Given the promising solutions brought by negotiation, recent discovery approaches based on negotiation models arise. A set of these works have been already presented in the related work section. These works have three common problems. Firstly, the negotiation systems presented by these works lack the realism in simulating perfectly the real human interactions and behaviors during the negotiation process. In fact, only a unique negotiation strategy based on one negotiation model is adopted for simulating providers' and clients' behaviors. In real life each client and provider has his/her own strategy to negotiate. In these works, all negotiators are obliged to follow the same strategy in order to reach their different goals. In this paper, our aim is to relax this assumption by introducing a system which applies more than one strategy based on different negotiation models. In our work, we adopt the argumentative and the heuristic models. The formal description of the adopted strategies is out of the scope of this paper, since our main interest is to present our discovery process based on hybrid negotiation. Secondly, all these works assume that the negotiation process is static i.e., once the negotiation strategy is defined by developers it will be applied similarly with all providers without considering their different context and/or characteristics. Finally, dependencies between the different negotiations are missed. Each bilateral negotiation involved in the whole negotiation process is conducted regardless to other processes which result in longer negotiation process and sub-optimal offers. In our study, we solve these problems by considering three main contributions. The first contribution is to make more realistic the negotiation process by simulating some details observed during the real humans' interactions. The second contribution is to make dynamic the negotiation process. The final contribution is to consider dependencies between the concurrent bilateral negotiations.

3.1 The Simulation of the Real Humans' Interactions

The simulation of humans' interactions implies the creation of an artificial world of agents that are in interaction and that have humans' capabilities to interact with their environment and to react according to their own perception, goals, and beliefs.

Since systems are substituting the real human interactions, they have to be as much as possible close to the humans' behaviors as well as to their strategies during the negotiation. The second feature observed in human interactions, but that is missed in the current systems dealing with the discovery of the best service is the exchange of explanations between the different parties. In our work, we solve conflicts among negotiators by enabling the two parties to exchange explanations through arguments. In works dealing with the service discovery based on argumentative based models the explanatory role is not yet studied. The third feature characterizing the real human interactions that is missed in some of the existing works is related to the providers' and clients' role in generating offers. Information about the opponent's offers is used to propose more attractive offers during the next negotiation rounds.

3.1.1 An Hybrid Strategy

During their daily real business interactions, sellers and buyers are using different strategies to convince each others about their offers and preferences. Each negotiator has his/her own way to convince his/her opponent and to generate offers. Obliging all providers and all clients to adopt the same strategy is not possible since some of them haven't essential information and backgrounds to use systems based on some kind of negotiation models. For example, if the negotiation system implements an experience based strategy, users who haven't yet experience and who want to purchase the service for the first time, will find problems in using this system. Also other models such as the argumentation based model require a predefined set of knowledge and arguments. However, sometimes users aren't able to formulate readily suitable arguments to convince their opponents and to express fluently all their preferences. Implementing different strategies will improve the discovery system usability level. Usability is defined by three features namely, the system's effectiveness, the system's efficiency, and the user's satisfaction (Frøkjær et al., 2000). Effectiveness is related to how much the system is able to respond the users' requests. Satisfaction is related to the comfort and subjective user's evaluation of the interaction. The efficiency is related to how the system is able to achieve the best result with less effort and the minimum time. A system qualified by a high usability level is better suited to be consumed by different types of users. Along with such a system, users having different requirements are more luckily to find the functionalities and the conditions helping them to discover their services. Consequently, the reputation of the discovery system will be improved. When adopting different strategies we try to balance between the cost and the effectiveness. The argumentative model is powerful enough to achieve an agreement but it's costly in terms of consumed resources and complexity. While a less powerful model such as the heuristic model can improve the complexity and decrease the chances to achieve an agreement. Moreover, contexts and attitudes of each provider are different, so it's difficult to apply the same strategy with all providers. The client has to collect information about the provider's attitude and accordingly he/she must choose the adequate strategy that complies with the providers' characteristics.

3.1.2 The Role of Clients and Providers in Giving Explanations

The provider's role in bringing explanation for users about some aspects that may be not obvious for them is missed within the existing discovery approaches. This is possible with the negotiation via the adoption of the explanatory arguments. For example, the client can ask the provider if it's possible to have a discount or a higher service availability level. Also he/she can inquire further details about the service functionalities and about the reason of uttering a given offer. Within the negotiation mechanism, the provider is able to respond to such requests and to give additional explanations. In works adopting the argument based models in order to select the best service, other types of arguments are used, including rewards and threats. These arguments have the effect of inducing and convincing negotiators rather than giving explanations. In the case of an explanatory argument about the last offer, the opponent examines firstly the argument and accordingly he/she can decide if he/she will accept or refuse it. If the opponent's beliefs and preferences comply with the explanatory argument then he/she will accept it. Otherwise, this explanation will be used by the opponent during the building of his/her next offers. The opponent will have an idea about the proponent's preferences and beliefs, and accordingly he/she can formulate more attractive offers to him/her. Explanatory arguments have the role of resolving conflicts among the negotiators' preferences. Clients and providers use them to explain the reason of refusing the opponent's offer. This information will be then retained by the opponent and used to exchange more

attractive offers. This will make faster the discovery process and helps to resolve gradually conflicts. Also, the quality of the last offer will be improved, since it will be the result of exchanging more attractive proposals.

3.1.3 The Role of Clients and Providers in Generating Offers

In some existing works, the provider has the role of generating offers, while the user has only to evaluate offers (Napoli et al., 2013). In our work, users and providers generate and evaluate offers. Our main interest is satisfying the users' goals. To do so, users must have their own strategies and tactics to generate offers reflecting their preferences as is done during the real human interactions. These proposals will be used by providers as basis to the construction of their next offers. Both parties must evaluate and generate offers because this will make smarter and faster the negotiation process. Evaluating only offers doesn't show the preferences of the negotiators, rather it reveals only their positions. Thus, a rational client respectively (provider) who wants to gain the negotiation and to reach as faster as possible an agreement must generate offers rather than evaluating only the provider's (the client's) offers. This will help both negotiators to construct faster an idea about each other's preferences, and then to reach more rapidly an agreement.

3.2 The Dynamic Negotiation Process

Most of works dealing with the selection of the best service by adopting a negotiation approach assume that the negotiation process is static. In our work, we make dynamic the negotiation process by selecting the appropriate negotiation strategy to be adopted with each provider. In fact, the choice of the best strategy depends on different parameters including the provider's context, reputation, and preferences. Before starting the negotiation, the system must gather information about providers. Accordingly, the best strategy to be adopted with these providers is selected. For example, systems must have an idea about the provider's context such as, if the provider is selling during a high or a sale season. In the latter case, the provider is willing to make more concession. Accordingly, the system can choose a negotiation strategy that doesn't need big efforts in convincing the provider such as a heuristic based strategy. However, if the provider is in a high season which means that there is a big competition among providers, then the system may choose a stronger strategy such as an argument based strategy. Reputation is also important. Reputation is determined based on the attitude of the provider during his/her past interactions. The system should know if the provider is a stubborn negotiator or a collaborative one. Accordingly the appropriate negotiation strategy is adopted. Moreover, knowing the provider preferences toward the values of the QoS parameters can give us an idea about the distance between the user's offers and those of the provider. This distance represents evidence about how will be difficult the conversation with one provider. If this distance is large, the client will find difficulties to convince the provider. It will be easier to convince each other if they have closer preferences. In the first case, convincing the provider will be a hard task; a negotiation strategy based on strong arguments must be adopted. However, in the second case, an experience or a heuristic based model can sufficiently make the task. By considering one of the latter information about each provider, the best strategy that must be applied with each provider is determined. Choosing the best strategy that fits the provider's profile, avoids for example the adoption of less powerful negotiation model with the more stubborn providers as well as a heavy strategies with the less stubborn ones. In the first case, we are increasing the chance of achieving an agreement. However, in the second case we are reducing the consumption of unnecessary resources.

3.3 The Dependency among the Bilateral Negotiations

In the existing works, dependencies between the different negotiations involved in the whole negotiation process are not considered. In our work, the relation between the concurrent bilateral negotiation processes is ensured through an intermediate party. During each negotiation round, the intermediate party gathers information about offers proposed by each provider from each client's delegate and communicates this information to other client's agent instances. In our work, we assume that there is a competition between providers and also cooperation relation between clients' agent delegates. Each provider's aim is to present a more attractive offer to the user. Each client's agent is able to be informed about other providers' offers through their correspondent clients' agent instances. The client's agent will use this information to induce the provider's agent to make better offers than those of its agent's competitors. For example, if one of the client's agents knows that the other providers'

agents offered more attractive proposals, it will then refuse the offer proposed by the provider's agent with which it is currently dealing. Also, it will induce the provider to make more attractive offers. Consequently the competition between providers will be more intense. This competition will be in the favour of the user and will accelerate the negotiation process progression. Each provider will try to make more interesting offers in order to get the deal. Thus, providers and clients will exchange more and more attractive offers until achieving an agreement. This will speed up the negotiation process and scale down distances between the negotiators' preferences. When the cooperation among clients' instances and the competition among providers are adopted together, clients are luckiest to get at the end the best offer.

4 A CASE STUDY

In this section, we give an example of a buying-selling transaction that illustrates how the negotiation protocols presented by Moschoyiannis et al., (2009), and Venugopal et al., (2008) are applied. We make some modification in these protocols to show benefits of considering dependencies between the bilateral negotiations. In future papers, we will demonstrate through different types of experiments the other contributions.

In this example, we consider that the client's agent aims is to benefit from a service with a low price and high levels of availability, and reliability. However, providers' agents' ambition is to have the opposite.

4.1 the Protocols Description

In order to illustrate our hybrid approach, we use the standard alternating-offers protocol and also the protocol presented by Moschoyiannis et al.. The latter protocol is composed of a set of moves which are respectively, m1:request(negotiation), m2:refuse (negotiation), m3: accept(negotiation), m4: offer(x), m5: accept(x), m6: refuse(x), m7: challenge(x), m8: argue(x), m9: withdraw(x). It specifies also when a particular move is made in the course of a negotiation dialogue by defining the pre-condition and the post-condition of each move. For instance, the move challenge can be uttered if only there is beforehand an offer move. The post-condition of the challenge move is to utter an argue message or a withdraw message. We extend this protocol to make possible the proposal of more than one message at once. For example, for the message m4 we can reply by an m7 and m4 messages, instead of m7 only. Also we enable the utterance of an offer move after an argue move. Also, we make changes in order to adapt the protocol to the negotiation over the QoS parameters rather than it is presented in the context of the resource allocation. In the following, we denote by A1 the first client's delegate agent, and by P1 its correspondent provider's agent.

The standard alternating-offers is a bilateral negotiation protocol. Negotiators exchange only proposals, counter-proposals, accept and reject messages. We denote by A2 the second client's delegate agent, and by P2 its correspondent provider's agent. Since the presented negotiation protocols cover more than one potential alternative scenario, we will be restricted to the more principal negotiation scenario. Furthermore, we only describe informally the proposals, arguments, and messages. In this example, we keep static the availability and the reliability parameters values and we focus only on the price.

4.2 The Main Scenario

Step1
Negotiation 1
A1
m1:Request(negotiation)
Negotiation 2
A2
Offer (50 $, 0.8, 0.9)
Step2
Negotiation 1
P1
m3:Accept(negotiation)
Negotiation 2
P2
Counter-offer (59$, 0.8, 0.9)
Step3
Negotiation 1
A1
m4:Offer(50 $, 0.8, 09)
Negotiation 2
A2
Refuse(59 $, 0.8, 0.9)
Step4
Negotiation 1
P1
m7: challenge (50 $, 0.8, 0.9).
-Why do you proposed as price 50 $, as availability 0.8, and as reliability 0.9?
- What argument you can give to let me believing about this offer?
m4:Offer (58$, 0.8, 0.9)

Negotiation 2
P2
Offer (58 $, 0.8, 0.9)
Step5
Negotiation 1
A1
m8:argue(I need it for a daily use. I would like to have a high availability and reliability level with a reasonable price).
Negotiation 2
A2
Counter-offer(54$, 0.8, 0.9)
Step6
Negotiation 1
P1
-IF the m8 complies with the P1' goals then it will accept the P1' offer. And the process ends with an agreement.
-ELSE, P1 refuses the A1' offer. End of the process without agreement. P1 can give a proposal.
m4: Offer (57 $, 0.8, 0.9)
Negotiation 2
P2
Offer (56 $, 0.8,0.9)
Step7
Negotiation 1
A1
m7: challenge (57 $, 0.8, 0.9)
-Why do you proposed as price 57 $, as availability 0.8, and as reliability 0.9?
Negotiation 2
A2
Accept(56 $, 0.8,0.9)
Step8
Negotiation 1
P1
m8: argue ({If you accept this proposal, you will have a discount}).
Step9
Negotiation 1
A1
m4: Offer (55 $, 0.8, 0.9)
Step10
Negotiation 1
P1
m5:Accept (55$, 0.8, 0.9).

In step1, the agent A1 makes a request move. Simultaneously, the client's agent A2 makes an initial offer. In the second step, P1's agent accepts the A1's request, and the negotiation starts. In the second bilateral negotiation, agent P2 doesn't agree with the A2's offer and makes a counter-offer. In step 3, the A1's agent generates an offer according to its preferences and beliefs. In parallel, A2 refuses the last offer of P2. In step 4, P1 makes a challenge move, and it proposes a first offer. In parallel, P2 proposes a cheaper price than that it offered in the previous step. In step 5, A1 sends to P1 an explanatory argument in which it explains its reason of making such an offer. Indeed, the client's agent wants to have a high availability and reliability levels in addition to a low price since the client will consume daily the service. This explanatory argument helps provider's agent to change its anterior beliefs to be more flexible and closer to the client's preferences. In fact, closer the agents' preferences, the least conflicts are. In the second negotiation, the A2's agent makes a concession and proposes 54 $ as a price. In step 6, the intermediate party's agent gathers information about the last providers' offers and communicates them to the other clients' delegates. Each delegate, during each round must inform its opponent's agent about this information and induce it to make more interesting offers. It also tells its opponent about the offers that were refused by the other client's agents. According to its policies and strategies, each provider's agent decides which offer to propose. For example, in the step6, P2 proposes as a price 56$ which is lower than 58$, proposed by P1 during the step4. It's a time lost for the provider's agent to propose an offer that is less pertinent to the client. By knowing the offers refused by the other clients' instances, the provider's agent can understand better the client's preferences. Consequently, it can propose more interesting offers. For example in the step4, P1 avoids to offer as a price 59 $, since this proposal was refused by the other client's agent in the step3. The consideration of the relation between the two bilateral negotiations makes faster the achievement of an agreement. In step 7, A1 makes a challenge move. In parallel A2 accepts the P2's offer, and the second negotiation process ends. In step 8, P1 makes a reward argument in which it promises A1 a discount if it accepts the offer. According to its goals, A1 can either accept or refuse this argument. If it accepts the argument, then an agreement is achieved, and the client will have a discount. In step 9, A1 doesn't accept the P1's argument, and it proposes 55$ as a price. In step 10, P1 examines the A1's last offer and according to its preferences and beliefs it decides if it will accept, refuse the offer or make a withdraw move. In our case, P1 accepts the A1's offer, and an agreement is achieved. Between the two final offers, the coordinator's agent will choose the one which decrease the price, which is in this case the P1's offer.

5 CONCLUSION

Changes on the Web service environment and on users' requirements have led to new works having as goal the establishment of discovery systems that can respond to users' requests and assure the credibility of results. In this context, several approaches have been applied including the negotiation which leads to significant results related mainly to the improvement of the success rate and the optimization of the search time. In this paper, we have introduced new aspects which are missed in these approaches. The first aspect is related to the consideration of the real life negotiation characteristics during the simulation. These characteristics are the use of different negotiation strategies during interactions, the exchange of explanations between negotiators, and the role of clients and providers in generating offers. The adoption of an hybrid strategy makes the discovery system more realistic and oriented to the largest number of users. The second contribution is to make dynamic the negotiation process. This has the effect of making the negotiation more efficient by choosing for each provider the most suitable negotiation strategy that fits his/her reputation, preferences and context. The third contribution is the consideration of dependencies among the concurrent negotiations. This will impact the quality of the final offer which will be also the result of the competition between providers. Moreover, this will make faster the discovery process since the client will avoid the evaluation of offers that weren't accepted by other clients' instances during the past rounds. Also the providers will avoid proposing offers that were refused by the other clients' instances during the previous rounds. Our future work, will be devoted initially to the presentation of a formal definition of our negotiation approach and secondly for the development and the experimentation of our system.

REFERENCES

Bannock Consulting, 2000. *Quality of Service Parameters for Internet Service Provision*, European Commission, DG Information Society.

Frøkjær E., Hertzum M., and Hornbæk K., 2000. Measuring usability: are effectiveness, efficiency, and satisfaction really correlated, *In Proc. CHI, Washington.*

Chaib-draa B., Jarras I., and Moulin B., Chaib-draa B., 2001. Systèmes multiagents: Principes généraux et applications, *In: Agent et Systèmes Multi-Agents, Hermès, Paris.*

Ran S. P., 2003. A Model for Web Services Discovery with QoS, *SIGecom Exchange*.

Thio N., and Karunasekera S., 2005. Automatic Measurement of a QoS Metric for Web Service Recommendation, *the 2005 ASEC*.

Comuzzi M. and Pernici B., 2005. An Architecture for Flexible Web Service QoS Negotiation, *EDOC 2005*.

Zhou C. and Chen H., 2005. An Objective and Automatic Feedback Model for Qos Evaluation, *ICIS '09 Proceedings of the 2nd ICIS, ACM New York.*

Platzer C., Rosenberg F., and Dustdar S., 2007. Enhancing Web Service Discovery and Monitoring with Quality of Service Information, *In: Securing Web Services: Practical Usage of Standards and Specifications, Idea Publishing Inc*.

Bentahar J., Maamar Z., Wan W., Benslimane D., Thiran P., and Subramanian S., 2008. Agent-based communities of Web services: An argumentation-driven approach, *Service Oriented Comput. Appl., vol. 2, no. 4, pp. 219–238*.

Venugopal S., Chu X., and Buyya R., 2008. A Negotiation Mechanism for Advance Resource Reservation using the Alternate Offers Protocol. *Proceedings of the (IWQoS 2008)*.

Zhou C., and Chen H., 2009. An Objective and Automatic Feedback Model for Qos Evaluation*, ICIS '09 Proceedings of the 2nd ICIS.*

Al-Masri E., and Mahmoud Q. H., 2009. A Broker for Universal Access to Web Services. *2009 Seventh Annual Communications Networks and Services Research Conference.*

Bromuri S., Urovi V., Morge M., Stathis K., and Toni F., 2009. A multi-agent system for service discovery, selection and negotiation. *AAMAS.*

Albreshne A., Fuhrer P., and Pasquier J., 2009. *Web Services Technologies: State of the Art*, University of Fribourg, Switzerland.

Moschoyiannis S., Krause P., Bryant D. and McBurney P., 2009. Verifiable Protocol Design for Agent Argumentation Dialogues *In: 3rd IEEE ICDET, Istanbul, TURKEY.*

Chen H-P., Zhang C., and. Yang G., 2010. A Model for Managing and Discovering ServicesBased on Dynamic Quality of Services. Journal of Networks.

Siala F., and Ghédira K., 2011. A multi-agent selection of web service providers driven by composite qos. *In ISCC.*

Julie D., and Kumar K., 2012. Optimal Web Service Selection Scheme With Dynamic QoS Property Assignment. *IJART, Vol. 2 Issue 2*.

Ben Lakhal R., and Chainbi W., 2012. A Multi-Criteria Approach for Web Service Discovery, *MobiWIS 2012.*

Napoli C. Di, Pisa P., and Rossi S., 2013. A market-based negotiation mechanism for QoS-aware service selection, *The 15th International Workshop on Agent-Mediated Electronic Commerce.*

Kouki J., Chainbi W., and Ghedira K., QoS Synchronization of Web Services: A Mutli Agent-Based Model Based Model, *10th International Symposium on Distributed Computing and Artificial Intelligence.*

Task Placement in a Cloud with Case-based Reasoning

Eric Schulte-Zurhausen and Mirjam Minor
Institute of Informatik, Goethe University, Robert-Mayer-Str.10, Frankfurt am Main, Germany
{*eschulte, minor*}*@informatik.uni-frankfurt.de*

Keywords: Workflow, Cloud Management, Task Placement.

Abstract: Moving workflow management to the cloud raises novel, exciting opportunities for rapid scalability of workflow execution. Instead of running a fixed number of workflow engines on an invariant cluster of physical machines, both physical and virtual resources can be scaled rapidly. Furthermore, the actual state of the resources gained from cloud monitoring tools can be used to schedule workload, migrate workload or conduct split and join operations for workload at run time. However, having so many options for distributing workload forms a computationally complex configuration problem which we call the *task placement problem*.
In this paper, we present a case-based framework addressing the task placement problem by interleaving workflow management and cloud management. In addition to traditional workflow and cloud management operations it provides a set of task internal operations for workload distribution.

1 INTRODUCTION

Rapid scalability is one of the most important characteristics of cloud computing according to the NIST definition (US Department of Commerce, 2011). In classical workflow management, scalability of workflow execution is achieved by traditional load balancing. A load balancer distributes the workload across multiple workflow engines in order to maintain an acceptable performance level of the workflow management system (WFMS) (Jin et al., 2001). Moving workflow management to the cloud raises novel, exciting opportunities for rapid scalability of workflow execution. Instead of running a fixed number of workflow engines on an invariant cluster of physical machines, both physical and virtual resources can be scaled rapidly. Furthermore, the actual state of the resources gained from cloud monitoring tools can be used to schedule workload, migrate workload or conduct split and join operations for workload at run time. However, having so many options for distributing workload forms a computationally complex configuration problem which we call the *task placement problem*. The Workflow Management Coalition ({Workflow Management Coalition}, 1999) defines a workflow as followed: *The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules*.

A task also called activity is defined as: *A description of a piece of work that forms one logical step within a process. An activity may be a manual activity, which does not support computer automation, or a workflow (automated) activity. A workflow activity requires human and/or machine resources(s) to support process execution*.
In this work we use the term task configuration. A task configuration is a set of tasks with their relationships and also a set of parameter. These parameters are not needed for the tasks themselves but for the handling of the tasks through the system.
A cloud configuration is in our work a set of PMs with their resources and a set of VMs assigned to the PMs and sharing some or all of the resources owned by the PM.
We define a task placement as the assignment of automated tasks to virtual machines (VM) and furthermore the assignment of VMs to physical machines (PM). So a task placement is a mapping of a task configuration to a cloud configuration. Our task placement problem includes two kinds of problems: the job placement problem as in (Sharma et al., 2011) and a VM placement problem as in (Jiang et al., 2012). The job placement problem is the problem to assign jobs to VMs and the VM placement problem is the problem to assign VMs to PMs.
In this paper we present a case-based framework addressing the task placement problem by interleaving workflow management and cloud management. In

addition to traditional workflow and cloud management operations it provides a set of task internal operations for workload distribution. The notion of a case is used to represent a task placement problem. A stack of placement operators allows to organize the case representation in layers. Instead of solving the task placement in a computationally complex optimization problem, case-based reasoning (Aamodt and Plaza, 1994) is used to retrieve similar task placements from the past that can be reused in the current situation. In Section 2, we discuss related work from workflow management and cloud management. Section 3 resumes and extends a layered VM placement model from Maurer et al. (Maurer et al., 2013) that we use for a task placement model in Section 4. Section 5 describes a case-based recommender system that suggests a set of promising placement operators based on the experience stored in the case base. Section 6 addresses a planned evaluation. Finally, we discuss future work and draw some conclusions in Section 7.

2 RELATED WORK

Our definition of a task placement is different form the definition of a task placement used in Sharma et al. (Sharma et al., 2011). In their work the tasks are jobs placed on VMs, but the jobs are independent from each other. Workflow tasks are ordered and their execution order is strict. In their work they describe a task placement constraint. Not all tasks or jobs can be placed on all available VMs due to kernel, operating system or other restrictions. The task placement constraint restricts the task placement.

Tang et al. describe in their work (Tang et al., 2007) the problem to allocate applications with dynamically changing demands to VMs and how to decide how many VMs must be started. They called this problem application placement and their problem is similar to Sharma et al. (Sharma et al., 2011).

The problem to assign workflow tasks to VMs is not new. Wu et al. describe in their work (Wu et al., 2013) their approach to solve the task placement problem for scientific workflow with meta-heuristics. They called this Task-to-VM assignment and implement SwinDeW-C, a cloud workflow system. In their work (Maurer et al., 2013) Maurer et al. present an approach to manage a cloud configuration on different layers. They divide the resource allocation into three parts and called them escalation levels.

The first level is for individual applications. The possible actions are:

- Increase / Decrease incoming bandwidth share by x%
- Increase / Decrease outgoing bandwidth share by x%
- Increase / Decrease memory by x%
- Increase / Decrease CPU share by x%
- Add / Remove allocated storage by x%
- Outsource (move application) to other cloud
- Insource (accept application) from other cloud
- Migrate application to different VM

The second level is for VMs. The possible actions are:

- Increase / Decrease incoming bandwidth share by x%.
- Increase / Decrease outgoing bandwidth share by x%.
- Increase / Decrease memory by x%.
- Add / Remove allocated storage by x%.
- Increase / Decrease CPU share by x%.
- Outsource (move VM) to other cloud.
- Insource (accept VM) from other cloud.
- Migrate VM to different PM.

And the third level is for physical machines:

- Add x computing nodes.
- Remove x computing nodes.

Maurer et al. use the escalation levels to guarantee the scalability and efficient resource utilization of their system in order to fulfill the Service Level Agreements. To manage the resources they used two different types of knowledge management - a rule-based and a case-based reasoning approach.

3 MANIPULATION OPERATORS

In our approach we use a system based on the escalation levels introduced by Maurer et al. (Maurer et al., 2013). We extend this approach by so-called manipulation operators (MO) that manipulate the task and cloud configuration that will be introduced in Section 4. Furthermore, we add the workflow level on top of Maurer's escalation levels namely the application, VM and PM levels. The resulting hierarchy of levels is as follows:

- 1: Workflow level
- 2: Application level
- 3: Virtual machine level
- 4: Physical machine level

We denote the three lower levels as the resource level. The resource level includes the application, virtual and physical machine level. The workflow level supports operations to manipulate a workflow so that a better utilization of the resources can be achieved. The following workflow operations also have a hierarchical order:

- 1: Task split / join
- 2: Data split / join
- 3: Migrate task to different VM

The task split operation divides a task into a bunch of parallel tasks with different parameters. For example: I = $\{image_1,..., image_n\}$ is a set of images. Two different, mutually independent types of render algorithms $r_1(I), r_2(I)$ are given. Task $t_1(r_1(I), r_2(I))$ should perform both algorithms and task t_2 evaluates the result. A task split means that we remove t_1 and replace it with two tasks t_{11} and t_{12}, each of which performs only one render algorithm $t_{11}(r_1(I)), t_{12}(r_2(I))$. Because of the split it is now possible to run t_{11} and t_{12} on different VMs and a duration speed up is possible. The task join operation is the inverse action. With this operator the task t_{11} and t_{12} would be joined to t_1. A join can be useful to reduce the number of running VMs or to reduce the number of tasks that run on a single VM.

The data split operation replicates a task and divides the data that is to be processed. Similar to the previous example, I = $\{image_1,...$, $image_n\}$ is a list of images and task $r_1(I)$ is an algorithm to render images. The task t_1 is the task to render all images $t_1(r_1(image_1,...,image_n))$. The data split operation replicates t_1 into a bunch of parallel tasks where each task processes a subset $I\prime \subseteq I$, for example $t_{11}(r_1(images_1,...images_i))$, $t_{12}(r_1(images_{i+1},...,images_n))$. The data join operation is the inverse action similar to the task join operation. It combines t_{11} and t_{12} to t_1.

The migrate operation migrates a task between VMs. We include this operation despite of the fact that the migration of a task can be considered an application migration. The border between the application level and the workflow level is more distinct due to this decision.

4 PLACEMENT MODEL

We model the task placement problem as a mapping problem between a task configuration and a cloud configuration. A task configuration is a tuple $TC = \{T,P,F\}$. T is a set of tasks $\{t_1,t_2,...,t_n\}$. P is a set of tuples $\{p_1,p_2,...,p_n\}$ whose elements p_i describe the parameters for task t_i. The parameters for each task can be the expected execution time on a default VM for each MBit of data volume, the expected upcoming data volume, the system prerequisites, whether a task split is possible or not and if a data split is possible or not. F is a partial order $\prec$ on T specifying a precedence relation on T. F is induced by the order of tasks given in the workflow. In case that t_1 and t_2 are ordered in a sequence within the workflow, t_2 can only be started when t_1 has finished execution and, thus, (t_1,t_2) is taking part in F.

The cloud configuration CC is a tuple {PM, VM, CCP}. PM is a set of physical machines. Each physical machine is represented as a tuple *(*{incoming bandwidth, outgoing bandwidth, CPU, memory, storage, costs per time unit}). This is the representation of the hardware, offered by the physical machine. The costs per time unit is an abstract value which represents the costs, particularly energy, that occur when the machine is running. VM is, similar to PM, a set of virtual machines. Each virtual machine is represented as a tuple of properties *(*{incoming bandwidth, outgoing bandwidth, CPU, memory, storage, speed up}) similar to a physical machine except speed up. The speed up represents a factor that influenced the expected execution time of a task. The expected execution time is determined by a default VM configuration as follows. Let $vm_1, vm_2 \in VM$ and vm_1 is the default VM so the speed up sp_1 of vm_1 is 1. Let the quantity of the available resources for vm_2 be higher than the quantity of the available resources for vm_1. So the speed up of sp_2 is $\leq sp_1$. We consider the run time r_i of a task t_i roughly as the product of the speed up and the expected execution time, so that the run time is $r_i * sp_2 \leq r_i * sp_1$.

The cloud configuration placement CCP is a set of tuples of assignments from VMs to PMs. Each VM $vm_i \in VM$ must be assigned to exactly one $pm_j \in PM$, but a PM can contain more than one VM, for example $\{(vm_1,pm_1),(vm_2,pm_1)\}$.

Now we consider the task placement as a tuple {TC, CC, TP}. TC is the task configuration and CC is the cloud configuration. TP is a set of tuples which represent the assignment of the tasks to a VM and their specified order. TP = $\{tp_1,...,tp_i\}$ with $tp_i = \{vm_i, vmt_i, vmtf_i\}$, $vm_i \in VM, vmt \subseteq T$ denotes the set of tasks that is actually running on vm_i, and $vmtf \subseteq T \times T$ specifies the order constraints on tasks.

5 CASE-BASED TASK PLACEMENT

In case-based reasoning, a problem is solved by finding a similar case stored in the case base and reusing it in the new problem situation (Aamodt and Plaza, 1994). A case is defined as an old problem including a solution for this distinct problem. In our approach a case is a set of two task placements tp_t and tp_{t+1}. The problem task placement tp_t is the placement at time t and the solution task placement tp_{t+1} is a reconfigured version of tp_t at point $t+1$. In addition to tp_{t+1}, the solution includes a list of manipulation operators that is required to transform tp_t into tp_{t+1}. To reuse such a case means to replay the manipulation operators that have been used to transform tp_t into tp_{t+1} for the current situation, that is the current task placement. The similarity of two task placements can be determined by measuring the number of manipulation operations that is required to transform the one into the other. In addition, the expected costs, the expected performance and the expected number of SLA violations can be compared.

Aamodt and Plaza described the case-based reasoning (CBR) process as a cycle with four steps as shown in Figure 1. The four steps are:

- *Retrieve* the most similar case or cases
- *Reuse* the information and knowledge in that case to solve the problem
- *Revise* the proposed solution
- *Retain* the parts of this experience likely to be useful for future problem solving

In the retrieve phase, one or more cases with the highest similarity to the current problem are chosen. The similarity of two task placements depend mainly on the cloud configuration, the SLA requirements and the task configuration. In a first step we assume that the SLA requirements are constant. We further assume that the task configuration in a case base can always lead back to the same workflow template through the workflow manipulation operators introduced in Section 3. But the approach is not limited to that. In (Minor et al., 2007) it is shown how to define the similarity between two workflows. The idea is to represent a workflow as a graph and determine the similarity between two graphs.

With the assumption of a constant set of SLA and the workflow template we can define a neighborhood between task placements. The neighborhood between two task placements is defined by the manipulation operations. Let $TP_1 = \{TC_1, CC_1, TP_1\}$ and $TP_2 = \{TC_1, CC_1, TP_2\}$ be two different task placements with the same task and cloud configuration but

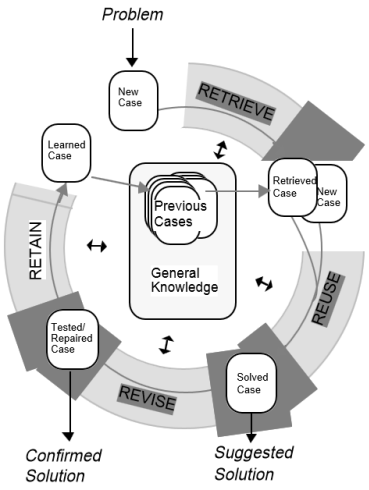


Figure 1: CBR cycle introduced by Aamodt and Plaza.

different task placements and let $\mathrm{Dis}(TP_1, TP_2)$ be a function which determines the distance between two task placements. Be $TP_1 = \{(vm_1, (t_1)), vm_2, (t_2, t_3))\}$ and $TP_2 = \{(vm_1, (t_1, t_2)), vm_2, (t_3))\}$. The distance between them is the number of manipulation operations that is required to convert TP_1 into TP_2. The difference in this example is that the task t_2 in TP_1 is running on VM vm_1 and the task t_2 in TP_2 is running on vm_2. With the operation "'*Migrate application to different VM*"' and the assumption that a task can be considered as an application, $\mathrm{Dis}(TP_1, TP_2) = 1$.

After the retrieve phase follows the reuse phase. In this phase the task placement will be modified by the manipulation operators listed in the solution part of the case. In the revise phase the new case will be evaluated and in the retain phase the new case will be stored in the case base.

6 EVALUATION OF THE CONCEPT

To evaluate the case-based approach we are planning to measure three values: the performance of the workflow execution p, the overall costs oc and the robustness ro for an experimental set of workflows. At this point, the set T includes all tasks of a workflow. That means that one task placement covers the whole workflow. But we consider that when the workflow is divided into parts of currently active tasks and builds a task placement for each of this part, the overall costs will be reduced because the needed infrastructure in terms of VMs and PMs will be very close to the actually required resources. So when the workflow is divided into several task placements we call each place-

ment a Task Placement Segment (TPS). However, the problem with the TPSs is that when there are many TPSs and they are paired very differently, it will take more time to reconfigurate the cloud and thus the performance is reduced.

The robustness in our case is an indicator for measuring the changes within a sequence of TPSs. If the difference between subsequent pairs of TPSs is high, the robustness will be low. To measure the deviation of a TPS from its predecessor, we determine the number n of needed manipulation operations to transform one cloud configuration into another, resulting in $q = \frac{1}{n+1} * 100$. The robustness value qo of the entire sequence aggregates the particular deviation values, for instance by a weighted sum with a logarithmic factor. The performance of the workflow execution depends on the expected execution time r_i of the tasks, the speed up sp_j of the VMs, the reconfiguration time wft_k of the workflow level operators and the reconfiguration time cct_k of the resource level operators of the current TPS tp_k. Each TPS includes one task placement tp_k. The performance is described as: $p = \sum_{k=1}^{n} max_{i \in tp_k}(sp_i * \sum_{j \in vmt_i} r_j) + wft_k + cct_k$. The value of the expected execution time is not just the sum of all expected executions within a TPC, because VMs execute tasks parallel so we search for the maximum makespan of the VMs which is $max_{i \in tp_k}(sp_i * \sum_{j \in vmt_i} r_j)$. In addition, the speed up factor might consider the type of the task. Some tasks could be more memory or network intensive than others which leads to a different speed up for every task depending on the task himself. As a solution, the speed up could be determined for a set of reference task. However, this would require additional effort in classifying workflow tasks.

The overall costs are the sum of the costs of the PMs cpm and the costs of the VMs cvm: $oc = cpm + cvm$. $cvm = \sum_{k=1}^{n} \sum_{i \in tp_k} c_i * \sum_{j \in vmt_i} r_j$. $\sum_{k=1}^{n}$ is the sum over all TPS, $\sum_{i \in tp_k}$ is the sum over all VMs and $c_i * \sum_{j \in vmt_i} r_j$ is the sum over all products of the expected execution r_j and the costs c_i of VM i. The costs for the PMs are similar, in short the costs of the PM pmc multiplied with the highest run time of the VM assigned to the PM. So the formula is: $cpm = \sum_{k=1}^{n} (max_{i \in tp_k}(\sum_{j \in vmt_i} r_j) * pmc)$. We are planning to measure these values to assess the target configuration that is suggested by the retrieval results.

7 SUMMARY AND FUTURE WORK

The task placement problem for workflows in a cloud environment requires an intelligent solution due to its complexity. We think the problem is more difficult than the VM placement problem because it increases the VM placement problem by an additional layer.

In this paper, we present our approach that uses case-based reasoning to find good task placements without a recalculation of the configuration on every single step of the workflow. We belief reasoning techniques are feasible and useful for task placement. Case-based reasoning is a valid method to develop a solution.

The work is still in an early phase of development. It provides a representation and a case-based solution for the task placement problem. In a next step, we will finish the implementation of the prototype and conduct an experimental evaluation. Furthermore, we will deploy a more dynamic approach to determine the changes in speed up of a virtual machine when the manipulation operators change the resources of the VM. Another issue of our future work is to determine the granularity in which the task placement segment should be chosen. To achieve a solution for this, we will implement a configurable prototype in order to conduct further experiments with different setups. The set of values to be measured might be adjusted after first experimental results have been achieved.

We expect the following benefits of the approach. The deep integration of workflow management and cloud management creates novel business opportunities for cloud providers in the area of cloud-based workflow services. Furthermore, the preference on cases with a robust configuration will hopefully reduce the re-configuration costs. Additionally, the number of SLA violations is considered by the approach and, thus, will most probably be reduced. We expect a significantly better performance of the workflow execution service in comparison to simply migrating a traditional workflow management system into a cloud infrastructure as a whole. The novel manipulation operators at workflow level facilitate both scalability at the task and at the data level.

REFERENCES

Aamodt, A. and Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications*, 7(1):39–59.

Jiang, J. W., Lan, T., Ha, S., Chen, M., and Chiang, M. (2012). Joint VM placement and routing for data center traffic engineering. In *INFOCOM, 2012 Proceedings IEEE*, page 28762880.

Jin, L.-j., Casati, F., Sayal, M., and Shan, M.-C. (2001). Load balancing in distributed workflow management system. In *Proceedings of the 2001 ACM symposium on Applied computing*, page 522530.

Maurer, M., Brandic, I., and Sakellariou, R. (2013). Adaptive resource configuration for cloud infrastructure

management. *Future Generation Computer Systems*, 29(2):472–487.

Minor, M., Schmalen, D., Koldehoff, A., and Bergmann, R. (2007). Structural adaptation of workflows supported by a suspension mechanism and by case-based reasoning. In Reddy, S. M., editor, *Proceedings of the 16th IEEE Internazional Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'07), June 18 - 20, 2007, Paris, France*, pages 370–375. IEEE Computer Society, Los Alamitos, California. Best Paper.

Sharma, B., Chudnovsky, V., Hellerstein, J. L., Rifaat, R., and Das, C. R. (2011). Modeling and synthesizing task placement constraints in google compute clusters. In *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, SOCC '11, page 3:13:14, New York, NY, USA. ACM.

Tang, C., Steinder, M., Spreitzer, M., and Pacifici, G. (2007). A scalable application placement controller for enterprise data centers. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, page 331340, New York, NY, USA. ACM.

US Department of Commerce, N. (2011). Final version of NIST cloud computing definition published. Final Version of NIST Cloud Computing Definition Published.

{Workflow Management Coalition} (1999). Workflow management coalition glossary & terminology. last access 05-23-2007.

Wu, Z., Liu, X., Ni, Z., Yuan, D., and Yang, Y. (2013). A market-oriented hierarchical scheduling strategy in cloud workflow systems. *The Journal of Supercomputing*, 63(1):256–293.

The Management System of INTEGRIS
Extending the Smart Grid to the Web of Energy

Joan Navarro[1], Andreu Sancho-Asensio[1], Agustín Zaballos[1], Virginia Jiménez-Ruano[1], David Vernet[1] and José Enrique Armendáriz-Iñigo[2]
[1]*La Salle – Ramon Llull University, 08022 Barcelona, Spain*
[2]*Universidad Pública de Navarra, 31006 Pamplona, Spain*
{*jnavarro, andreus, zaballos, vjimenez, dave*}*@salleurl.edu, enrique.armendariz@unavarra.es*

Keywords: Web-based Services, Intelligent Agents, Distributed Databases, Security, Smart Grids.

Abstract: The recent growth experimented by the Internet has fostered the interaction of many heterogeneous technologies under a common environment (i.e., the Internet of Things). Smart Grids entail a sound example of such situation where several devices from different vendors, running different protocols and policies, are integrated in order to reach a common goal: bring together energy delivery and smart services. Latest advances on this domain have led to effective architectures that support this idea from a technical perspective, but fail at providing powerful tools to assist this new business model. Hence, the purpose of this paper is to present a novel unified and ubiquitous management interface, driven by an intelligent system, that uses the advantages featured by the Web of Things to manage the Smart Grid. Therefore, this work opens a new path between the Internet of Things and the Web of Things resulting in a new concept coined as the Web of Energy.

1 INTRODUCTION

The ever growing set of features provided by the Internet has led into a standard communications framework that eases the deployment and development of distributed and heterogeneous applications. Such evolution has driven a rising interest in connecting several gadgets to the Internet, ranging from mobile phones to home appliances, including special systems such as traffic lights or embedded devices. This has led to a new form of distributed system, referred to as the Internet of Things (IoT), that consists in (1) uniquely identifying every object in the network, (2) using the Internet as the communications infrastructure, and (3) providing a lightweight interface to rapidly access everywhere.

Recent advances on Smart Grids have explored the feasibility of considering the electricity distribution network as a particular case of the IoT (Guinard et al., 2011; Zaballos et al., 2011). Certainly, this specific domain poses appealing challenges in terms of integration, since several distinct smart devices (also referred to as Intelligent Electronic Devices or IEDs) from different vendors—often using proprietary protocols and running at different layers—must interact to effectively deliver energy and provide a set of enhanced services and features (also referred to as smart functions) to both consumers and producers (prosumers) such as network self-healing, real-time consumption monitoring, and asset management (INTEGRIS, 2011). Although the latest advances on the IoT field have definitely contributed to the physical connection of such an overwhelming number of smart devices, several issues have arisen when attempting to provide a common management and monitoring interface for the whole Smart Grid (INTEGRIS, 2011; Aman et al., 2013).

Indeed, integrating the heterogeneous data generated by every device on the Smart Grid (e.g., wired and wireless sensors, smart meters, distributed generators, dispersed loads, synchrophasors, windmills, solar panels, communication network devices) into a single interface has emerged as a hot research topic. So far, some experimental proposals (Guinard et al., 2011) have been presented to face this issue by using the Web of Things (WoT) concept to access a mashup of smart devices and directly retrieve their information using reasonably thin protocols (e.g., HTTP, SOAP) (Guinard et al., 2010).

However, the specific application of these approaches to real-world environments is fairly doubtful because (1) they may open new security breaches (Zeng et al., 2011; Bou-Harb et al., 2013) (i.e., end-users could gain access to critical equipment), (2)

there are no mature electric devices implementing *WoT-compliant* standards available in the market (INTEGRIS, 2011), and (3) industry is averse to include foreign modules (i.e., web servers) on their historically tested and established—but poorly evolved—proprietary systems (Gungor et al., 2013).

Therefore, the European project INTelligent Electrical GRId Sensor Communications (INTEGRIS) (INTEGRIS, 2011) has explored a new way to overcome these issues and, thus, provide a management interface for the Smart Grid inspired by the WoT. More specifically, the aim of INTEGRIS is twofold: on the one hand, it implements an ICT infrastructure—based on the IoT paradigm—to handle the Smart Grid security, storage, and communications requirements (Navarro et al., 2012). On the other hand, it uses a cognitive-inspired intelligent multi-agent system to manage the whole Smart Grid and link it with end-users using a WoT-based approach, which results in a new bridge between the IoT and WoT.

The purpose of this paper is to present this multi-agent intelligent system that extends the WoT approach and implements the management system used in the INTEGRIS' real-world domain. This proposal, which leads in a new form of the WoT coined as the Web of Energy (WoE), is targeted to provide a secure, context-aware, and uniform web-based novel environment to effectively manage, monitor, and configure the whole Smart Grid. Moreover, conducted developments proof the feasibility and reliability of our approach and encourage practitioners to further research towards this direction.

The remainder of this paper is organized as follows. Section 2 reviews the related work and justifies the proposal. Then, Section 3 details the reference architecture for the Smart Grid and Section 4 introduces how it is integrated into the Web of Things. Finally, Section 5 concludes the paper.

2 RELATED WORK

Service composition, heterogeneous devices interaction, and a close contact with the real-life demands are some of the features that have positioned Smart Grids as an appealing landscape for deploying the latest advances concerning the IoT and WoT. Over the last years, practitioners have directed their efforts on enabling communications between distinct types of devices spread across the different network facilities that compose the Smart Grid (Zaballos et al., 2011). Certainly, the IoT and the WoT approaches have promoted such advances in the sense that IEDs are no longer considered as isolated entities but referred to as gears of a complex and heavily coupled distributed system (Guinard et al., 2011; Guinard et al., 2010). While the IoT has provided a mature approach to enable hardware communication on the Smart Grid (Zaballos et al., 2011; INTEGRIS, 2011), management, monitoring, and grid configuration requirements envisage the need of using either centralized Network Management Systems (NMSs) (Gungor et al., 2013) or WoT-based approaches (Guinard et al., 2011; Priyantha et al., 2008).

Although centralized NMSs are rapid to deploy and easy to maintain, their lack of scalability, single point of failure exposure, and bottleneck effect vulnerability make them unfeasible for real-world applications (Aman et al., 2013). These issues are driving system designers to explore distributed solutions for the Smart Grid domain such as the Representational State Transfer (REST) and Web Services (Guinard et al., 2010). On the one hand, RESTful strategies (Cubo et al., 2012; Luckenbach et al., 2005) are targeted to ease the development of scalable services and applications by (1) uniformly identifying every smart object through a unique Uniform Resource Identifier (URI), (2) using HTTP stateless communications, (3) representing resources through standard human readable formats (e.g., XML), and (4) using simple parsing algorithms—typically embedded on tiny web servers—to interact with every IED (Guinard et al., 2011; Duquennoy et al., 2009). On the other hand, strategies based on Web Services pursue the same goal but implement some advanced facilities (e.g., UDDI server, WSDL, security) rather than using light weight protocols (Guinard et al., 2010; Priyantha et al., 2008).

Despite the aforementioned benefits featured by Web Services and RESTful frameworks, the established electric industry is still far from widely adopting them as a standard solution for the Smart Grid domain (INTEGRIS, 2011). In addition, both solutions may incur in some critical security issues that are not affordable in this domain (Zeng et al., 2011; Bou-Harb et al., 2013). Therefore, in the INTEGRIS project we have split the Smart Grid into two isolated layers: a lower layer that encompasses IEDs' communications following the IoT schema—described in (Navarro et al., 2012)—, and an upper layer—herein presented—that covers prosumers interactions with the grid following the WoT approach. To successfully bridge both layers and meet the stringent security and scalability constraints demanded by the Smart Grid (Aman et al., 2013; Navarro et al., 2012), we have implemented a multi-agent intelligent system that (1) is aware of all heterogeneities of the lower

layer (Zaballos et al., 2011), (2) predicts future situations (Gama, 2010), (3) builds a comprehensive system model (Navarro et al., 2012), and (4) delivers it to a uniform interface for user interaction using a RESTful approach. The following section reviews the key parameters to be extracted from the lower layer and presents the internals of this cognitive-inspired intelligent subsystem.

3 A REFERENCE MODEL FOR SMART GRIDS

Actually, splitting the ICT system that holds the ambitious requirements of Smart Grids (Aman et al., 2013) into the two aforesaid layers enables (1) providing an extra security barrier by hiding electric-devices and grid topology from unauthorized end-users, (2) building scalable mechanisms to collect data from IEDs, (3) addressing the electric functions (e.g., voltage monitoring) separately from user-driven functions (e.g., power flow management) and communication issues (i.e., overlay networks (Zaballos et al., 2011)), (4) integrating heterogeneous protocols and technologies, and (5) providing a uniform management interface (INTEGRIS, 2011). The purpose of this section is to present the architecture that supports this idea and announce the key parameters used to link both layers.

From our real-world experiences collected during the INTEGRIS project, we have found that dividing the Smart Grid into these logical layers poses some critical difficulties arisen from the fact that typically, IEDs are closed devices that do not allow implementing custom developments (e.g., security or information-exchange protocols)—as novel experimental devices do (Guinard et al., 2011). Therefore, we proposed a new device coined as I-Dev (Navarro et al., 2012) that behaves as a frontier between these two layers and implements (1) a communications subsystem that allows heterogeneous network coexistence, (2) a security subsystem that provides a reliable and secure low layer communications infrastructure, (3) a distributed storage subsystem that smartly stores all data generated by IEDs, and (4) an intelligent cognitive-inspired subsystem that is aware of all events arisen from any subsystem of the network.

In order to avoid the bottleneck effect and thus deploying a scalable ICT infrastructure, two distinct roles have been assigned to I-Devs: the Perception Action Agent (PAA) and the Domain Management Agent (DMA). Therefore, the whole Smart Grid is spread into several regions (e.g., low voltage substations) referred to as I-Domains (Navarro et al., 2012), where in each I-Domain a DMA and several PAAs interact with deployed IEDs using the IoT approach (Zaballos et al., 2011).

As depicted in Fig. 1, PAAs—placed in the bottom layer—(1) gather information from their associated IEDs, (2) report their findings to their assigned DMA, and (3) execute the commands ordered by it as a result of a collaborative decision between all DMAs in the Smart Grid. Hence, I-Devs behaving as DMAs bridge the bottom layer and the top layer while ensuring scalability and a safe isolation between both sides. Likewise, the upper layer is committed to interface with the management side of all the smart functions provided by the Smart Grid (e.g., supervisory control and data acquisition (SCADA), advanced distributed automation (ADA), distributed energy resource (DER), automatic meter reading (AMR), advanced metering infrastructure (AMI), NMS, or quality of service) taking benefit from the ubiquitous features provided by the WoT approach.

As the lower layer has been broadly addressed in (Navarro et al., 2012; Zaballos et al., 2011), the remainder of this section is devoted to (1) summarize the functionalities of every subsystem running in any I-Dev and (2) specify the key monitoring and configuration parameters to be delivered to the herein proposed upper layer.

3.1 Distributed Storage Subsystem

Smart Grids intrinsically generate vast amounts of heterogeneous information (i.e., every IED may have its own proprietary data format) that need to be effectively stored in order to be later processed by the aforementioned smart functions. In this context, data are generated at different points of the grid and need to be reliably replicated in order to afford site failures and boost their availability through a distributed storage system. Typically, classical relational databases are unable to handle neither the dynamic nature, nor the scalability requirements, nor the stringent computing and storage capabilities of smart devices (Aman et al., 2013). Hence, we developed our own distributed storage architecture running a custom replication protocol that smartly placed data to optimize the performance of smart functions being executed at the I-Devs layer. This replication protocol is aimed to provide a consistent view of any datum—stale but consistent versions are useful for non critical functions such as average consumption monitoring—at any situation. To achieve this commitment data is replicated and partitioned following a hierarchical hybrid approach between state-machine and synchronous primary copy replication that attempts to

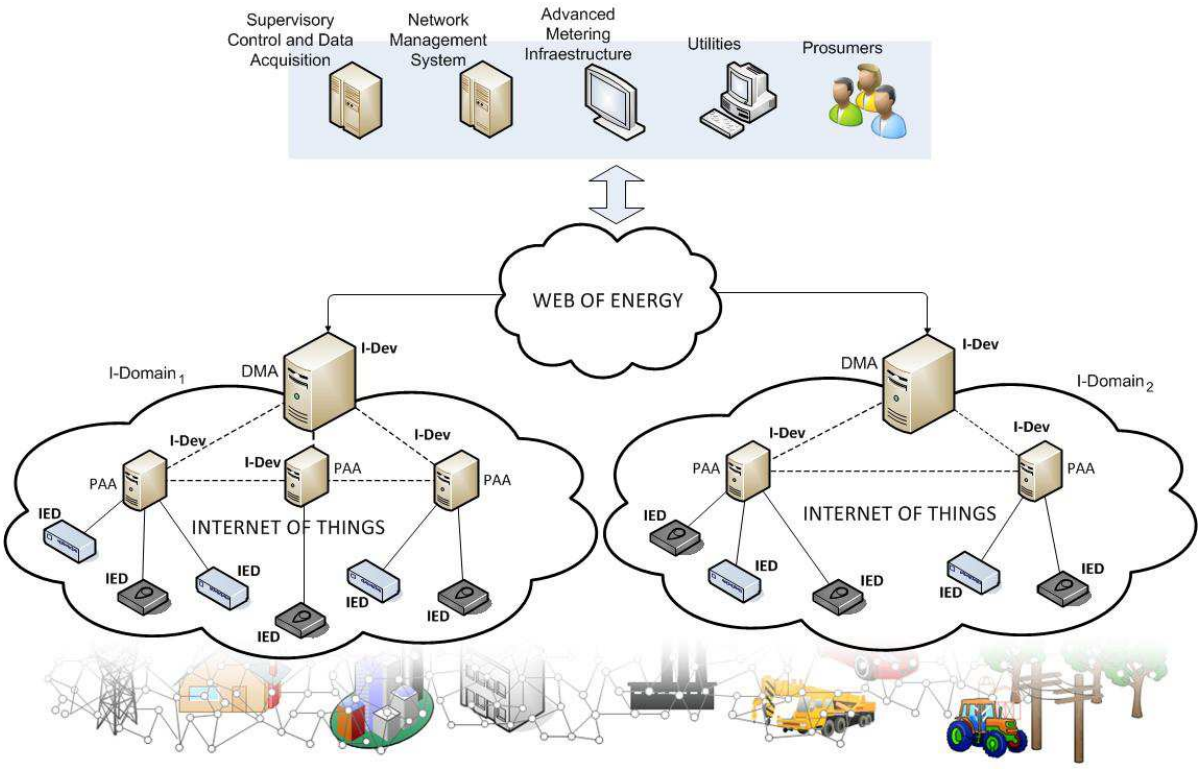


Figure 1: Deployed reference architecture for Smart Grids.

keep system scalability by epidemically updating data across I-Devs (Navarro et al., 2012). This strategy converts the storage infrastructure composed by I-Devs as a dynamic set of nested onion layers that store eventually consistent versions of data.

Manually configuring and monitoring this distributed system is unfeasible because (1) there is an overwhelming number of devices involved on the replication process, (2) there are several interdependencies with other subsystems, and (3) the system conditions may change abruptly. Therefore, we propose to deliver the following key performance metrics and configuration parameters to an intelligent system who is aware of the status of the whole Smart Grid, in order to get a reliable set of actions to be performed on the storage domain. Specifically, we need to track the amount of read/write operations, response time, replication depth, required consistency degree, and replication hierarchy layout (Navarro et al., 2012). The following subsection describes the communications and security subsystem and details which metrics are delivered to the intelligent system.

3.2 Communications and Security Subsystem

ICTs, trust management, and technological integration play an essential role when coordinating all IEDs to solve every smart function. However, they have to be carefully addressed since the electrical distribution infrastructure encompasses aerial and underground areas that current communications technologies are unable to reach (Zaballos et al., 2011). Current approaches in this domain rely on the Internet network to boost their performance but inherit the same threats and critical risks in terms of cyber-security (Bou-Harb et al., 2013). Therefore, communications and security in Smart Grids are crucial for the survival and feasibility of the global electricity distribution concept (Navarro et al., 2012). So far, very few standards concerning security (e.g., IEC62351, NISTIR7268) have been proposed to address the specific issues posed by Smart Grids, which still leaves the network and its links open to cyber-security attacks that may produce Denial of Service (DoS) and eavesdropping of critical network management messages. Cyber attacks in the communications network are aimed to bring the maximum damage, exploit the greatest benefits and take advantage of the network structure or from protocols vulnerabilities for malicious purposes (e.g., extended power outages and destruction of power equipment (Bou-Harb et al., 2013)). Latest advances on network resilience protocols (Chen et al., 2012) implemented on I-Devs permit them to sustain from temporal node

failures or disconnections as long as countermeasures can be activated.

However, these countermeasures may prevent other subsystems to behave properly (e.g., the distributed storage system may be unable to collect data from a given PAA) and thus drive the whole Smart Grid to a panic situation. Therefore, our proposal sends the following key communications and security metrics of every I-Dev to an intelligent system that owns a global view of the Smart Grid: connectivity—the number of links to the Smart Grid control center—, security performance—number of packet errors, number of encryption errors, number of unsuccessful access attempts, number of I-Dev reboots, number of network overload situations and other DoS attacks, and number of packet retransmissions—, quality of service performance—amount of flow rates, discarding probabilities and queue lengths—, and delay performance—average link delays. Then, the intelligent subsystem examines these parameters and selects the best security and communications action (e.g., isolating an I-Dev) in order to achieve the best Smart Grid overall performance.

3.3 Intelligent Subsystem

Although designing and modeling every subsystem of the Smart Grid separately eases the development process, we have observed that it is unfeasible to allow them running on their own (i.e., without considering the whole grid status) (Navarro et al., 2012). Considering the vast amount of data and distinct events continuously arisen from the grid, the management of such a system cannot fully rely on an expert. Therefore, we have chosen an intelligent multi-agent system to interprete the aforesaid key parameters of every subsystem in the grid in order to build a comprehensive model of the whole system. Specifically, this context requires an online learning scheme able to (1) put together the parameters of each subsystem, (2) rapidly adapt to the constant changes of the Smart Grid, (3) provide an accurate estimation concerning the whole grid status, and (4) come up with the best configuration at every subsystem to reach the optimal overall performance. To this regard, we have selected an enhanced version of the eXtended Classifier System (XCS) (INTEGRIS, 2011) that diligently meets these requirements as shown in what follows.

In fact, XCS, a cognitive-inspired algorithm, is targeted to evolve a population [P] of classifiers. At the end of the learning process, the population is expected to acquire a high quality model using these classifiers. Each one consists of a production rule—composed by an antecedent and a consequent—and

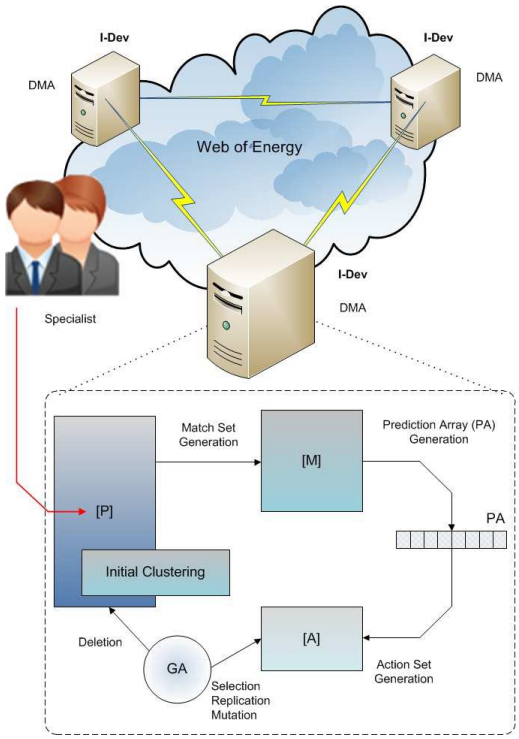


Figure 2: Intelligent subsystem learning architecture. DMAs use the Web of Energy to share their experiences collected at every I-Domain and build a global Smart Grid knowledge model.

a set of parameters that evaluate the quality of the rule. The antecedent part of the rule contains the input variables collected by PAAs from every subsystem and related to the environment (i.e., what the algorithm "senses"). Likewise, the consequent part contains the action predicted for the future status of the Smart Grid. The main parameters that evaluate the quality of every rule are (1) an estimation of the reward that will be received if the predicted status is triggered, (2) the expected error, (3) the experience of the classifier, and (4) the fitness of the classifier.

Thoroughly, XCS follows the online learning process depicted in Fig. 2 at the DMA: it starts with an empty population and it learns by sampling new training examples. This learning process creates the match set [M] containing all the classifiers that match with the current example (INTEGRIS, 2011). If [M] does not contain a minimum user-defined threshold of different actions, the system generates new ones arbitrarily. Then, the action to be proposed to the actuators (i.e., PAAs) is selected via a fitness-weighted average of all matching classifiers in [M], forming the action set [A]. All classifiers in [A] predict the same action and hence they share the evaluation payoff in a niching scheme. At the end of the iteration, a genetic algorithm is applied at [A] to discover new promising

rules, for instance ***if*** *data_response_time* $\in [10,100]$ *ms* ***and*** *... **and** reconnections* $\in [40,60]$ ***then*** *block I-Dev3*.

As the XCS is a reinforcement learning approach, training and scalability need to be properly considered: First, as it is unfeasible to manually label all examples of the training set, we have used an on-line clustering approach based on k-Nearest Neighbors (Navarro et al., 2012) to label them and thus let the system autonomously learn from the environment. Last, in order to meet the intrinsic scalability requirements of the Smart Grid, a unique centralized thinking unit is avoided by deploying a DMA and its associated PAAs on every I-Domian.

However, the distributed nature of this cognitive subsystem entails further concerns on how integrating all the knowledge to an accurate set of human-readable rules, which enables to think globally and act locally. The following section is devoted to detail how these rules are delivered across the upper layer of the herein presented WoT approach.

4 FROM THE IoT TO THE WoE

In fact, distributing the knowledge building layer of the cognitive subsystem (i.e., DMAs) to meet the scalability constraints posed by the Smart Grid definitively hampers the learning process. As shown in Fig. 2, refusing a centralized architecture forces locally collected rules at every I-Domain to be shared among all DMAs of the Smart Grid—the underlying hypothesis in rule sharing is that similar structures (i.e., I-Domains) require similar configurations (i.e., knowledge model)—, which may lead to some conflicting situations arisen from the fact that the cognitive subsystem is unable to reach 100% of accuracy (Navarro et al., 2012). Note that rules shared between I-Domains are those with a classification accuracy greater than a user-defined minimum threshold. Although the dynamics of this on-line learning architecture allow removing these conflicting rules without shutting down or reseting the affected DMA, a reliable decision process to conduct this action is mandatory.

Therefore, as depicted in Fig. 2, we have introduced a new role on the Smart Grid: the expert system; which is the combination of the aforesaid software entities and a specialist that owns enough experience related to the electricity domain. To this regard, this specialist is continuously analyzing the results provided by the intelligent system and thus, learning from the Smart Grid in order to supplement software suggestions by deciding which rules must be deleted from every DMA (also referred to as conflict resolution). Likewise, this expert can introduce new knowledge based on his expertise to the cognitive subsystem by forcing the usage of new rules. We have found (INTEGRIS, 2011; Navarro et al., 2012) that this expert system greatly enhances the performance of the intelligent system because it ensures that critical decisions are taken consistently.

However, the feasibility of this approach relies on a uniform interface that fits with the intrinsic distributed nature of the Smart Grid and enables its effective management. So far, existing preliminary solutions (INTEGRIS, 2011) achieve such commitment by using centralized management systems (e.g., SCADA) that are unable to fully integrate the long-term requirements and applications posed by the Smart Grid domain (Gungor et al., 2013). Hence, we have deployed a WoT-inspired infrastructure—coined as WoE—on top of the Smart Grid that links all I-Domains and permits a bidirectional communication between electricity domain (i.e., bottom layer in Fig. 1) and the application domain (i.e., top layer in Fig. 1).

To this concern, every I-Dev in the Smart Grid has been labelled with an URI, which enables specialists and all entities residing at the top layer of Fig. 1 using asynchronous communications to (1) incorporate new knowledge or erase existing rules that enter in conflict with previously discovered ones, (2) reset and manipulate the configuration parameters that control every subsystem—especially the cognitive one—, and (3) obtain accurate statistics from every DMA, and thus from the whole Smart Grid as depicted in Fig. 3.

More specifically, Fig. 3 depicts some of the web interfaces that integrate the heterogeneous framework found on the bottom layer of the Smart Grid—and linked using a IoT approach—to a uniform environment. The top left screenshot in Fig. 3 shows the configuration interface of the distributed subsystem (i.e., replication depth, cache size, number of layers (Navarro et al., 2012)) running at I-Domain 6. The top right screenshot in Fig. 3 shows the configuration of the communications and security subsystem running at I-Domain 3. Finally, the bottom screenshots in Fig. 3 are devoted to depict the two layers of the cognitive system.

The bottom left screenshot in Fig. 3 depicts the management interface of the intelligent subsystem running at I-Domain 9; first, it shows a list of three PAAs, below it depicts the predicted I-Domain status and the selected action in [P]. Additionally, the bottom right screenshot in Fig. 3 depicts the collected data at PAA 2 from I-Domain 7. In this way, specialists can monitor the (1) effects of applying a given ac-

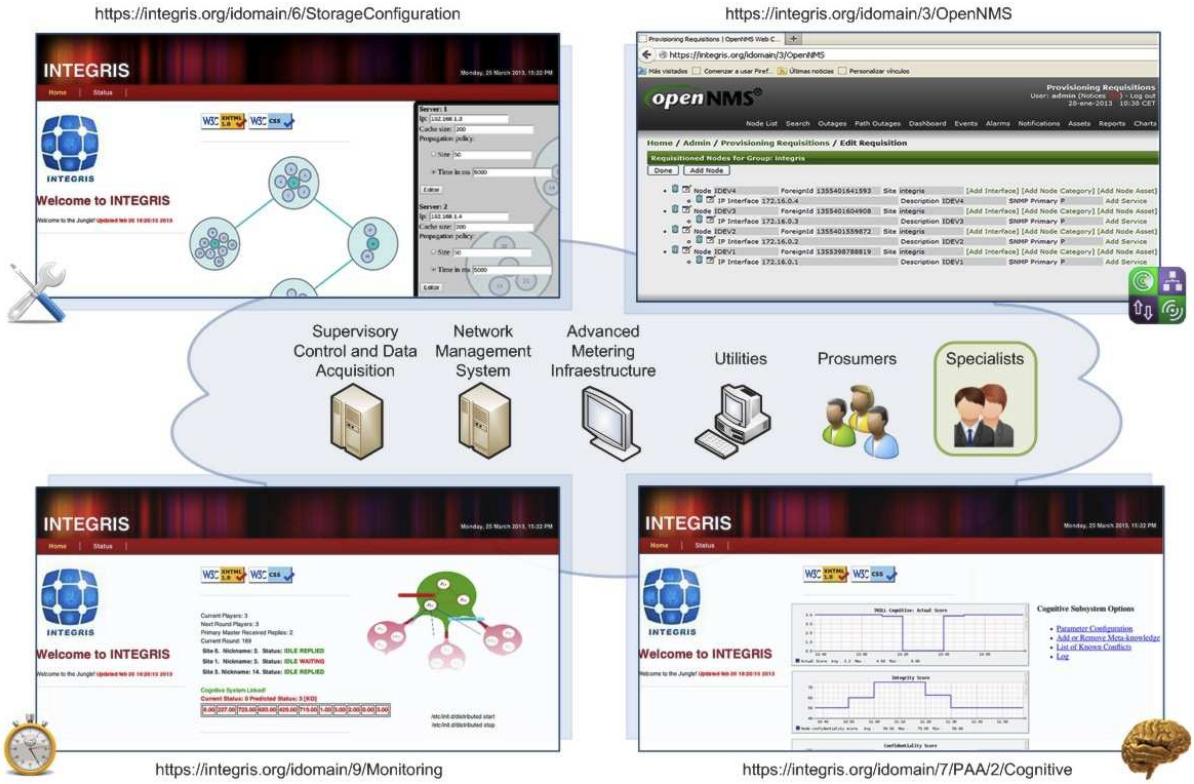


Figure 3: The Web of Energy. A uniform interface to support smart functions and manage the associated distributed, communications, security, and intelligent subsystems.

tion, (2) set of locally and globally collected rules, (3) parameters perceived at every PAA, and (4) training accuracy at a glance.

Thoroughly, the herein presented approach provides a uniform web-based interface referred to as WoE that safely isolates the electricity grid domain from its applications. More specifically, this system includes an ad-hoc distributed storage architecture to support the massive amount of data generated by the grid, that delivers these data through a secured communication network to a multi-agent—able to face the intrinsic dynamic nature of Smart Grids—intelligent subsystem that processes the events and changes arisen in this domain.

5 CONCLUSION

This paper presents a particular application of the WoT to the concrete scenario of power networks. We have conducted our experimentation over the system presented in (Navarro et al., 2012), where an IoT-based infrastructure enabled machine-to-machine interactions between small and resource-constrained devices on the Smart Grid domain. Thus, we have extended the IoT concept by providing a bidirectional human-to-machine interface—inspired by the WoT—that results in a ubiquitous energy control and management system coined as Web of Energy. This proposal combines the web-based visualization and tracking tools with the Internet protocols, which enables a uniform access to all devices of the Smart Grid.

In order to provide such an effective and reliable management interface aimed to address the heterogeneous nature of devices residing on the grid, we have deployed an intelligent subsystem devoted to (1) learn from the real-world events, (2) predict future situations, and (3) assist on the decision making process. This intelligence layer is composed by means of a multi agent system to meet the scalability requirements of the Smart Grid. Moreover, it builds a knowledge model in terms of production rules, implements its own apportionment of credit mechanism (i.e., uses reinforcement learning), and has an ad-hoc rule discovery technique based on a genetic algorithm. However, we have seen that the intelligent system is unable to cope with the complexities of the Smart Grid that hamper the optimal learning performance (i.e., 100% accuracy). Therefore, we have successfully

included the role of the specialist in the presented WoE approach. This entity is targeted to dynamically rectify the intelligent subsystem outcomes and improve the global grid performance. Note that the aforesaid WoE framework eases the specialists commitment considerably, in the sense that they are able to interact with the different modules that control the Smart Grid through a pervasive and user friendly web-based interface rather than traditional roughly command lines.

Finally, we have demonstrated the feasibility of our proposal by running it on the real-world scenario defined by the INTEGRIS project (INTEGRIS, 2011). Our collected experiences show that this uniform management interface—depicted in Fig. 3—plays a key role in the process of development and standardization of current and new smart functions. Certainly, this paper encourages practitioners to conduct future work in this direction by defining new electric applications following this layered scheme.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Union European Atomic Energy Community Seventh Framework Programme (FP7/2007-2013 FP7/2007-2011) under grant agreement 247938 for Joan Navarro, Agustín Zaballos by Generalitat de Catalunya for its support under grant 2013FI_B2 00089 for Andreu Sancho-Asensio, and by the Spanish National Science Foundation (MEC) (grant TIN2012-37719-C03-03) for José Enrique Armendáriz-Iñigo.

REFERENCES

Aman, S., Simmhan, Y., and Prasanna, V. K. (2013). Energy management systems: State of the art and emerging trends. *IEEE Communications Magazine*, 51(1):114–119.

Bou-Harb, E., Fachkha, C., Pourzandi, M., Debbabi, M., and Assi, C. (2013). Communication security for Smart Grid distribution networks. *IEEE Communications Magazine*, 51(1):42–49.

Chen, P.-Y., Cheng, S.-M., and Chen, K.-C. (2012). Smart attacks in Smart Grid communication networks. *IEEE Communications Magazine*, 50(8):24–29.

Cubo, J., Brogi, A., and Pimentel, E. (2012). Towards behaviour-aware compositions of things in the Future Internet. In *Proceedings of the 2nd International Workshop on Adaptive Services for the Future Internet and 6th International Workshop on Web APIs and Service Mashups*, WAS4FI-Mashups '12, pages 28–35, New York, NY, USA. ACM.

Duquennoy, S., Grimaud, G., and Vandewalle, J.-J. (2009). The Web of Things: Interconnecting devices with high usability and performance. In *International Conference on Embedded Software and Systems, ICESS '09, Hangzhou, Zhejiang, P. R. China, May 25-27, 2009*, pages 323–330, Hangzhou, Zhejiang, P. R. China. IEEE.

Gama, J., editor (2010). *Knowledge Discovery from Data Streams*. Advances in Database Systems. Chapman and Hall/CRC, Florida, USA, first edition.

Guinard, D., Trifa, V., Karnouskos, S., Spiess, P., and Savio, D. (2010). Interacting with the SOA-Based Internet of Things: Discovery, query, selection, and on-demand provisioning of Web Services. *Services Computing, IEEE Transactions on,*, 3(3):223–235.

Guinard, D., Trifa, V., Mattern, F., and Wilde, E. (2011). From the Internet of Things to the Web of Things: Resource-oriented architecture and best practices. In *Architecting the Internet of Things*, pages 97–129. Springer Berlin Heidelberg.

Gungor, V. C., Sahin, D., Kocak, T., Ergüt, S., Buccella, C., Cecati, C., and Hancke, G. P. (2013). A survey on smart grid potential applications and communication requirements. *IEEE Trans. Industrial Informatics*, 9(1):28–42.

INTEGRIS (2011). INTEGRIS FP7 Project INTelligent Electrical Grid Sensor communications. ICT-Energy-2009 call (number 247938).

Luckenbach, T., Gober, P., Arbanowski, S., Kotsopoulos, A., and Kim, K. (2005). TinyREST-A protocol for integrating sensor networks into the Internet. In *in Proc. of REALWSN*.

Navarro, J., Zaballos, A., Sancho-Asensio, A., Ravera, G., and Armendariz-Iñigo, J. E. (2012). The information system of INTEGRIS: INTelligent Electrical GRId sensor communications. *Industrial Informatics, IEEE Transactions on*, PP(99):1.

Priyantha, N. B., Kansal, A., Goraczko, M., and Zhao, F. (2008). Tiny Web Services: design and implementation of interoperable and evolvable sensor networks. In *Proceedings of the 6th International Conference on Embedded Networked Sensor Systems, SenSys 2008, Raleigh, NC, USA, November 5-7, 2008*, pages 253–266, St. Louis, Missouri, USA. ACM.

Zaballos, A., Vallejo, A., and Selga, J. M. (2011). Heterogeneous communication architecture for the Smart Grid. *IEEE Network*, 25(5):30–37.

Zeng, D., Guo, S., and Cheng, Z. (2011). The Web of Things: A survey (invited paper). *JCM*, 6(6):424–438.

CLOUD COMPUTING PLATFORMS AND APPLICATIONS

FULL PAPERS

Monitoring Large Cloud-Based Systems

Mauro Andreolini[1], Marcello Pietri[2], Stefania Tosi[2] and Andrea Balboni[2]
[1]*Department of Physics, Computer Science and Mathematics, University of Modena and Reggio Emilia, Via Campi 213/a, 41125, Modena, Italy*
[2]*Department of Engineering "Enzo Ferrari", University of Modena and Reggio Emilia, Via Vignolese 905/b, 41125, Modena, Italy*
mauro.andreolini@unimore.it, {marcello.pietri,stefania.tosi, andrea.balboni}@unimore.it

Keywords: Monitoring Architecture, Cloud Computing, Large-scale, Scalability, Multi-tenancy.

Abstract: Large scale cloud-based services are built upon a multitude of hardware and software resources, disseminated in one or multiple data centers. Controlling and managing these resources requires the integration of several pieces of software that may yield a representative view of the data center status. Today's both closed and open-source monitoring solutions fail in different ways, including the lack of scalability, scarce representativity of global state conditions, inability in guaranteeing persistence in service delivery, and the impossibility of monitoring multi-tenant applications. In this paper, we present a novel monitoring architecture that addresses the aforementioned issues. It integrates a hierarchical scheme to monitor the resources in a cluster with a distributed hash table (DHT) to broadcast system state information among different monitors. This architecture strives to obtain high scalability, effectiveness and resilience, as well as the possibility of monitoring services spanning across different clusters or even different data centers of the cloud provider. We evaluate the scalability of the proposed architecture through a bottleneck analysis achieved by experimental results.

1 INTRODUCTION

Cloud Computing is the most adopted model to support the processing of large data volumes using clusters of commodity computers. According to Gartner, Cloud Computing is expected to grow 19% in 2012, becoming a $109 billion industry compared to a $91 billion market last year. By 2016, it is expected to be a $207 billion industry. This esteem compares to the 3% growth expected in the overall global IT market. Several companies such as Google (Dean and Lopes, 2004), Microsoft (Calder et al., 2011), and Yahoo (Shvachko et al., 2010) process tens of petabytes of data per day coming from large data centers hosting several thousands nodes. According to (Gantz and Reinsel, 2012), from 2005 to 2020, the digital universe will grow by a factor of 300, from 130 EB to 40000 EB, or 40 trillion GB (more than 5200 GB per person in 2020). From now until 2020, the digital universe will about double every two years.

In order to satisfy service level agreements (SLAs) and to keep a consistent state of the workflows in this tangled layout, such growing large infrastructures are usually monitored through a multitude of services that extract and store measurements regarding the performance and the utilization of specific hardware and software resources. These monitoring tools are operated by cloud providers and offered to the services' owners, but also ad-hoc monitoring solutions are designed in order to satisfy the requirements of big companies which own their private cloud infrastructures. For example, Sony uses the closed-source Zyrion Traverse database (Zyrion, 2013) to claim the monitoring of over 6000 devices and applications over twelve data centers across Asia, Europe and North America. The virtual data layer within the solution collects half a million resource data streams every five minutes.

This scenario requires the design of an advanced monitoring infrastructure that satisfies several properties:

1. **Scalability.** It must cope with a large amount of data that must be collected, analyzed, stored and transmitted at real-time, so as to take timely corrective actions to meet SLAs.
2. **Effectiveness.** It must provide an effective view of the system state conditions that can be used for management purposes and to identify the causes of observed phenomena. It must also adapt its monitoring functions to varying conditions in order to accommodate variable resources, system errors, and changing requirements.

3. **Resilience.** It must withstand a number of component failures while continuing to operate normally, thus ensuring service continuity. Single points of failure must be avoided for providing persistence of service delivery.
4. **Multi-tenancy.** It must be able to monitor applications distributed over different data centers in order to better perform troubleshooting activities in dynamic environments such as cloud scenarios.

We state that none of the existing solutions fulfills all these requirements. In this paper we overcome state-of-the-art limits with a novel open-source monitoring infrastructure. We propose a hybrid architecture for a quasi real-time monitoring of large-scale, geographically distributed network infrastructures spread across multiple data centers, designed to provide high scalability, effectiveness and resilience. Here, the term *hybrid* refers to the use of two different communication schemes: a *hierarchical* one and a *P2P-based* one. Each data center is equipped with its own decoupled monitoring infrastructure; each monitor adopts a hierarchical scheme that ensure scalability with respect to the number of monitored resources, in a subset of the whole architecture. Communications between data centers are performed through the root managers, software modules responsible for orchestrating the whole process. The root managers of every decentralized monitor are connected through a custom communication module that implements the P2P Pastry DHT routing overlay (Rowstron and Druschel, 2001). In this way, a service distributed across several data centers can be jointly monitored through the appropriate root managers.The internal operations of the monitor are geared towards effectiveness objectives. We provide real-time access to single performance samples or graphs, as well as more sophisticated analysis that aim at identifying system or application states for anomaly detection, capacity planning, or other management studies. Every single component in the infrastructure is designed to be resilient to failures. Whenever possible, we enrich the existing software modules with redundancy and failover mechanisms. Otherwise, we automatically restart the modules in case of failure.

The rest of this paper is organized as follows. Section 2 evaluates the current state-of-the-art in the area of large-scale system monitoring. Section 3 describes the design decisions supporting the described requirements, provides a high level architecture of the entire monitoring infrastructure, motivates the choice of the software components and discusses various implementation details. Section 4 investigates the theoretical scalability limits of the proposed architecture figured out from experimental scenarios. Finally, Section 5 concludes the paper with some remarks and future work.

2 RELATED WORK

Current state-of-the-art monitoring tools do not guarantee scalability, effectiveness, resilience and multi-tenancy objectives.

Fully centralized monitors cannot scale to the desired number of resource data streams. For example, the prototype system introduced in (Litvinova et al., 2010), which uses Ganglia (Massie et al., 2004) and Syslog-NG to accumulate data into a central MySQL database, shows severe scalability limits at only 64 monitored nodes, each one collecting 20 resource data streams every 30 seconds. Here, the main bottleneck is related to the increasing computational overhead occurring at high sampling frequencies. On the other hand, lowering the sampling frequency (commonly, once every five minutes) can make it difficult to spot rapidly changing workloads which in turn may entail the violation of SLAs (Keller and Ludwig, 2003).

Concerning resilience, the vast majority of both open-source and commercial monitoring infrastructures like OpenNMS (Surhone et al., 2011), Zabbix (Olups, 2010), Zenoss (Badger, 2008) and Cacti (Kundu and Lavlu, 2009) are not adequate or designed to address failures, especially if combined with the ability to gather and support millions of resource data streams per second.

In terms of effectiveness, most open-source monitoring tools only partially address this aspect. For example, Graphite (Davis, 2013) and Cacti provide only trending analyses, Nagios (Josephsen, 2007) provides alerting, while Chukwa (Rabkin and Katz, 2010) and Flume (Hoffman and Souza, 2013) are designed exclusively to collect resource data streams or logs. Also current decentralized, per-data-center, hierarchical monitors such as Ganglia (Sacerdoti et al., 2003) are limited to efficiently compute averages of measures spanning over several nodes. However, the complexity of current workloads in modern data centers calls for more sophisticated processing, such as the identification of correlations among different resource data streams, or the detection of anomalies in the global system state.

Astrolabe (Renesse et al., 2003) is a hybrid solution that combines a hierarchical scheme with an unstructured P2P routing protocol for distributed communications as our proposal does. While it is resilient and highly scalable in terms of data collection and storage, it lacks in effectiveness and its manageability is a complex task since it incurs a lot of network traf-

fic. Unstructured systems do not put any constraints on placement of data items on peers and how peers maintain their network connections and this solution suffers from non-deterministic results, high network communication overload and non-scalability of bandwidth consumption (Lv et al., 2002).

While collection and network monitoring were addressed in many works with significant results (Babu et al., 2001; Cranor et al., 2003; Voicu et al., 2009), the state-of-the-art technology in multi-tenant monitoring is a very niche field. In fact, none of the previous works deals with a multi-tenant environment. At the best of our knowledge, the only open contribution in this sense is given by (Hasselmeyer. and d'Heureuse, 2010): it extends monitoring based on data stream management systems (DSMS) with the ability to handle multiple tenants and arbitrary data; however it does not address resilience in terms of single points of failure, it has no implemented prototype, and it does not present any type of analysis to support the proposed architectural choices.

3 ARCHITECTURE DESIGN

The early decisions that inspired the design of the proposed architecture share four important goals: (1) to dominate the complexity of the monitoring problem (*Scalability*), (2) to tune the monitoring activities according to different objectives (*Effectiveness*), (3) to avoid single points of failure (*Resilience*), and (4) to monitor services spanning across different clusters or data centers (*Multi-tenancy*). This section details the architecture design of our proposal, with particular emphasis to the design decisions that allow the achievement of the mentioned goals.

3.1 High Level Architecture

Figure 1 and Figure 2 present the high level architecture of the monitoring infrastructure.

We propose a hybrid architecture using a hierarchical communication scheme to ensure scalability and a P2P-based communication scheme to allow multi-tenancy. In our opinion, a hybrid solution is the only viable alternative for scaling to an arbitrary number of data centers and the huge problem size makes it literally impossible to deploy any kind of centralized infrastructure. Even worse, service centralization would not be fault-tolerant. For these reasons, each cluster in our architecture is equipped with an independent monitoring infrastructure.

In order to scale to millions of data streams per sample interval, it is mandatory to shift preliminary computations (such as the sampling of a resource and the performing of sanity checks on sampled data) as close as possible to the edge of the monitored infrastructure. Failure to do so would result in a system that unnecessarily processes potentially useless data. For this reason, collected resource data streams are initially filtered (or marked as invalid, anomalous) on the monitored nodes where a *collection agent* receives the samples from several probe processes. Probe processes are responsible for collecting periodically performance and/or utilization samples regarding a set of hardware and software resources. The collection agent performs preliminary validity checks on them, that are executed through dynamic, pluggable modules that receive in input the data stream and respond with TRUE or FALSE. If at least one check fails, the stream is tagged as invalid, but it is never discarded; this facilitates later debugging operations. The following checks are implemented now: missing value, value out of range, sequence of null values.

Then, the collection agent updates the resource data streams and sends them to a set of associated collector nodes. We consider both the sending of uncoded (without compression) and coded (lossless compression) data. A detailed description of the collection agent has been presented by the authors in (Andreolini et al., 2012).

The collector node is the main component of the *distributed cluster data filter*. It receives the checked and coded resource data streams, performs the necessary decoding, applies low cost analyses on decoded data, and stores their results for a real-time plot or further analysis. In the former case, processing stops and the user is able to see immediately the behavior of the resource data streams. In the latter case, data is made available to the *distributed analyzer system*. Its purpose is to compute more sophisticated analyses on the resource data streams, such as aggregation of information coming from different clusters, identification of correlated components in the system, anomaly detection and capacity planning. The data streams resulting from these analyses are persistently stored in the *distributed data storage*. Here, data is available as (key, value) pairs, where "key" is a unique identifier of a measure and "value" is usually a tuple of values describing it (e.g., timestamp, host name, service/process, name of the monitored performance index, actual value).

The information regarding the data center asset is stored in a *distributed configuration database*. In this way, we strive to avoid possible inconsistencies mainly due to a service being migrated or receiving more resources. The monitoring infrastructure associates data streams to the identifiers of the corre-

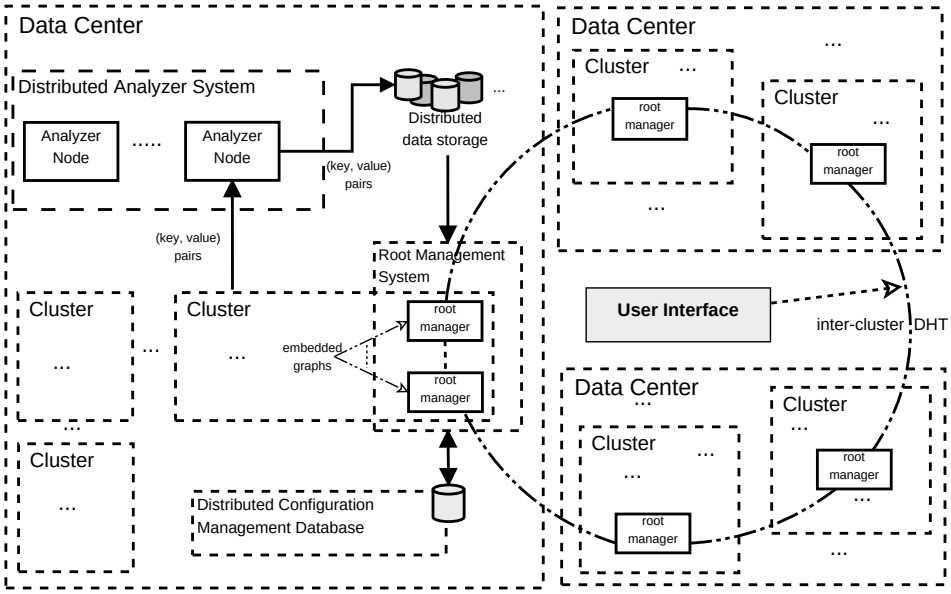


Figure 1: Monitoring system architecture overview.

sponding monitored resource.

Each monitoring infrastructure is orchestrated by a *root management system*, a software component that organizes the workflow of monitoring operations and provides a programmable monitoring interface to the user. All the root managers dislocated on different data centers are interconnected by an efficient DHT overlay routing network. In this first version of our prototype, the other main task carried out by a root manager is to forward early notifications of anomalies in the internal state of some resources to other interested, subscribed root managers. In this way, it is possible to anticipate the performance degradation of services depending on these failing resources.

We used exclusively open-source tools that can be modified and adapted for our goals. We used GNU/Linux Debian, Ubuntu and Fedora OSs in different experimental testbeds, enhanced with the software packages from the Cloudera repository (CDH4). The languages used for the deployment of our modules are Bash (v4.2.36), Python (v2.7), Java (v1.6), JavaScript and C (where efficiency is needed, such as in our modified monitor probes). The batch processing framework is Hadoop, version 2.0. Our choice is motivated by the dramatic scalability improvement with respect to traditional RDBMS-based data storage architectures under random, write-intensive data access patterns (Leu et al., 2010). To avoid single points of failure and to ensure service continuity, we enforce redundancy of every component of the monitoring architecture. Whenever possible, we deploy our solution using software that can be easily replicated. In other cases, we wrap the component through custom scripts that detect failures and restart it, in case.

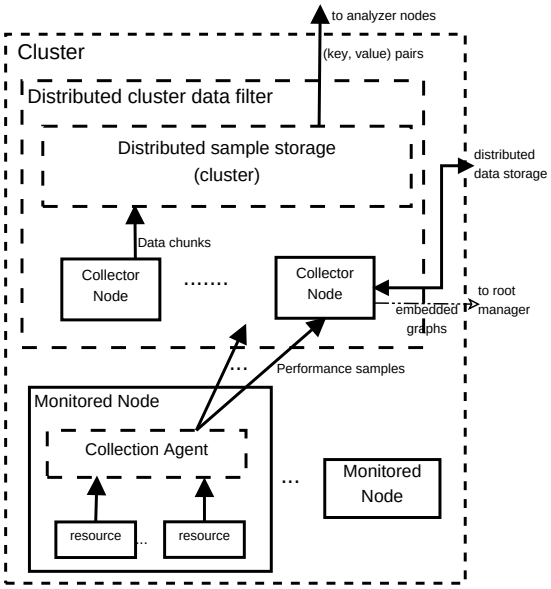


Figure 2: Cluster architecture.

3.2 The Distributed Cluster Data Filter

The resource data streams gathered by the collection agent are sent to the distributed cluster data filter, shown in Figure 3. Here, a collector process receives the checked and coded resource data streams.

The received data streams are decoded and later

analyzed to extract a representation of the cluster state, thus guaranteeing the effectiveness of our monitoring infrastructure. Resource data streams coming from different nodes are processed through sophisticated analyses that aim to guarantee high accuracy and significantly reduced human intervention. In order to support real-time analytics at large scale, at this level we adopt analytic approaches having linear computational complexity and adaptive implementation. Linear solutions permit to understand system behavior in real-time, so as to diagnose eventual problems and take timely corrective actions to meet service level objectives. Adaptivity allows analytic approaches to accommodate variable, heterogeneous data collected across the multiple levels of abstraction present in complex data center systems. Example analyses we implemented at this stage include:

1. computing moving averages of resource data streams, in order to provide a more stable representation of a node status;
2. aggregating (both temporally and spatially) node state representations to obtain global views of the cluster state conditions;
3. extracting trends for short-term prediction of resource consumption and of cluster state conditions;
4. detecting state changes and/or anomalies occurring in data streams for the erase of alarms and the adoption of recovering strategies;
5. correlating node state representations in order to identify dependencies among different nodes in the cluster and to exclude secondary flows.

Nodes and cluster state representations are then sent to two different storages: one for real-time plotting of the decoded and analyzed resource data streams, and one for non-real-time later processing at highest levels.

The former storage for real-time plotting is handled by OpenTSDB (Sigoure, 2010), a software for the storage and configurable plotting of time series. We have chosen OpenTSDB because it is open-source, scalable, and it interacts with another open-source distributed database, HBase (George, 2011). It retains time series for a configurable amount of time (defaults to forever), and it allows to create custom graphs on the fly. We have modified few parts of OpenTSDB (shown in Figure 4) in order to plot a simple but real-time prediction of resources trend. This analysis is performed using a linear regression and a Gaussian kernel.

The latter storage for non-real-time processing, called *data synk*, receives data destined to further

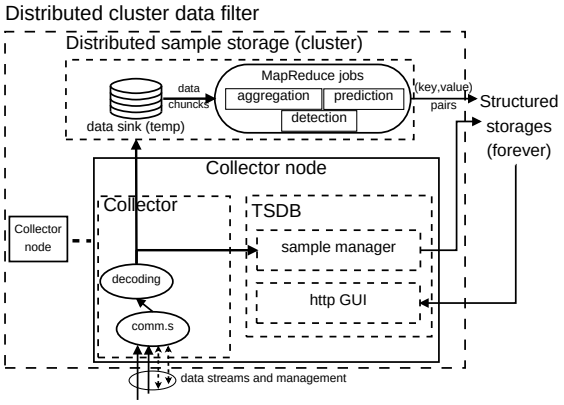


Figure 3: Distributed cluster data filter.

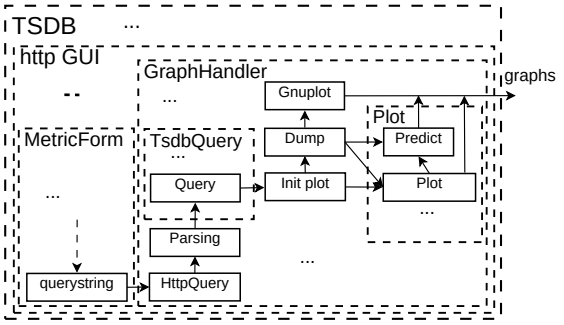


Figure 4: OpenTSDB schema improvements for prediction.

processing performed by the distributed analyzer described in the following subsection. This solution reduces the number of files generated from one per node per unit time to a handful per cluster (Andreolini et al., 2011). To enhance the performance of the storage engine, we chose to pack the resource data streams (few bytes per each) in larger chunks (64KB by default) and to write them asynchronously to a distributed file system that can be scaled to the appropriate size by easily adding back-end nodes. In order to provide a homogeneous software layer (eg., Hbase coupling) and an open-source platform, and in order to support a map-reduce paradigm, the best possible choice is the Hadoop Distributed File System (HDFS). It allows extremely scalable computations, it is designed to run on commodity hardware, it is highly fault-tolerant, it provides high throughput access to application data, and it is suitable for applications that have large data sets.

Every time new samples are added to the resource data streams and representations, an extra overhead is paid due to data storage. As previous literature shows, in this scenario characterized by frequent, small, random database requests (Andreolini et al., 2011), write operations to secondary storage do suffer from scalability issues. To reduce this overhead, write oper-

ations should be grouped and batched to secondary storage. We believe that the map-reduce paradigm (Dean and Lopes, 2004) is well suited to this purpose. If needs be, several collectors can be added to scale the acquisition process to the desired number of resource data streams. The collector is designed to scale up to thousands of data streams, providing that limitations on the maximum number of TCP connections and open files can be raised. In GNU/Linux, this can be easily achieved by changing some system parameter and/or by recompiling the Linux kernel and the GNU C library.

3.3 The Distributed Analyzer System

The distributed analyzer system is composed by a set of analyzer nodes (Figure 5). Each analyzer node runs arbitrary batch jobs that analyze the state representation data streams of nodes and clusters. At this stage, we admit the implementation of more computational expensive analyses with respect to those applied at the cluster level. Now, analyses are applied only to small sets of representative information (i.e., nodes and cluster state representations) from which we require to obtain relevant information for management with high levels of accuracy.

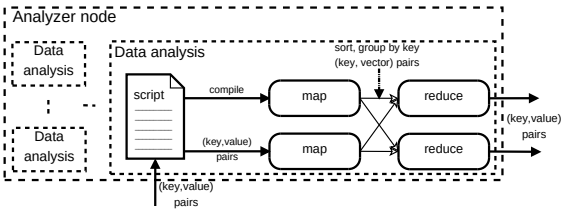


Figure 5: Analyzer node.

For example, analyses implemented at data center level are:

1. aggregation of cluster state representations to obtain global views of the data center state conditions;
2. long-term prediction of clusters and data center state conditions computed at different temporal scales and with different prediction horizons;
3. detection of changes and anomalous events in data center state conditions with the identification of which node(s) in the different clusters is the culprit.

The batch jobs first read the necessary data streams (*map*) from the distributed cluster data filter storages, and then run the appropriate scripts (*reduce*) for analysis. The adoption of map-reduce allows to perform sophisticated analysis over the collected resource data streams in a scalable fashion with commodity hardware (or even in a leased platform such as Amazon EC2). On the contrary, the most advanced state-of-the-art monitors compute at most moving averages of regular windows of past samples. To the best of our knowledge, this paper represents one first step beyond this limit in a quasi real-time scenario.

The result of the map-reduce paradigm is a reduced set of (key, value) pairs that is written to the distributed data storage. The goal shared by these operations is to compute a reduced state information of each cluster, and to produce few figures of merit that show the health status of the entire data center. Through these results, it is possible to tell whether the service is about to misbehave or not and, in the former case, also to tell which resource(s) and which node(s) are the culprits.

We choose the Pig framework for the implementation of the analysis scripts (Olston et al., 2008). Pig offers richer data structures over pure map-reduce, for example multivalued and nested dictionaries. Each Pig script is compiled into a series of equivalent map-reduce scripts that process the input data and write the results in a parallel way. Our scripts implement the analyses mentioned above. Further analyses can be easily supported by our architecture and implemented to satisfy more sophisticated requests.

3.4 The Distributed Data Storage

Both the reduced streams representing the system state and the resource data streams processed by OpenTSDB must be written into a data storage. For the sake of performance, it is possible to avoid the reuse of the same structured storage. As matter of facts, the data storage:

- must scale with an increasing number of data streams;
- must be fault tolerant;
- should be designed towards the data management.

In this context, we choose Apache HBase (George, 2011) also because of the fact that it includes the homogeneity and the reuse of components. In our architecture, the HBase storage is responsible to preserve all the analyzed information about nodes, clusters and data center. Apache HBase is a distributed column-oriented database built on top of HDFS, designed from the ground-up to scale linearly just by adding nodes. It is not relational and it does not support SQL, but thanks to the proper space management properties, it is able to surpass a traditional RDBMS-based system by hosting very large and sparsely pop-

ulated tables on clusters implemented on commodity hardware.

3.5 The Distributed Configuration Database

In the proposed architecture, a configuration database is needed to store all information related to the asset of a cluster. Asset-related information includes a description of the resource metadata (name, id), placement (IP of the hosting node or virtual machine), sampling period, and a description of the time interval during which the resource is supposed to be assigned to a service. We think that it is a good idea to use off-the-shelf Configuration Management DataBase Systems (CMDBs). A CMDB is a repository of information related to all the components of an information system, and contains the details of the configuration items in the IT infrastructure. However, the majority of CMDBs is not natively fault tolerant. We address this shortcoming by replicating both its Web front-end and DB back-end.

The configuration management database of our choice is OneCMDB. It is an open-source CMDB for data centers that can store configurations such as hardware, software, services, customers, incidents, problems, RFCs and documents. OneCMDB conforms to IT Management best practice declared by the Information Technology Infrastructure Library. It adopts a client-server paradigm and it is used in large production environments with thousands of configuration items. An enhanced graphical user interface enables more effective system operations. Scaling and fault tolerance are not included in the majority of CMDBs, including OneCMDB. To address these shortcomings, we replicate both its front-end and back-end. The front-end is replicated through highly scalable Linux Virtual Server (LVS). The architecture of the server cluster is fully transparent to root managers, and the root managers interact as if it was a single high-performance virtual server. The back-end is replicated through an off-the-shelf, load balanced, master-slave MySQL setup (v. 5.1).

3.6 The Root Management System

As shown in Figure 6, the root management system is composed of three distinct areas: orchestration, communication and failover.

The orchestration module is the heart of the monitoring system since it orchestrates the operations of the other aforementioned components (collector, data filter, analyzer). One of its main tasks is to trigger and

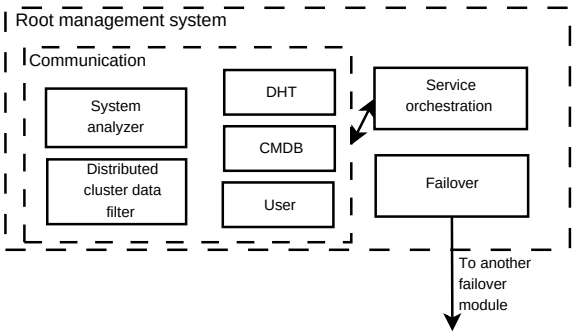


Figure 6: The root management system.

to abort the execution of batch jobs in the distributed cluster data filter and in the analyzer nodes.

The communication module is a simple messaging system used to interact with the other components of the monitoring architecture in order to communicate relevant information (such as anomalies in some resource state) to other monitoring systems dislocated in different data centers. The root manager node also receives commands from the user interface; these commands are forwarded to and processed by the orchestration module. The user interface is basically a Web-based application running on any selected node. It manages the resources owned by an application and provides a programmable dashboard with figures of merit, diagrams and configuration parameters (monitored nodes, resources, performance indexes, sampling intervals). Each cluster and each monitored process is represented using embedded OpenTSDB graphs, while the system view is represented using a similar but customized interface that supports also long-term predictions, aggregation analysis, detection and capacity planning.

The failover module ensures fault tolerance by identifying which root managers are compromised and by restoring a safe state. To this purpose, each root manager runs part of the replica of the other root managers in the same data center. If a root manager fails, the replica becomes the master until the former one is restored.

When a service is installed on the nodes, the collection and analysis processes supply this information to the root management system, which stores it into the distributed configuration database. At each application deployment, a list of the involved nodes is defined. A unique key is associated to this list; both the key and the list are shared through the DHT with each root management system. The root management system responsible for the largest number of involved nodes selects its best root manager on the basis of multiple configurable metrics. Finally, the selected root manager becomes the service leader.

The root managers are connected as previously shown in Figure 1. Each data center is composed by a set of root manager nodes connected through a Pastry-based Distributed Hash Table (DHT). We chose Pastry (Rowstron and Druschel, 2001) because it is a generic, scalable and efficient substrate for P2P applications that forms a decentralized, self-organizing and fault-tolerant overlay network. Pastry provides efficient request routing, deterministic object location, and load balancing in an application-independent manner. Furthermore, it provides mechanisms that support and facilitate application-specific object replication, caching, and fault recovery. The DHT communication module implements all the needed overlay routing functions.

The root management system is built upon a set of custom Python and Java modules. The DHT is implemented through the freepastry libraries. The publish-subscribe mechanism used to broadcast alerts to the interested root managers is implemented through Scribe (Castro et al., 2002). We previously discussed these aspects from a security point-of-view in (Marchetti et al., 2010).

We implemented the user interface using the Django MVC framework and the JQuery library to enhance the presentation of data. The responsiveness of the application is improved through the adoption of AJAX-based techniques and the Web server Apache v.2.2.

4 ANALYSIS

We perform experimental analyses for evaluating the ability of the proposed monitoring architecture in satisfying all requirements of scalability, effectiveness, resiliency and multi-tenancy. Due to the limited space, in this section we only report analysis results about the scalability of our solution. We evaluate the scalability of the proposed architecture in terms of number of monitored resource data streams. In particular, we aim to find out:

- how many resource data streams can be monitored per node (intra-node scalability);
- how many nodes can be monitored in a cluster (intra-cluster scalability).

Highest level scalability (intra-data center scalability) is left for future extensions and strongly depends on both resource behaviors and aggregation results obtained through analytics computed in the distributed analyzer system. In this paper, we used the Amazon EC2 IaaS platform. In the considered infrastructure, the backing storage is shared across the instances (EBS), and the theoretical network connectivity is up to 1Gbps. The virtual machines are running instances of the TPC-W and RUBiS benchmark suites. MapReduce jobs queries are used for data distribution and analysis. We perform Map-Reduce versions of several performance analyses having different computational costs, including the moving average and the Principal Component Analysis (PCA) over more than 1 hour of data collected from 2048 monitored nodes. We emphasize that the results are strongly influenced by the resource consumption of the TSDB component, and the tuning of this trade-off is out of the scope of this paper. However, we measure that the PCA (i.e., the most computational expensive analysis) requires an average of 5 minutes when computed over 8 collector nodes using around the 85% of CPU (the 12.5% was used for collector process). This result shows that the behavior of a single cluster during the collection of over more than 6M of resource data streams per second can be analyzed (in batches) within quasi real-time constraints.

In each monitored node, one probe is dedicated to system-related performance monitoring through the output of the vmstat and sar monitors. The remaining probes are process-related through pidstat and nethogs2 monitors. This system probe collects 25 different performance indexes, while each process probe collects 23 different resource data streams. The sampling interval is configured at 1 second for each probe in order to emulate the most challenging scenario.

4.1 Intra-node Scalability

In the first experimental testbed, we evaluate how many resource data streams can be handled for each monitored node. We use one collector node and one analyzer node running a single script that computes the moving average for every resource data stream. The detail of the resources of the monitored node is the following: micro instance, 613 MB memory, up to 2 EC2 Compute Units (Dual-Core AMD Opteron(tm) Processor 2218 HE, cpu 2.6 GHz, cache size 1,024 KB), EBS storage, dedicated network bandwidth of theoretically 100 Mbps per node.

Table 1 reports the average resource consumption (percentage of CPU, memory (RAM) and network (NET) utilization) of the collection agent as a function of the number of monitored resource data streams. We performed tests on both uncoded (*without compression*) and coded (*lossless compression*) data in order to evaluate the impact of compression on the scalability of the different resources. Then, we evaluate how the use of the *Adaptive algorithm* that we proposed in (Andreolini et al., 2012) improves the scalability

Table 1: Average resource utilization of the collection agent.

#probes	#resource data streams	Without compression			Lossless compression			Adaptive algorithm		
		CPU (%)	RAM (%)	NET (%)	CPU (%)	RAM (%)	NET (%)	CPU (%)	RAM (%)	NET (%)
1	25	0.0	0.4	0.005	0.3	0.4	0.002	0.1	0.5	0.001
2	48	0.1	0.5	0.009	0.5	0.5	0.004	0.1	0.5	0.002
4	94	0.1	0.6	0.019	1.1	0.6	0.009	0.2	0.7	0.004
8	186	0.1	1.0	0.041	1.8	0.9	0.019	0.3	1.0	0.008
16	370	0.3	1.4	0.085	2.9	1.4	0.041	0.4	1.4	0.016
32	738	0.5	2.5	0.173	4.1	2.6	0.083	0.6	2.7	0.032
64	1474	0.6	4.7	0.352	6.0	4.8	0.162	0.8	4.6	0.069
128	2946	0.9	9.4	0.681	9.8	9.3	0.337	1.2	9.5	0.127
256	5890	2.5	18.7	1.392	23.1	18.3	0.641	3.1	18.8	0.266

Table 2: Average resource utilization of the collector in the distributed cluster data filter.

#monitored nodes	#resource data streams	Without compression		Lossless compression		Adaptive algorithm	
		CPU (%)	NET (%)	CPU (%)	NET (%)	CPU (%)	NET (%)
1	2946	0.1	0.971	0.6	0.450	0.1	0.189
2	5892	0.1	1.943	0.9	0.899	0.1	0.355
4	11784	0.2	3.838	2.0	1.797	0.2	0.748
8	23568	0.4	7.763	3.6	3.594	0.4	1.463
16	47136	0.9	15.421	8.1	7.186	0.9	3.001
32	94272	1.9	31.055	17.1	14.374	1.9	5.872
64	188544	3.2	61.980	33.6	28.751	3.2	11.711
128	377088	-	-	69.9	57.539	6.1	23.404
256	754176	-	-	-	-	12.5	47.096
512	1508352	-	-	-	-	23.7	93.691

of our architecture. The Adaptive algorithm is able to adapt the frequency of sampling and data updating to minimize computational and communication costs, while guaranteeing high accuracy of monitored information. From these tests, we see that at intra-node level, sending data streams has a negligible impact on the network bandwidth, despite the fact that it is reduced of about 50% by using lossless compression and more than 80% by using the Adaptive algorithm. We see also that the most used resource without data compression or with Adaptive algorithm is the memory, while with lossless compression the most used resource is the CPU. At 128 probes, both the CPU and memory utilizations are less than 10%. This threshold is commonly used as the largest fraction of resource utilization that administrators are comfortable devoting to monitoring. We have adopted this threshold as our target maximum resource utilization for the monitoring system. Hence, on each monitored node, we can collect up to 128 probes for a total of 2,946 resource data streams per second. We recall that a period of one second is much shorter than commonly adopted sampling periods that typically do not go below one minute.

4.2 Intra-cluster Scalability

In the following set of experiments, we consider nodes within a cluster, monitored with the same probe setup. We measure the resource consumption of the collector at cluster level with or without compression efforts and with the Adaptive algorithm.

Table 2 reports the average resource consumption of the collector node as a function of the number of monitored nodes. From this table, we see that without compression the most used resource is the network that allows the monitoring of at most 64 nodes (or 188,544 resource data streams) in a cluster. On the contrary, compressing data strongly impacts the CPU utilization. Despite that, the compression of data allows to monitor more than 128 nodes or $2,946 \cdot 128 = 377,088$ resource data streams per second. By using the Adaptive algorithm we are able to monitor up to 512 nodes per collector, meaning 1.5M resource data streams per second.

As further result, we add collector nodes and increment the number of monitored hosts to evaluate the scalability of the distributed cluster data filter. Table 3 reports the average resource utilization across the collector nodes.

We keep adding collectors up to 2,048 monitored nodes. We also add more HDFS and HBASE nodes

to support the write throughput when the number of nodes becomes higher than 256. We keep 256 as limit in the number of nodes since overcoming the 50% of incoming network bandwidth of the collector node means overcoming the 100% of outcoming bandwidth as can be inferred by Figure 3. In this scenario, by using the Adaptive algorithm we are able to monitor about 6M resource data streams by using an average 12.5% of CPU and 47.3% of network bandwidth.

Table 3: Average resource utilization of a collector process over the distributed cluster data filter.

#monitored nodes	#resource data streams	collector		
		#nodes	CPU (%)	NET (%)
256	754176	1	12.5	47.096
512	1508352	2	12.8	48.327
1024	3016704	4	12.2	46.851
2048	6033408	8	12.4	46.908

This analysis on scalability reveals that the proposed architecture is able to collect and process:

- more than 2900 resource data streams per second, from 128 probes, on a single monitored node, with a resource utilization $<$10%;
- more than 754000 resource data streams per second, from 256 different monitored nodes using a single collector node;
- more than 6000000 resource data streams per second per cluster.

By using the TSDB component (see Figure 3), every collector node provides the real-time plotting. In Table 4, we report the resource consumption of this process. In this testbed we request an increasing number of graphs (from 10 to 100) and we set a refresh rate of 15 seconds for each graph.

As for the collector process, the memory consumption of the TSDB component is negligible with respect to the CPU consumption. The TSDB process uses about the 66% of CPU while plotting 100 graphs (i.e. 30000 resource data streams) for each collector node every 15 seconds. Moreover, Table 4 shows that both the incoming and outcoming network bandwidth consumptions are negligible if compared to the network consumptions of the collector process. By using the 12.5% and the 66.4% of CPU for the collector and TSDB respectively, more than the 20% of spare CPU can be used for other purposes like the execution of the Distributed sample storage jobs.

Table 4: Average resource utilization of a TSDB process over the distributed cluster data filter.

#graphs	#resource data streams	CPU (%)	NET In (%)	NET Out (%)
10	4500	10,3	0,077	0,131
25	11250	25,1	0,163	0,265
50	22500	49,8	0,329	0,538
100	30000	66,4	0,432	0,714
100	45000	98,2	0,671	1,099

5 CONCLUSIONS

In this paper, we proposed a novel hybrid architecture for monitoring large-scale, geographically distributed network infrastructures spread across multiple data centers. Architectural choices are made in order to satisfy scalability, effectiveness, resiliency and multi-tenancy requirements. These choices are mandatory when you have to support gathering and analysis operations of huge numbers of data streams coming from cloud system monitors. The proposed architecture is already integrated with on-line analyzers working at different temporal scales. Our preliminary experiments show the potential scalability limits of the monitoring system: more than 6M of resource data streams per cluster, per second. All these operations of data streams are carried out within real-time constraints in the order of seconds thus demonstrating that huge margins of improvement are feasible.

Future work includes the evaluation of traffic scalability between data centers for common analytics for monitoring and the comparison with respect to state-of-the-art system architectures (Rabkin and Katz, 2010; Surhone et al., 2011; Sacerdoti et al., 2003; Renesse et al., 2003).

REFERENCES

Andreolini, M., Colajanni, M., and Pietri, M. (2012). A Scalable Architecture for Real-Time Monitoring of Large Information Systems. In *NCCA'12, 2nd IEEE Symposium on Network Cloud Computing and Applications*. IEEE Computer Society.

Andreolini, M., Colajanni, M., and Tosi, S. (2011). A Software Architecture for the Analysis of Large Sets of Data Streams in Cloud Infrastructures. In *CIT'11, 11th IEEE International Conference on Computer and Information Technology*. IEEE Computer Society.

Babu, S., Subramanian, L., and Widom, J. (2001). A Data Stream Management System for Network Traffic Management. In *NRDM'01, 1st Workshop on Network-Related Data Management*.

Badger, M. (2008). *Zenoss Core Network and System Monitoring*. Packt Publishing Ltd, Birmingham, UK.

Calder, B. et al. (2011). Windows Azure Storage: a highly available cloud storage service with strong consis-

tency. In *SOSP'11, 23rd ACM Symposium on Operating System Principles*. ACM.

Castro, M., Druschel, P., Kermarrec, A.-M., and Rowstron, A. (2002). Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC)*.

Cranor, C., Johnson, T., and Spataschek, O. (2003). Gigascope: a stream database for network applications. In *SIGMOD'03, 2003 ACM SIGMOD International Conference on Management of Data*. ACM.

Davis, C. (2013). Graphite - Scalable Realtime Graphing. http://graphite.wikidot.com.

Dean, J. and Lopes, J. (2004). MapReduce: Simplified Data Processing on Large Clusters. In *OSDI'04, 6th Symposium on Operating Systems Design and Implementation*. USENIX Association.

Gantz, J. and Reinsel, D. (2012). The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. http://www.emc.com/leadership/digital-universe/iview/big-data-2020.htm.

George, L. (2011). *HBase: the definitive guide*. O'Reilly Media, Sebastopol, CA.

Hasselmeyer., P. and d'Heureuse, N. (2010). Towards holistic multi-tenant monitoring for virtual data centers. In *NOMS'10, 2010 IEEE/IFIP Network Operations and Management Symposium Workshops*. IEEE Computer Society.

Hoffman, S. and Souza, S. D. (2013). *Apache Flume: Distributed Log Collection for Hadoop*. Packt Publishing, Birmingham, UK.

Josephsen, D. (2007). *Building a Monitoring Infrastructure with Nagios*. Prentice Hall, Upper Saddle River, NJ.

Keller, A. and Ludwig, H. (2003). The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and Systems Management*.

Kundu, D. and Lavlu, S. (2009). *Cacti 0.8 Network Monitoring*. Packt Publishing, Birmingham, UK.

Leu, J. S., Yee, Y. S., and Chen, W. L. (2010). Comparison of Map-Reduce and SQL on Large-Scale Data Processing. In *ISPA'10, 1st International Symposium on Parallel and Distributed Processing with Applications*. IEEE Computer Society.

Litvinova, A., Engelmann, C., and Scott, S. L. (2010). A proactive fault tolerance framework for high-performance computing. In *PDCN'10, 9th IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN2010)*. ACTA Press.

Lv, Q., Cao, P., Cohen, E., Li, K., and Shenker, S. (2002). Search and replication in unstructured peer-to-peer networks. In *ICS'02, 16th International Conference on Supercomputing*. ACM.

Marchetti, M., Colajanni, M., and Messori, M. (2010). Selective and early threat detectionin large networked systems. In *CIT'10, 10th IEEE International Conference on Computer and Information Technology*. IEEE Computer Society.

Massie, M. L., Chun, B. N., and Culler, D. E. (2004). The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing*.

Olston, C. et al. (2008). Pig Latin: a not-so-foreign language for data processing. In *SIGMOD'08, 2008 ACM SIGMOD International Conference on Management of Data*, New York, NY. ACM.

Olups, R. (2010). *Zabbix 1.8 network monitoring*. Packt Publishing, Birmingham, UK.

Rabkin, A. and Katz, R. (2010). Chukwa: a system for reliable large-scale log collection. In *LISA'10, 24th International Conference on Large Installation System Administration*. USENIX Association.

Renesse, R. V., Birman, K. P., and Vogels, W. (2003). Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Transactions on Computer Systems*.

Rowstron, A. and Druschel, P. (2001). Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *MIDDLEWARE'01, 3rd IFIP/ACM International Conference on Distributed Systems Platforms*.

Sacerdoti, F. D., Katz, M. J., Massie, M. L., and Culler, D. E. (2003). Wide Area Cluster Monitoring with Ganglia. *Cluster Computing*.

Shvachko, K. et al. (2010). The Hadoop Distributed File System. In *MSST'10, 26th Symposium on Massive Storage Systems and Technologies*. IEEE Computer Society.

Sigoure, B. (2010). OpenTSDB, a distributed, scalable Time Series Database. http://opentsdb.net.

Surhone, L. M., Tennoe, M. T., and Henssonow, S. F. (2011). *OpenNMS*. Betascript Publishing, Mauritius.

Voicu, R., Newman, H., and Cirstoiu, C. (2009). MonALISA: An agent based, dynamic service system to monitor, control and optimize distributed systems. *Computer Physics Communications*.

Zyrion (2010-2013). Traverse: distributed, scalable, high-availability architecture. http://www.zyrion.com/company/whitepapers.

Robust Performance Control for Web Applications in the Cloud

Hector Fernandez[1], Corina Stratan[1] and Guillaume Pierre[2]
[1]*Vrije Universiteit Amsterdam, Amsterdam, The Netherlands*
[2]*IRISA / University of Rennes 1, Rennes, France*
{*hector.fernandez, corina.stratan*}*@vu.nl, guillaume.pierre@irisa.fr*

Keywords: Cloud Computing, PaaS, Resource Provisioning, Web Applications.

Abstract: With the success of Cloud computing, more and more websites have been moved to cloud platforms. The elasticity and high availability of cloud solutions are attractive features for hosting web applications. In particular, the *elasticity* is supported through trigger-based provisioning systems that dynamically add/release resources when certain conditions are met. However, when dealing with websites, this operation becomes more problematic, as the workload demand fluctuates following an irregular pattern. An excessive reactiveness turns these systems into imprecise and wasteful in terms of SLA fulfillment and resource consumption. In this paper, we propose three different provisioning techniques that expose the limitations of traditional systems, and overcomes their drawbacks without overly increasing complexity. Our experiments conducted on both public and private infrastructures show significant reductions in SLA violations while offering performance stability.

1 INTRODUCTION

One of the major innovations provided by Cloud computing platforms is their pay-per-use model where clients pay only for the resources they actually use. This business model is particularly favorable for application domains where workloads vary widely over time, such as the domain of Web application hosting. Web applications in the cloud can request and release resources at any time according to their needs.

However, provisioning the right volume of resources for a Web application is not a simple task. Web applications are usually composed of multiple types of components such as Web servers, application servers, database servers and load balancers that distribute the incoming traffic across them. Complex performance behavior of these components makes it difficult to find the optimal resource allocation, even when the application workload is perfectly stable. The magnitude of the problem is further increased by the fact that Web application workloads are often very unstable and hard to predict. In the case of a sudden load increase there is a necessary tradeoff between reacting as early as possible to minimize the duration when the application underperforms because of insufficient processing capacity, and a slower approach to avoid situations where the load has already decreased when the new resources become available.

Faced with this difficult scientific challenge, the academic community has proposed a wide range of sophisticated resource provisioning algorithms (Dejun et al., 2011; Muppala et al., 2012; Urgaonkar et al., 2008; Vasić et al., 2012). However, we observe a wide discrepancy between these academic propositions and the very simple mechanisms that are currently available to cloud customers. These mechanisms are usually based on lower or upper thresholds on resource utilization. Crossing one of the thresholds triggers a pre-defined resource provisioning action such as adding or removing one machine.

We postulate three possible reasons why sophisticated techniques are not more widely deployed: *(i)* the gains of using sophisticated provisioning strategies are too low to be worth the effort; *(ii)* implementing and evaluating these techniques is a difficult exercise, which is why real cloud systems rely on simpler techniques; and *(iii)* academic approaches mostly focus on unrealistic evaluations using simple applications and artificial workloads (Do et al., 2011; Islam et al., 2012).

This paper investigate which of these possible causes are the real problems, and aims to propose automatic scaling algorithms which provide better results than the simple ones without overly increasing complexity. We implemented and installed several resource provisioning mechanisms in ConPaaS, an open source platform-as-a-service environment for hosting cloud applications (Pierre and Stratan, 2012). Our

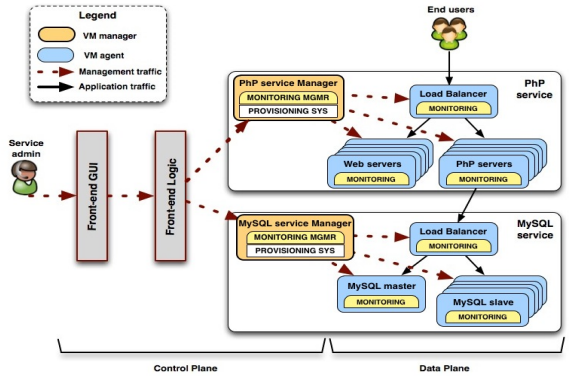


Figure 1: ConPaaS system architecture.

techniques use several levels of thresholds to predict future performance degradations, workload trend detection to better handle traffic spikes and dynamic load balancing weights to handle resources heterogeneity. We exercised these algorithms in realistic unstable workload situations by deploying a copy of Wikipedia and replaying a fraction of the real access traces to its official web site. Finally, we report on *(i)* implementation complexity; and *(ii)* potential gains compared to the threshold-based solution.

2 CONPAAS OVERVIEW

ConPaaS is an open-source runtime environment for hosting applications in Cloud infrastructures (Pierre and Stratan, 2012). In ConPaaS, an application is designed as a composition of one or more elastic and distributed *services*. Each service is dedicated to host a particular type of functionality of an application. ConPaaS currently supports seven different types of services: two web application hosting services (PHP and JSP); a MySQL database service; a NoSQL database service; a MapReduce service; a TaskFarming service; and a shared file system service.

Each ConPaaS service is made up of one manager virtual machine (VM) and a number of agent VMs.

Agent: The agent VMs hosts the necessary components to provide the service-specific functionality. Based on the performance requirements or the application workload, agent VMs can be started or stopped on demand.

Manager: Each service is controlled by one manager VM. The manager is in charge of centralizing monitoring data, controlling the allocation of resources assigned to the service, and coordinating reconfigurations so no loss of service is experienced when adding or removing resources.

Figure 1 shows the architecture of the ConPaaS deployment for a typical PHP web application backed by a MySQL database. The application is deployed using two ConPaaS services: the PHP service and the MySQL service. As illustrated in Figure 1, the manager VM of each ConPaaS service includes a performance monitoring and a resource provisioning system. The monitoring component is based on Ganglia (Ganglia monitoring system,), a scalable distributed monitoring system. The monitoring data is collected by the manager VM. For the PhP service, we implemented modules that extend Ganglia's standard set of monitoring metrics for measuring the request rate and response times of requests.

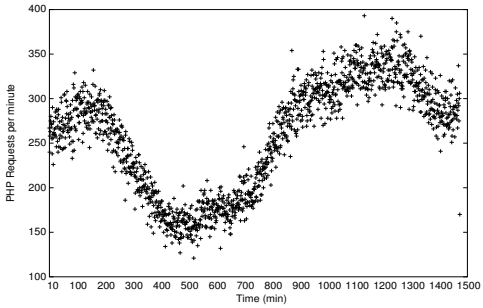
Each ConPaaS service can plug-in its own automatic resource provisioning policies, taking into account specificities of the service such as the structure of the service deployment, the complexity of the requests and service-specific monitoring data. So far our focus has been on the web hosting service, for which we implemented a number of automatic resource provisioning policies.

3 WIKIPEDIA APPLICATION

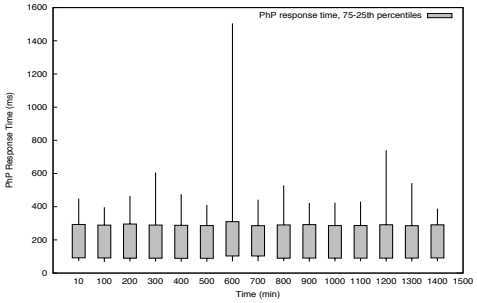
Most academic resource provisioning systems are evaluated using synthetic applications and workloads such as TPC-W, RuBiS and RuBBoS. However, these traditional benchmarks do not exhibit features of real Web applications such as traffic unstability as well as heterogeneous and constantly-changing request mixes (Cecchet et al., 2011). Instead of basing our evaluations on unrealistic workloads, we used the WikiBench benchmark (van Baaren, 2009). This benchmark uses a full copy of Wikipedia as the web application, and replays a fraction of the actual Wikipedia's access traces.

Hosting a copy of Wikipedia in ConPaaS requires two different services: a PHP web hosting and a MySQL service. The MySQL service is loaded with a full copy of the English Wikipedia articles as of 2008, which has a size of approximately 30GB. In the PHP service, the configuration was composed of one load balancer, one static web server and one or more PHP servers. We then use WikiBench to replay access traces from 2008 (Urdaneta et al., 2009). The trace contains requests for static and dynamic pages. When replaying it, the most important performance bottleneck is the application logic of the application: PHP requests are processed an order of magnitude slower than simpler static web pages. In this article we therefore focus on the automatic scaling of the PHP tier.

As an example, in Figure 2(a), we show the PHP workload sampled from one trace, as the number of PHP requests per minute during approximately one



(a) Workload intensity



(b) PHP request complexity

Figure 2: One day of Wikipedia trace.

day. Besides the obvious variations in number of requests per minute, requests are also heterogeneous in complexity. To illustrate this heterogeneity, in Figure 2(b) we present the distribution of the response time values for the PHP requests during the execution of the trace shown in Figure 2(a). For this experiment we used a fixed number of 5 PHP servers, which are sufficient for handling the workload from the access trace even at its peaks; this is why in this case the response time is not influenced by the intensity of the workload. However, the results show a relatively wide dispersion of the response time. The main reason for this dispersion is that the Wikipedia articles vary in complexity, requiring different amounts of information that needs to be retrieved from the database and assembled in a page. We also conclude from this experiment that, since the response time of the application varies and cannot be properly predicted even when the request rate is known, a provisioning algorithm aiming to keep the application's response time under a certain limit should take into account several monitoring parameters beyond the request rate and the response time.

4 PROVISIONING ALGORITHMS

We evaluate three resource provisioning algorithms: a trigger-based algorithm representing a baseline comparable to systems currently in production, and two new algorithms based on feedback techniques that alleviate the drawbacks of trigger-based systems by reacting in advance to traffic changes.

Algorithm 1: Trigger-based provisioning.

Data: Pre-defined metric threshold ranges, *SLO_threshold*
Result: Scaling decisions
while *auto-scaling is ON* **do**
  Collect monitoring data of each metric, $data_i$;
  if *no recent scaling operation* **then**
    if $avg(data_i)$>=SLO_threshold_max$_i$ *for at least one metric* **then**
      ADD resources;
    else if $avg(data_i)$<SLO_threshold_min$_i$ *for all metrics* **then**
      REMOVE resources;
    end
  end
  Wait until the next reporting period (5 minutes) ;
end

4.1 Trigger-based Provisioning

Existing cloud infrastructures adjust the amount of allocated resources based on a number of standard monitoring parameters. These are simple trigger-based systems which define threshold rules to increase or decrease the amount of computational resources when certain conditions are met. As an example, the Auto Scaling system offered by Amazon EC2 (Amazon Elastic Compute Cloud (EC2),) can be used to define rules for scaling out or back an application based on monitoring parameters like CPU utilization, network traffic volume, etc. Similar trigger-based techniques are also used in platforms such as RightScale (RightScale,) or OpenShift (OpenShift,).

For the sake of comparison, we designed and implemented a trigger-based provisioning mechanism in ConPaaS. This technique monitors the percentage of CPU usage and application response time metrics, and assigns a lower and upper threshold for each metric. It then creates or deletes VMs whenever one of the threshold is crossed, as described in Algorithm 1. Analogous to traditional trigger-based systems, the thresholds are statically defined by the user before execution.

Even though this algorithm is simple and widely used in cloud platforms, we will show that it is too reactive. This is due to two main factors:

1) Workload Heterogeneity: For some web sites, the workload fluctuates following an irregular pattern, with occasional short traffic spikes. An excessively reactive algorithm cannot handle these situations well, and will cause frequent fluctuations in the number of allocated resources; this has negative effects on the stability and performance of the system.

Considering the pricing model of cloud providers that charges users on a per-hour basis, trigger-based systems increase the infrastructure cost because they frequently terminate resources before this hour price boundary.

2) Resources Heterogeneity: The performance of virtual instances provided by current clouds is largely heterogeneous, even among instances of the same type (Dejun et al., 2009). Simple trigger-based provisioning systems do not take this heterogeneity into account, thus providing less efficient resource allocation.

We believe that trigger-based provisioning mechanisms can be improved without drastically increasing their complexity. We now present two techniques which aim at solving the aforementioned drawbacks by relying on predictive and more accurate methods.

4.2 Feedback Provisioning

To handle the workload heterogeneity and mitigate the reactiveness of trigger-based techniques, we designed and implemented an algorithm that relies on three simple mechanisms: integration of several metrics and automatic weighting of each metrics according to its predictiveness; estimating the overall workload trend; and definition of two levels of threshold ranges instead of one.

Weighted Metrics: As pointed out in (Singh et al., 2010), provisioning decisions solely made on the basis of request rate or percentage of CPU usage can incur errors by under- or over-provisioning an application. We propose instead to use a collection of metrics (e.g. response time, request rate and CPU usage) likely to indicate the need to scale the system. Our algorithm continuously measures the correlation between each metric's values and the actual scaling decisions. This method enables to better adapt the scaling decisions to the current workload requirements by identifying how metrics affect to the processing time. We thus assign a weight to each metric (correlation coefficient) according to its predictiveness. For example, when hosting the MediaWiki application, the hosting system needs to deal with requests having a wide diversity of computing complexity, and with significant variations in the ratios of simple vs. complex requests. Our algorithm therefore associates a higher weight to the CPU usage metric than to the request rate, as it affects to the response times.

Workload's Trend Estimation: a difficult problem in any autoscaling system is the fact that any decision has a delayed effect. Provisioning extra resources can take a couple of minutes before they are ready to process traffic. Besides, cloud computing resources are usually charged at a 1-hour granularity so the scaling decision has consequences at least during the first hours after provisioning. It is therefore important to estimate the medium-term trend of response times in addition to any short-term bursts. We therefore estimate the response time trend using past values over a period of the same order of magnitude (30 min in our experiments). We classify workloads in one out of three categories: stable, increasing or decreasing.

Two-level Thresholds: Initially, the users define a fixed threshold range based on their performance requirements (max and min required response time), denoted by *SLO thresholds* (Service Level Objective) in Figure 3. However, these upper and lower bounds

Algorithm 2: Feedback provisioning.

```
   Data: Pre-defined metric threshold ranges, SLO_threshold
   Result: Scaling decisions
 1 Create a queue to store historical workload, q;
 2 Establish two-level thresholds: pred_thr and reac_thr ;
 3 while auto-scaling is ON do
 4   Initialize variables s_bck ands_out to 0;
 5   Collect monitoring data (last ∼5min), data;
 6   Add the most recent response_time value to q;
 7   Estimate workload trend (last ∼30min) of q, td;
 8       - Increasing, td = 1; Decreasing, td = 0; Stable, td = -1;
 9   for each metric in data do
10     // Compute the weight w_i of metric_i
11     w_i = correlation_coefficient(metric_i, response_time);
12     if avg(metric_i) >= pred_thr_max_i then
13       // Increment chances of scaling_out
14       s_out = s_out + w_i;
15     else if avg(metric_i) < pred_thr_min_i then
16       // Increment chances of scaling_back
17       s_bck = s_bck + w_i;
18       else
19         // Decrease the chances
20         s_bck =s_bck - w_i; s_out =s_out - w_i;
21       end
22     end
23     if no recent scaling operation then
24       if avg(metric_i) >= reac_thr_max_i and td = 1 and s_out > s_bck
         then
25         ADD resources;
26       else if avg(metric_i) < reac_thr_min_i and td = 0 and s_out <
         s_bck then
27         REMOVE resources;
28       end
29       Reset s_bck ands_out to 0 ;
30     end
31     Wait until next reporting period (5 minutes);
32   end
33 end
```

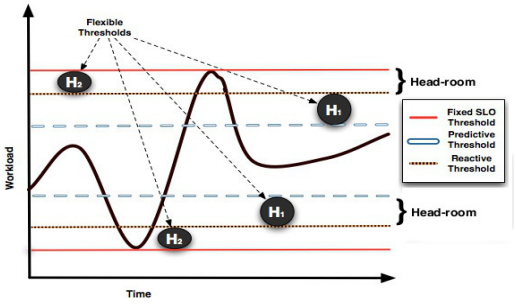


Figure 3: Two-level thresholds.

only indicate the resource under-utilization or when a SLO violation occurs. Thus, to prevent in advance these performance degradations, additional boundaries have to be defined. This mechanism establishes two levels of threshold ranges for each metric (CPU and response time) based on the bounds defined in the SLO by the user.

In Figure 3, these two extra thresholds called *predictive* and *reactive*, both with upper and lower bounds, create two "head-rooms" between the SLO threshold and them. The predictive head-room H_1 is intended to alert of future workload alterations when a metric exceeds its predictive bounds, thus increasing proportionally to its weight the chances to trigger scaling actions (denoted by *s_bck* and *s_out* in Algorithm 2 lines 12-17). Otherwise, the scaling chances will drop in the same proportion (Line 20). The reactive head-room H_2 is used to trigger scaling actions if the workload presents a *increasing or decreasing* variation (Lines 24-26), which may cause SLO violations if it keeps such a trend. This mechanism in conjunction with the workload trend estimation allow to better analyze the evolution of performance fluctuations, and as a consequence improves the accuracy of our decisions by reacting in advance. In the future, these two-levels of thresholds could be adjusted depending on the hardware configuration of each provisioned VM, as introduced in (Beloglazov and Buyya, 2010). To sum up, the feedback algorithm triggers a scaling action when a series of conditions are satisfied: (i) no previous scaling actions have been taken over the last 15min; (ii) the recent monitoring data have to exceed the predictive and reactive threshold ranges; (iii) the workload trend has to follow a constant pattern (increasing/decreasing). Although the combination of these techniques improves the accuracy of our measurements, the heterogeneous nature of the VM instances requires more flexible provisioning algorithms, as pointed out in (Jiang, 2012).

4.3 Dynamic Load Balancing Weights

The problem we consider here is the heterogeneity of cloud platforms. Different VMs have different performance characteristics, even when their specifications from the cloud vendor are the same. This issue can be addressed through various load balancing techniques, like assigning weights to the backend servers or taking into account the current number of connections that each server handles. Furthermore, the performance behavior of the virtual servers may also fluctuate, either due to changes in the application's usage patterns, or due to changes related to the hosting of the virtual servers (e.g., VM migration).

In order to address these issues in ConPaaS we implemented a weighted load balancing system in which the weights of the servers are periodically readjusted automatically, based on the monitoring data (e.g. response time, request rate and CPU usage). This method assigns the same weight to each backend server at the beginning of the process. The weights are then periodically adjusted (in our experiments, every 15min) proportionally with the difference among the average response times of the servers during this time interval. By adding this technique to the feedback-based algorithm, we noticed a performance improvement when running the benchmarks.

5 EVALUATION

To compare the provisioning algorithms described above, we ran experiments on two infrastructures: a homogeneous one (the DAS-4, a multi-cluster system hosted by universities in The Netherlands (Advanced School for Computing and Imaging (ASCI),)) and a heterogeneous one (Amazon Elastic Compute Cloud (EC2),). The goal of our experiments was to compare the algorithms by how well they fulfill the SLOs and by the amount of resources they allocate.

Testbed Configuration: As a representative scenario, we deployed the MediaWiki application using ConPaaS on both infrastructures, and we ran the Wikibench tools with a 10% sample of a real Wikipedia access trace for 24hours. We configured the experiments as follows: a monitoring window of 5min, a SLO of 700ms at the service's side (denoted by a red Line in Figures) and the same statistically-chosen performance threshold ranges for response time and CPU utilization. Note that, the weighted load-balancing provisioning technique was only evaluated on the heterogeneous platform Amazon EC2, as it only brings improvements in environments where VMs may have different hardware configurations.

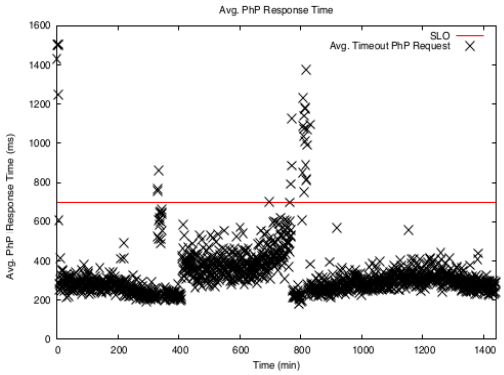


Figure 4: Response time on DAS4 – Trigger-based.

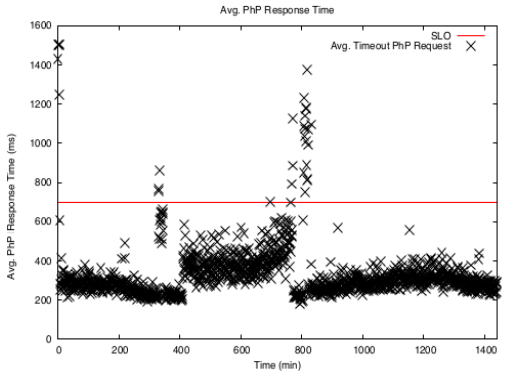


Figure 5: Response time on DAS4 – Feedback.

5.1 Homogeneous Infrastructure

Our experiments on DAS-4 rely on OpenNebula as IaaS (Sotomayor et al., 2009). To deploy the Wikipedia services, we used small instances for the PHP service (manager and agents) and a medium instance for the MySQL service (agent). In DAS-4, OpenNebula's small instances are VMs equipped with 1 CPU of 2Ghz, and 1GiB of memory, while medium instances are equipped with 4 CPU's of 2Ghz, and 4GiB of memory.

SLO Enforcement. Figure 4 and Figure 5 represent the degree of SLO fulfillment of the trigger-based and feedback algorithms, indicating the average of response times obtained during the execution of the Wikipedia workload trace. In Figure 4 and Figure 5, each cross represents the average of response time obtained at monitoring window. The results from Figure 4 show that the trigger-based provisioning algorithm provokes an important amount of SLO violations at certain moments in time, due to its excessively reactive behavior. As we mentioned, this algorithm fails easily during traffic spikes, as it adds or removes VMs without evaluating the workload trend. The feedback algorithm, as shown on Figure 5, handles the traffic spikes better and can minimize the amount of SLO violations; specifically, there were 31.72% less SLO violations in comparison with the trigger-based algorithm.

Resource Consumption. To better understand the behavior of both algorithms, we shall also focus on the resource consumption illustrated on Figure 6. The excessively reactive behavior of the trigger-based algorithm can be noticed in the time intervals around *t=350min* and *t=820min*, where two scaling operations under-provision the system during a short period of time. These provisioning decisions provoked the SLO violations that are visible in Figure 4 in the same intervals of time. Besides affecting the system's stability, such short fluctuations in the number of provisioned resource also raise the cost of hosting the application since more VM instantiations will be triggered. When using the feedback algorithm, the system makes provisioning decisions by analyzing the workload's trend. Scaling actions are only triggered when having *constant* alterations in the workload, thereby providing a more efficient resource usage. We can see that the provisioning decisions on Figure 6 match well with the workload variations depicted on Figure 2(a).

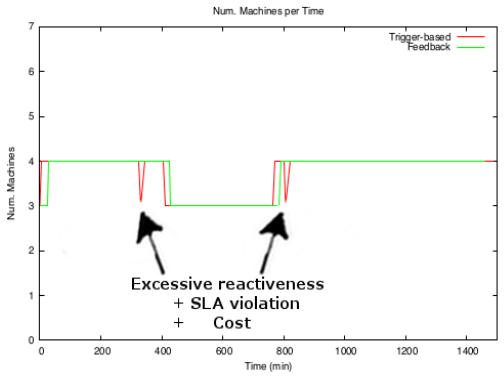


Figure 6: Resource consumption on DAS4.

Discussion. Both algorithms are best-effort regarding the SLO fulfillment, and thus they do not handle well short alterations of the workload (with duration in the range of minutes). The heterogeneity of the PHP requests, as well as the VM scheduling time (2-5min), are in part responsible of these SLO violations.

5.2 Heterogeneous Infrastructure

Our experiments on Amazon EC2 used small instances for the PHP service (manager and agents) and a medium instance for the MySQL service (agent). EC2 small instance are equipped with 1 EC2 CPU, and 1.7GiB of memory, while medium instances are

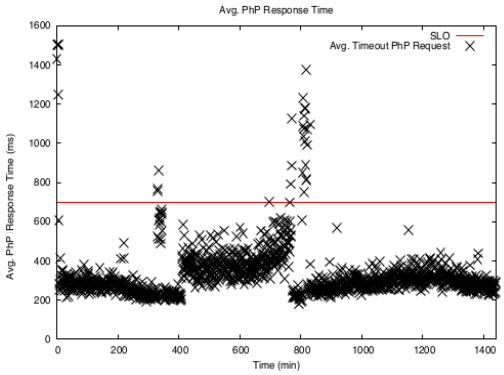


Figure 7: Resource consumption on EC2.

equipped with 2 EC2 CPU's, and 3.75GiB.

SLO Enforcement. Figure 8, Figure 9 and Figure 10 show the system performance of the trigger-based, feedback and dynamic load-balancing weights algorithms, respectively. As depicted on Figure 8, the performance of the trigger-based algorithm is even more unstable than in the case of the homogeneous infrastructure. Two of the three peaks in response time, at *t=300min* and *t=820min*, can be explained by the variations in the Wikipedia workload shown in Figure 2(a). However, there is a third peak between *t=400min* and *t=500min* that corresponds to an interval of time in which the workload trace shows a significant drop in the request volumes. During this period of time, the algorithm attempts to scale back the system but the new lower number of resources cannot handle the load, and the system is scaled out again. As this algorithm does not keep any history information, after a short time it attempts again to scale back the system, with the same result; thus, some oscillations occur in the number of allocated resources, causing also SLO violations.

Figure 9 and Figure 10 show that the feedback and dynamic load-balancing algorithm reduce the number of SLO violations and provide a more stable performance pattern. Specifically, with the feedback algorithm we obtained 41.3% less SLO violations than with the trigger-based algorithm, while the dynamic load balancing algorithm had 47.6% less violations than the trigger-based one. This improvement of 6.3% responds to an efficient distribution of the incoming traffic across the allocated resources.

Resource Consumption. As explained above, the trigger-based algorithm sometimes initiates series of scaling back operations quickly followed by scaling out operations, due to the fact that it does not use any history information that could prevent these oscillations. This can be seen on Figure 7, in the time interval between *t=400min* and *t=500min*. The feedback and dynamic load balancing algorithms have similar and more stable patterns of resource consumption, which show the benefits of using trend estimations and two-level threshold ranges.

Discussion. These runs show significant differences between the trigger-based algorithm on one hand, and the feedback and dynamic load balancing on the other hand. The results show that using a few heuristics as in the last two algorithms can help in improving the performance and stability of the system. We did not obtain however significant differences between the feedback and the dynamic load balancing algorithm. The last algorithm uses the additional technique of dynamically adjusting the load balancing weights, but the VMs participating in the experiments had relatively similar performance and adjusting their weights only made a small difference. On other infrastructures that are more heterogeneous than Amazon EC2, the dynamic load balancing technique may bring a greater performance improvement.

We also note that the frequent scaling actions launched by the trigger-based algorithm generally increase the infrastructure cost. An explanation comes from the frequent creation/removal of VM instances occurring within a time interval of 1hour. Even though this trace shows a clear daily variation which represents the traditional traffic pattern of web applications, more traces have been utilized to validate our autoscaling system in (Fernandez et al., 2014)

6 RELATED WORKS

There is a wide literature on issues related to dynamic resource provisioning for cloud web applications. However, most of these models require a deep understanding in mathematics or machine learning techniques which are not easily interpreted by non specialists. Besides the traffic in web applications is shaped by a combination of different factors such as diurnal/seasonal cycles that follows an irregular pattern, thus making extremely challenging the design and development of realistic and accurate provisioning mechanisms.

These well-known drawbacks force to IaaS like Amazon EC2, or PaaS like RightScale and OpenShift, to design simple trigger-based auto-scaling systems, instead of relying on approaches from academic research. Unfortunately, these scaling systems are imprecise and wasteful in terms of resource consumption and cost savings (Ghanbari et al., 2011).

As a consequence, more relevant and realistic academic approaches have been proposed over the last

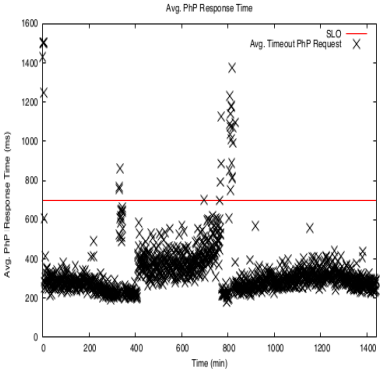


Figure 8: Response time on EC2 – Trigger-based.

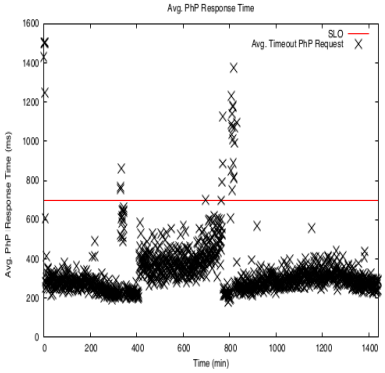


Figure 9: Response time on EC2– Feedback.

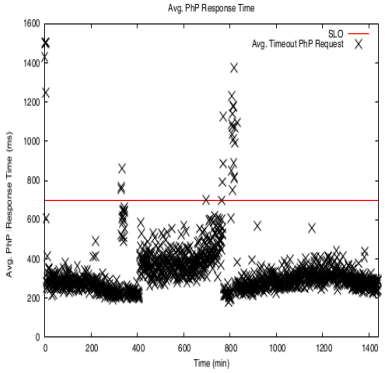


Figure 10: Response time on EC2– Load-balancing Weights.

years. (Urgaonkar et al., 2008) designed and implemented a predictive and reactive mechanism using a queuing model to decide the number of resources to be provisioned, and an admission control mechanism to face extreme workload variations. Even though the use of admission mechanisms enforce the performance requirements, it reduces the QoS of the service, and therefore affects user experience. Differently, (Muppala et al., 2012) proposed offline training techniques to gather information about the resource requirements of the current workload, and thereby to improve the accuracy of the scaling decisions. In the same vein, DejaVu (Vasić et al., 2012) and CBMG (Roy et al., 2011) built similar mechanisms to classify the workload need by analyzing recent traffic spikes or the customer behavior. However, these approaches require additional resources to identify the workload requirements, and an exhaustive knowledge of the deployed applications; thus preventing its integration in existing autoscaling systems.

As a proposal closely related to ours, (Ghanbari et al., 2011) designed an autoscaling system using control theoretic and rules-based models. The authors claimed for simpler provisioning mechanisms in comparison with the sophisticated academic approaches. However, factors such as resource heterogeneity were not addressed in this system.

7 CONCLUSION

Our study demonstrated that traditional trigger-based provisioning mechanisms do not handle workload variation well and in some situations cause instability, by changing the number of provisioned VMs often in short time intervals. This behavior has a negative effect on the application's response time, leading to violations of the SLO.

By using a few techniques that are relatively easy to implement we showed that we can significantly reduce the number of SLO violations and improve the performance stability. Although there is still room for improvement in our techniques, from the experimental results we can draw the conclusion that implementing provisioning mechanisms that go beyond simple triggers is effective and should be considered when hosting a cloud application.

ACKNOWLEDGEMENTS

This work is partially funded by the FP7 Programme of the European Commission in the context of the Contrail project under Grant Agreement FP7-ICT-257438 and the Harness project under Grant Agreement 318521.

REFERENCES

Advanced School for Computing and Imaging (ASCI). The Distributed ASCI SuperComputer 4. http://www.cs.vu.nl/das4/.

Amazon Elastic Compute Cloud (EC2). http://aws.amazon.com/ec2.

Beloglazov, A. and Buyya, R. (2010). Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers. In *Proc. 8th International Workshop on Middleware for Grids, Clouds and e-Science*.

Cecchet, E., Udayabhanu, V., Wood, T., and Shenoy, P. (2011). BenchLab: An open testbed for realistic benchmarking of web applications. In *Proc. WebApps*.

Dejun, J., Pierre, G., and Chi, C.-H. (2009). EC2 performance analysis for resource provisioning of service-oriented applications. In *Proc. 3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing*.

Dejun, J., Pierre, G., and Chi, C.-H. (2011). Resource provisioning of Web applications in heterogeneous clouds. In *Proc. Usenix WebApps*.

Do, A. V., Chen, J., Wang, C., Lee, Y. C., Zomaya, A., and Zhou, B. B. (2011). Profiling applications for virtual machine placement in clouds. In *Proc. IEEE CLOUD*.

Fernandez, H., Pierre, G., and Kielmann, T. (2014). Autoscaling in heterogenous in cloud infrastructures. In *Proc. IEEE IC2E*, page (to appear).

Ganglia monitoring system. http://ganglia.sourceforge.net/.

Ghanbari, H., Simmons, B., Litoiu, M., and Iszlai, G. (2011). Exploring alternative approaches to implement an elasticity policy. In *Proc. IEEE CLOUD*.

Islam, S., Keung, J., Lee, K., and Liu, A. (2012). Empirical prediction models for adaptive resource provisioning in the cloud. *Future Gener. Comput. Syst.*, 28(1):155–162.

Jiang, D. (2012). *Performance Guarantees For Web Applications*. PhD thesis, VU University Amsterdam.

Muppala, S., Zhou, X., Zhang, L., and Chen, G. (2012). Regression-based resource provisioning for session slowdown guarantee in multi-tier internet servers. *Journal of Parallel and Distributed Computing*, 72(3):362–375.

OpenShift. https://openshift.redhat.com/app/flex.

Pierre, G. and Stratan, C. (2012). ConPaaS: a platform for hosting elastic cloud applications. *IEEE Internet Computing*, 16(5):88–92.

RightScale. http://www.rightscale.com//.

Roy, N., Dubey, A., and Gokhale, A. (2011). Efficient autoscaling in the cloud using predictive models for workload forecasting. In *Proc. IEEE CLOUD*.

Singh, R., Sharma, U., Cecchet, E., and Shenoy, P. (2010). Autonomic mix-aware provisioning for non-stationary data center workloads. In *Proc. ICAC*.

Sotomayor, B., Montero, R. S., Llorente, I. M., and Foster, I. (2009). Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing*, 13(5):14 –22.

Urdaneta, G., Pierre, G., and van Steen, M. (2009). Wikipedia workload analysis for decentralized hosting. *Computer Networks*, 53(11):1830 – 1845.

Urgaonkar, B., Shenoy, P., Chandra, A., Goyal, P., and Wood, T. (2008). Agile dynamic provisioning of multi-tier internet applications. *ACM Trans. Auton. Adapt. Syst.*, 3(1):1–39.

van Baaren, E.-J. (2009). Wikibench: A distributed, wikipedia based web application benchmark. Master's thesis, VU University Amsterdam.

Vasić, N., Novaković, D., Miučin, S., Kostić, D., and Bianchini, R. (2012). DejaVu: Accelerating resource allocation in virtualized environments. In *Proc. ASPLOS*.

PAEAN4CLOUD
A Framework for Monitoring and Managing the SLA Violation of Cloud Service-based Applications

Yehia Taher[1], Rafiqul Haque[2], Dinh Khoa Nguyen[3] and Beatrice Finance[1]
[1]*Laboratoire PRiSM, Université de Versailles/Saint-Quentin-en-Yvelines, Versailles, France*
[2]*Laboratoire d'InfoRmatique en Image et Systèmes d'information, Université Claude Bernard Lyon 1, Lyon, France*
[3]*European Research Institute in Service Science, Tilburg University, Tilburg, The Netherlands*
{*yehia.taher, beatrice.finance*}*@prism.uvsq.fr, akm-rafiqul.haque@univ-lyon1.fr,dinhkhoanguyen@gmail.com*

Keywords: Business Transaction, Cloud Computing, Blueprint.

Abstract: Recently, Cloud Computing has become an emerging research topic in response to the shift from product oriented economy to service-oriented economy and the move from focusing on software/system development to addressing business-IT alignment. Technically speaking, Cloud Computing enables to build Cloud Service-Based Application (CSBA) which cater for the tailoring of services to specific business needs using a mixture of SaaS, PaaS and IaaS solutions - possibly from various providers. In other words, in the context of CSBAs, cloud services are rented by clients from providers instead of owning the services. Due ti this specific nature, SLA (Service Level Agreement) has become a very important and up-to-date issues in CSBAs. Therefore SLA turns to be critical for both cloud service clients and providers and needs constant monitoring for various reasons mostly detecting if any violation happens but also preventing the violation in efficient way. As in CSBAs a number of providers are involved, it is a challenge to detect and resist violations of multiple SLAs that engage different providers form different locations. To deal with such a problem, this paper introduces a framework called PAEAN4CLOUD. The framework comprises components for monitoring, detecting, and configuring SLAs. An algorithm is proposed for automatic detection of SLA violations. The configuration component underpins assembling CSBAs automatically at runtime. The components help in preventing SLA violations and optimizing application performance as well.

1 INTRODUCTION

Until today, the cloud computing paradigm has given the rise to a wide variety of services that are grouped and called Everything as a Service (XaaS). It promotes opportunities to transform an object into a service and deliver it to users over the Internet. It enables building *cloud Service-based application (CSBA)* that relies on cloud services such as SaaS, PaaS, IaaS for designing, implementing, deploying, executing, monitoring, managing applications. CSBAs are highly customizable and allows building applications by localizing the services for satisfying the very specific needs of a context *e.g.* organization or individual. CSBAs pose various challenges. The major challenge is *detecting* and *managing* different types of violations of service level agreement (SLA). SLA is a predominant entity in CSBAs. In CSBAs, clients rent services from providers instead of *buying services*. This means that a CSBA is composition of list of rented service. Thus, SLA becomes critical to both service clients and providers and it needs to be monitored constantly while a CSBA is running to detect violation and prevent the violation. As in CSBAs a number of providers are involved, detecting and resisting violations (of multiple SLAs that engage different providers form different locations) are enormously challenging. Therefore, an efficient and effective approach is paramount importance for CSBAs.

Monitoring and managing SLAs for service based applications (SBAs) have documented in various bodies of literatures, nonetheless, the solutions proposed in literature are not the best fit to CSBAs since the development landscape of these applications are different. This implies that the existing approaches cannot be reused in cloud environment. In addition, the state of the art lacks an efficient approach for detecting and resisting SLA violations in cloud-based environment.

The goal of this research is to address this shortcoming. To this end, a framework called

PAEAN4CLOUD is proposed in this paper. The framework comprises a list of components for monitoring, detecting, and configuring CSBAs. An algorithm is proposed in this paper for automatic detection of SLA violations. The configuration component enables assembling CSBAs automatically at runtime. It assists in preventing SLA violations and optimizing application performance as well. The remainder of this paper is organized as follows. Section 2 presents a motivating scenario. Section 3 describes the configuration and deployment of CSBAs. The patterns of SLA violation is explained in section 4. The PAEAN4CLOUD framework is described in section 5. Section 6 presents the algorithms that automate the violation detection and reconfiguration of CSBA. The related works is described in section 7. The conclusion is drawn in section 8.

2 MOTIVATING SCENARIO

This section illustrates a simple scenario of today's industrial reality. This scenario is an extension of Taxi Application Scenario that was originally co-developed with several IT companies like Telefonica, Telecom Italia, Ericson, 2ndQuadrant and SAP for the EC's 4CaaSt project (European Comission: Information & Communication Technologies Unit, 2010). The scenario promotes a comprehensive approach to exist so that the design, configuration, deployment, monitoring of SLA, and adaptation of a CSBA can occur.

`TaxiTilburg` is an application service provider that offers the taxi ordering application *TaxiOrdering-CSBA* in the form of an end-to-end process, as shown in Fig. 1. The process is composed of several steps including *Receive Order*, *Allocate Taxi*, *Calculate Route*, *Confirm Order*, and *Reject Order*. In order to implement the steps S1 (Receive Order), S4 (Confirm Order), S5 (Reject Order) of the TaxiOrdering-CSBA process, TaxiTilburg has developed an in-house SMS software component that is able to receive new orders by sms, and send order confirmations or rejections to customers. Unfortunately, due to lack of expertise, TaxiTilburg has to rely on third-party SaaS offerings for managing its taxi fleet and for calculating routes, which are the two necessary functionalities to implement step S2 (Allocate Taxi) and step 3 (Calculate Route) in its TaxiOrdering-CSBA process. These two required functionalities are captured as the two functional requirements *Fleet Management Service* and *Map Service*.

Figure 1 also shows the Quality-of-Service (QoS) offerings $(responseTime(s) \leq 15) \& (throughput(req/s) \geq 100)$ that TaxiTilburg has promised to its customers. To maintain this predefined QoSs, TaxiTilburg has distributed the QoS value of maximum response time to each step contained in TaxiOrdering-CSBA process. The QoS requirement of minimum throughput 100 req/s remains same for all steps, as shown in Figure 1.

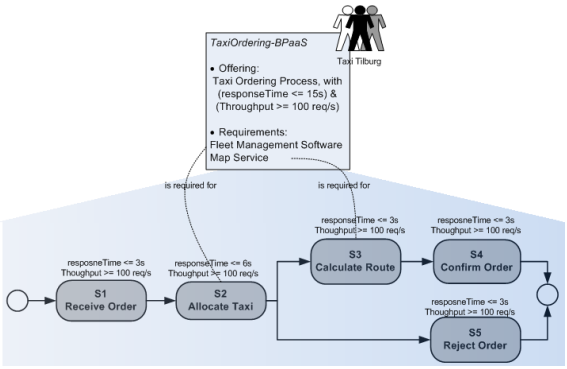


Figure 1: The process definition of TaxiOrdering-CSBA.

2.1 Configuring the CSBA

In order to deploy the TaxiOrdering-CSBA on the cloud, it is necessary to satisfy all the functional and QoS requirements of the TaxiOrdering-CSBA. We follow the *Blueprint Approach* to configure the TaxiOrdering-CSBA, so that all of its functional and QoS reuqirements can be correctly fulfilled. The concept "Blueprint" has defined in (Nguyen et al., 2012a) as "an uniform abstract description of a cloud service, i.e. SaaS, PaaS, or IaaS, that facilitates the CSBA developers with the selection, customization and composition of cross-layered cloud services". Depending on the abstraction layer of cloud services, there are three different types of blueprints *SaaS blueprint*,*PaaS Blueprint*, and *IaaS blueprint*. Figure 2 illustrates the configuration of TaxiOrdering-CSBA using the blueprint approach. There is a SaaS blueprint describing the `TaxiOrdering-CSBA` with its offering and requirements. The blueprint composition approach (Nguyen et al., 2012b) subsequently queries the marketplace repository to retrieve appropriate source blueprints whose offerings can fulfill the requirements of the `TaxiOrdering-CSBA` blueprint. The newly retrieved source blueprints may also contain requirements, e.g. the `FleetMgt-SaaS` blueprint. Hence, this querying and matchmaking task has to be executed iteratively until the entire configuration of the `TaxiOrdering-CSBA` no longer contains any further requirement.

The matchmaking between an offering and a requirement does not only take into account the func-

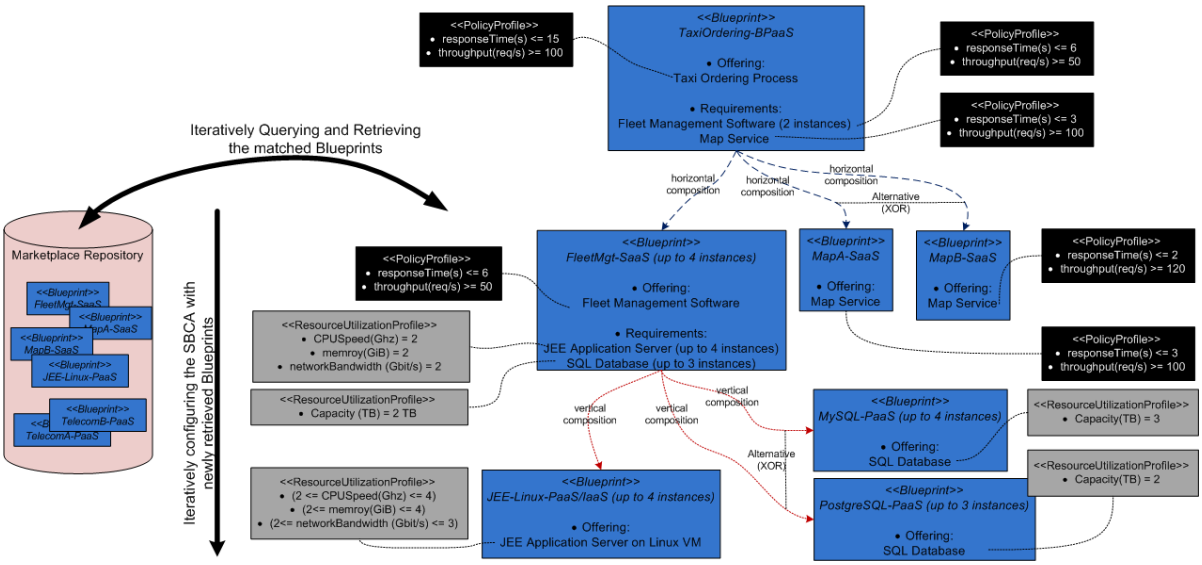


Figure 2: Blueprint Configuration for the Taxi Ordering CSBA.

tional matchmaking, but also the QoS matchmaking, i.e. whether the QoS properties specified in the $<<$ *PolicyProfile* $>>$ of the requirement can be satisfied by the QoS properties specified in the $<<$ *PolicyProfile* $>>$ of the offering. For instance in Figure 2, the two blueprints `MapA-SaaS` and `MapB-SaaS` are selected to fulfill the requirement `Map Service`, since their QoS properties regarding the maximum response time and minimum throughput satisfy the QoS properties specified for the requirement `Map Service`. Furthermore on the PaaS/IaaS level, matchmaking is also based on the resource provisioning specification specified in the so-called $<<$ *ResourceUtilizationProfile* $>>$, e.g. in Figure 2 the `FleetMgt-SaaS` blueprint has some resource provisioning properties specified for its requirement `JEE Application Server` regarding the required CPU speed, memory size and network bandwidth, and these properties can be satisfied by the resource provisioning properties of the `JEE-Linux-PaaS/IaaS` blueprint.

It is worth noting in Figure 2 that there could be alternative matchmaking results for fulfilling a requirement, e.g. for fulfilling the requirement `Map Service` of the `TaxiOrdering-CSBA` blueprint, and for fulfilling the requirement `SQL Database` of the `FleetMgt-SaaS` blueprint. The QoS properties of the alternative matchmaking results are different, which implies that there might exist alternative configurations for the `TaxiOrdering-CSBA` that could trigger a reconfiguration process in case there are some changes in the QoS requirements of the `TaxiOrdering-CSBA`.

The next step after configuring an CSBA is to select among the alternatives the optimum blueprint configuration that will be deployed on the physical cloud infrastructure. The selection strategy might depend on a variety of criteria such as the cost, licensing issues, or some private business constraints. For brevity reason we do not discuss the criteria for selecting a blueprint configuration in this paper. In Figure 2, a configuration has been chosen that involves the `MapA-SaaS` blueprint to fulfill the requirement `Map Service` of the `TaxiOrdering-CSBA` blueprint, and the `MySQL-PaaS` blueprint to fulfill the requirement `SQL Database` of the `FleetMgt-SaaS` blueprint.

2.2 SLA Specification

Each cloud service provider involved in the `TaxiOrdering-CSBA` configuration in Fig.2 promises to satisfy the stipulated Qualities of Services(QoS) through a Service Level Agreement (SLA) with his consumer. Hence, as soon as a blueprint configuration is finalized, SLAs are generated among all the involved cloud service providers. On the SaaS layer, the SLAs mainly contain the promised maximum response time of handling a request and the minimum throughput (in terms of number of request per seconds). On the PaaS layer, the PaaS providers mainly specify in their SLAs the *number of instances* of their offerings that can be provided to a single user, and the resource capacity of their offerings including for instance the *CPU speed* (in GHz), *memory* (in GiB), *networkBandwidth* (in Gbit/s), and storage *capacity* (in TB). Table 1 summarizes the SLA specifications of all the cloud service offerings in the scenario.

Table 1: SLA specification of Cloud Services.

Cloud Service	SLA specification for consumers
TaxiOrdering-CSBA	$(responseTime(s) \leq 15) \& (throughput(req/s) \geq 100)$
MaMapA-SaaS	$(responseTime(s) \leq 3) \& (throughput(req/s) \geq 100)$
MapB-SaaS	$(responseTime(s) \leq 2) \& (throughput(req/s) \geq 120)$
FleetMgt-SaaS	$(numberOfInstance \leq 4) \& (responseTime(s) \leq 6) \& (throughput(req/s) \geq 50)$
MySQL-PaaS	$(numberOfInstance \leq 4) \& (capacity(TB) = 3)$
PostgreSQL-PaaS	$(numberOfInstance \leq 3) \& (capacity(TB) = 2)$
JEE-Linux-PaaS	$(numberOfInstance \leq 3) \& (2 \leq CPUSpeed(Ghz) \leq 4) \& (2 \leq memory(GiB) \leq 4) \& (2 \leq networkBandwidth(Gbit/s \leq 3)$

2.3 Deploying the CSBA

In the next phase, a deployment architecture has to be developed based on the selected optimum blueprint configuration. The purpose of this deployment architecture is to serve as a cookbook for cloud administrator to deploy and maintain the CSBA on the cloud infrastructure. In this architecture, the blueprints will be transformed into a set of concrete, multi-layered cloud service instances and network links that collaboratively form the deployment topology of the CSBA. Figure 3 illustrates the deployment architecture for the `TaxiOrdering-CSBA`, in which:

- Two instances of the `MySQL-PaaS` should be deployed
- Two instances of the `FleetMgt-SaaS` blueprint should be deployed on two instances of the `JEE-Linux-PaaS/IaaS`.
- A network link should be deployed to connect the two `MySQL-PaaS` instances with the two `JEE-Linux-PaaS/IaaS` instances.
- One instance of the `MapA-SaaS` should be deployed
- A network link should be deployed to connect the two `FleetMgt-SaaS` instances with the `Map-SaaS` instance and with all the prospective `TaxiOrdering-CSBA` instances.

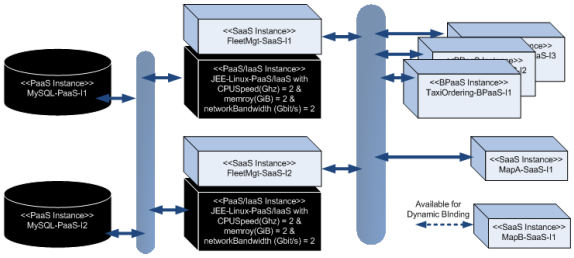


Figure 3: Deployment Architecture of the Taxi Ordering CSBA.

Maintaining the QoS levels that have been specified in the SLAs is indispensable for all the service providers providing platform service, software services, and physical resource services that are used for building the TaxiOrdering-CSBA application that will underpin the process. At runtime, a framework is required for monitoring the execution of TaxiOrdering-CSBA process steps, detecting the occurrence of violations or potential violations of QoS constraints, and preventing these violations. PAEAN4CLOUD is proposed for addressing these requirements as an efficient solution is missing.

3 PATTERNS OF SERVICE LEVEL AGREEMENT VIOLATION

In this section, we describe the Patterns of SLA Violation (PSLAV) that occur in workstreams while a CSBA is running. Additionally, the formal semantics of these patterns is provided in a lightweight style using First Order Logic (FOL). The key purpose of discovering and formalizing these patterns is to make the PAEAN4CLOUD aware of SLA violations and enable it for detecting SLA violations automatically. In addition, the patterns enable PAEAN4CLOUD detecting and preventing SLA violations proactively and efficiently. It is worth noting that, the violations reported in this paper have discovered through a study on the cloud SLA samples provided in various literature e.g. (Wu and Buyya, 2010), (Emeakaroha et al., 2012a), and (DeveloperWorks, 2010).

Before defining the PSLAVs, the concepts SLA and violation of SLA are defined.

Definition 1. *Let 'Σ' be the SLA that contains a list of quality constraints 'Q^C', security constraints 'S^C', privacy constraints 'P^C', and regulatory constraints 'R^C' such that,*

$$\Sigma := (Q^C \wedge S^C \wedge P^C \wedge R^C)$$

Since CSBA relies on third party cloud service providers, an SLA involves a 'consumer' and one to

many 'providers'. For example, a client "X" buys infrastructure services from Amazon[1], platform service from google [2], and software service from SalesForce [3]. These providers must satisfy the obligations they promise to the client "X".

Definition 2. *A violation 'Γ' indicates either a cloud service consumer or a provider fails to satisfy one or more constraints contained in the agreement.*

The SLA violation patterns are classified into QoS constraint violation patterns, security constraint violation patterns, privacy constraint violation patterns, and regulatory constraint violation patterns. The term 'constraint' refers to obligations related to QoS, privacy, regulatory, and security which are stipulated and must be satisfied by the involved service providers. These patterns are described in the following. Notably, this paper covers the well-understood SLA violation patterns only.

- **QoS Constraint Violation Patterns.** Quality constraints violation is denoted by 'Γ_Q^C'. In relation to QoS[4], the following violation patterns are observed.
  - *Reliability Violation Pattern.* $\forall CSBA,\ \exists\Gamma_Q^C$ *if* $(Availability(X^S)) \rightarrow \neg satisfied$, where X^S denotes any cloud service. If availability of a service is not at *agreed level* then quality constraint is violated and the service is not reliable any longer.
  - *Throughput Violation.* $\forall(CSBA),\ \exists\Gamma_Q^C$ *if* $(Throughput(X^S)) \rightarrow \neg satisfied$. In this pattern, throughput of infrastructure, platform, and software services are not met by providers.
  - *Response Time Violation.* $\forall(CSBA),\ \exists\Gamma_Q^C$ *if* $(T_R(X^S)) \rightarrow \neg satisfied$. In this pattern, service providers fail to satisfy response time threshold promised to consumers.
  - *Processing Time Violation.* $\forall(CSBA),\ \exists\Gamma_Q^C$ *if* $(T_P(X^S)) \rightarrow \neg satisfied$. In this pattern, providers fail to meet the processing time threshold.
  - *Delay Time Violation.* $\forall(CSBA),\ \exists\Gamma_Q^C$ *if* $(T_d(X^S)) \rightarrow \neg satisfied$. If the maximum delay threshold is not satisfied then it results quality constraint violation.
- **Security Constraint Violation Patterns.** 'Γ_S^C' represents security constraints violation. The security constraint violation patterns are listed below.
  - *Data Durability Constraint Violation.* $\forall(CSBA),\ \exists\Gamma_S^C$ *if* $(D^d(D^s)) \rightarrow \neg satisfied$ where, $D^s \in P^s$. D^s, P^s and D_d^C represent database service, platform service, and data durability respectively. The data durability violation happens in database services. The violation happens if a database service provider fails to persist data of CSBA users. For database service, durability is an important issue.
  - *Breaching Data Confidentiality.* $\forall(CSBA),\ \exists\Gamma_S^C$ *if* $(D_c^C(D^s)) \rightarrow \neg satisfied$, where D_c^C represents data confidentiality. If confidentiality of a data is breached then it is treated as data confidentiality violation.
  - *Breaching Data Integrity.* $\forall(CSBA),\ \exists\Gamma_S^C$ *if* $(D_I^C(D^s)) \rightarrow \neg satisfied$, where, D_I^C denotes data integrity constraint. Data integrity refers to consistent and accurate data. Data integrity violation happens if a database service provider fails to maintain desired level of data consistency and accuracy.
- **Privacy Constraint Violation Patterns.** $\forall(CSBA),\ \exists\Gamma_S^C$ *if* $(P^C(D^s)) \rightarrow \neg satisfied$, where P^C denotes the privacy constraint and 'Γ_P^C' privacy constraints. In this violation, the providers fails to isolate client's data.
- **Regulatory Policy Violation Patterns.** 'Γ_R^P' denotes the violation of regulatory policies related to cloud services. The list below shows the patterns of the regulatory policy violations.
  - *Data Retention Policy Violation.* $\forall(CSBA),\ \exists\Gamma_R^P$ *if* $(D_R^P(D^s)) \rightarrow \neg satisfied$, where '$D_R^P$' represents data retention policy. In this violation, providers fail to comply with the data retention policies.
  - *Data Deletion Policy Violation.* $\forall(CSBA),\ \exists\Gamma_R^P$ *if* $(D_D^P(D^s)) \rightarrow \neg satisfied$, where '$D_D^P$' is data deletion policy. If a data service provider fails to comply with the data deletion policies then it results violation.

[1] http://aws.amazon.com/

[2] https://developers.google.com/appengine/

[3] http://www.salesforce.com/eu/?ir=1

[4] Most of the QoS constraints listed in this paper have adapted from (DeveloperWorks, 2010)

4 PAEAN4CLOUD FRAMEWORK

In this section, the PAEAN4CLOUD framework is described. This architecture of PAEAN4CLOUD framework is shown in Fig. 4.

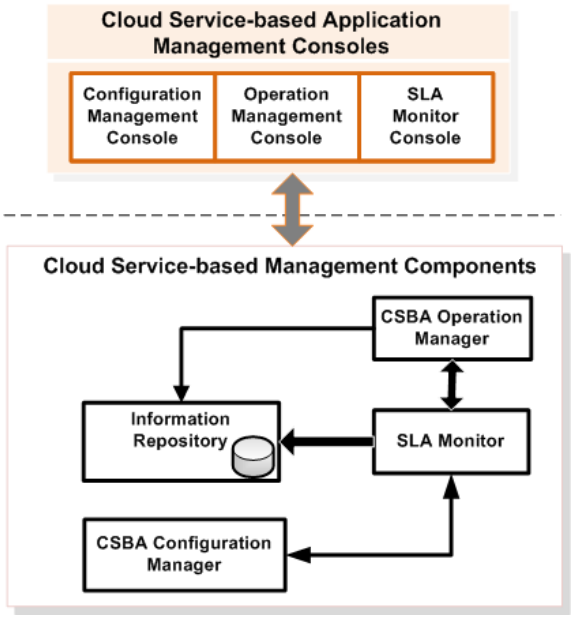


Figure 4: The Architecture of PAEAN4CLOUD.

The architecture comprises the front-end management console and Back-end management components. The console assists end-users in managing and monitoring SLA violations and the back-end management components automate the management and monitoring tasks.

The console contains four *buttons* that cater for performing management tasks. The management console relies on SOAP/RPC protocol to communicate with the back-end components that essentially perform management tasks.

The back-end consists of CSBA configuration manager, CSBA operation manager, SLA monitor, SLA violation manager, and a repository. These components are described briefly in the following,

- *CSBA Configuration Manager.* This component is used for (re-)configuring the cloud services that are incorporated in CSBAs. The configuration manager contains blueprints of CSBAs. An user carries out the modification (if necessary) and the configuration manager that in essence is a component automates these changes. The special purpose of this component is reconfiguring CSBA *on the fly* at runtime for preventing SLA violations.
- *CSBA Operation Manager.* CSBA operation manager is an application management component that manages operations performed at runtime by CSBAs.
- *Information Repository.* While an instance is running, several operations are performed. The operation logs are archived in information repository so that the history of the operations can be used by other components. The repository stores monitring information as well.
- *SLA Monitor.* This component observes the running instances. The roles of the SLA monitor are as follows:
  - watching operation cycles
  - storing monitoring information in the repository
  - detecting violations of SLA
  - reporting violations to the SLA violation manager

  In this paper, we develop an algorithm (presented in 5) for detecting the SLA violations. The SLA monitor relies on this algorithm. The proposed algorithm is designed using the SLA violation patterns defined in section 3.
- *SLA violation manager*: The primary role of the SLA violation manager is preventing violations. The violation manager communicates with SLA monitor that detects violation and sends report to violation manager that acts to prevent the violation. In addition, it communicates with repository and fetch necessary information. A component called *performance analyzer* is integrated with SLA violation manager. The performance analyzer carries out analysis on the history of the instances, predicts the potential violations. Based on this prediction, the violation manager takes necessary actions in particular, reconfigure CSBAs to resist the potential violations proactively. The next section describes reconfiguration of CSBA.

5 DETECTING SLA VIOLATION AND RECONFIGURING CSBA

The SLA monitor of PAEAN4CLOUD relies on the algorithms proposed in this paper for automated reasoning with CSBAs. These algorithms are presented and discussed in this section. Additionally, the reconfiguration of CSBA is discussed here.

5.1 Detecting SLA Violation

The violation detection algorithms are the key contributions of this paper. Algorithm 5.1 and 5.2 are designed for automated reasoning with CSBA to detect SLA violations. It is worth noting that privacy and regulatory policy violations out of the scope.

In this section, the detection of violations is discussed using the TaxiOrdering-CSBA scenario (described in section 2). The TaxiOrdering-CSBA is configured by integrating a business process service, software services, platform services, and infrastructure service that are provided by different vendors. The configuration of CSBA is discussed in Section 2.1.

The central focus of TaxiOrdering-CSBA scenario is the SLA between the client *TaxiTilburg* and the cloud service providers include BestHosting, TelecomProvider A and B, Auto Inc., and MapProvider A and B. As TaxiTilburg is yet another service provider for the end-users, the quality of the services promised by the service providers is critical to TaxiTilburg. Besides, the data security is always crucial for the client TaxiTilburg.

Upon receiving a request placed by the customer 'X', a process instance is created. For this instance, the process execution starts with activity *Allocate Taxi* (S_2). Then, the SLA monitor is called and software services are invoked if they are available. If a required software service is not *available*, the monitor detects the unavailability and reports it to the SLA manager.

For S_2, the *expected values* for the QoS parameters *responsetime* is *6 seconds* and *expected throughput* is 100 requests per second. However, the *actual value* that is taken for response time is *8 seconds* which is greater than the expected value. SLA monitor detects this violation of quality constraints and reports it to SLA Manager (See section 4 for detailed description of the SLA manager). SLA manager prevents the violation of *response time* of *Taxi Ordering Process*.

The *response time* violation of *Taxi Ordering Process* results in end-client dissatisfaction that influences the customer in choosing another taxi service provider. In order to avoid such consequence, the SLA manager acts proactively based on history of completed activities.

Furthermore, for each process instance that is completed successfully, the data is saved in the database permanently. The DatabaseProviderA grantees data security in particular, the durability of the data and consistency of the database to TaxiTilburg. The SLA monitor can detect if data is lost or the database is inconsistent. Algorithm 5.2 is used for automated reasoning with TaxiOrdering-CSBA. After data of an end-client (the client who ordered for a taxi) is stored in the TaxiOrdering database, the application manager of TaxiOrdering-CSBA access the database remotely for retrieving detail of a customer. The application manager executes query and retrieve the detail of the client. If complete detail is not found in the database then the SLA monitor reports this as a violation of security constraint, to the SLA manager.

Algorithm 5.1: Quality Constraint Violation Detection (*CSBA*).

Input:
i. X^s that includes the cloud services such that,
$(B^s_P, S^s, P^s, I^s) \sqsubseteq X^s$
ii. QoS parameters P^{QoS} contained in SLA.
Output: Detect and Report the violation of the QoS constraints
Global Variables:
$\lambda \leftarrow$ cloud service based application(CSBA)
$X^s \leftarrow$ cloud services, $B^s_P \leftarrow$ business process services
$S^s \leftarrow$ software services, $P^s \leftarrow$ platform services
$I^s \leftarrow$ infrastructure services, $Monitor \leftarrow$ SLA monitor
$M^{SLA} \leftarrow$ SLA Manager, $R \leftarrow$ information repository
$\Gamma^C_Q \leftarrow$ quality constraints, $T^R \leftarrow$ response time
$T^P \leftarrow$ processing time, $D^T \leftarrow$ delay time
$Availability \leftarrow$ Service Availability, $A \leftarrow$ Process Activity
$Throughput \leftarrow$ the performance rate of services
$B^P_I \leftarrow$ Process Instance, $EValue \leftarrow$ Expected Value
$AValue \leftarrow$ Actual Value, $ERate \leftarrow$ Expected Throughput
$ARate \leftarrow$ Actual Throughput, $H \leftarrow$ Processing History
procedure (*Initialize*)
- **for each** inbound request 'R_i' to λ
- **if** ($\lambda \neq \varnothing$
  - **then** {*Call Monitor* **and** *invoke* B^S_P;
  - **else** *Cancel* 'R_i';
- **if** $B^s_P \neq \varnothing$
  - **then**
    - **if** (B^s_P.*Availability* = "Yes")
      - **then** *Instantiate* B^I_P
    - (**Comment**: a process instance B^I_P is created in this step).
      - **else** *Report* Γ^C_Q to M^{SLA} **and** *Store* H in R;
  - **else** *Cancel* 'R_i' **and** *Terminate* 'B^P_i';
- **for each** B^I_P
- *execute* ($A_i \in B^s_P$);
- **for each** A
- (**Comment:** Assume that 'A' is an operational activity
- **if** $A \neq \varnothing$
  - **then** {**invoke** (S^s **and** P^s **and** $I^s \sqsubseteq X^s$);
  - **else** *Cancel* 'R_i' **and** *Terminate* 'B^P_i';
- **if** ((S^s. *Availability* = "yes") **and**
- (P^s. *Availability* = "yes")
- **and** (I^s. *Availability* = "yes"))
  - **then**
    - (*Compute AValue* (T^R, D^T for X^s)
    - **and** *ARate* (*Throughput* for X^s)
    - **and** T^P for A);
  - **else**
    - **Report** Γ^C_Q to M^{SLA} **and**
    - *Store* H in R;
- **if** (*AValue* (T^R, D^T) > *EValue* (T^R, D^T)&&
- *ARate* (*Throughput*) > *ERate* (*Throughput*))&&
- *AValue*(T^P) > *AValue*(T^P) for A)
  - **then**
    - **Report** Γ^C_Q to M^{SLA} **and**
    - *Store* H in R;
  - **else** {**Execute** Next A;
- **for all** $A_i \sqsubseteq B^s_P$
- **repeat**
- *Check* SLA Violation
- **until** Complete the execution of B^s_P;

Additionally, the application manager checks consistency of the TaxiOrdering database either period-

Algorithm 5.2: (SecurityConstraintViolationDetection).

Input:
i. $S^s, D^s \in P^s$
ii. Security parameters '$P^{Security}$' contained in SLA.
Output: Detect and Report the violation of security constraints
Global Variables:
$S^s \leftarrow$ Software Services
$P^s \leftarrow$ Platform Services
$D^s \leftarrow$ Database Services
$Monitor \leftarrow$ SLA Monitor
$M^{SLA} \leftarrow$ SLA Manager
procedure $(Initialize)$
/ ∗ *Durability Violation Detection* ∗ /
for each *Query* to D^s
if $(D^s \neq \varnothing)$
then *Call Monitor* **and** *Run* SELECT D_i FROM D^s;
else *Cancel Query*;
if *Query* $(D^s) \rightarrow D^i$ Not Found
then *Report* Γ_S^C to M^{SLA} **and** *Store H* in *R*;
/ ∗ *Consistency Violation Detection* ∗ /
for each Database Checking '*ChkDB*' to D^s
if $(D^s \neq \varnothing)$
then *Call Monitor* **and** *Run* DBCC CHECKDB
/* CHECKDB command includes CHECKALLOC CHECKTABLE, and CHECKCATALOG commands */
else *Cancel ChkDB*;
if *ChkDB* $(D^s) \rightarrow \neg$ *consistent* D^s
then *Report* Γ_S^C to M^{SLA} **and** *Store H* in *R*;

ically or when a database transaction is happened. For any consistency checking request placed by the application manager, SLA monitor is called. The application manager access databases and run query such as, an SQL statement 'DBCC CHECKDB' on TaxiOrdering database for checking consistency. If SLA monitor detects inconsistency in TaxiOrdering database, then it reports a violation to SLA manager. It is worth noting that in this example only SQL commands are used for query database and checking consistency because the TaxiOrdering is a mySQL database. The proposed solution supports executing other commands as well since it supports integrating other database services. In essence, the query statements and consistency checking depends on the database service integrated with CSBAs.

The SLA manager calls the configuration manager - depending on the violation pattern - for reconfiguring the CSBAs. In the next section, the strategies for reconfiguring CSBAs are discussed.

5.2 Reconfiguring the Service-based Cloud Application

After being triggered by the SLA Manager, the Configuration and Deployment Manager is responsible for making an informed decision on the preventive reconfiguration strategies and actions for the BPaaS based on the violation patterns that have been detected in the previous BPaaS instances. The aim is to enable future BPaaS instances to meet the expected SLA with their clients. Figure 5 illustrates the decision tree inside the Configuration and Deployment Manager, in which each SLA Violation pattern detected in the previous section 5 is tackled by specific reconfiguration strategies, which lead to specific reconfiguration actions. In the following, each strategy and action is explained in detail:

- RS1- Find alternative SaaS: This strategy can be used when a constituent SaaS of a BPaaS is not available for invocation (*Availability Constraint Violation*) or its QoS (response time and throughput) performance is decreasing. The idea is to find an alternative substitution for this SaaS. This strategy leads to the reconfiguration action *RA1- Dynamic SaaS Binding*, which dynamically binds all the upcoming BPaaS instances to the new SaaS.
- RS2- Scale up SaaS: This strategy can be used when the response time constraint of a constituent SaaS is violated (*ResponseTime Constraint Violation*). The idea is to prevent this violation for the future BPaaS instances by resizing the infrastructure resources of the running SaaS instances, including resizing and rebooting their Virtual Machines with more computing resources (*RA2-Resizing and Rebooting VM*) and reconfiguring their network links with less latency (*RA3-Reconfiguring Network Link*).
- RS3- Scale out SaaS: This strategy can be used when the throughput constraint of a constituent SaaS is violated (*Throughput Constraint Violation*). The idea is to prevent this violation for the future BPaaS instances by deploying new instances of this SaaS (*RA4- Deploying new Instances*) and reconfiguring the network link of all the SaaS instances with larger bandwidth (*RA4-Reconfiguring Network Link*).
- RS4- Find alternative data service provider: This strategy can be used when a violation in data security and confidentiality is detected by the SLA Manager (*Data Security Constraint Violation*). The idea is to switch to a new Data service provider that provides better security mechanism and guarantee. This strategy leads to the

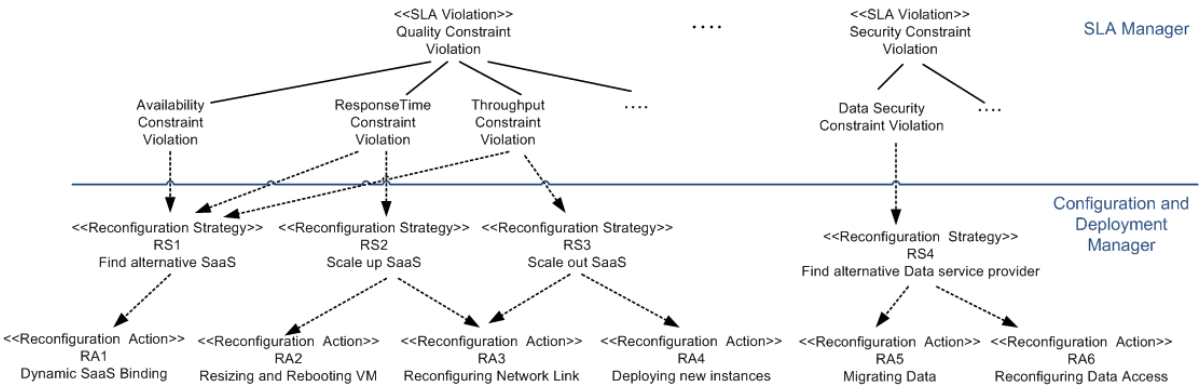


Figure 5: Reconfiguration Strategies and Actions for each SLA Violation Pattern.

process of migrating data from the old to the new provider (*RA5- Migrating Data*), and reconfiguring the Data Access mechanism (*RA6- Migrating Data*)

In case several violation patterns are detected, multiple reconfiguration strategies could be combined to perform preventive actions. For instance, in case both the response time and throughput constraint of a SaaS are violated, one could follow both RS2 and RS3 to resize the current SaaS instances with more resource and to deploy new instances of the SaaS.

In our running example in section 2, the TaxiOrdering-BPaaS is configured using the blueprint approach that results into a blueprint configuration in Figure 2. Based on the blueprint configuration, a deployment configuration has been developed in Figure 3, which serves as a cookbook for cloud administrator to actually deploy and maintain the TaxiOrdering-BPaaS, so that all the SLA constraints between the TaxiOrdering-BPaaS with its clients can be satisfied. Nevertheless, at peak time period (e.g. new year's eve), the number of instances of the TaxiOrdering-BPaaS may increase dramatically and hence lead to several SLA violations. The previous Section 5 points out the four sample violation cases that may occur to the TaxiOrdering-BPaaS.

- V1: Response time of the step (S2-Allocate Taxi) increases, i.e. the *actual response time* is 8 s whilst the *expected response time* is only 6s. This is due to the response time constraint violation of the FleetMgt-SaaS.
- V2: Throughput of the step (S2-Allocate Taxi) decreases, i.e. the *actual throughput* is only 8 req/s whilst the *expected throughput* is 10 req/s. This is due to the throughput constraint violation of the FleetMgt-SaaS.
- V3: Step (S3-Calculate Route) cannot be executed. This is due to the availability constraint violation of the MapA-SaaS.
- V4: Data security is violated. This is due to the data security constraint violation of the MySQL-PaaS.

With help of the reconfiguration decision tree, the Configuration and Deployment Manager is able to perform preventive actions for the future instances of the TaxiOrdering-BPaaS. Table 2 explains which reconfiguration strategy and action are performed for which violation cases in our running example. Please note that the details of how to perform a reconfiguration action strongly depends on the blueprint configuration of the TaxiOrdering-BPaaS introduced in Section 2.1 that provides different alternative configurations. In this paper, we do not discuss the approaches of selecting a particular configuration.

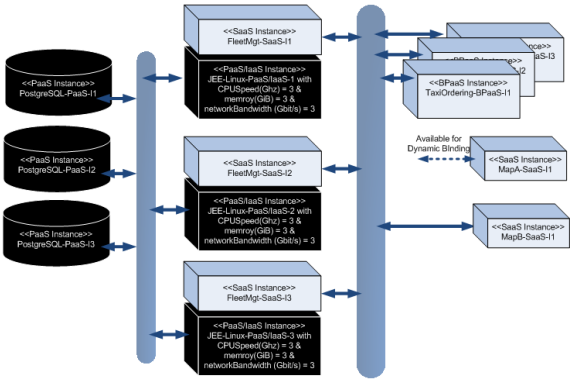


Figure 6: The deployment architecture of the reconifgured TaxiOrdering-BPaaS.

Performing the reconfiguration actions will result in a different deployment configuration of the TaxiOrdering-BPaaS. Figure 6 illustrates this adapted deployment configuration of the TaxiOrdering-BPaaS

- 3 instances of the FleetMMapB-SaaS is deployed on 3 instances of the Linux-JEE-PaaS/IaaS, each

Table 2: Reconfiguration Strategies and Actions for the sample TaxiOrdering-BPaaS.

Cloud Service	SLA Violation	Reconfiguration Strategy & Action	Details
FleetMgt-SaaS	V1: Response Time Constraint	RS2 $\Longrightarrow$ RA2	Resizing the VM with CPUSpeed(Ghz)=3, memory(GiB)=4.
FleetMgt-SaaS	V2: Throughput Constraint	RS3 $\Longrightarrow$ RA3 & RA4	Deploying another instance of the stack FleetMgt-SaaS + JEE-Linux-PaaS/IaaS. Reconfiguring the network link of all three JEE-Linux-PaaS/IaaS instances with networkBandwidth(Gbit/s) = 3
MapA-SaaS	V3: Availability Constraint	RS1 $\Longrightarrow$ RA1	Dynamically rebind to the new MapB-SaaS instance.
MySQL-PaaS	V4: Data Security Constraint	RS4 $\Longrightarrow$ RA5 & RA6	Migrate all the data of the FleetMgt-SaaS to 2 new instances of PostgreSQL-PaaS. reconfiguring the data access layer of the FleetMgt-SaaS.

with 3Ghz CPU speed, 3 GiB memeory and 3 Gbit network link

- 3 instances of the PostgreSQL databases are used by the FleetMgt-SaaS instances to store data.
- A MapB-SaaS instance used to performed the "Step 3- Calculate Route", instead of the MapA-SaaS instance.

6 RELATED WORKS

This section describes the related works in the area of monitoring and management the SLA violation of the CSBA. A number of remarkable works related to monitoring and management SLAs of the service oriented and Grid applications have been presented in many literature *e.g.* (Comuzzi et al., 2009), (Fu and Huang, 2006), (Koller and Schubert, 2007), (Boniface et al., 2007), (Wood et al., 2009). However, a very little works are found in the area of monitoring and managing SLAs of the cloud service based system. From the conceptual point of view, monitoring SLA in the area of cloud, grid and conventional service oriented domains are the same, yet the methods of monitoring SLAs of the systems serving these domains differ. Therefore, since this paper is focused on detecting the violation of the SLAs within cloud environment, the related works focusing on monitoring SLA for detecting violation and reconfiguration of cloud services is discussed in this section.

In (Emeakaroha et al., 2012b), a framework called *DeSVi* is proposed to detect SLA violation in cloud infrastructure. The goal of this framework is similar to PAEAN4CLOUD. The framework is proposed for monitor the resources and component perform casebases reasoning for detecting the violation by utilizing the notion of knowledge database. Additionally, the framework relies on the service level objective. DeSVi contains a application deployment component that facilitates deploying application automatically.

Hammad (Hammadi and Hussain, 2012) proposes framework for monitoring QoS. It comprises two modules include reputation assessment module and transactional risk assessment module. It assists in assessing performance based on QOSs observed and also assists in making decision whether or not a cloud service customer should migrate to another provider.

SageShift (Sukwong et al., 2012) consists of a Virtual Machine(VM) admission control - Sage, and a hypervisor - shift. The key focus of the SageShift is preventing SLA violations. SageShift assess the feasibility of SLA based on the incoming requests and prevent SLA violations by adjusting the required amount of resources dynamically and determining the sequence of VM execution. Unfortunately, the potential SLA violations with respect to software service and platform service cannot be traced as SageShift oversights these service issues.

(Chandrasekar et al., 2012) proposes a QoS monitor that reads data and a offers a solution that is built upon the Markov Chain [5] to establish trust between service consumer and provider.

A holistic approach called OPTIMIS was proposed by (Ferrer et al., 2012) to optimize entire service life cycle including monitoring services. The authors describe various aspects of service monitoring and managing SLAs between service provider and consumer. They stress on a list issues that need to be improved for optimizing the performance of current solutions for monitoring the cloud services. However, the authors do not offer any optimizing method. Essentially, SLA monitoring is rather a small part of the

[5] www.math.rutgers.edu/courses/338/coursenotes/chapter5.pdf

OPTIMIS approach.

Most of the SLA monitoring approaches discussed in the above focus on the QoS aspect of services. Nonetheless, PAEAN4CLOUD is able to detect the QoS as well as security constraint violations in the cloud environment. To the best of our knowledge, existing SLA monitoring solutions do not address the security constraint violations. The SLA monitor of the PAEAN4CLOUD relies on the *category-based violation* detection approach that enhance the capability of the monitoring system for identifying violations efficiently. Additionally, the category-based violation detection approach assists in deciding the most suitable reconfiguration strategy for avoiding the violation of SLAs. None of the SLA monitoring approaches discussed in the above deals with the category-based violation detection of cloud services.

7 CONCLUSION AND FUTURE WORKS

In this paper, we have highlighted the challenges of monitoring and predicting of SLA violations in cloud service based applications. Thus, we motivated emerging requirements for the monitoring, detection, and configuring of SLA violation in a real world scenario. We argued that the specific nature of cloud service based applications using a mixture of SaaS, PaaS and IaaS solutions - possibly from various providers, makes classic approaches incapable of meeting these requirements. Consequently, we proposed a PAEAN4CLOUD, a generic framework monitoring, detecting, and configuring SLA violations. The framework relies on components and algorithms for automated reasoning with CSBA for detecting SLA. Currently we are exploring the idea of root cause analysis for preventing SLA violations in such as way that monitoring can play a pivotal role to predict violations of the QoS requirements as stipulated in the SLA and to suggest (manually or automatically) some adjustment to enforce the running application to comply with the agreed-on SLA. The adjustment could be (1) to change the service infrastructure in such way, that the application will be processed and respect the SLA. Another possibility is to (2) apply self-scaling (up/out) techniques to enable/enforce the application to comply with the SLA of the application as a whole.

REFERENCES

Boniface, M. J., Phillips, S., Perez, A. S.-M., and Surridge, M. (2007). Dynamic service provisioning using gria slas. In *NFPSLA-SOC'07*. Event Dates: 17th September 2007.

Chandrasekar, A., Chandrasekar, K., Mahadevan, M., and Varalakshmi, P. (2012).

Comuzzi, M., Kotsokalis, C., Spanoudakis, G., and Yahyapour, R. (2009). Establishing and monitoring slas in complex service based systems. In *ICWS*, pages 783–790.

DeveloperWorks (2010). Review and summary of cloud service level agreement.

Emeakaroha, V. C., Ferreto, T. C., Netto, M. A. S., Brandic, I., and Rose, C. A. F. D. (2012a). Casvid: Application level monitoring for sla violation detection in clouds. In *COMPSAC*, pages 499–508.

Emeakaroha, V. C., Netto, M. A. S., Calheiros, R. N., Brandic, I., Buyya, R., and Rose, C. A. F. D. (2012b). Towards autonomic detection of sla violations in cloud infrastructures. *Future Generation Comp. Syst.*, 28(7):1017–1029.

European Comission: Information & Communication Technologies Unit (2010). 4CaaSt: Building the PaaS Cloud of the Future. Project Objectives Document.

Ferrer, A. J., Hernndez, F., Tordsson, J., Elmroth, E., Ali-Eldin, A., Zsigri, C., Sirvent, R., Guitart, J., Badia, R. M., Djemame, K., Ziegler, W., Dimitrakos, T., Nair, S. K., Kousiouris, G., Konstanteli, K., Varvarigou, T., Hudzia, B., Kipp, A., Wesner, S., Corrales, M., Forg, N., Sharif, T., and Sheridan, C. (2012). Optimis: A holistic approach to cloud service provisioning. *Future Generation Computer Systems*, 28(1):66 – 77.

Fu, W. and Huang, Q. (2006). Grideye: A service-oriented grid monitoring system with improved forecasting algorithm. In *GCC Workshops*, pages 5–12.

Hammadi, A. M. and Hussain, O. (2012). A framework for sla assurance in cloud computing. In *AINA Workshops*, pages 393–398.

Koller, B. and Schubert, L. (2007). Towards autonomous sla management using a proxy-like approach. *Multiagent and Grid Systems*, 3(3):313–325.

Nguyen, D. K., Lelli, F., Papazoglou, M. P., and van den Heuvel, W.-J. (2012a). Blueprinting approach in support of cloud computing. *Future Internet*, 4(1):322–346.

Nguyen, D. K., Lelli, F., Papazoglou, M. P., and van den Heuvel, W.-J. (2012b). Issue in automatic combination of cloud services. In *ISPA*, pages 487–493.

Sukwong, O., Sangpetch, A., and Kim, H. S. (2012). Sageshift: Managing slas for highly consolidated cloud. In *INFOCOM*, pages 208–216.

Wood, T., Shenoy, P. J., Venkataramani, A., and Yousif, M. S. (2009). Sandpiper: Black-box and gray-box resource management for virtual machines. *Computer Networks*, 53(17):2923–2938.

Wu, L. and Buyya, R. (2010). Service level agreement (sla) in utility computing systems. *CoRR*, abs/1010.2881.

SHORT PAPERS

Methodology to Determine Relationships between Performance Factors in Hadoop Cloud Computing Applications

Luis Eduardo Bautista Villalpando[1,2], Alain April[1] and Alain Abran[1]
[1]*Department of Software Engineering and Information Technology - ETS, University of Quebec, Montreal, Canada*
[2]*Department of Electronic Systems, Autonomous University of Aguascalientes, Aguascalientes, Mexico*
lebautis@correo.uaa.mx, {alain.april, alain.abran}@etsmtl.ca

Keywords: Cloud Computing, Measurement, Performance, Taguchi Method, ISO 25010, Maintenance, Hadoop Mapreduce.

Abstract: Cloud Computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources. Cloud Computing users prefer not to own physical infrastructure, but instead rent Cloud infrastructure, a Cloud platform or software, from a third-party provider. Sometimes, anomalies and defects affect a part of the Cloud platform, resulting in degradation of the Cloud performance. One of the challenges in identifying the source of such degradation is how to determine the type of relationship that exists between the various performance metrics which affect the quality of the Cloud and more specifically Cloud applications. This work uses the Taguchi method for the design of experiments to propose a methodology for identifying the relationships between the various configuration parameters that affect the quality of Cloud Computing performance in Hadoop environments. This paper is based on a proposed performance measurement framework for Cloud Computing systems, which integrates software quality concepts from ISO 25010 and other international standards.

1 INTRODUCTION

Cloud Computing (CC) is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction (Mell and Grance 2011). Some CC users prefer not to own physical infrastructure, but instead rent Cloud infrastructure, or a Cloud platform or software, from a third-party provider. These infrastructure application options delivered as a service are known as Cloud Services.

Service models for CC are categorized as: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) (ISO/IEC 2011). The model that relates the most to the software engineering community is the SaaS model. Software engineers focus on software components, and customers use an IT provider's applications running on a Cloud infrastructure to process information according to their processing and storage requirements. One of the main characteristics of this type of service is that customers do not manage or control the underlying Cloud infrastructure (including network, servers, operating systems, and storage), except for limited user-specific application configuration settings.

Performance measurement models (PMMo) for CC, and more specifically for Cloud Computing Applications (CCA), should propose a means to identify and quantify "normal application behaviour," which can serve as a baseline for detecting and predicting possible anomalies in the software (i.e. applications in a Cloud environment) that may impact in a Cloud application. To be able to design such PMMo for CCA, methods are needed to collect the necessary base measures specific to performance, and analysis models must be designed to analyze and evaluate the relationships that exist among these measures.

One of the challenges in designing PMMo for CCA is how to determine what type of relationship exists between the various base measures. For example, what is the extent of the relationship between the amount of physical memory used and the amount of information to process by an application? Thus, this work proposes the use of a

methodology based on the Taguchi method to determine how closely the performance parameters (base measures) involved in the performance analysis process are related. The Taguchi method combines industrial and statistical experience, and offers a means for improving the quality of manufactured products. It is based on the "robust design" concept, popularized by Taguchi, according to which a well designed product should cause no problems when used under specified conditions (Taguchi, Chowdhury et al. 2005). Although the experiment presented in this paper was not developed in a CC production system, the main contribution of this work is to propose the Taguchi method as a way to determine relationships between performance parameters of CCA.

This paper is structured as follows. Section 2 presents background of concepts related to the performance measurement of CCA and introduces the MapReduce programming model, which is used to develop CCA. In addition, section 2 presents the PMFCC, which describes the key performance concepts and sub concepts identified from international standards. Section 3 presents the method for examining the relationships among the performance concepts identified in the PMFCC. In this section, an experimental methodology based on the Taguchi method of experimental design, is used and offers a means for improving the quality of product performance. Section 4 presents the results of the experiment and analyzes the relationship between the performance factors of CCA. Finally, section 5 presents a synthesis of the results of this research and suggests future work.

2 BACKGROUND

2.1 Performance Analysis in Cloud Computing Applications

Researchers have studied the performance of CCA from various viewpoints. For example, Jackson (Jackson et al., 2010) analyzes high performance computing applications on the Amazon Web Services cloud, with the objective of examining the performance of existing CC infrastructures and creating a mechanism to quantitatively evaluate them. His work is focused on the performance of Amazon EC2 as a representative example of the current mainstream of commercial CC services, and its potential applicability to Cloud-based environments in scientific computing environments. He quantitatively examines the performance of a set of benchmarks designed to represent a typical High Performance Computing (HPC) workload running on the Amazon EC2 platform. Timing results from different application benchmarks are used to compute a Sustained System Performance (SSP) metric, which is a derived measure for measuring the performance delivered by the workload of a computing system. According to the National Energy Research Scientific Computing Center (NERSC) (Kramer et al., 2005), SSP is useful for evaluating system performance across any time frame, and can be applied to any set of systems, any workload, and/or benchmark suite, and for any time period. In addition, SSP measures time to solution across different application areas, and can be used to evaluate absolute performance and performance relative to cost (in dollars, energy, or other value propositions). In his work, Jackson shows that the SSP metric has a strong correlation between the percentage of time an application spends communicating and its overall performance on EC2. Also highlighted, the more communication there is, the worse the performance became. Jackson concludes that the communication pattern of an application can have a significant impact on performance.

Other researchers focus on applications in virtualized Cloud environments. For instance, Mei (Mei et al., 2010) studies the measurement and analysis of the performance of network I/O applications (network-intensive applications) in these environments. The aim of his research is to understand the performance impact of co-locating applications in a virtualized Cloud, in terms of throughput performance and resource sharing effectiveness. Mei addresses issues related to managing idle instances, which are processes running in an operating system (OS) that are executing idle loops. Results show that when two identical I/O applications are running together, schedulers can approximately guarantee that each has its fair share of CPU slicing, network bandwidth consumption, and resulting throughput. It also shows that the duration of performance degradation experienced is related to machine capacity, workload level in the running domain, and the number of new virtual machine (VM) instances to start up.

Although these publications present interesting methods for performance measurement of CCA, the approaches used were from an infrastructure perspective and did not consider CCA performance factors from a software engineering perspective. This work bases the performance evaluation of CCA on frameworks developed for data intensive

processing i.e. like Hadoop and MapReduce, and by integrating software quality concepts from ISO 25010, as well as frameworks for Cloud Computing Systems (CCS) performance measurement. This approach was taken as a novel way to apply concepts of software engineering to the new paradigm of cloud computing.

2.2 The ISO 5939 Measurement Process Model

The purpose of a measurement process, as described in ISO 15939 (ISO/IEC 2008), is to collect, analyze, and report data relating to the products developed and processes implemented in an organizational unit, both to support effective management of the process and to objectively demonstrate the quality of the products.

ISO 15939 defines four sequential activities in a measurement process: establish and sustain measurement commitment, plan the measurement process, perform the measurement process, and evaluate the measurement. These activities are performed in an iterative cycle that allows for continuous feedback and improvement of the measurement process, as shown in Figure 1.

The first two activities recommended by the ISO 15939 measurement process, which are: 1) establish measurement commitment; and 2) plan the measurement process, were addressed in the work, "Design of a Performance Measurement Framework for Cloud Computing" (PMFCC) (Bautista et al., 2012). In this paper, the bases for the measurement of Cloud Computing concepts that are directly related to performance are defined. The PMFCC identifies terms associated with the quality concept of performance, which have been identified from international standards such as ISO 25010 and those of the European Cooperation on Space Standardization. The PMFCC proposes a combination of base measures to determine the derived measures of a specific concept that contributes to performance analysis.

2.3 Performance Measurement Framework for Cloud Computing

2.3.1 Jain's System Performance Concepts and Sub Concepts

A well known perspective for system performance measurement was proposed by Jain (Jain, 1991), who suggests that a performance study must first define a set of performance criteria (or characteristics) to help carrying out the system measurement process. He notes that system performance is typically measured using three sub concepts, if it is performing a service correctly: 1) responsiveness, 2) productivity, and 3) utilization, and proposes a measurement process for each. In addition, Jain notes that there are several possible outcomes for each service request made to a system, which can be classified in three categories. The system may: 1) perform the service correctly, 2) perform the service incorrectly, or 3) refuse to perform the service altogether. Moreover, he defines three sub concepts associated with each of these possible outcomes which affect system performance: 1) speed, 2) reliability, and 3) availability. Figure 2 presents the possible outcomes of a service request to a system and the sub concepts associated with them.

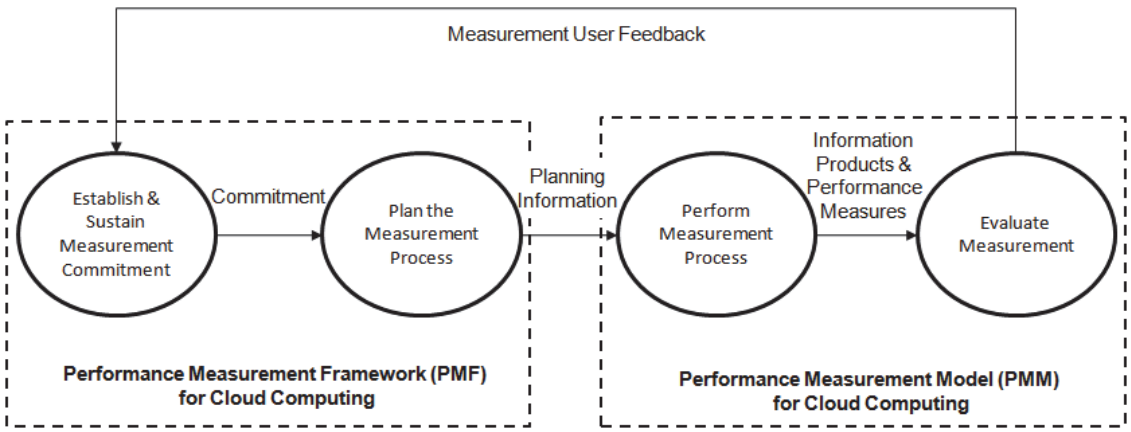


Figure 1: Sequence of activities in a measurement process (adapted from the ISO 5939 measurement process model (ISO/IEC 2008)).

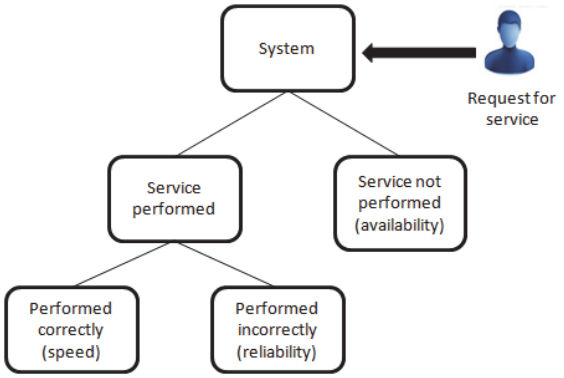


Figure 2: Possible outcomes of a service request to a system, according to Jain.

2.3.2 Definition of Cloud Computing Application Performance

The ISO 25010 (ISO/IEC 2011) standard defines software product and computer system quality from two distinct perspectives: 1) a quality in use model, and 2) a product quality model. The product quality model is applicable to both systems and software. According to ISO 25010, the properties of both determine the quality of the product in a particular context, based on user requirements.

Based on Jain's performance perspectives and the main ISO 25010 product quality characteristics, we propose the following definition of CCA performance measurement:

> *"The performance of a Cloud Computing application is determined by analysis of the characteristics involved in performing an efficient and reliable service that meets requirements under stated conditions and within the maximum limits of the system parameters."*

Although at first sight this definition may seem complex, it only includes the sub concepts necessary to carry out CCA performance analysis.

2.3.3 Definition of the Performance Measurement Framework for Cloud Computing

Performance measurement concepts and sub concepts have previously been related using a proposed relationship model which was described in detail in the PMFCC (Bautista et al., 2012) (see in Figure 3). This model presents the logical sequence, from top to bottom, in which the concepts and sub concepts appear when a performance issue arises in a Cloud Computing System (CCS).

In Figure 3, system performance is determined by two main sub concepts: 1) performance efficiency, and 2) reliability. We have observed that when a CCS receives a service request, there are three possible outcomes (the service is performed correctly, the service is performed incorrectly, or the service cannot be performed). The outcome will determine the sub concepts that will be used for performance measurement. For example, suppose that the CCS performs a service correctly, but, during execution, the service failed and was later reinstated. Although the service was ultimately performed successfully, it is clear that the system availability (part of the reliability sub concept) was compromised, and this affected CCS performance.

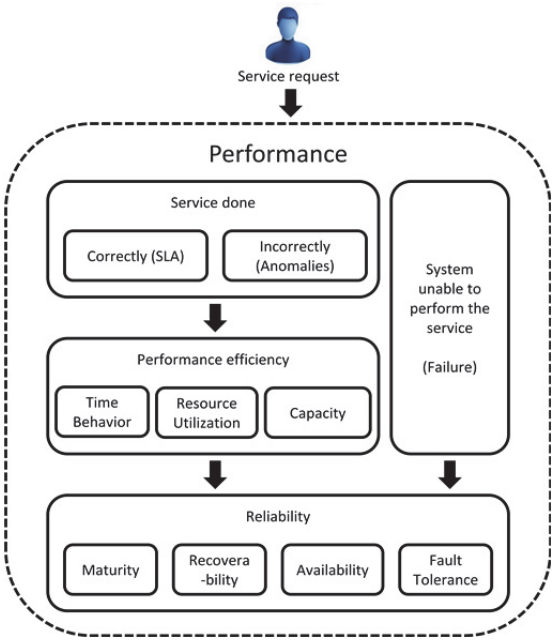


Figure 3: Model of the relationships between performance concepts and sub concepts.

Thus, PMFCC defines the base measures related to the performance concepts that represent the system attributes, and which can be measured to assess whether or not the CCA satisfies the stated requirements. These base measures are grouped into collection functions, which are responsible for conducting the measurement process using a combination of base measures through a data collector. They are associated with the corresponding ISO 25010 quality derived measures, as presented in Table 1.

An example of using the framework is: how can be measured the CC availability concept (presented in Table 1) using the PMFCC? As a first step, it needs three collection functions: 1) the time

function, 2) the task function, and 3) the transmission function. The time function can use several different measurements, such as CPU utilization by the user, job duration, and response time. These base measures can be obtained using a data collector, and then send the measures to a time function that calculates a derived measure of the time concept. An intermediate service will be designed to combine the results of each function in order to calculate a derived measure of the availability that contributes to CC performance, as defined in the framework.

Table 1: Functions associated with Cloud Computing performance concepts.

Base Measures	Collection Functions for Measures	ISO 25010 Derived Measures
Failures avoided Failures detected Failures predicted Failures resolved	Failure function	Maturity Resource utilization Fault tolerance
Breakdowns Faults corrected Faults detected Faults predicted	Fault function	Maturity Fault tolerance
Tasks entered into recovery Tasks executed Tasks passed Tasks restarted Tasks restored Tasks successfully restored	Task function	Availability Capacity Maturity Fault tolerance Resource utilization Time behaviour
Continuous resource utilization time Down time Maximum response time Observation time Operation time Recovery time Repair time Response time Task time Time I/O devices occupied Transmission response time Turnaround time	Time function	Availability Capacity Maturity Recoverability Resource utilization Time behaviour
Transmission errors Transmission capacity Transmission ratio	Transmission function	Availability Capacity Maturity Recoverability Resource utilization Time behaviour

2.3.4 Hadoop Mapreduce

Hadoop is an Apache Software Foundation's project, and encompasses various Hadoop subprojects. The Hadoop project develops and supports the open source software that supplies a framework for the development of highly scalable distributed computing applications designed to handle processing details, leaving developers free to focus on application logic (Hadoop, 2012). MapReduce is a Hadoop subproject which is a programming model with an associated implementation for processing and generating large datasets.

According to Dean (Dean and Ghemawat, 2008), programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. Authors like Lin (Lin and Dyer, 2010) point out that today's issue, which is the need to tackle large amounts of data, is addressed by a divide-and-conquer approach, where the basic idea is to partition a large problem into smaller sub problems. Those sub problems can be processed in parallel by different workers; for example, threads in a processor core, cores in a multi-core processor, multiple processors in a machine, or many machines in a cluster. The intermediate results of each individual worker are combined to yield the final output.

3 METHODOLOGY

3.1 Definition of the Problem

To design the proposed collection functions proposed in the PMFCC (see in Table 1), it is needed to determine how the various base measures are related and to what degree. Studying these relationships enables assess the influence each of them has on the resulting derived measures. The PMFCC shows many of the relationships that exist between the base measures that have a major influence on the collection functions. In CCA, and more specifically in the MapReduce applications, there are over a hundred base measures (including system measures) which could potentially contribute to the analysis of CCA performance. A selection of these measures has to be included in the collection functions so that the respective measures can be derived, and from there an indication of the performance of the applications can be obtained.

There are two key design problems to be solved here: 1) establish which base measures are interrelated, and 2) determine how much the interrelated measures contribute to each of the collection functions.

In traditional statistical methods, thirty or more observations (or data points) are typically needed for

each variable observed, in order to gain meaningful insight. In addition, a few independent variables are needed for the experiments designed to uncover potential relationships among them. These experiments must be performed under certain predetermined and controlled test conditions.

However, this approach is not appropriate here, owing to the large number of variables involved and the time and effort that would be required, which is much more than we have allowed for in this step of the research. Consequently, we have to resort to an analysis method that is better suited to our constraints, specific problem and study area. A possible candidate approach is Taguchi's experimental design method, which investigates how different variables affect the mean and variance of a process performance characteristic helping in determining how well the process is functioning.

This method only requires a limited number of experiments, but is more efficient than a factorial design in its ability to identify relationships and dependencies. The next section describes the method and the concepts to be used.

3.2 Taguchi's Method of Experimental Design

Taguchi's Quality Engineering Handbook (Taguchi et al., 2005) describes the Taguchi method of experimental design, which was developed by Dr. Genichi Taguchi, a researcher at the Electronic Control Laboratory in Japan. This method combines industrial and statistical experience, and offers a means for improving the quality of manufactured products. It is based on the "robust design" concept, popularized by Taguchi, according to which a well designed product should cause no problems when used under specified conditions.

According to Cheikhi (Cheikhi and Abran 2012), Taguchi's two phase quality strategy is the following:

- Phase 1: The online phase, which focuses on the techniques and methods used to control quality during the production of the product.
- Phase 2: The offline phase, which focuses on taking those techniques and methods into account before manufacturing the product, that is, during the design phase, the development phase, etc.

One of the most important activities in the offline phase of the strategy is parameter design. This is where the parameters are determined that make it possible to satisfy the set quality objectives (often called the objective function) through the use of experimental designs under set conditions. If the product does not work properly (does not fulfil the objective function), then the design constants (also called parameters) need to be adjusted so that it will perform better. Cheikhi explains that this activity includes several steps, which are required to determine the parameters that satisfy the quality objectives (output).

According to Taguchi's Quality Engineering Handbook, orthogonal arrays (OA) organizes the parameters affecting the process and the levels at which they should vary. The OA show the various experiments that will need to be conducted in order to verify the effect of the factors studied on the output. Taguchi's method tests pairs of combinations, instead of having to test all possible combinations (as in a factorial experimental design). With this approach, we can determine which factors affect product quality the most in a minimum number of experiments.

Taguchi's OA arrays can be created manually or they can be derived from deterministic algorithms. They are selected by the number of parameters (variables) and the number of levels (states). An OA array is represented by Ln and Pn, where Ln corresponds to the number of experiments to be conducted, and Pn corresponds to the number of parameters to be analyzed. Table 2 presents an example of Taguchi's OA L4, meaning that 4 experiments are conducted to analyze 3 parameters.

Table 2: Taguchi´s Orthogonal Array L4.

No. of Experiments (L)	P1	P2	P3
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

An OA cell contains the factor levels (1 and 2) that determine the types of parameter values for each experiment. Once the experimental design has been determined and the trials have been carried out, the performance characteristic measurements from each trial can be used to analyze the relative effect of the various parameters.

Taguchi´s method is based on the use of the signal-to-noise ratio (SNR), which is a measurement scale that has been used in the communications industry for nearly a century for determining the extent of the relationship between the quality factors in a measurement model. The SNR approach involves the analysis of data for variability, in which an input-to-output relationship is studied in the

measurement system. To determine the effect each parameter has on the output, the SNR (or SN number) is calculated by the formula 1. In this formula y_i is the mean value and s_i is the variance (y_i is the value of the performance characteristic for a given experiment).

$$SN_i = 10 \log \frac{\bar{y}_i^2}{s_i^2} \quad (1)$$

where

$$\bar{y}_i = \frac{1}{N_i} \sum_{u=1}^{N_i} y_{i,u}$$

$$s_i^2 = \frac{1}{N_i - 1} \sum_{u=1}^{N_i} \left(y_{i,u} - \bar{y}_i \right)$$

i=Experiment number
u=Trial number
Ni=Number of trials for experiment i

To minimize the performance characteristic (objective function), the following definition of the SNR should be calculated:

$$SN_i = -10 \log \left(\sum_{u=1}^{Ni} \frac{y_u^2}{N_i} \right) \quad (2)$$

To maximize the performance characteristic (objective function), the following definition of the SNR should be calculated:

$$SN_i = -10 \log \left(\frac{1}{N_i} \sum_{u=1}^{N_i} \frac{1}{y_u^2} \right) \quad (3)$$

Once the SNR values have been calculated for each factor and level, they are tabulated as shown in Table 3, and then the range R (R = high SN - low SN) of the SNR for each parameter also is calculated and entered into Table 3.

Table 3: Rank for SNR values.

Level	P1	P2	P3
1	$SN_{1,1}$	$SN_{2,1}$	$SN_{3,1}$
2	$SN_{1,2}$	$SN_{2,2}$	$SN_{3,2}$
3	$SN_{1,3}$	$SN_{2,3}$	$SN_{3,3}$
Range	R_{P1}	R_{P2}	R_{P3}
Rank	---	---	---

According to Taguchi's method, the larger the R value for a parameter, the greater its effect on the process.

3.3 Experiment

3.3.1 Experimental Setup

All the experiments were conducted on a DELL Studio Workstation XPS 9100 with an Intel Core i7 12-core X980 processor running at 3.3 GHz, 24 GB DDR3 RAM, a Seagate 1.5 TB 7200 RPM SATA 3Gb/s disk, and a 1 Gbps network connection. We used a Linux CentOS 5.8 64-bit distribution and Xen 3.3 as the hypervisor. This physical machine hosts five virtual machines (VM), each with a dual-core Intel i7 configuration, 4 GB RAM, 10 GB virtual storage, and a virtual network interface type. In addition, each VM executes the Apache Hadoop distribution version 0.22.0, which includes the Hadoop Distributed File System (HDFS) and MapReduce framework libraries. One of these VM is the master node, which executes NameNode (HDFS) and JobTracker (MapReduce), and the rest of the VM are slave nodes running DataNodes (HDFS) and JobTrackers (MapReduce).

The Apache Hadoop distribution includes a set of applications for testing the performance of a cluster. According to Hadoop (Hadoop 2012), these applications can test various cluster characteristics, such as network transfer, storage reliability, cluster availability, etc. Four applications were selected to obtain performance measures from the Hadoop cluster as for example; the amount of physical memory used by a Job is a measure that varies according to values given to configuration parameters, such as the number of files to process, the amount of information to process, etc. The viewpoint taken for the selection of the above applications is that it is possible to use the same type's o parameters to configure each application as well as cluster machine.

Below is a brief description of the applications used in the experiments:

1. TestDFSIO. This is a MapReduce application that reads and writes the HDFS test. It executes tasks to test the HDFS to discover performance bottlenecks in the network, to test the hardware, the OS, and the Hadoop setup of the cluster machines (particularly the NameNode and the DataNodes), and to determine the speed of the cluster in terms of I/O.
2. TeraSort. The goal of this application is to sort large amounts of data as fast as possible. It is a benchmark application that combines HDFS testing, as well as testing of the MapReduce layers of a Hadoop cluster.
3. MapRedReliability. This program tests the

reliability of the MapReduce framework by injecting faults/failures into the Map and Reduce stages.

4. MapRedTest. This application loops a small job a number of times, placing the focus on the MapReduce layer and its impact on the HDFS layer.

To develop the set of experiments, three parameters were selected, which can be set with different values for each type of application. These parameters are: 1) the number of files to process, 2) the total number of bytes to process, and 3) the number of tasks to execute in the cluster. Also, a number of different MapReduce base measures such as *Job Duration, Job Status, Amount of Amount of physical memory used, etc.* were selected as possible quality objectives (objective function). These base measures are related to one or more of the performance derived measures identified in the PMFCC.

3.3.2 Definition of Factors and Quality Objective

In a virtualized Cloud environment, Cloud providers implement clustering by slicing each physical machine into multiple virtual machines (VM) interconnected through virtual interfaces. So, we established a virtual cluster with the features mentioned above, in order to obtain representative results.

Fifty experiments were performed to test the Hadoop virtual cluster, varying the three parameters mentioned previously. In each experiment, four different applications were executed, and performance results were recorded for their analysis.

In this way, the set of experiments investigates the effect of the following variables (or control factors, according to the Taguchi terminology) on the output dependent variable:

- Number of files to be processed by the cluster
- Total number of bytes to be processed by the cluster
- Number of tasks into which to divide the Job application

According to Taguchi, quality is often referred to as conformance to the operating specifications of a system. To him, the quality objective (or dependent variable) determines the ideal function of the output that the system should show. In our experiment, the observed dependent variable is the following:

- *Amount of physical memory used by the Job (Mbytes)*

3.3.3 Experiment Development

According to the Hadoop documentation, the number of files and the amount of data to be processed by a Hadoop cluster will be determined by the number of processors (cores) available and their storage capacity. Also, the number of tasks to be processed by the cluster will be determined by the total of number of processing units (cores) in the cluster. Based on the above premises and the configuration of the experimental cluster, we have chosen two levels for each parameter in the experiment. We determine the different levels of each factor in the following way:

- Number of files to process:
  - Small set of files: fewer than 10,000 files for level 1;
  - Large set of files: 10,000 files or more for level 2.
- Number of bytes to process, as determined by the storage capacity of the cluster:
  - fewer than 10,000 Mbytes to process for level 1 (a small amount of data to process);
  - 10,000 or more Mbytes to process for level 2 (a large amount of data to process).
- Number of tasks to create, determined, according to the MapReduce framework, by the number of processing units (cores) in the cluster and the number of input files to process. Since the cluster has a total of 10 cores, we decided to perform tests with:
  - fewer than 10 tasks for level 1;
  - 10 or more tasks for level 2.

Table 4 Presents a summary of the factors, levels, and values for this experiment.

Table 4: Factors and Levels.

Factor Number	Factor Name	Level 1	Level 2
1	Number of files to process	< 10,000	≥10,000
2	Number of MB to process	< 10,000	≥10,000
3	Number of tasks to create	< 10	≥10

Using Taguchi's experimental design method, the selection of the appropriate OA is determined by the number of factors and levels to be examined. The resulting OA array for this case study is L4 (presented in Table 2). The assignment of the various factors and values of this OA array is shown in Table 5.

Table 5: Matrix of Experiments.

No. of the Experiment (L)	Number of Files	Number of Bytes (MB)	Number of Tasks
1	< 10,000	< 10,000	< 10
2	< 10,000	≥ 10,000	≥ 10
3	≥ 10,000	< 10,000	≥ 10
4	≥ 10,000	≥ 10,000	< 10

Table 5 shows the set of experiments to be carried out with different values for each parameter selected. For example, experiment 2 involves fewer than 10,000 files, the number of bytes to be processed is greater than or equal to 10,000 Mbytes, and the number of tasks is greater than or equal to 10.

A total of 50 experiments were carried out by varying the parameter values. However, only 12 experiments met the requirements presented in Table 5. This set of 12 experiments was divided into three groups of four experiments each (called trials). The values and results of each experiment are presented in Table 6.

Taguchi's method defined the SNR used to measure robustness, which is the transformed form of the performance quality characteristic (output value) used to analyze the results. Since the objective of this experiment is to minimize the quality characteristic of the output (amount of physical memory used per Job), the SNR for the quality characteristic "the smaller the better" is given by formula 4, that is:

$$SN_i = -10\log\left(\sum_{u=1}^{Ni}\frac{y_u^2}{N_i}\right) \quad (4)$$

The SNR result for each experiment is shown in Table 7.

According to Taguchi's method, the factor effect is equal to the difference between the highest average SNR and the lowest average SNR for each factor. This means that the larger the factor effect for a parameter, the larger the effect the variable has on the process, or, in other words, the more significant the effect of the factor. Table 8 shows the factor effect for each variable studied in the experiment.

Table 6: Trials, experiments, and resulting values.

Trial	Experiment	Number of Files	Mbytes to Process	Num. of Tasks	Physical Memory (Mbytes)
1	1	10	3	1	185.91
1	2	10	10,000	10	270.65
1	3	10,000	450	10	1589.26
1	4	10,000	10,000	2	105.77
2	1	100	33	2	761.18
2	2	100	10,00	100	605.77
2	3	96,000	29	42	3259.75
2	4	10,000,000	10,000,000	4	100.95
3	1	100	300	1	242.75
3	2	1,000	10,000	1,000	900.95
3	3	1,000,000	3,300	10	770.65
3	4	10,000,000	50,000	2	1112.16

Table 7: SNR results.

Experiment	Number of Files	Mbytes to Process	Number of Tasks	Physical Memory Trial 1	Physical Memory Trial 2	Physical Memory Trial 3	SNR
1	< 10,000	< 10,000	< 10	185.91	761.18	242.75	0.0906
2	< 10,000	≥ 10,000	≥ 10	270.65	605.77	900.95	0.5046
3	≥10,000	< 10,000	≥ 10	1589.26	3259.75	770.65	0.2665
4	≥10,000	≥10,000	< 10	105.77	100.95	1112.16	-0.6263

Table 8: Factor Effect on the Output Objective

	Number of Files	Mbytes to Process	Number of Tasks
Average SNR at Level 1	0.2976	0.1785	-0.2678
Average SNR at Level 2	-0.1799	-0.4028	0.3855
Factor Effect (difference)	0.4775	0.5813	0.6534
Rank	**3**	**2**	**1**

4 RESULTS

4.1 Analysis and Interpretation of Results

Based on the results presented in Table 8, we can observe that:

- Number of tasks is the factor that has the most influence on the quality objective (physical memory used) of the output observed, at 0.6534.
- Number of Mbytes to process is the second most influential factor, at 0.5813.
- Number of files to process is the least influential factor in this case study, at 0.4775.

Figure 4 presents a graphical representation of the factor results and their levels.

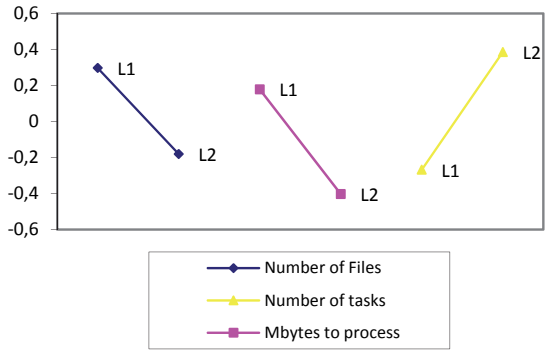


Figure 4: Graphical representation of factors and their SNR levels.

To represent the optimal condition of the levels, also called the optimal solution of the levels, an analysis of SNR values is necessary in this experiment. Whether the aim is to minimize or maximize the quality characteristic (physical memory used), it is always necessary to maximize the SNR parameter values. Consequently, the optimum level of a specific factor will be the highest value of its SNR. It can be seen that the optimum level for each factor is represented by the highest point in the graph (as presented in Figure 4); that is, L1, L1, and L2 respectively.

Using the findings presented in Tables 7 and 8 and in Figure 4, we can conclude that the optimum levels for the factors in this experiment based on the experimental configuration cluster are:

- Number of files to process: The optimum level is fewer than 10,000 files (level 1).
- Total number of Mbytes to process: The optimum level is fewer than 10,000 Mbytes (level 1).
- Number of tasks to be created to divide the Job: The optimum level is greater than or equal to 10 tasks or more per Job (level 2).

4.2 Statistical Data Analysis

The analysis of variance (ANOVA) is a statistical technique usually used in the design and analysis of experiments. According to Trivedi (Trivedi, 2002), the purpose of applying the ANOVA technique to an experimental situation is to compare the effect of several factors applied simultaneously to the response variable (quality characteristic). It allows the effects of the controllable factors to be separated from those of uncontrolled variations. Table 9 presents the results of this analysis of the experimental factors.

As can be seen in the contribution column of Table 9, these results can be interpreted as follows (represented graphically in Figure 5):

- Number of tasks is the factor that has the most influence (43% of the contribution) on the physical memory in this case study.
- Total number of bytes to process is the factor that has the second greatest influence (34% of the contribution) on the processing time.
- Number of files is the factor with the least influence (23% of the contribution) on the processing time in the cluster.

Table 9: Analysis of Variance (ANOVA).

Factors	Degrees of Freedom	Sum of Squares (SS)	Variance (MS)	Contribution (%)	Variance ration (F)
No. of files	1	0.2280	0.2280	23	
Total no. of bytes to process	1	0.3379	0.3379	34	2
No. of tasks	1	0.4268	0.4268	43	3
Error	0	0.0000	0.0000		
Total	3	0.9927			
Error estimate	1	0.2280			

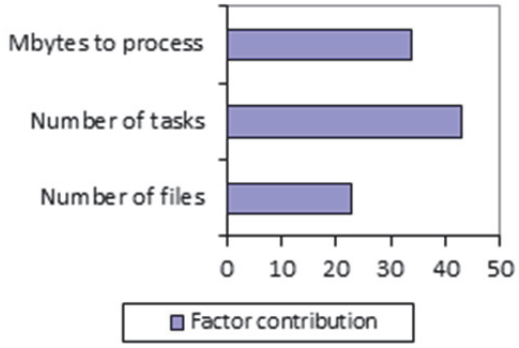


Figure 5: Percentage contribution of factors.

In addition, based on the column related to the variance ratio F shown in Table 9, we can conclude that the following:

- The factors Number of tasks and Number of Mbytes to process have the most dominant effect and the second most dominant effect on the output variable respectively.
- According to Taguchi's method, the factor with the smallest contribution is taken as the error estimate. So, the factor Total number of files to process is taken as the error estimate, since it corresponds to the smallest sum of squares.

The results of this case study show, based on both the graphical and statistical data analyses of the SNR, that the number of tasks into which to divide the Job in a MapReduce application in our cluster has the most influence, followed by the number of bytes to process, and, finally, the number of files.
To summarize, when an application is developed in the MapReduce framework to be executed in this cluster, the factors mentioned above must be taken into account in order to improve the performance of the application, and, more specifically, the output variable, which is the amount of physical memory to be used by a Job.

5 SUMMARY

One of the challenges in CC is to deliver good performance to its end users. In this paper, we present the results of using a method that determines the relationships among the CCA performance parameters. This proposed method is based on a performance measurement framework for Cloud Computing (PMFCC) system, which defines a number of terms that are necessary to measure the performance of CCS using software quality concepts. The PMFCC defined several collection functions which are automated to obtain derived measures and enable analysis of the performance of a CCA. One of the challenges we faced in designing these functions was to decide how to determine the extent to which the base measures are related, and their influence in the analysis of CCA performance. To address this challenge, we proposed the use of Taguchi's method of experimental design.

Using this experimental design method, we carried out experiments to analyze the relationships between the configuration parameters of several Hadoop applications and their performance quality measures based on the amount of physical memory used by a Job. We found that there is a strong relationship between the number of tasks executed by a MapReduce application and the amount of physical memory used by a Job. Our next research activity will be to reproduce this experiment in a production environment, in order to confirm these 'trial group' results with greater certainty. Also, this early research work serves as a basis for a next activity that will need to determine the most important relationships between the performance concepts defined in the PMFCC and enable us to propose a robust model for CCA performance analysis.

Further research is also needed on the design of measurement models and mechanisms to analyze the performance of a real Cloud Computing application, which could also contribute to further validate our proposed method. Such evaluation work would include performance concepts related to software, hardware, and networking. These concepts would be mapped to the collection functions identified in the PMFCC previously developed in order to improve it. We expect that it will be possible, based on this work, to propose a robust model in future research that will be able to analyze Hadoop cluster behavior in a real Cloud Computing environment. This would allow real time detection of anomalies that affect CCS and CCA performance.

REFERENCES

Bautista, L., A. Abran, et al. (2012). *"Design of a Performance Measurement Framework for Cloud Computing."* Journal of Software Engineering and Applications 5(2): 69-75.

Cheikhi, L. and A. Abran (2012). *"Investigation of the Relationships between the Software Quality Models of ISO 9126 Standard: An Empirical Study using the Taguchi Method."* Software Quality Professional Magazine.

Dean, J. and S. Ghemawat (2008). *"MapReduce:*

simplified data processing on large clusters." Communications of the ACM 51(1): 107-113.

Hadoop, A. F. (2012). *"What Is Apache Hadoop?"*, from http://hadoop.apache.org/.

ISO/IEC (2008). *ISO/IEC 15939:2007 Systems and software engineering* — Measurement process. Geneva, Switzerland, International Organization for Standardization.

ISO/IEC (2011). *ISO/IEC 25010: Systems and software engineering – Systems and software product Quality Requirements and Evaluation (SQuaRE) – System and software quality models*. Geneva, Switzerland, International Organization for Standardization: 43.

ISO/IEC (2011). *ISO/IEC JTC 1 SC38:Study Group Report on Cloud Computing*. Geneva, Switzerland, International Organization for Standardization.

Jackson, K. R., L. Ramakrishnan, et al. (2010). *Performance Analysis of High Performance Computing Applications on the Amazon Web Services* Cloud. IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), Washington, DC, USA, IEEE Computer Society.

Jain, R. (1991). *The Art of Computer Systems Performance Analysis*: Techniques for Experimental Design, Measurement, Simulation, and Modeling. New York, NY, John Wiley & Sons - Interscience.

Kramer, W., J. Shalf, et al. (2005). *The NERSC Sustained System Performance (SSP) Metric*. California, USA, Lawrence Berkeley National Laboratory.

Lin, J. and C. Dyer (2010). *Data-Intensive Text Processing with MapReduce*. University of Maryland, College Park, Manuscript of a book in the Morgan & Claypool Synthesis Lectures on Human Language Technologies.

Mei, Y., L. Liu, et al. (2010). *Performance Measurements and Analysis of Network I/O Applications in Virtualized Cloud*. IEEE International Conference on Cloud Computing, CLOUD 2010, Miami, FL, USA, IEEE.

Mell, P. and T. Grance (2011). *The NIST Definition of Cloud Computing*. Gaithersburg, MD, USA, Information Technology Laboratory, National Institute of Standards and Technology: 2-3.

Taguchi, G., S. Chowdhury, et al. (2005). *Taguchi's Quality Engineering Handbook*, John Wiley & Sons, New Jersey.

Trivedi, K. S. (2002). *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. New York, U.S.A., John Wiley & Sons, Inc.

A Cloud-based GWAS Analysis Pipeline for Clinical Researchers

Paul Heinzlreiter[1,2], James Richard Perkins[3], Oscar Torreño[5,1], Johan Karlsson[5,6], Juan Antonio Ranea[5], Andreas Mitterecker[7,1], Miguel Blanca[4] and Oswaldo Trelles[5,1]

[1]*RISC Software GmbH, Softwarepark 35, 4232 Hagenberg, Austria*

[2]*Leibniz Supercomputing Centre (LRZ), Boltzmannstr. 1, 85748 Garching, Germany*

[3]*Research Laboratory, University Hospital-IBIMA, Málaga, Spain*

[4]*Allergy Unit, University Hospital-IBIMA, Málaga, Spain*

[5]*Department of Computer Architecture, University of Málaga, Málaga, Spain*

[6]*Integromics S.L., Avenida de la Innovación 1, 18100 Armilla, Granada, Spain*

[7]*Institute of Bioinformatics, Joh. Kepler University Linz, Linz, Austria*

paul.heinzlreiter@risc-software.at

Keywords: Cloud Computing, Bioinformatics, Biomedicine.

Abstract: The cost of obtaining genome-scale biomedical data continues to drop rapidly, with many hospitals and universities being able to produce large amounts of data. Managing and analysing such ever-growing datasets is becoming a crucial issue. Cloud computing presents a good solution to this problem due to its flexibility in obtaining computational resources. However, it is essential to allow end-users with no experience to take advantage of the cloud computing model of elastic resource provisioning. This paper presents a workflow that allows the end-user to perform the core steps of a genome wide association analysis where raw gene-expression data is quality assessed. A number of steps in this process are computationally intensive and vary greatly depending on the size of the study, from a few samples to a few thousand. Therefore cloud computing provides an ideal solution to this problem by enabling scalability due to elastic resource provisioning. The key contributions of this paper are a real world application of cloud computing addressing a critical problem in biomedicine through parallelization of the appropriate parts of the workflow as well as enabling the end-user to concentrate on data analysis and biological interpretation of results by taking care of the computational aspects.

1 INTRODUCTION

This paper presents *genCloud*, a workflow allowing the end-user to employ various state of the art tools for the analysis of genome wide association studies (GWAS). A workflow describes a series of activities as part of a process. These activities are typically linked and information is transferred from one activity to another. Such links represent dependencies (for example, one activity must occur before another). Currently, running these GWAS applications requires computational infrastructure and knowledge of specific tools, including the command line, scripting, and the R computational language. These tools often require an amount of computational resources greater than what is typically available in a moderate size hospital, which is a typical environment where such an analysis takes place. The workflow described here presents a contribution to the areas of cloud computing and biomedicine allowing end-users with basic computational knowledge, such as clinical scientists, to analyse their own experiments. This is possible because the workflow is made available to end-users through the jORCA software client (Martin-Requena et al., 2010). By presenting a case study of such an analysis, using real life data we apply cloud computing to solve an important biomedical problem in a real-world situation. Our biomedical use case is particularly well suited to cloud computing because it can require very different amounts of computational resources depending on the size of the analysis.

In terms of the underlying cloud infrastructure the work presented in this paper offers several distinguishing characteristics as compared to standard cloud installations:

- User authentication based on grid technology.
- Specific support for transferring big datasets

through grid computing protocols.

- Enactment of specific biomedical and bioinformatics workflows through an user-friendly software client, which significantly lowers the level of computational experience required by the end-users.

1.1 Applying Cloud Computing for Workflows

Cloud computing (Mell and Grance, 2011) specifically addresses requirements of the biomedical workflow solution described here:

- *Dynamic Instantiation of Additional Resources through Elasticity:* The capacity needed for running user applications can easily be increased or decreased either automatically through a scheduling component or following a user request without requiring interaction with the service provider.
- *Flexible Configuration of Workflow Modules without Administrator Intervention:* This approach is specifically viable in a scientific environment, where the users often rely on specific libraries and software packages to perform their research. The domains of biomedicine and bioinformatics – which are addressed within the scope of the work described here – are no exception to this rule: For example the statistical software environment R (R Development Core Team, 2008) is heavily used in these research domains.
- *Storing readily Configured Instances for reuse through Snapshotting Mechanisms:* An infrastructure-as-a-service (IaaS) approach is used giving the user the utmost flexibility by providing access to fully configurable virtual instances. Similarly the instances involved in an analysis workflow can be stored as snapshots and later reused, allowing the user to recreate the analysis setup when data sources are updated (for example for new genome releases), or to allow other users to recreate their original analysis. Such reproducibility of published results is a recurrent problem in biomedical research (Button et al., 2013) (Mobley et al., 2013).

1.2 Related Work

Considerable work has been done in the field of bioinformatics in connecting independent web-services (WS) into higher-level workflows solving more complex problems.

Taverna (Oinn et al., 2004) is a widely used tool in bioinformatics for composing web-services into workflows and enacting them. The stand-alone tool allows users to access various web-service repositories and connecting the services graphically. It is important to note that the focus of Taverna is not specific for cloud computing environments but instead it focuses on web-services which could in turn be deployed on cloud computing environments.

Galaxy (Afgan et al., 2010) is an open, web-based platform for biomedical and bioinformatics research. The tool allows users to create and execute workflows either from the web browser or through the API provided. Users can register and use the system on a public galaxy server. Additionally it is possible to deploy the tool easily on top of commodity hardware or in cloud environments such as Amazon EC2 and Openstack (Le Bras and Chilton, 2013). CloudBioLinux (Krampis et al., 2012) and CloudMan (Afgan et al., 2012) allow the researchers to easily and quickly get access to a functional compute infrastructure. This infrastructure can be configured in a matter of minutes and finalized when is no longer required, using the CloudMan tool. The infrastructure provides access to a set of bioinformatics tools, as specified in the CloudBioLinux virtual machines. After the configuration of the compute infrastructure, the Galaxy workflow engine is deployed. With the galaxy tool deployed and the compute infrastructure configured, the end-users could focus on creating analysis workflows and running them on the infrastructure being configured, rather than focussing on specific details of configuration and deployment.

2 WORKFLOW

This section describes the biomedical workflow and its underlying technologies in detail. Starting from the workflow itself, the focus is being put on the underlying cloud technology being applied during its execution, such as data handling and user authentication. While the specific workflow as described in section 2.1 serves as proof of concept, the underlying cloud, data transfer, and authentication solutions as well as the client software presented in section 2.5 can also be used for other computational workflows and are not limited to the bioinformatics and biomedical domains.

2.1 GWAS Workflow

An overview of the bioinformatics workflow is given in Figure 1. The user begins by uploading CEL files to the cloud data storage. Each CEL file contains the raw data generated from a single nucleotide poly-

morphism (SNP) – an area of variation in the human genome – microarray analysis for a single patient containing details of the genotype. The amount of data involved can range from a few megabytes to potentially terabytes in size, depending on the number of patients included in the study. Once the CEL files have been uploaded, the birdseed algorithm (Korn et al., 2008), which is implemented in the Affymetrix Power Tools package, is executed. It produces a table of genotype calls for each of the different probes on the microarray. Each probe represents a different SNP. Next this table is filtered to remove SNPs that do not vary across different patients, and that could not be called accurately by the birdseed algorithm. This is achieved through the use of a Python script that parses the output of the birdseed algorithm. Once this table has been filtered, an additional python script must be called to convert the birdseed output, which is specific for Affymetrix data, into a variant call format (VCF) file, the standard representation of a genotype for biomedical data. This script must be called separately for each CEL file, which is specific for each patient. Therefore it is an embarrassingly parallel problem and as such very suitable for a data-parallel execution on cloud infrastructures. Finally the user can download the set of VCF files and can then apply standard web-based tools to view the data, or further packages for downstream analysis. Given that this is a standard file format for several types of genomic biomedical data, it is the starting point for many well established further workflows. Several of these workflows are currently under development.

The workflow is using already available software for all of its stages. The specific novelty is given by its execution on a community cloud environment, which has been extended in functionality specifically regarding the data transfer between the stages of the workflow.

It should also be noted that the workflow as presented here is designed to analyse data produced using the Affymetrix SNP chip microarray platform. However it is also possible to adapt the workflow to other types of input data by modifying instances of the *genCloud* cloud images or creating new ones as necessary to implement related workflows processing data from different platforms, as well as sequencing data.

Taken together, these raw data analysis workflows and different downstream analysis workflows will provide a flexible, easy to use service enabling a wide variety of potential analysis workflows, for various genomics technologies.

Although the workflow as shown in Figure 1 only contains a small number of steps, it is computationally quite involved and contains the key steps of an initial GWAS data analysis. Also the different steps have varying computational requirements. Therefore we have chosen to implement this workflow as a first step for testing the different tools available and ensuring its validity for more complex analysis.

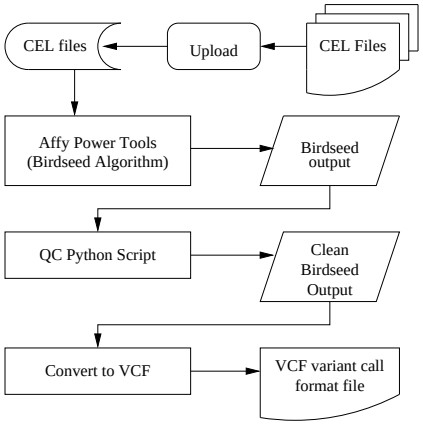


Figure 1: The GWAS workflow.

2.2 Cloud Computing Infrastructure

The cloud computing infrastructure is provided on a small community cloud installation running OpenStack (Pepple, 2011) as cloud middleware. Besides offering IaaS cloud services, as an improvement beyond the state of the art the cloud installation has been extended to support efficient transfer of large data sets through the GridFTP protocol (Allcock et al., 2003). In addition, the data storage is built on top of the distributed file system Ceph (Weil et al., 2006a), which is accessed by OpenStack through the RADOS Gateway (Ceph Team, 2012) implementing the API being used by the OpenStack services. Ceph is made accessible for OpenStack services like object containers and persistent volumes. The object containers can be used as storage for files similar to a directory in a file system. However OpenStack containers cannot be nested. Persistent volumes on the other hand are equivalent to ordinary hard disks and can be dynamically attached to running OpenStack instances.

To glue all of these components together, standard web-service protocols have been used. To enable the transfer of mass data into OpenStack object containers they have been made accessible for the GridFTP data transfer protocol by mounting them into the local file system of a GridFTP server. The same mounting approach is applied for providing access to the containers from inside the running virtual instances.

This underlying infrastructure enables the configuration and execution of arbitrary workflows such as the *genCloud* workflow discussed in this paper.

2.3 Data Storage and Transfer

The Ceph distributed file system (Weil et al., 2006a) is used as the basic data storage system in our cloud setup. Its main advantage is given by the fact that it does not have a single point of failure due to not requiring a specific master node. The storage server is determined by the CRUSH data distribution mechanism (Weil et al., 2006b).

To make the Ceph storage accessible to the OpenStack services, a web-service enabled access mechanism is exposed through the RADOS Gateway, which implements the web-service interfaces defined by the Amazon S3 (Amazon Web Services, 2013) and OpenStack Swift storage services on top of Ceph.

In addition to the web-service based data access mechanisms, we have configured the OpenStack containers to be accessible for the mass data transfer protocol GridFTP. Since a GridFTP server operates on a local file system, the object containers - which are normally accessed through web-service calls - need to be mounted into the local file system of the GridFTP server.

This is achieved by the Cloudfuse daemon, which makes the files within the object container accessible through POSIX file system calls.

The user authentication services represent another crucial part of the setup of the GridFTP server. They are performed using Grid credentials, namely X.509 certificates. These can be either long-lived personal user credentials or short-lived credentials being generated by a service such as MyProxy (Novotny et al., 2011) upon user request.

Our setup of GridFTP works with short-lived credentials, which are generated by MyProxy upon a user request and can be used by the GridFTP service to authenticate itself towards remote servers whilst performing the data transfer requested by the user.

To ensure a reliable transfer of the data between different GridFTP servers, we are relying on the GlobusOnline service (Allen et al., 2011), which manages the data transfers on the user's behalf, as soon as a GridFTP server has been registered as an endpoint of the service.

2.4 User Authentication

The user authentication for the whole OpenStack installation is performed through the Lightweight Directory Access Protocol (LDAP) (Howes and Smith, 1995) thus allowing for an easy and central management of the local credentials for all services. The management of the OpenStack users follows the central grid computing (Foster and Kesselman, 2003) concept of a Virtual Organization (VO) (Foster et al., 2001) which is grouping users from different organizations, who are for example working on a shared project. Within OpenStack the VOs are mapped onto user groups called tenants, thus enabling the participants of one project to easily share resources among themselves.

The VO concept is used to assign the users to specific OpenStack tenants and also to ease the management of access rights, which are assigned at the level of VOs. A good example is given by the access to the local GridFTP server, enabling users to access the service and to use GlobusOnline to transfer data to the local OpenStack containers. All the access rights based on the VOs are represented within the central LDAP tree of the installation, which is subsequently queried by the different services during user authentication.

2.5 Client Software

jORCA (Martin-Requena et al., 2010) is a user-friendly software client for using web-services. The application was specifically designed to help users take advantage of computational resources made available as web-services, i.e. discover web-services, display available parameters, request information and finally execute the web-service. To enable all these activities, jORCA uses metadata repositories (i.e. containers of meta-information), with information about available web-services and data types. The MAPI library (Karlsson and Trelles, 2013) provides the unification of metadata information. By using this library, it is possible to extend the execution functionality of jORCA with components called workers.

2.5.1 Web-service Invocation and Data Transfer

Support for invoking a RESTful WS protocol / frontend from jORCA is also implemented, providing an interface between jORCA and the underlying cloud infrastructure. This interface provides operations to submit new jobs, to cancel previous ones, to poll for status and to retrieve intermediate and final results. To submit a new job the user has to fill the required WS parameters as references to data already uploaded on cloud data storage. Once the frontend receives the job submission it contacts the job scheduler to dispatch the new task. The scheduler will decide depending on the previously mentioned parameters if it needs to create a new instance and also if some idle instances could be deleted. The scheduler will use euca2ools (Debian Wiki, 2012) to perform the described operations.

At the end of the job submission the front-end will return a new unique URL resource to the user which can be cancelled, polled for status, intermediate results, and, when ready, the final result. The intermediate and final results will be data references to the results stored on the data storage. The user can now choose to download the data from the data storage (using the jORCA plugin developed) and/or submit the result data references as input to another WS. This greatly facilitates the invocation of a series of WS as a workflow because the intermediate data is already available on the infrastructure for a subsequent WS.

The potentially large input and output data sizes when invoking web-services deployed on the cloud infrastructure made it essential to use a well-established protocol to transfer large data sets reliably and securely. The Globus Online (GO) initiative (Allen et al., 2011) uses GridFTP to transfer large amounts of data. Additionally, GO provides easy-to-install software for client-side transfer of data and also supports long-running data transfers where GO mediates the data transfer. The actual data is transferred directly between two GridFTP servers being registered as GO endpoints; GO only monitors and controls the transfer.

jORCA can initiate a GO-mediated data transfer either upon user request or automatically if the metadata of the web-service to be invoked indicates that it requires the input data to be transferred via GO prior to invocation. To transfer data from the local computer running jORCA, it is necessary to previously install and configure the Globus Connect software, which is acting as a local GO endpoint. The transfer status – running, finished, or error – is displayed to the end-user through jORCA. Once finished, the necessary information about the uploaded file is transferred to the service which, in turn, will use GO to move the data if necessary.

2.6 Mapping the Workflow onto the Cloud Infrastructure

The biomedical workflow described in 2.1 consists of a sequence of computational steps, which are interconnected by data stored in POSIX-compliant files. The output of one step is stored on the file system, from where it is read by the next module within the workflow.

To enable a flexible execution of the workflow on top of the above-described cloud infrastructure and across multiple cloud instances, the intermediate data files are being stored in OpenStack containers.

The current cloud setup enables the containers to be mounted on the local file system of the running instances. Therefore the existing programs and scripts accessing local files do not need to be adapted to run on the cloud infrastructure. When a user wishes to execute the workflow, the first step is to upload the input data and store it in a container.

Since the initial input data will typically come from an external source, the GO service can be invoked through jORCA to transfer the data to the local cloud installation.

Sequential parts of the workflow consisting of multiple consecutive stages can either be collocated on one instance or distributed across multiple instances depending on their computational load.

The parts of the workflow, which are executed in parallel, are replicated across multiple instances running concurrently.

The instances monitor the availability of their input data: as soon as the corresponding input dataset is available in the specified container, the instance starts to copy the input data from the container to its local hard disk and to subsequently execute its computational algorithm. As soon as a group of collocated stages on one instance is finished, the output is written to a container, enabling the next stage of the workflow running on a different instance to proceed by consuming the newly produced intermediate data in the same way.

3 RESULTS

The workflow described in section 2.1 has been executed on a small community cloud installation using the technologies described in section 2.6.

The user starts by uploading the input data for the workflow, such as the SNP and CEL data, to an OpenStack container using GO through the jORCA client.

To perform the data processing workflow, a simple bash-shell script has been set up which calls the executables performing the computational as well as data management steps within the workflow.

The infrastructure enables the containers to be mounted into the local file system of a running virtual instance, and thus the data access methodology of the original non-cloud-aware workflow modules remains valid.

Initial runtime measurements have shown that the last step of the workflow – converting the Birdseed output into a VCF file – is by far the most time-consuming (see tables 1, 2, and 3). A typical use-case involves processing a large number of CEL-files and the VCF conversion of a single CEL-file takes significantly longer than the rest of workflow combined. Given these facts, it is clear that the parallelization of

the conversion to VCF is key to improving the process.

Therefore the following parallelization approach has been realized:

- All steps before the VCF conversion are executed sequentially on one instance.
- The VCF conversion is parallelized at the file level by assigning the conversion tasks for different files to different machine instances.

The workflow starts with the sequential component retrieving all the input data required for the birdseed algorithm from the OpenStack container and storing it on a local disk within the instance. It is important to note that while the data is directly accessible through the mounted container, the copy operation speeds up the computational step, which only has to access a local disk as compared to web-service calls for accessing the OpenStack container.

After the data has been made available locally the birdseed algorithm and the filtering of its results are performed in sequence before the output of the filtering step is again stored in the container.

Making the filtered data accessible within the container enables the data-parallel execution of the final VCF generation step across different cloud instances.

For the initial prototype all instances are executing the same shell script and accessing the same container. The existence of a specific file marks whether the sequential part of the workflow is still active.

While this is the case the other instances taking part in the parallel VCF generation load the required input data for the last workflow step from the container onto their local hard disks. Thus this data transfer overlaps in time with the computation of the first steps of the workflow.

Since the input data for the VCF generation includes human genome data of significant size, this overlapping reduces the overall processing time of the workflow.

As soon as the previous steps of the workflow have been completed, the responsible instance also loads the input data for the last step. Meanwhile the other instances can already start the computation within the last step of the workflow. As mentioned before, the completion of the previous workflow steps is indicated by the existence of a specific file in the shared container, which can be tested by the script being executed by all instances.

To properly distribute the files, which need to be converted to the VCF format between the involved instances the same lock-by-file approach is performed:

Within a loop, the script iterates over the set of files to be converted, and adds a corresponding lock file to the container marking each file that is currently being or has already been processed. If an input file is already locked by another instance, it is skipped and the local instance continues with the next file available.

While this approach does not guarantee the avoidance of concurrent access to the same file, is has been chosen as the synchronization mechanism for this first workflow prototype due to its simplicity.

3.1 Test Runs

Test runs of this setup have been performed based on the following input data set:

- 8 CEL files with 66 MB each
- Human genome data slightly bigger than 3 GB

The following cloud resources have been used to perform the computations:

- 5 instances with 4 GB of RAM and two Intel Core2 Duo CPUs with 2.4 GHz each. However for the execution only one CPU per instance was used.

One of the instances was used to perform the sequential part and afterwards all five instances processed the eight CEL files in parallel.

Table 1: Runtimes measured for the sequential part.

Workflow Step	**Runtime**
Input data upload	62 s
Birdseed algorithm	263 s
Filtering SNPs	26 s
Storing Result in Container	3 s
Sum	**354 s**

Within the parallel execution the data upload has only been performed once for each instance. The uploading times for the input data for the sequential and parallel parts of the workflow are different due to the different types of input data of the different workflow stages.

The result data upload in tables 2 and 3 refers to the result data of the sequential part of the workflow.

Table 2: Maximum runtimes measured for the parallel part.

Workflow Step	**Max Runtime**
Genome data upload	558 s
Result data upload	8 s
VCF conversion	5207 s
Sum	**5773 s**

Table 3: Mean runtimes measured for the parallel part.

Workflow Step	Mean Runtime
Genome data upload	547.80 s
Result data upload	2.80 s
VCF conversion	4970.33 s
Sum	**5430.93 s**

3.2 Application of the Workflow to Real Data

We have shown an example usage scenario for this workflow, where it was used to analyse the genotypes of patients with cross intolerance to non-steroidal anti inflammatory drugs, one of the most important drug allergies, which can lead to potentially fatal reactions (Ayuso et al., 2013). A group of over 100 patients were genotyped, along with over 100 control subjects. More details on the data and its analysis are given in (Cornejo-García et al., 2013). This approach led to the discovery of a number of SNPs that show a strong association with the pathology. These SNPs are currently being further investigated in a new cohort of patients and through the use of in vitro experimental assays.

4 DISCUSSION AND CONCLUSIONS

We have presented a novel cloud-based workflow, *genCloud*, which allows an end-user to perform a number of important and computationally intensive steps in the analysis of genome wide association data. We believe that close collaboration between target end-users and developers is essential for the development of such a service and to ensure that user demands are met. We also believe that cloud based systems are the most suitable and flexible solutions for the computational needs of clinical research groups in hospitals and medical research groups for two main reasons:

- They require minimal computational know-how from the end-user in terms of installation, administration and scripting. The user can therefore focus on understanding the details of the tools themselves and the biological interpretation of the results, rather than their implementation.
- They can also be scaled according to the user's needs: the analysis presented here focussed on a small, proof of principle analysis, with 8 samples. However, some GWAS experiments produce 100s, even 1000s of samples, requiring more computational resources. This isn't an issue for a cloud based system: the cloud instances started for the analysis can be as large and as numerous as required. The upload of the genome data - representing the biggest input dataset by far - is only required once. Afterwards multiple sets of SNP files can be processed by the workflow.

It should be made clear that the workflow described here is only able to handle data from the Affymetrix SNP chip microarray platform. Given the ever growing popularity of whole genome sequencing data, as well as its ever reducing cost, the next step will be to extend the workflow for this kind of data. A typical whole genome sequencing file with a high coverage can have more than 100 GB per individual. The task to extract the genotypes out of these files can take around a week and will take around 1 TB temporary hard disk size per individual. As this step is completely independent for each individual it will benefit greatly from a cloud-based implementation, since this will enable flexible scaling and parallelisation. Moreover, given that the output of the analysis of this data is also a VCF file, it could be combined with the same downstream analysis workflows that are used for the current data.

The runtime measurements presented in Section 3.1 have been collected from a workflow of instances being started by the user before the actual workflow was invoked by starting the workflow script on each instance through an interactive shell. While the automatic invocation of instances stored as snapshots has already been performed for other applications on the same infrastructure it has not been applied yet for the workflow described here. The automatic deployment of new instances will be realised as part of the scheduler component of the infrastructure, which is still in the design phase.

Future work will focus on the integration of the above-mentioned cloud technologies and our community cloud with the Galaxy workflow engine subsequently enabling us to provide a much better and simplified user experience to medical and research staff, who are not experts in cloud computing and workflows. As additional steps we will investigate possibilities of automatic exploitation of cloud elasticity to enable scalable workflows being able to handle different input sizes.

ACKNOWLEDGEMENTS

This publication is supported by the European Community through the FP7 IAPP project Mr. SymBioMath, grant agreement number 324554.

REFERENCES

Afgan, E., Baker, D., Coraor, N., Chapman, B., Nekrutenko, A., and Taylor, J. (2010). Galaxy cloudman: delivering cloud compute clusters. *BMC Bioinformatics*, 11(Suppl 12)(S4).

Afgan, E., Chapman, B., Jadan, M., Franke, V., and Taylor, J. (2012). Using cloud computing infrastructure with cloudbiolinux, cloudman, and galaxy. *Current Protocols in Bioinformatics*.

Allcock, W., Bester, J., Bresnahan, S., Plaszczak, P., and Tuecke, S. (2003). Gridftp: protocol extensions to ftp for the grid. Technical Report GFD-R-P.020, Open Grid Forum. Proposed Recommendation.

Allen, B., Bresnahan, J., Childers, L., Foster, I., Kandaswamy, G., and Kettimuthu, R. (2011). Globus online: Radical simplification of data movement via saas. Technical Report Preprint CI-PP-5-0611, Computation Institute, The University of Chicago.

Amazon Web Services (2013). Amazon simple storage service (amazon s3). http://aws.amazon.com/s3/.

Ayuso, P., Blanca-López, N., Doña, I., Torres, M., Guéant-Rodriguez, R., Canto, G., Sanak, M., Mayorga, C., Guéant, J., Blanca, M., and Cornejo-García, J. (2013). Advanced phenotyping in hypersensitivity drug reactions to nsaids. *Clinical and Experimental Allergy*, 43(10):1097–1109.

Button, K., Ioannidis, J., Mokrysz, C., Nosek, B., Flint, J., Robinson, E., and Munaf, M. (2013). Power failure: why small sample size undermines the reliability of neuroscience. *Nature Reviews Neuroscience*, 14(5):365–376.

Ceph Team (2012). Rados gateway - ceph documentation. http://eu.ceph.com/docs/wip-3060/radosgw/.

Cornejo-García, J., Liu, B., Blanca-López, N., na, I. D., Chen, C., Chou, Y., Chuang, H., Wu, J., Chen, Y., Plaza-Serón, M., Mayorga, C., Guéant-Rodríguez, R., Lin, S., Torres, M., Campo, P., Rondón, C., Laguna, J., Fernández, J., Guéant, J., Canto, G., Blanca, M., and Lee, M. (2013). Genome-wide association study in nsaids-induced acute urticaria/angioedema in spanish and han-chinese populations. *Pharmacogenomics*. in press.

Debian Wiki (2012). euca2ools - debian wiki. https://wiki.debian.org/euca2ools.

Foster, I. and Kesselman, C., editors (2003). *The Grid 2: Blueprint for a New Computing Infrastructure*. Elsevier.

Foster, I., Kesselman, C., and Tuecke, S. (2001). The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 3(15).

Howes, T. and Smith, M. (1995). A scalable, deployable directory service framework for the internet. Technical Report UM-CITI 95-7, University of Michigan.

Karlsson, J. and Trelles, O. (2013). Mapi: a software framework for distributed biomedical applications. *Journal of Biomedical Semantics*, 4(4).

Korn, J., Kuruvilla, F., McCarroll, S., Wysoker, A., Nemesh, J., Cawley, S., Hubbell, E., Veitch, J., Collins, P., Darvishi, K., Lee, C., Nizzari, M., Gabriel, S., Purcell, S., Daly, M., and Altshuler, D. (2008). Integrated genotype calling and association analysis of snps, common copy number polymorphisms and rare cnvs. *Nature Genetics*, 10(40):1253–1260.

Krampis, K., Booth, T., Chapman, B., B. Tiwari, M. B., Field, D., and Nelson, K. (2012). Cloud biolinux: preconfigured and on-demand bioinformatics compu ting for the genomics community. *BMC Bioinformatics*, 13(1):42.

Le Bras, Y. and Chilton, J. (2013). Deploying production galaxy instances on openstack with cloudbiolinux and cloudman. https://www.e-biogenouest.org/resources/243.

Martin-Requena, V., Rios, J., Garcia, M., Ramirez, S., and Trelles, O. (2010). jorca: easily integrating bioinformatics web services. *Bioinformatics*, 26(4):553–559.

Mell, P. and Grance, T. (2011). The nist definition of cloud computing. Technical Report 800-145, National Institute of Standards and Technology.

Mobley, A., Linder, S., Braeuer, R., Ellis, L., and Zwelling, L. (2013). A survey on data reproducibility in cancer research provides insights into our limited ability to translate findings from the laboratory to the clinic. *PLoS One*, 8(5). e63221.

Novotny, J., Tuecke, S., and Welch, V. (2011). An online credential repository for the grid: Myproxy. In *Proceedings of the Tenth International Symposium on High Performance Distributed Computing*.

Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M., Wipat, A., and Li, P. (2004). Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054.

Pepple, K. (2011). *Deploying OpenStack*. O'Reilly Media, first edition.

R Development Core Team (2008). *R: A language ad environment for statistical computing*. R Foundation for Statistical Computing.

Weil, S., Brandt, S., Miller, E., Long, D., and Maltzahn, C. (2006a). Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th Symposium on Operating System Design and Implementation*, pages 307–320.

Weil, S., Brandt, S., Miller, E., and Maltzahn, C. (2006b). Crush: controlled, scalable, decentralized placement of replicated data. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*.

Architectural Design of a Deployment Platform to Provision Mixed-tenancy SaaS-Applications

Matthias Reinhardt[1], Stefan T. Ruehl[2], Stephan A. W. Verclas[3], Urs Andelfinger[1] and Alois Schütte[1]
[1]*University of Applied Sciences Darmstadt, Haardtring 100, 64295 Darmstadt, Germany*
[2]*Clausthal University of Technology, Albrecht-von-Groddeck-Str. 7, 38678 Clausthal-Zellerfeld, Germany*
[3]*T-Systems International GmbH, Heinrich-Hertz-Str. 1, 64295 Darmstadt, Germany*
matthias.reinhardt@stud.h-da.de, {urs.andelfinger, alois.schuette}@h-da.de, stefan.t.ruehl@tu-clausthal.de, stephan.verclas@t-systems.com

Keywords: Software Architecture, Software Design, Web and Internet Services.

Abstract: Software-as-a-Service (SaaS) is a delivery model whose basic idea is to provide applications to the customer on demand over the Internet. In contrast to similar but older approaches, SaaS promotes multi-tenancy as a tool to exploit economies of scale. This means that a single application instance serves multiple customers. However, a major throwback of SaaS is the customers' hesitation of sharing infrastructure, application code, or data with other tenants. This is due to the fact that one of the major threats of multi-tenancy is information disclosure due to a system malfunction, system error, or aggressive actions by individual users. So far the only approach in research to counteract on this hesitation has been to enhance the isolation between tenants using the same instance. Our approach (presented in earlier work) tackles this hesitation differently. It allows customers to choose if or even with whom they want to share the application. The approach enables the customer to make that choice not just for the entire application but specifically for individual application components and the underlying infrastructure. This paper contributes to the mixed-tenancy approach by introducing a software pattern that allows to establish communication between Application Components that are deployed following the mixed-tenancy paradigm. The contributions of this paper are evaluated based on a representative example that employs all possible kinds of communication.

1 INTRODUCTION

Software-as-a-Service (SaaS) is a delivery model whose basic idea is to provide applications to the customer on demand over the Internet. SaaS promotes multi-tenancy (MT) as a tool to exploit economies of scale. This means that a single application instance serves multiple customers. However, even though multiple customers use the same instance, each of them has the impression that the instance is designated only to themselves. This is achieved by isolating the tenants' data from each other (Chong and Carraro, 2006).

In contrast to single-tenancy, MT has the advantage that IT-Infrastructure may be used most efficiently as it is possible to host as many tenants as possible on the same instance. Thus, operational and maintenance cost of the application is decreased. However, one of the major threats of MT applications is information disclosure due to data breach (Bezemer and Zaidman, 2010; Cloud Security Alliance, 2013). This may occur through a system error, malfunction, or destructive action. So far this problem has only been tackled by proposing new approaches to implement and improve the tenant isolation on a single instance.

The approach presented in (Ruehl et al., 2012; Ruehl et al., 2013), however, is different as it strives to solve the problem by finding a hybrid solution between multi-tenancy and single-tenancy. We refer to this approach as *mixed-tenancy*. The approach tries to emphasize both the customers' concerns about sharing infrastructure as well as the operator's desire to utilize infrastructure as efficiently as possible.

The approach that is supposed to answer these questions starts by building an MT application based on the concept of Service-oriented Architecture (SOA). This means the so-called composite SaaS-applications are composed of a number of application components (AC) that each offer atomic functionality. The approach enables the customer to choose if or even with whom they want to share a specific AC

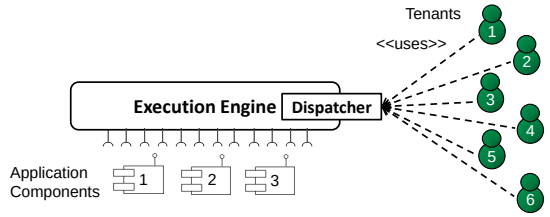


Figure 1: Architectural overview.

and the underlying infrastructure stack. This way customers may declare their requirements toward a deployment of their application variant.
As an example please refer to Figure 1. It illustrates a setting in which six tenants are using an application that is composed of three ACs. Example requirements that would be possible are:

1. AC 1 is not handling any critical data, thus all tenants feel comfortable to share instances and the underlying Virtual Machines.
2. AC 2, however, handles semi-sensitive data. Thus, tenant 2, for example, only wants to share an instance with European companies but not with competitors. However, on a Virtual Machine (VM) level, tenant 2 agrees to share with all customers that are not competitors.
3. AC 3 is handling very sensitive data. Thus, tenants will require quite restrictive deployments. Tenant 4, for example, stated that they do not wish to share an instance with competitors and only with companies from the USA.
4. Tenant 6 is special in the sense that they demand a single instance for their private use of every AC and VM.

In (Ruehl et al., 2013) a semantic model is presented that allows customers to describe their constraints towards a mixed-tenancy deployment. Based on this model the Deployment Configuration Generator (DCG) may be used to capture customer constraints. Furthermore, once the constraints were added into the DCG, a deployment is calculated that applies to all expressed customer constraints and is optimal according to resource consumption. This deployment is described in a transfer document that we refer to as deployment configuration (DC). This will be discussed in detail by Section 3. Based on the DC, a given multi-tenancy enabled composite application may be deployed according to the mixed-tenancy paradigm.
This paper's contribution is a software pattern that may be applied to establish communication between ACs deployed in a mixed-tenancy environment. The pattern shall be able to solve the special challenges that appear in such an environment. First of all, this pattern shall be able to enforce the customer given constraints for the data communication between components. And secondly, the pattern promotes the possibility to migrate existing multi-tenancy applications into a mixed-tenancy environment without having to change the application.

In order to do that the paper is structured as follows. After a discussion of related work, the paper will start with an analysis of requirements that architecture of the mixed-tenancy platform has to meet. The paper continues by developing the pattern to establish communication. This is done by first discussing existing patterns with respect to the defined requirements, and combine those to satisfy the requirements. In order to be capable of evaluating the communication pattern, it is necessary to develop a mechanism that can automatically deploy an application according to a DC. This is presented by Section 5. Section 6 will conduct an evaluation of the presented pattern by analyzing specifically built scenarios that represent a wide variety of possible cases.

The paper concludes with summarizing the results and drawing a conclusion.

2 RELATED WORK

SaaS in general as well as the configuration issues and challenges related to it were explored in (Sun et al., 2008). The aspect of functional variability has been discussed in some work. As, for example, in (Mietzner et al., 2009) the author discusses bringing such flexibility to the process layer of process-based, service-oriented SaaS-applications. This is done by creating variability descriptors that are transformed into a BPEL process model. In (Lizhen et al., 2010) an approach is presented to describe variability for SaaS-applications. This approach can systematically describe variability points and their relationships, and it assures the quality of the configuration inputs made by the customers. This work focuses on the creation of descriptions of functional variability. Furthermore, there are approaches with the goal to realize MT that meets privacy and performance requirements. In (Guo et al., 2007), for example, a framework for multi-tenant aware SaaS-applications including data isolation, performance isolation or configuration is described. Our approach, in contrast, allows the customer to give input for how and with whom their ACs are deployed. Data security is achieved as tenants do not share the same AC instances. To the best of our knowledge there is no similar approach. In addition, approaches that realize tenants' isolation on an MT instance may be applied.

3 REQUIREMENTS ANALYSIS

This section's purpose is to discuss the necessary requirements towards realizing a mixed-tenancy platform. The platform consists of two major tasks. The first is the automatic deployment, whose job is to deploy the application according to the description of the DC. A prototypical implementation of the deployment task will be discussed in section 5. This may require that multiple VMs need to be created and ACs, that make up the application, will be instantiated multiple times. The DC is the output of the aforementioned DCG and describes the deployment that is to be realized. Thus, the DC defines how many instances of a specific component shall be instantiated on a particular server and which tenants are allowed to access them. It captures a deployment that applies to all constraints of all tenants using the application. Additionally, the DC was created to be optimal according to resource consumptions in order to satisfy the service providers' need for efficient use of infrastructure. Figure 2 illustrates the structure of the DC in Entity-Relationship Notation. The figure captures the relationship between the important entities (Servers, ACs, ACIs, Tenants and Users). These entities have been included in an XML-schema, from which a valid DC can be created by defining an XML-file which refers to the schema. This allows us to decouple the DCG from the rest of the platform.

The second important task of the platform is the communication. This task of the platform is in charge of establishing communication between the deployed component instances. Since it is our aim to keep the adaptation progress of an application at a minimum, one requirement of the communication task is the separation of the application logic and the platform logic. This may be realized by introducing a new layer above the original application logic. This platform logic layer manages the communication. Furthermore, it is important, that the platform's communication applies to the customers' wishes. If a tenant denies the common deployment with another tenant, the platform has to make sure that this can happen under no circumstances. Hence, it has to be avoided, that the data communication of two tenants, who may are competitors, is redirected through a common point of intersection. This is the second requirement that needs to be fulfilled by the communication pattern. It is based on the respect for the customers' wishes. If a customer denies to share a component with his competitor, it is not legitimate to redirect both their data traffic through a global interstation. Another reason is a possible high amount of data traffic which could slow down the running application.

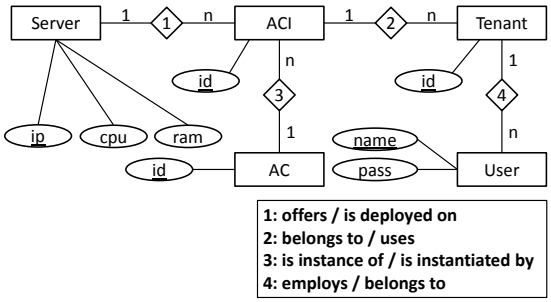


Figure 2: Structure of Deployment Configuration (ERM).

4 PATTERN SELECTION

The main goal of this paper is to develop a proper design pattern that allows to establish communication by realizing the prerequisites, that were introduced in the last chapter. To achieve this, we investigate well-established design patterns as well as intuitive approaches. We chose a simple example of a DC which contains a special case. In the following, the example will be applied to every pattern. This allows to illustrate how the platform would behave in common situations as well as in the special case. Additionally, it will be possible to compare the individual results in order to select the most appropriate pattern. The comparison is shown in Figure 3 via visualizations of the structure and the sequence (UML sequence diagrams).

The example consists of two tenants who are using an application with three components. There are five application component instances (ACI) where their border color indicates the tenant belonging. The regular case is demonstrated at component 1 and 3 where both tenants use exclusively one instance. As to component 2 there is only one instance which is shared between tenant 1 and 2. As discussed in the last section it is important that the platform maintains decentralized communication and separation of the platform logic in both cases. Another element which exists in all diagrams is the Execution Engine (EE). It belongs to the platform and has the knowledge of all component-instance locations and tenant belongings. It may fulfill additional tasks depending on the specific patterns it is used in.

The first pattern we came up with we called **delegator**. Within this pattern the EE's job is to delegate communication. When a user or a component instance wants to use a service of another component it asks the delegator which instance of that component belongs to the current tenant. This control communication is represented by the dashed arrows in Figure 3. The EE responds with the whereabouts in form of

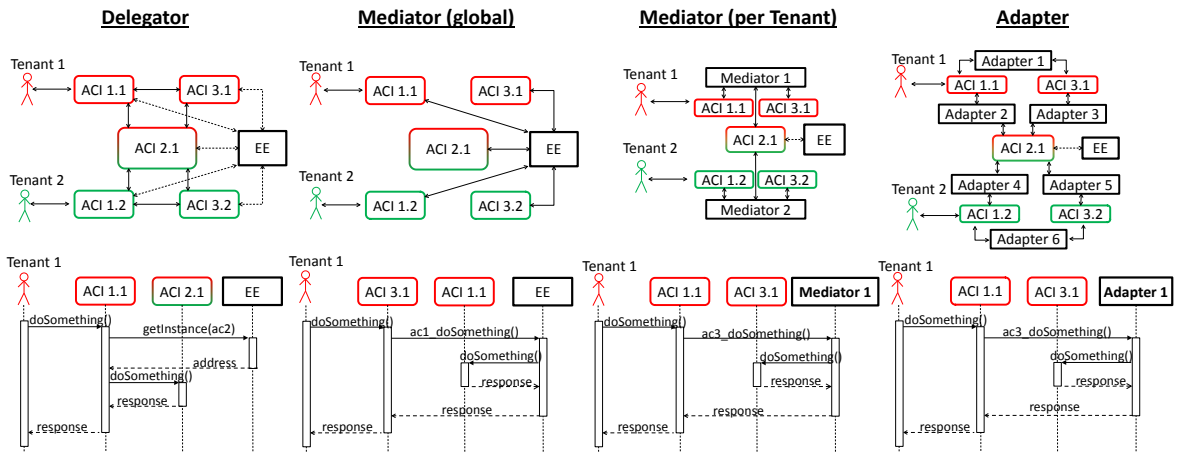


Figure 3: Overview of different Patterns to tackle the Communication Challenge.

an address. Thereby a contact can be established between the two instances or the user and the instance. This direct communication is represented by the solid lines. Changes of the locations are no issue, because the contact to the EE will happen before every function call of another component instance.

With respect to the requirements, there is a problem with the separation of the platform logic. The single instance of component 2 knows multiple instances of a component because it is shared by multiple tenants. Thus, we need platform logic in the component, which is not desired because it complicates the adaptation process of an application to the platform. However, the pattern meets the requirement for decentralized communication that applies to customers' constraints.

Secondly, we engaged in with the **mediator** pattern provided by Gamma et al. (Gamma et al., 1994). The Mediator is an object which encapsulates the interaction between an amount of objects. Thus, the objects are able to communicate without directly connecting to each other. Through our investigation two ways to implement the pattern into the platform emerged – global and concrete mediator.

The **global mediator** is characterized by a single mediator object which is represented by the EE. There are solid lines in Figure 3 between the component instances and the mediator. This means that all data communication between the component instances passes through the global mediator, unlike the delegator pattern where the data communication happens directly between the component instances.

With the use of the global mediator, there is no communication logic inside the components. They just have to be able to communicate with the EE. Thus, there is no problem with the special case of component 2, where one instance is used by two tenants at once. But because of the passing of the complete data communication through the EE, we detected a problem with the other requirement. If the customer requested the separation to another customer, it is not legitimate to redirect both their data communication through a common point of intersection, which, unfortunately, is the case here. Another negative aspect is a possible overload of the global mediator in a larger scenario, which could slow down the system as the global mediator becomes a bottleneck for all communication.

Another way to adapt the mediator pattern is shown by Figure 3 by the use of multiple mediator objects, one **mediator per tenant**. In contrast to the global mediator the data communication which passes a mediator is related to the same tenant. Because of this, we have successfully respected the requirement of decentralized communication.

In case of component 2 there exists only one instance used by both tenants. The instance has to decide which mediator to contact if it wants to call a function of another component. Clearly, there is communication logic necessary inside the components which stands in conflict with the second requirement.

Another pattern we ported onto the platform is the adapter pattern which was also given by Gamma et al. (Gamma et al., 1994). Usually, it will be applied to establish communication despite incompatible interfaces. For example, when third-party components should be included into a software without the possibility to adjust the interfaces, an adapter is an appropriate solution. In figure 3 there exists an adapter between every two component instances that will eventually need to communicate. Such adapters encapsulate the functionality of their related component instances.

In this setup we follow the requirement of de-

centralized communication. The data communication passing an adapter is related to the same tenant. With respect to the second requirement we have noticed a violation in the case of component 2. A common used component instance must decide which adapter to address. This decision has to be made depending on the current accessing tenant. Thus, communication logic is necessary inside the components.

So far none of the investigated patterns have matched both previously defined requirements. On this account we created a new pattern resulting the positive aspects of the previous patterns. It is called Connector because of its behavior. It has no relation to other homonymous design patterns which can be found in literature (e.g. Acceptor and Connector (Schmidt, 1995)).

Figure 4 describes the structure of the pattern. For every component instance there exists a related connector element. The component instances are solely able to initiate a connection to their connector. This is geared to the global adapter pattern, where there also was no platform logic located in the component instances. In contrast to the global mediator, this approach is distributing the platform intelligence into multiple connector elements and not into a single object like a mediator. Besides, there is a direct communication between the connector-elements. This works just like in the delegator pattern.

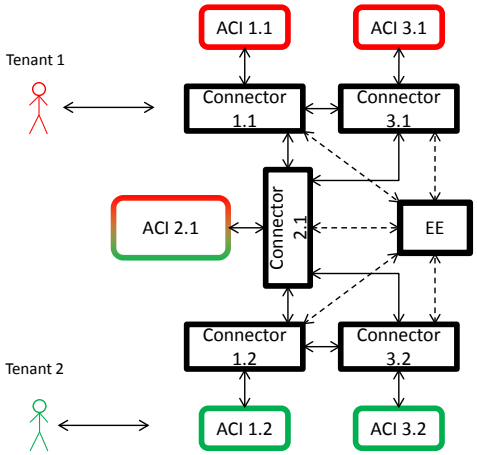


Figure 4: Example of the connector pattern.

In Figure 5 it is demonstrated how the communication works. If a component instance respectively a user wants to retrieve data from another component, it is calling the specific function at the related connector. Thus, here is no platform logic necessary. The component instance treats the function just like a local one. The connector is now handling the establishment of the connection. The EE acts just like the delegator element. The only difference is that the connector is requesting the address or the reference of the corresponding connector of the target component instance - not of the component-instance. After the actual function call took place, the result will be returned to the initial calling component instance. As illustrated by Figure 5, a connector encapsulates the needed functions of all other components for the related component instance and the provided functions to be called by the other component instances.

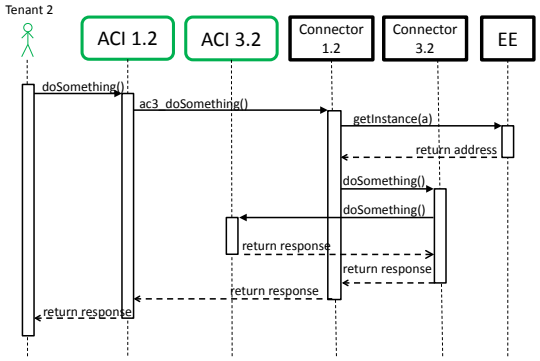


Figure 5: Sequence diagram of the connector pattern.

The pattern fits the requirements. As already explained in the last section, we have achieved a decentralized communication and the separation of platform logic. Because of this, we elect the connector pattern to the best way for solving the discussed problem. A summary of the investigated patterns and their ratings is shown in Table 1.

Table 1: Overview of the results of pattern analysis.

Pattern	Decentralized Communication	Separation of Platform Logic
Delegator	✔	✗
Mediator (global)	✗	✔
Mediator (per tenant)	✔	✗
Adapter	✔	✗
Connector	✔	✔

5 PROTOTYPE REALIZATION

Since we were able to identify a mixed-tenancy architecture, there is the need to prove its functionality and practicability. The intent of this section is the description of a case-study which has been realized using the connector pattern. This provides a base for an evaluation of the architecture in the next section. This

section addresses the requirements of the deployment task which have been defined in Section 3.

The main focus of the project relied on how to automatically deploy multiple instances of different components, which afterwards shall communicate with each other. In Figure 6 you can see the automatic deployment of an exemplary mixed-tenancy application. There is a basic VM that is in charge of the deployment. At first the deployment of multiple target VMs starts, which is followed by the deployment of application component instances (ACI) and the related connector elements. The locations and number of deployments executed by the EE are given. They result from the customers' requirements.

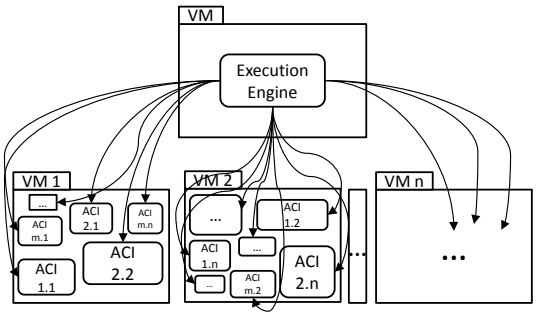


Figure 6: Deployment of application components.

The idea was to develop a mixed-tenancy platform. This means, that it is possible to attach existing multi-tenancy applications to the platform in order to make it mixed-tenancy capable. Thus, we realized the deployment task essentially using VMware vSphere[1] technolog and GlassFish[2] application servers. We used the VMware Virtual Infrastructure Java API (VI java)[3], which is an open source project initiated by VMware. The advantages against the official VMware vSPhere Management SDK are its higher performance and easier handling (Jin, 2013).

The ACs are encapsulated in web archives (WAR-files) and located at the same VM as the EE. During the execution process of a DC, these WAR-files will be duplicated and modified before they will automatically be uploaded and deployed on the target VM. To deploy an ACI we used the GlassFish asadmin tool. To allow or deny the access to an ACI we set the deployment descriptor of the WAR-file according to the information of the submitted DC. The modification of each concrete ACI is necessary, because the deployment constructor consists of many entries that cannot be identical if two ACIs should be deployed on the same server. This way each ACI is unique and can be identified.

The restricted access of AC instances gains security benefits over a pure multi-tenancy deployment. This is due to the fact that access restrictions to different instances of the same AC increases tenants' isolation.

The communication between the connector elements is realized via REST interfaces. To transmit the necessary number of VMs and ACIs we have defined a specific XML schema. A document following this schema contains a number of servers and instances of each component of the application. Furthermore, for each instance it is declared which tenants are allowed to access it. The implementation of SSL encryption would be a further step towards increasing tenants' isolation. This will be tackled in future.

6 EVALUATION

Based on the realized prototype of a platform introduced in the previous section, it is now possible to evaluate the planned mixed-tenancy architecture. Obviously, the platform is not runnable on its own, as it is supposed to deploy and run a multi-tenancy application. Because of this, we need to develop a dummy multi-tenancy application and port it onto the platform. We made sure, that the application comprises all possible special cases as well as the regular cases. This means that it is built op of multiple components that interact with each other directly or indirectly. An indirect interaction occurs whenever a component instance or a tenant advises a second component instance to call a function of a third component instance. When it comes to the properties of mixed-tenancy, we need to test a row of things.

In order to test the functionality of the developed platform, we made sure that all possible forms of customer constraints will be deployed without malfunctions. At first we have to distinguish between the server and the component layer. At both layers there exist multiple isolation levels.

At first we describe the levels at the component layer. *Shared* means that there is only one instance of the component, which is used by multiple tenants simultaneously. *Separate* on the contrary represents single instances of the component owned by each tenant. At last there is a level called *mixed* which comprises all remaining cases. At this level, there is at least one component instance that is used by more than one but not all existing tenants.

Regarding the server level, this is quite similar. *Shared* means that all tenants share one server where all component instances are deployed. At the *separate*

[1]http://www.vmware.com/de/products/vsphere/

[2]https://glassfish.java.net/

[3]http://vijava.sourceforge.net/

level each tenant needs a single server where all his component instances are deployed. *Mixed* means that at least two tenants are using component instances on the same server.

After the identification and the classification of all realistic cases, we combined the server and component layer. Since each of the layers has three levels, it is resulting in 9 permutations. Three of the 9 permutations are not realistic, e.g. if two tenants share one component instance, they cannot possess their own servers. This finally leads to 6 possibilities.

All test cases that belong to the same of the 6 resulting classes are equivalent. This means that we just need to test one to prove its functionality. For example, if we can successfully deploy a shared instance of component A on a server, then this result represents the operativeness of the shared deployment of component B. Thus, we elected one test case for each of the 6 classes. For each one we have tested direct and indirect access of the deployed component instances and verified the effect was as anticipated.

Due to the fact that all our test cases run through with success, we can say that our realized mixed-tenancy architecture which is embodied by the connector pattern is a working solution for the discussed problem. The evaluation has proven that our defined prerequisites were successfully enforced for the dummy application.

However, there are throwbacks coming from the utilized pattern. First of all performance will be decreased. This is due to the communication behavior, where ACs do not communicate directly with each other anymore. Instead communication is handled by connectors. This causes overhead. Secondly, if an operator decides to host a multi-tenancy application following the mixed-tenancy approach, it is necessary to create the connectors for all application components involved. The effort necessary to do that could only be minimized if it would be possible to automatically generate connectors through code inspection.

7 SUMMARY AND CONCLUSION

In the first chapter we introduced this paper with the characteristics of multi-tenancy as well as the advantages and disadvantages of cloud providers and their tenants. We explained why there is a demand for a better solution when it comes to the client's privacy needs and the resulting consequences for the provider. For example, when a client does not want to share a part of an application with his competitor, the provider needs to deploy the whole application twice, which leads to higher costs.

After that we introduced the concepts of mixed-tenancy as a possible solution. In order to verify these concepts as a solution we developed a prototypical platform which is capable of deploying multi-tenancy applications in the form of mixed-tenancy. The main goal of this paper was to identify or design an architecture for this platform. After defining several requirements, we investigated common design patterns regarding their applicability. This leads to the connector pattern which is a combination of the previously analyzed patterns. Since it matched the requirements, we selected the connector pattern as mixed-tenancy architecture.

Due to an evaluation of the realized platform, we were able to make sure that it is possible to port simple multi-tenancy applications onto the platform. During the porting process it was obvious that the connector pattern separates the platform logic from the application logic as well as it decentralizes the data communication.

Furthermore, we successfully verified that none of the tenants can access a component instance that does not belong to him. Through carefully selected test cases, we made sure that all kinds of DCs were covered.

However, these results come with a price. The pattern decreases the performance of an application since it produces an overhead in the communication among ACs. Furthermore, it is required that for every AC a connector is created. That creates an additional effort for an operator that wants to deploy a multi-tenancy application following the mixed-tenancy paradigm.

The main goal in the future should be the realization of a stable mixed-tenancy deployment platform. It shall be able to migrate an existing real-world multi-tenancy application onto this platform. For this paper we only evaluated an example application that realizes certain styles of communication. Due to its logical partition in a few components, it turned out to be a good test application and it was possible to gain indications that mixed-tenancy may be realized. However, in order to gain further conclusions and the applicability of the mixed-tenancy approach in real-world scenarios, it will be necessary to evaluate the migration of a real application.

Although, the focus of this paper was not based on performance, we aspire to optimize it in future research. Thus, a detailed performance analysis is still open for future research. So far the only security measures that have been implemented are basic authentication and user impersonation. The mixed-tenancy concept allows to deploy further security measures to isolate tenants even further (e.g. on a network level). However, this has not been investigated further yet,

as it was our goal to create a working platform with initial security.

Finally, one additional point that is still open for future research is the platform's lifecycle. In a real-world environment it is highly desired to have a environment that is both scalable and elastically adaptive. As a first step these characteristics were not addressed by this paper but bear great opportunities for future research.

REFERENCES

Bezemer, C.-P. and Zaidman, A. (2010). Multi-tenant SaaS applications: maintenance dream or nightmare? In *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, IWPSE-EVOL '10, page 88–92, New York, NY, USA. ACM.

Chong, F. and Carraro, G. (2006). Architecture strategies for catching the long tail. *Microsoft MSDN*.

Cloud Security Alliance (2013). The notorious nine: Cloud computing top threats in 2013. Technical report.

Gamma, E., Helm, R., and Johnson, R. E. (1994). *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley Longman, Amsterdam, 1st ed., reprint. edition.

Guo, C. J., Sun, W., Huang, Y., Wang, Z. H., and Gao, B. (2007). A framework for native multi-tenancy application development and management. In *The 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (CEC-EEE 2007)*, pages 551–558, Tokyo, Japan.

Jin, S. (2013). VMware infrastructure (vSphere) java API. http://vijava.sourceforge.net/faq.php.

Lizhen, C., Haiyang, W., Lin, J., and Pu, H. (2010). Customization modeling based on metagraph for multi-tenant applications. In *5th International Conference on Pervasive Computing and Applications*, pages 255–260, Maribor, Slovenia.

Mietzner, R., Metzger, A., Leymann, F., and Pohl, K. (2009). Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications. In *Workshop on Principles of Engineering Service Oriented Systems, ICSE*, volume 0, pages 18–25, Los Alamitos, CA, USA. IEEE Computer Society.

Ruehl, S. T., Andelfinger, U., Rausch, A., and Verclas, S. A. (2012). Toward realization of deployment variability for software-as-a-service applications. In *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, pages 622 –629.

Ruehl, S. T., Wache, H., and Verclas, S. A. W. (2013). Capturing customers requirements towards mixed-tenancy deployments of SaaS-Applications. In *2013 IEEE 6th International Conference on Cloud Computing (CLOUD)*, pages 462 –469.

Schmidt, D. C. (1995). Acceptor and connector: A family of object creational patterns for initializing communication services. In *In Proceedings of the European Pattern Language of Programs conference*, pages 10–14.

Sun, W., Zhang, X., Guo, C. J., Sun, P., and Su, H. (2008). Software as a service: Configuration and customization perspectives. In *Services Part II, IEEE Congress on*, volume 0, pages 18–25, Los Alamitos, CA, USA. IEEE Computer Society.

DDSF: A Data Deduplication System Framework for Cloud Environments

Jianhua Gu, Chuang Zhang and Wenwei Zhang
School of Computer Science and Technology, High Performance Computing R&D Center
Northwestern Polytechnical University, Xi'an, China
gujh@nwpu.edu.cn, pers@mail.nwpu.edu.cn

Keywords: Cloud Storage, Data Deduplication, Hash Table Partition, Index File System.

Abstract: Cloud storage has been widely used because it can provide seemingly unlimited storage space and flexible access way, while the rising cost of storage and communications is an issue. In this paper, we propose a Data Deduplication System Framework(DDSF) for cloud storage environments. The DDSF consists of three major components, the client, fingerprint server and storage component. The client component divides file into chunks and calculates hash value of each chunk. And it only sends the chunk whose hash value has not existed in fingerprint server component to the storage component, so it can reduce consumption of communications and storage space. We developed an Index File System(IFS) to manage the metadata of user file in fingerprint server. The fingerprint server component maintains a hash table containing the hash values of chunks already stored in storage component. This paper presents a two-level indexing mechanism to improve the both spatial and temporal overhead to access to the hash table. The first level employs the Bloom Filter and the second level uses hash table partition mechanism. In order to reduce the possibility of accidental collision of data block hash value, we use two hash algorithms to calculate the hash value of the chunk. We further conduct some experiments in our framework. The results of these experiments demonstrate that framework proposed in this paper can improve storage space utilization and reduce consumption of communications.

1 INTRODUCTION

Cloud storage has become an effective way for people to store their massive data, due to its high reliability, high scalability(Bonvin, 2010). However, with the increase of data quantity, the costs of the storage and communications increase exponentially, so it's very important to seek an effective storage method to improve the utilization of storage space and accelerate the storing speed. According to (Soghoian, 2011), an average of 60% of data may be deduplicated in Dropbox. The duplicated data may introduce more overhead of storage space and communications. Data deduplication is technique for eliminating duplicate copies of repeating data, and it is used to improve storage utilization and can also be applied to network data transfers to reduce the number of bytes that must be sent. Data deduplication can detect and remove duplicated data based on two different levels, the file-level or block-level(Maddodi,2010). Cross-user deduplication is used in practice to maximize the benefits of data deduplication, which identifies redundant data among different users, and removes the redundancy and just saves a single copy of the duplicate data. So data deduplication in cloud storage system can not only achieve higher storage efficiency, saving the cost of cloud storage providers, but also can reduce storage and communication costs for consumers who use the cloud storage system.

This paper presents a Data Deduplication System Framework for cloud storage ----DDSF, which adds deduplication mechanism to the common cloud storage system to reduce data redundancy without changing the common cloud storage framework. It can solve the problem of low utilization of storage space in the common cloud storage.

The DDSF is composed of three components, the client component, the fingerprint server component and the storage component. The fingerprint server component is responsible for storing and retrieving hash value of data block, named fingerprint. Before storing a file, the client component will divide the file into blocks and calculate the hash value of each

block. By communicating with the fingerprint server component, the client component can know which block has already existed in the storage component, and then it will only transmit the block which is not in the storage component. So it can effectively reduce the bandwidth consumption of communications.

In DDSF, we use two hash algorithms to calculate the hash value of the data block, which can greatly reduce the possibility of accidental collision(Henson, 2003) (Black, 2006). The size of hash table is increasing as amount of data stored in the cloud is increasing, so it's quite inefficient to retrieve the hash table. We design a two-level indexing mechanism of hash table to speed up the hash table retrieval. And we also develop a file system, named IFS(Index File System). The IFS is the key function of the metadata management and access. The metadata is the information about the user's file, and that is the position of each block's hash value in the hash table. The IFS uses tree structure to organize directory. The content of file in IFS is no longer real data of user file, but the addresses of all chunks belonging to this file in the hash table. We use the system image that stores a copy of the IFS and log file of operations on IFS to enhance the persistence and robustness of the IFS. In our implementation of DDSF, we use HDFS (Hadoop Distribute File System) as a storage system(Borthakur, 2007). HDFS is a distribution file system run on commodity hardware and it was developed by Apache Software Foundation for managing massive data.

The rest of this paper is organized as follows. Related works are discussed in Section Ⅱ. The proposed system framework and its main components are discussed in detail in Section Ⅲ. Some experiments are conducted in Section Ⅳ. Conclusion is in Section Ⅴ.

2 RELATED WORKS

As the amount of data stored in the cloud storage system increase, the size of the hash table will also increase, so how to efficiently search the hash table has become a key issue of data deduplication. In (Zhu, 2008), the authors described a Data Domain deduplication system, in which proposed to use multiple indexes and search the entire index table step-by-step to improve efficiency. But it has much more storage space overhead. In this paper, we develop a two-level index mechanism to improve the both spatial and temporal overhead. The first level index is based on the Bloom Filter (Broder,2003) (Bloom, 1970) technology, and the second level index uses partition mechanism which divides hash table into pieces of suitable size . The Bloom Filter can use the memory space efficiently. It maintains a binary vector data structure in memory and can quickly test whether a certain element is member of the set. False positive matches are possible, but false negatives are not; i.e. a query returns either "definitely not in set" or "possibly in set". For the first case, we can conclude that this element does not exist immediately, and for the second case, we should do further search in second level.

Several deduplication storage systems have been previously designed, including Venti (Sean, 2002), Extreme Binning (Bhagwat, 2009), DeDe (Austin, 2009), DeDu (Sun, 2011). Venti is a data archiving system based on network, and it uses unique hash values to identify data blocks and an append-only way to store data. And it lacks support for data deleting function, therefore it is not suitable for the cloud storage environment. Extreme Binning stores a separate entire file to a separate backup node. And each backup node is autonomous and manages its own index table and data, without sharing information among backup nodes and knowing each other. It does deduplication only in one backup node, so it is not suitable for massive data global deduplication. DeDe is a block-level deduplication cluster file system without central coordination. However it's only suitable for a virtual machine environment to do deduplication on the virtual image, not suitable for general data storage system. DeDu is a file-level deduplication system and entire file similarity detection is in the client. But the repetition rate of the file-level is not very high generally, so the deduplication rate is relatively low.

In summary, this paper proposes a block-level and online deduplication data framework. The file chunking and data block fingerprint calculation is placed in the client, which can reduce the server calculated pressure. Moreover it identifies duplicate data before data transmission, so it can reduce the amount of data transmission and bandwidth consumption.

3 FRAMEWORK

The proposed framework named DDSF is shown in Figure 1. The DDSF is composed of three components, the client component, the fingerprint server component and the storage component.

- **Client Component:** In this component, file is divided into blocks(chunks) and the hash value of every block is calculated, then it sends these hash values to the fingerprint server component, which tests whether the data block is repeated in storage component and returns the results of test to client component. The client component then decides whether transmit the data block to the storage component based on these results. When users need to download a file, the chunk merging module can download the all chunks of a file from storage component directly and merges these chunks into a file.
- **Fingerprint Server Component:** There is a file system, named IFS, that includes two main functions: metadata management and hash table management in this component. The metadata management mainly manages metadata of user's file and operations on files. The hash table management maintains the hash table of all blocks and efficiently access to the hash table. Each hash table entry contains the block's hash value, the actual address of data block in the storage system, the repeat number of block and so on.
- **Storage Component:** We use HDFS as storage system, and in fact the data blocks of user files are stored in this place.

DDSF realizes online deduplication. The client component can know which data block is already stored in the storage component before sending the data through interaction with fingerprint server component. So it can save storage space and reduce bandwidth consumption with only sending the no duplicated block to HDFS. The client component and fingerprint server component are the main function components of the framework.

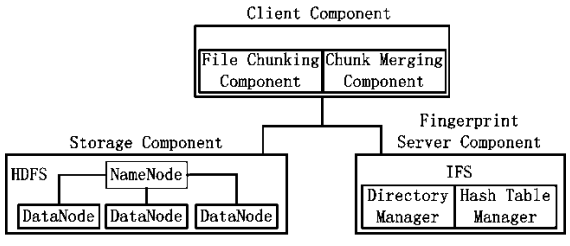


Figure 1: Framework Architecture.

3.1 Fingerprint Server Component

3.1.1 Index File System

The main function of this component is achieved through IFS, it is a file system. The IFS is the key function of the user file's metadata management and access. The metadata includes file's attributes and addresses of file data block, and the later are the pointers to an entry of hash table. For each user's file, the IFS will create a corresponding file in IFS, the *shadow file* or *metadata file*. Instead of real content of a user file, the metadata of user file are stored in the shadow file.

When fingerprint server component receives the command from client component to store a new file to storage component, it will create a new metadata file in IFS according to the file name from client component. And then fingerprint server component tests if the hash value of chunk from client component has already existed in the hash table. If so, it will increase the counter of hash table entry by 1 and store the address of this entry to the metadata file. If not, it will insert a new entry about the hash value and address of this block stored in storage component into the hash table, and then store the address of the new entry to the metadata file.

When fingerprint server component receives the command from client component to download a file from storage component, it gets the address of hash table entry from metadata file. According to the address, fingerprint server component gets the address of chunk stored in the storage component form the hash table and sends the address of chunk to client component, which will directly download the chunk in term of address from the storage component.

The IFS realizes metadata management and hash table management functions. The IFS uses a tree structure to manage the directory of metadata file. It creates a child node for each user on the tree and each user's directory structure is stored in a sub-tree rooted on that node. And each user can only access the files in his/her own directory. The hash table management is the key of IFS.

3.1.2 Two-level Index Mechanism

The IFS maintains a hash table. Each record of the hash table stores one data block's address in storage component, hash value of the block and the number of repetition of the data block. With the increasing of number of chunks stored in system, both storage and retrieval the hash table are very complex issue. In order to speed up the access to the hash table and to decrease amount of storage space, we use Bloom Filter and hash table partition technique. The two-level index mechanism is shown in Figure 2. The first level index is completed by the Bloom Filter,

and the second level index is completed by searching for a slice of the hash table. The pseudo-code of searching fingerprint with the two-level index mechanism is shown in Algorithm 1.

```
Algorithm 1 Searching fingerprint
Bool  Search_fingerprint(fingerprint)
Begin
  16_bit_fp=To_16_bit(fingerprint);
  if(hit_in_Bloom_Filter(16_bit_fp))
     if(found_in_slice(16_bit_fp))
        return TRUE;
     else return FALSE;
  else return FALSE;
End
```

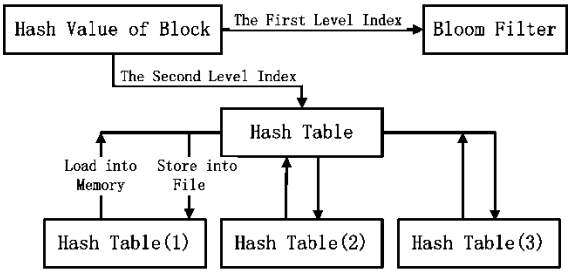


Figure 2: The two-level index mechanism.

The Bloom Filter table can reside in memory, so it has higher query efficiency. A bit in Bloom Filter represents the status weather a hash value exists in the hash table. We use the hash value calculated by the MD5 algorithm as main index value, and the hash value length is 128 bits. Obviously, if we want to use all of values generated by MD5 to create the Bloom Filter table, its size will be 2^{128} bits, which is too big to load into memory. So we design a compressed hash value indexing technique. It divides 128-bit hash value into eight 16-bit-parts, and then does Exclusive OR operation on these 8 parts in order to get a new 16-bit hash value. We use this new 16-bit hash value to create the Bloom Filter table. The Bloom Filter table size will be 2^{16}(64K), so it can reside in memory.

Once the fingerprint server component receives the hash value sent by the client component, it tests the status of the specific bit in the Bloom Filter table with the hash value. If the bit is not set, that means this chunk is not duplicated. Then client component will store this data block into HDFS directly. And the fingerprint server component adds the location of data block in HDFS and hash value to the hash table as an entry. Finally it will set the bit in Bloom Filter according to the hash value. If the bit is set, then it will perform the next level index operation.

In second level indexing, the hash table is divided into multiple parts and each part is separately stored in different directory on the IFS as a file. The directories are created according to 16-bit value by Exclusive OR operation, and each possible value corresponds to a directory. And for all hash values of hash table in the same directory, they have the same 16-bit value. In the indexing process, it will search the specified directory according to the 16-bit value generated by a hash value. And the slice in this directory will be loaded into memory, so it can ensure not only the size of the hash table in memory, but the performance of the indexing. The hash value will be judged as a new one if it does not exist in the slice, so the data block should be stored into HDFS, then the storage address and hash value should be stored in this slice.

The hash table located in different directory can be divided into several fixed-size slices according to the record number, and each slice is a hash sub-table. In order to prevent a single hash table becomes too large, we can set up a maximum record number, such as 16M. When a hash table size exceeds this maximum, then the system will create a new hash table. This will ensure the appropriate size of the hash table.

3.1.3 System Image and Operation Log

We develop the system image and log file of operations on the IFS to enhance its robustness to recover from mistakes. System image is used to store persistently the structure of file directory in memory, and operation log file is used to record user's operation information on IFS. When the system is activated, it simultaneously reads the system image and operation log from the disk, then combines these two files to generate new system image and simultaneously generates the directory structure of IFS in memory. Finally it deletes the old operation log and recreates a new blank log file. The system will periodically store the structure of file directory in memory into the system image, and update the operation log file.

For the concurrency issue of access to operation log information, we set two buffers and allow only one user to access one buffer at a certain moment. When one user access to the first buffer to record operation information, the buffer will be locked, then the system sends the operation information in first buffer to the second buffer and unlocks the first buffer. After the first buffer is unlocked, other user can access the first buffer, and the second buffer can write these information into the log file. Because the

I/O operation is performed in the second buffer, the second user can immediately write operation log when the previous user leaves from the first buffer, so it can greatly improve the response speed for multiple users to write log file.

3.2 Client Component

Client component is a tool through which user accesses to the cloud storage system. The client consists of file chunking module and chunk merging module.

- **File Chunking Module:** In this module, a file is broken down into chunks of fixed size or unfixed size, and the hash values for each chunk are calculated using both MD5 and SHA-1 algorithm. Then the client sends the command of storing file and the file chunk's hash value list to the fingerprint server component. The fingerprint server component creates a new file in the IFS to store the metadata of user file, and searches the hash table to test which chunk already exists according to the list of the hash values. The fingerprint server component will inform the client component to send the chunks which have not been duplicated to HDFS directly.
- **Chunk Merging Module:** When user wants to download a file from cloud, this module will consult the fingerprint server component to obtain all chunks' address of this file, and then directly access to HDFS to get chunks block by block according to these chunks' addresses, eventually this module merges these chunks into a file.

In all, the client component can know whether the chunk is repeated before chunk transmission and only sends the unrepeated chunk, which can reduce traffic and bandwidth consumption.

4 EXPERIMENTS

Our experiment platform was set up on 6 computers. Four computers serve as the cloud storage component, including one NameNode and three DataNodes. The fifth computer serves as the fingerprint server component, and the last one serves as the client component. The detailed configuration information is listed in Table 1.

Table 1: Configuration of Six Computers.

Hosts	CPU	RAM	Hard Disk	IP Address
NameNode	Intel(R) Xeon(R) E5540@2.53GHz	8GB	120GB	10.128.12.4
DataNode1	Intel(R) Xeon(R) E5540@2.53GHz	8GB	120GB	10.128.12.5
DataNode2	Intel(R) Xeon(R) E5540@2.53GHz	8GB	120GB	10.128.12.6
DataNode3	Intel(R) Xeon(R) E5540@2.53GHz	8GB	120GB	10.128.12.7
Hash Server	Intel(R) Xeon(R) E5540@2.53GHz	8GB	120GB	10.128.12.8
Client	IntelCore(TM)2 Duo E8400 @3.0GHz	2GB	120GB	10.128.12.9

4.1 File Consistency

In our experiment, we uploaded some files into DDSF, then downloaded these files and compared the downloaded files with the source files to check whether these files are consistent after deduplication. These files are built for a special purpose and consist of pictures, audio, binary, text and video files. We completed three experiments and set the data block size to 1M. Details of the source file size, the space used after deduplication and file consistency are given in Table 2.

Table 2: File Consistency.

Test	Number of Files	Source File Size	Size after Deduplication	File Consistency
1	20	125MB	87MB	Y
2	45	560MB	425MB	Y
3	76	1.2GB	726MB	Y

According to the results shown in Table 2, three tests have a different number of file, file type, file size and repetition rate, and all files are consistent in DDSF. Therefore our framework can maintain file consistency and integrity regardless of the file type and size.

4.2 Writing Efficiency

In this part, we will discuss the writing efficiency with and without DDFS. We used 6 groups of data sets and the data repetition rates were 0%, 20%, 40%, 60%, 80% and 100% respectively. The size of each group was 2.3G and the chunk size was 10M. We uploaded each data set and measured its loading time with and without DDSF. The time it takes to upload files with and without DDSF is shown in Figure 3.

In figure 3, data repetition rate is plotted on the x-axis and time taken to save the file is plotted on the y-axis. As shown in the figure 3, with data repetition rate increasing, the time taken to save files with DDSF reduces rapidly. When the data repetition rate is 5%, the time taken to save the same files with DDSF and without DDSF is the about same.

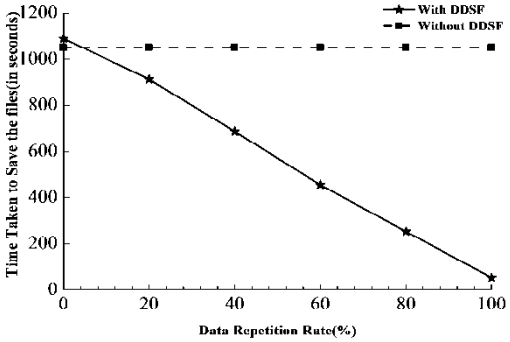


Figure 3: Time to Save files with and without DDSF.

4.3 Reading Efficiency

This part introduces the results of reading efficiency with and without DDSF. We separately download the previously stored data sets and measured each download time. The results are shown in figure 4.

As shown in figure 4, the average time taken to read files without DDSF is about 253s, and the time with DDSF is about 260s. The time increases by only 2.4% with DDSF than without DDSF. As mentioned before, there is an average of 60% of duplicated data in Dropbox. If using DDSF, we can save nearly 60% of storage space, although the time taken to read files will increase by 2.4%, whereas the time taken to upload files will reduce by 56.7%. In summary, the performance of the entire storage system can be largely improved by using DDSF.

4.4 Repetition Rate

In this section, we discuss the relationship among different block size, the amount of data and the data repetition rate. The experimental files are extracted from eight computers in our lab, and the total size of the data is 512GB. The block size has two levels, which are KB level and MB level. Each level is divided into three sub-levels which are 8KB, 16KB, 64KB, 1MB, 2MB, and 4MB. The amounts of data are 16GB, 32GB, 64GB, 128GB, 256GB and 512GB. Figure 5 shows the relationship between block size, amount of data, the number of hash table's record and the number of repeated record.

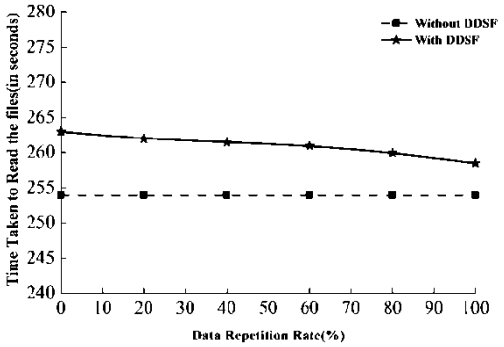


Figure 4: Time to Read files with and without DDSF.

As shown in figure 5-1, in a certain amount of data, the number of hash table's record will increase with the decrease of the block size. If the block size is 8KB and the amount of data is 512GB, the number of hash table's record is 10^7. That number will be 10^5 if the block size is 4MB. Meanwhile, the size of hash table is liner growth with the increase of data.

In figure 5-2, the number of repeated hash value will increase when the amount of data increases or the block size reduces.

The repetition rates are shown in table 3. BS denotes Block Size, RR denotes Repetitive Rate，AOD denotes Amount of Data. With the growth of block size, the repetition rate decreases.

The repetition rate decreases with the growth of block size. This is because the repetition rate is lower with the larger block size. Because the growth rate of the number of block is higher than that of the number of repeated hash entry, the repetition rate will reduce when amount of data increase.

$$\text{Repetition\_Rate} = \text{NRHE} / \text{NB} \quad (1)$$

where NRHE denotes the number of repeated hash entry and NB denotes the number of block.

Table 3: Repetition rates.

RR \ BS / AOD	8KB	16KB	64KB	1M	2MB	4MB
16G	38.4%	38.0%	36.9%	34.4%	33.2%	31.3%
32G	28.6%	28.2%	27.3%	25.1%	23.8%	22.6%
64G	22.9%	22.1%	20.8%	16.5%	15.4%	14.2%
128G	17.5%	16.9%	15.2%	12.6%	11.7%	11.0%
256G	15.9%	14.6%	14.1%	13.2%	10.9%	9.4%
512G	14.9%	13.0%	12.8%	11.2%	10.3%	8.7%

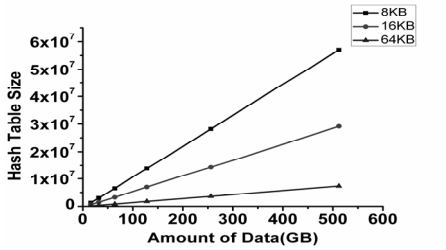


Figure 5.1.

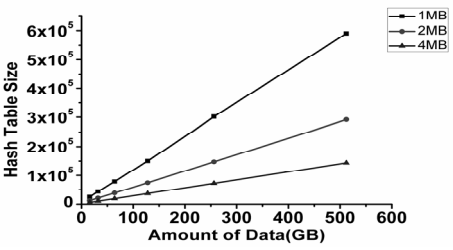


Figure 5.2.

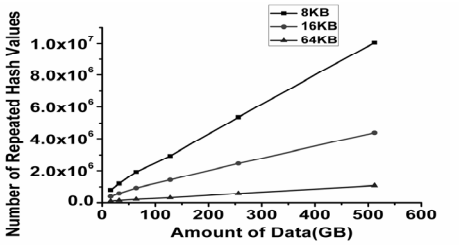


Figure 5.3.

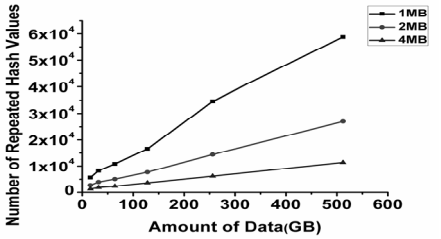


Figure 5.4.

4.5 Indexing Efficiency

We will discuss the efficiency of the two-level indexing mechanism. We use five different hash tables obtained in the section 4.4 and measure the time to retrieve hash table. These hash tables are generated in the condition that the chunk size is 8KB and the amounts of data are 16GB, 32GB, 64GB, 128GB and 256GB. The number of hash tables' records is 3×10^6, 5×10^6, 1×10^7, 2×10^7 and 3×10^7 respectively. We do four experiments for each hash table. The results are shown in Figure 6.

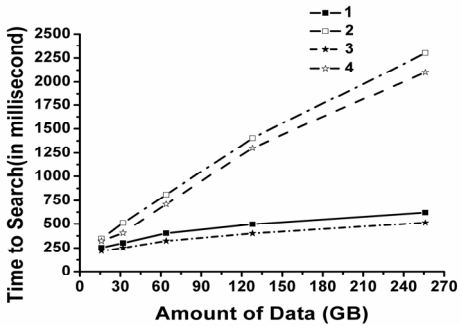


Figure 6: The efficiency of Indexing.

Table 4: The four tests.

Test	Number of hash values	Number in Bloom Filter	Number in hash table
1	5000	5000	5000
2	5000	5000	4000
3	5000	4000	4000
4	5000	4000	3000

We use five thousands of hash values to retrieve the hash table in each experiment. The features of these hash values are shown in table 4.

In figure 6 shows that, the time in test 2 is much more than that in test 1. Because searching hash values which have not existed in the hash table will need to search the entire slice, which will cost more time. The time in test 3 is less than that in test 1, because some hash values don't need to do further searching if they do not hit in Bloom Filter.

5 CONCLUSIONS

In this paper, we propose a deduplicated cloud storage framework---DDSF. The framework is designed to reduce storage space and bandwidth consumption to upload data to cloud. The client component consults the fingerprint server component to determine whether the chunk is repeated before sending the chunk to cloud, and it only sends the unrepeated chunk to the underlying storage system, so it can reduce the amount of communication and improve the storage space utilization.

This paper presents an efficient two-level indexing mechanism in order to improve the retrieval efficiency of the hash table. We employ the Bloom Filter and hash table partition mechanism. It divides the hash table into multiple parts and stores these parts in different directory according to a 16-bit value generated by hash value. When searching a hash value, it will only load the slice which has the same 16-bit value with the hash value into memory, so it can further accelerate the retrieval speed and improve efficiency. This paper also introduces the

system image and operation log. The system image preserves the directory structure of the entire system, and the operation log records user operations on directory, so the robustness of the IFS can be enhanced.

The retrieval of hash tables is the system bottleneck. In future work, we will develop more efficient indexing mechanism to reduce the indexing time so as to further optimize performance.

ACKNOWLEDGEMENTS

This research is supported by the project of Science and Technology Department of Shaanxi Province and by the National High-tech R&D Program of China (863 Program) under Grant No.2009AA01Z142.

REFERENCES

Nicolas Bonvin, 2010. A self-organized, fault-tolerant and scalable replication scheme for cloud storage[A]. *Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC'10*. New York: ACM Press, 2010. 205~216

C. Soghoian, 2011. How dropbox sacrifices user privacy for cost savings. http://paranoia.dubfire.net/2011/04/how-dropbox-sacrifices-userprivacy-for.html, April 2011, online.

Maddodi, S., 2010. Data Deduplication Techniques and Analysis[A]. *In Proceedings-3rd International Conference on Emerging Trends in Engineering and Technology, ICETET 2010*. New Jersey: IEEE Compute Society Press, 2010. 664~668.

V. Henson, 2003. An analysis of compare-by-hash. *in Proceedings of the 9th conference on Hot Topics in Operating Systems Lihue*, Hawaii, 2003, pp.13-18.

J. Black, 2006. Compare-by-hash: A reasoned analysis. *in USENIX Association Proceedings of the 2006 USENIX Annual Technical Conference,* 2006, pp.85-90.

D. Borthakur, 2007. The Hadoop Distributed File System: Architecture and Design. 2007. URL: http://hadoop.apache.org/hdfs/docs/current Ihdfs_design.pdf.

Benjamin Zhu, 2008. Avoiding the disk bottleneck in the data domain deduplication file system[A]. *In Proc. of the 6th Usenix Conf. on File and Storage Technologies (FAST 2008)*. Berkeley: USENIX Association, 2008. 269–282.

Broder A. Z., Mitzenmacher M, 2003. Network applications of bloom filters: A survey. Internet Mathematics, 2003, 1(4): 485−509.

Q. Sean, D. Sean, 2002. Venti: A New Approach to Archival Data Storage. *In Proceedings of the 1st USENfX Conference on File and Storage Technologies, ed. Monterey*, CA: USENIX Association, 2002, pp. 89-101.

D. Bhagwat, 2009. Extreme Binning: Scalable, Parallel Deduplication for Chunk-based File Backup. *in 2009 IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems Mascots*, 2009, pp. 237-245.

T. C. Austin, 2009. Decentralized deduplication in SAN cluster file systems. *in Proceedings of the 2009 conference on USENIX Annual technical conference*, San Diego, CaJifornia. 2009, pp. 101-114.

Zhe SUN, 2011. DeDu: Building a Deduplication Storage System over Cloud Computing. *in Proceedings of the 2011 15th International Conference on Computer Supported Cooperative Work in Design(CSCWD)*, pp. 348 – 355.

B. Bloom, 1970. Space/Time Tradeoffs in Hash Coding with Allowable Errors. Communications of the ACM, 13(7):422-426, 1970.

Everest: A Cloud Platform for Computational Web Services

Oleg Sukhoroslov and Alexander Afanasiev
Institute for Information Transmission Problems, Russian Academy of Sciences, Bolshoy Karetny per. 19, Moscow, Russia
oleg.sukhoroslov@gmail.com

Keywords: Service-Oriented Computing, Computational Web Services, Web Service API, REST, Service Framework, Cloud Platform, Platform as a Service.

Abstract: The ability to effortlessly reuse and combine existing computational tools is an important factor influencing research productivity in many scientific domains. While the service-oriented approach proved to be essential in order to enable wide-scale sharing of applications, we argue that its full potential in scientific computing is still not realized. In this paper, we present Everest, a cloud platform that supports publication, sharing and reuse of scientific applications as web services. The underlying approach is based on a uniform representation of computational web services and its implementation using REST architectural style. In comparison with existing work, Everest has a number of novel features such as the use of PaaS model, flexible binding of services with externally provisioned computing resources and remotely accessible API.

1 INTRODUCTION

Modern scientific research is often associated with complex computations and use of high performance computing resources. In their research scientists actively use software applications that implement computational algorithms, methods and models.

The ability to reuse existing computational tools is one of important factors influencing research productivity. However, such software often requires specific expertise in order to install, configure and run it that is beyond the expertise of an ordinary researcher. This also applies to configuration and use of high performance computing resources to run the software. Finally, researchers increasingly need to combine multiple tools in order to solve a complex problem, which brings an important issue of application composition.

The aforementioned problems can be addressed by provision of scientific applications in the form of remotely accessible, interoperable services. The use of service-oriented approach can enable wide-scale sharing, publication and reuse of applications, as well as automation of scientific tasks and composition of applications into new services (Foster, 2005). While the underlying principles of this approach are well-known, it is still an open question how to implement it in scientific computing in order to realize its full potential.

So far, most efforts in this area were focused on the provision of remote access to scientific tools via convenient web user interfaces. Examples of such approach include grid portals (Kacsuk, 2011), science gateways (Miller et al., 2010) and scientific hubs (McLennan and Kennell, 2010). While being successful among unskilled users, such systems do not actually expose applications as web services or provide programming interfaces thus limiting opportunities for application reuse, composition and integration with external applications.

This approach is in stark contrast to Web 2.0 applications and cloud computing services that support programmatic access via web service based APIs (Programmable Web, 2013). The proliferation of Web APIs has spawned development of mashups (Yu et al., 2008) that combine data, presentation and functionality from multiple services. Web service composition tools, such as Yahoo! Pipes, provided convenient interfaces for building mashups and making them available to everyone as new services.

This aspect is largely ignored in existing web-based scientific environments. As a rule, such systems do not provide tools for application composition or support workflows only on the level of computational jobs. The notable exception is Galaxy platform (Afgan et al., 2011) that supports tool composition and sharing of produced workflows. At the same time, Galaxy also doesn't expose tools and workflows as services thus limiting their use outside the platform.

Other efforts were focused on building tools for transformation of scientific applications into web services (Delaitre et al., 2005; Krishnan et al., 2009). These tools don't provide user interfaces beyond basic web forms for service invocation and rely on existing solutions for web service composition. While more powerful and well-aligned with SOA principles, this approach requires more effort in order to build a convenient environment for scientists. Service developers need an infrastructure to host services. The environment should provide mechanisms for service discovery, invocation and composition taking into account security requirements. These mechanisms should be accessible via convenient user interfaces facilitating the use of services for problem solving.

Both approaches also require considerable efforts to integrate an environment with high performance computing resources and grid infrastructures needed to run applications. As a rule, existing systems are tied to a single computing infrastructure and doesn't allow users to attach external resources.

The paper presents Everest, a cloud platform for computational web services that addresses discussed issues. It combines both approaches discussed above by exposing computational applications as web services with a uniform interface and implementing a web user interface for creating, sharing and accessing services. In contrast to previous work (Afanasiev et al., 2013), all functionality of the platform is provided remotely using the Platform as a Service model.

The paper is structured as follows. Section 2 discusses the model, interface and implementation of computational web services that underpin the proposed approach. Section 3 describes architecture and components of Everest platform in its current implementation. Section 4 concludes and discusses future work.

2 COMPUTATIONAL WEB SERVICES

2.1 Service Model

Computational web services (CWS) are the main entities managed by Everest. On the conceptual level, CWS represent a special type of web services targeted at processing computationally intensive requests. Such services should support management of long-running jobs and transfer of job data.

In contrast to generic web service interfaces to computing infrastructures, such as grids, CWS are specialized in running specific applications, i.e., solving specific classes of problems. Therefore a request to CWS normally doesn't contain an executable, but instead represents a set of input parameters describing a problem to be solved. We will refer to such requests as service-level jobs or just jobs. The job results can be represented as a set of output parameters in the same fashion.

It is responsibility of a CWS implementation to translate service-level jobs to one or more compute jobs submitted to underlying computing infrastructure in order to obtain desired results. Therefore CWS implement more specialized and high level interfaces than computing infrastructures. This makes it possible to hide the complexity of running compute jobs from service users and to enable transparent use of resources from multiple infrastructures.

In contrast to stateful web services typically found in enterprise systems, CWS process each incoming job in isolation. The only state managed by CWS is the state of processed jobs, so all data needed for a job should be provided in a request. While such restriction leaves out interactive session-based applications, it aligns well with the majority of computational tools such as solvers. This restriction also contributes to scalability properties of CWS.

The above description of CWS is rather general and can be related to a large class of services found not only in the scientific computing domain. Nevertheless, it explains the motivation and reasons behind implementation of CWS in Everest.

2.2 Service Interface

In technical terms, CWS can be implemented using any web service technology or style. Such freedom and lack of standards for implementation of CWS led to a multitude of approaches introduced by different systems. In order to facilitate reuse and composition of CWS implemented by different parties it is crucial to unify service interfaces.

The described model of CWS makes it possible to introduce a uniform service interface consisting of four operations:

- Job submission (as a set of input parameters);
- Retrieval of job state and results (as a set of output parameters);
- Job cancellation;
- Retrieval of service description (including description of input and output parameters).

All services implementing this interface support the same set of operations but can accept and return different sets of parameters. The last operation enables introspection of service parameters in order to

facilitate construction of job submission requests and processing of job results.

The described uniform interface follows an approach used by the HTTP protocol (Fielding et al., 1999) which defines a standard set of methods to indicate the desired action to be performed on the web resource identified by URI. This approach and the underlying REST architectural style (Fielding, 2000) proved to be essential to make the Web successful. In contrast, SOAP-based web services (Curbera et al., 2002) encourage creation of specialized interfaces and operations which provides a greater flexibility but complicates service reuse.

2.3 Interface Implementation

Using the REST architectural style the described uniform interface can be implemented as follows (Afanasiev et al., 2013). CWS represents a RESTful web service (Richardson and Ruby, 2008) identified by a Service URL. A job managed by the service is identified by a Job URL.

The Service resource supports the following HTTP methods:

- GET, which returns service description;
- POST, which performs job submission and returns a Job URL.

The Job resource created during job submission supports the following methods:

- GET, which returns the job state and results (if any available);
- PUT, which enables changing of the job state (e.g., job cancellation);
- DELETE, which destroys the job resource and deletes its data.

An additional File resource can be introduced to identify files passed to or returned by a service via its parameters. In such case parameter value contains a file URL. This enables passing large amount of data, which is particularly important for scientific computing, via appropriate data transfer mechanisms, such as HTTP, FTP or GridFTP.

Consider resource representation formats and means of describing service parameters.

The most widely used data representation formats for web services are XML and JSON. Among these JSON has been chosen for the following reasons. First, JSON provides more compact and readable representation of data structures, while XML is focused on representation of arbitrary documents. Second, JSON supports native integration with JavaScript language facilitating creation of web user interfaces for CWS.

The description and validation of service parameters can be accomplished by means of JSON Schema (JSON Schema, 2013), a de facto standard for defining the structure of JSON data.

2.4 Service Implementation

Consider an implementation of computational web service. Just like its interface, the inner workings of CWS follow a common pattern. A service listens to incoming job requests over HTTP and performs the following steps for each request:

- Authenticate and authorize the client;
- Parse and validate input parameters from the request;
- Translate input parameters to a compute job specification (executable, arguments, input and output files, etc.);
- Submit the compute job to configured computing resource;
- Monitor compute job state and provide this information to the client;
- Retrieve compute job results upon job completion;
- Translate compute job results to output parameters;
- Pass output parameters to the client.

Most of these steps can be implemented in the same fashion for any service disregarding its application domain. The only application specific parts are the ones that deal with processing of input and generation of output parameters. This makes it possible to implement a software framework, which provides a generic service skeleton that can be configured with the application specific parts (Afanasiev et al., 2013).

3 EVEREST PLATFORM

Everest is a cloud platform for computational web services that is based on considerations presented in the previous section. It implements a development framework and a hosting environment for CWS that adhere to the described uniform interface.

In contrast to traditional service development tools, Everest follows the Platform as a Service cloud delivery model by providing all its functionality via remote interfaces. A single instance of the platform can be accessed by many users in order to create, run and share services with each other without the need to install additional software on users' computers.

Another distinct feature of Everest is the ability to connect services with external computing

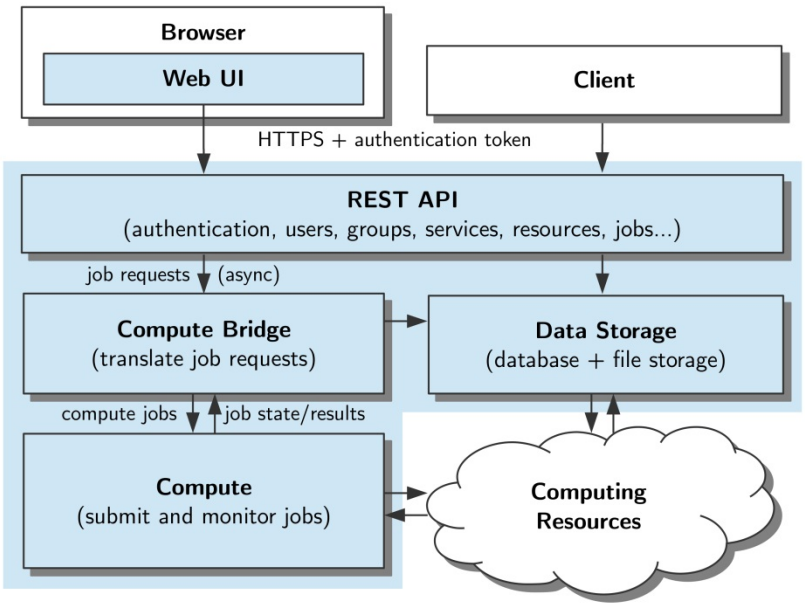


Figure 1: Architecture of Everest platform.

resources. That means that service developer can provide computing resource for running service jobs. This feature is useful in situations when platform's computing infrastructure has limited capacity or service developers need more control over an execution environment. A service user can also override the default resource by providing another resource for running her jobs.

The architecture of Everest is represented in Figure 1. Consider each of the platform's components in detail.

3.1 REST API

REST API is the platform's application programming interface implemented as a RESTful web service. It serves as a single entry point for all clients, including the web user interface.

The API includes operations for accessing and manipulating entities managed by the platform such as users, user groups, services, jobs and resources. In particular, the API implements operations from the uniform interface of CWS described in Section 2. It also provides additional operations related to services, such as service configuration and service discovery.

For each incoming request the API performs authentication of a client. The default authentication mechanism is implemented by means of OAuth bearer tokens (Jones and Hardt, 2012). A client can obtain a token by providing user credentials, i.e., username and password.

Upon successful authentication, the API also performs authorization of the requested action. Each entity managed by the platform has its' owner. The default security policy allows access to the entity only to its' owner. An owner can modify this policy, e.g., a service owner can specify a white list of users or user groups that are allowed to use the service.

The API relies on the data storage component to read and write information about platform entities. It also communicates with the compute bridge by passing it incoming job requests.

3.2 Web User Interface

Web user interface (Web UI) provides a convenient graphical interface for interaction with the platform. It is implemented as a JavaScript application that can run in any modern web browser without installation of additional software.

Web UI provides access to all functionality of the platform. It is built directly on top of the REST API, i.e., it uses the same interface as all other platform clients. While technically more challenging than traditional server-side web interface generation, this approach allowed us to reduce the server complexity and directly test the REST API.

The most important parts of Web UI are service configuration and job submission interfaces.

Service configuration interface is used to create new and edit existing services. It is implemented as a set of web forms that enable a user to specify all required information about a service including:

- Service metadata (name, description, etc.);
- Input parameters;
- Output parameters;
- Job template;
- Required files;
- Computing resource to run service jobs;
- Security configuration (white list, etc.).

Job template represents an application specific part of service configuration that is used by the platform to translate service requests to compute jobs. It includes the following information:

- Job command template that supports input parameter substitution;
- Mapping of input parameters to job input files;
- Mapping of job output files to output parameters.

Job submission interface is used to submit job requests to services. This interface is dynamically generated for each service according to the description of its input parameters. This information is also used to validate the request before its submission to the API. The implemented approach frees service developer from manual implementation of job submission forms.

3.3 Compute Bridge

Compute bridge is the core component of Everest that performs translation of service requests (service-level jobs) to compute jobs. It acts as a mediator between REST API and Compute subsystem that manages execution of compute jobs.

All job requests coming to REST API are asynchronously forwarded to the bridge. For each request the bridge performs translation of input parameters to a compute job specification according to the service configuration. The bridge also downloads input files that are referenced in the request.

A compute job specification produced by the bridge includes a command to be run, a list of job input files, a list of output files and a resource to run the job. The job specification is passed to the Compute subsystem for execution. The bridge also subscribes to notifications about the job state changes and translates these changes to the data storage.

Upon the job completion the bridge performs translation of job output files to output parameters according to the service configuration and saves the final result in the data storage.

3.4 Compute

Compute subsystem manages execution of compute jobs received from the bridge on computing resources attached to the platform. It performs all routine tasks related to staging of input files, submitting a job, monitoring a job state and downloading job results. All job state changes are translated to the bridge. Compute subsystem also monitors the state of all resources attached to the platform.

A computing resource can be attached to the platform by any user. A resource owner can configure a policy for accessing the resource. Any allowed user can bind the resource to any service.

Currently two approaches for integration with computing resources have been implemented. These approaches represent different tradeoffs between ease of integration and resource protection.

The first approach relies on existing remote access mechanisms supported by resources, such as SSH. In this case such mechanism is configured to accept credentials provided by the platform, e.g., SSH keypair. This enables the platform to directly execute any commands on the attached resource. Such approach makes it easy to attach computing servers or clusters without the need to install additional software on a resource. However, it also brings some issues. For example, sometimes it can be desirable to restrict commands that can be run by the platform, or a user can't provide full access to its account due to resource usage policy. This approach also doesn't support integration with resources that are not accessible remotely, such as desktop computers or resources behind a firewall.

The second approach addresses the mentioned issues by running a special agent on each attached resource. The agent acts as a mediator between the platform and the resource. This approach requires deployment of additional software on resources, but enables implementation of arbitrary security policies on the agent level and integration with resources behind a firewall. The communication between an agent and the platform is implemented through the WebSocket protocol (Fette and Melnikov, 2011). Upon startup an agent initiates connection with the platform to establish a bidirectional communication channel. This channel is used only for control and status messages. Job data transfer is performed by an agent via the HTTP protocol.

Currently the Compute subsystem doesn't perform resource selection during the submission of compute job. It is assumed that each service has only one resource linked with it. A user can override this resource with another one in a job request. In any case, the job specification passed to the Compute contains a single resource reference.

3.5 Data Storage

Data storage component implements long-term storage of information related to all entities managed by the platform. It is based on MongoDB (MongoDB, 2013), a document-oriented database system. Native support for JSON data structures with dynamic schemas proved to be useful during the platform development. The data storage also relies on GridFS feature of MongoDB for storing job data and other files.

4 CONCLUSIONS

The paper presented Everest, a cloud platform that supports development and hosting of computational web services. In comparison with existing work, Everest has a number of novel features such as the use of PaaS model, flexible binding of services with externally provisioned computing resources and remotely accessible API. While the platform doesn't provide its own infrastructure to run compute jobs as classic PaaS examples, it can handle the problems of resource allocation, job management, data transfer and so on without the interference of users.

Everest is work in progress. The platform is currently undergoing experimental evaluation and pilot deployment. The results of this work and application case studies will be presented in future publications. Future work will also address remaining gaps in platform's functionality and other challenges, such as development of programming APIs, supporting service composition, implementation of job scheduling mechanism enabling binding of multiple resources to a service, integration with grid infrastructures, and optimization of data transfer for services handling large amounts of data.

ACKNOWLEDGEMENTS

The work is supported by the Russian Foundation for Basic Research (grant No. 14-07-00309 A).

REFERENCES

Foster, I. (2005). Service-Oriented Science. *Science*, 308(5723), 814-817.

Kacsuk, P. (2011). P-GRADE portal family for grid infrastructures. *Concurrency and Computation: Practice and Experience*, 23(3), 235-245.

Miller, M. A., Pfeiffer, W., & Schwartz, T. (2010). Creating the CIPRES Science Gateway for inference of large phylogenetic trees. In *Gateway Computing Environments Workshop (GCE)*, 2010 (pp. 1-8). IEEE.

McLennan, M., Kennell, R. (2010). HUBzero: A Platform for Dissemination and Collaboration in Computational Science and Engineering. *Computing in Science and Engineering*, 12(2), pp. 48-52.

ProgrammableWeb (2013). *ProgrammableWeb - Mashups, APIs, and the Web as Platform.* http://www.programmableweb.com/.

Yu, J., Benatallah, B., Casati, F., & Daniel, F. (2008). Understanding mashup development. *Internet Computing*, IEEE, 12(5), 44-52.

Afgan, E., Goecks, J., Baker, D., Coraor, N., Nekrutenko, A., Taylor, J. (2011). Galaxy - a Gateway to Tools in e-Science. In: K. Yang, Ed. (ed) *Guide to e-Science: Next Generation Scientific Research and Discovery*, Springer, pp. 145-177.

Delaitre, T., Kiss, T., Goyeneche, A., Terstyanszky, G., Winter, S., Kacsuk, P. (2005). GEMLCA: Running Legacy Code Applications as Grid Services. *Journal of Grid Computing*, Vol. 3. No. 1-2, pp. 75-90.

Krishnan, S., Clementi, L., Ren, J., Papadopoulos, P., Li, W. (2009). Design and Evaluation of Opal2: A Toolkit for Scientific Software as a Service. In *2009 IEEE Congress on Services (SERVICES-1 2009)*, pp.709-716.

Afanasiev, A., Sukhoroslov, O., Voloshinov, V. (2013). MathCloud: Publication and Reuse of Scientific Applications as RESTful Web Services. In *Parallel Computing Technologies (PaCT 2013). Lecture Notes in Computer Science*, Vol. 7979, Springer, pp. 394-408.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999). *Hypertext transfer protocol—HTTP/1.1*. Internet RFC 2616.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Ph.D. dissertation, University of California, Irvine.

Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., & Weerawarana, S. (2002). Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI. *Internet Computing*, IEEE, 6(2), 86-93.

Richardson, L., & Ruby, S. (2008). *RESTful Web Services*, O'Reilly.

JSON Schema (2013). *JSON Schema and Hyper-Schema.* http://json-schema.org/.

Jones, M., & Hardt, D. (2012). *The OAuth 2.0 Authorization Framework: Bearer Token Usage*. RFC 6750.

Fette, I., & Melnikov, A. (2011) *The WebSocket Protocol*. RFC 6455, Internet Engineering Task Force.

MongoDB (2013). *MongoDB*. http://www.mongodb.org/

Economical Aspects of Database Sharding

Uwe Hohenstein[1] and Michael C. Jaeger[2]
[1]Siemens AG, Corporate Technology, CT RTC ITP SYI-DE, Otto-Hahn-Ring 6, D-81730 Muenchen, Germany
[2]Siemens AG, Corporate Technology, CT CSG SWI OSS, Otto-Hahn-Ring 6, D-81730 Muenchen, Germany
{uwe.hohenstein, michael.c.jaeger}@siemens.com

Keywords: Public Cloud, Database Sharding, Economical Aspects, Scale Out, NoSQL.

Abstract: Database sharding is a technique to handle large data volumes efficiently by spreading data over a large number of machines. Sharding techniques are not only integral parts of NoSQL products, but also relevant for relational database servers if applications prefer standard relational database technology and also have to scale out with massive data. Sharding of relational databases is especially useful in a public cloud because of the pay-per-use model, which already includes licenses, and the fast provisioning of virtually unlimited servers. In this paper, we investigate relational database sharding thereby focussing in detail on one of the important aspects of cloud computing: the economical aspects. We discuss the difficulties of cost savings for database sharding and present some surprising findings on how to reduce costs.

1 INTRODUCTION

Recently, a new storage technology named NoSQL has come up (NoSQL, 2013). The NoSQL idea takes benefit, among others, from placing data on several nodes, being able to store massive data just by adding nodes and thereby parallelizing operations. However, there are applications in industrial environments that want or have to retain standard relational database systems (RDBSs) because of its ad-hoc query capability and the query power of SQL – both mostly missing in NoSQL – besides being a well-established and mature technology.

In contrast to the *scale out* of NoSQL products, RDBSs are designed to *scale up* by using bigger machines, several disks and advanced concepts such as table partitioning, frequently sticking to a single RDBS server for query processing. However, there is always a bleeding edge of what is possible so that one cannot "go big" forever (Kharchenko, 2012).

Database sharding for larger data volumes is one solution to scale out. The tables are sharded, i.e., are split by row and spread across multiple database servers, the *shard members* or *partitions*, according to a distribution key. The distribution key specifies a range value for each shard member. The main advantage of database sharding is the ability to grow in a linear fashion as more servers are included to the system. Another positive effect is that each database shard can be placed on separate hardware thus enabling a distribution of the database over a large number of machines, resulting in less competition on resources such as CPU, memory, and disk I/O. This means that the database performance can be spread out over multiple machines taking benefit from high parallelism. Since the number of rows in each table in each database is also reduced, performance is improved by having smaller search spaces and reduced index sizes. In addition, if the partitioning scheme is based on some appropriate real-world segmentation of the data, then it may be possible to infer and to query only the relevant shard. Moreover, there will be an increase of availability since data is distributed over shards: If one shard fails, other shards are still available and accessible. And finally, (Kharchenko, 2012) shows that sharding can save costs compared to scaling up.

RDBSs, and particularly RDBS sharding, make also sense for cloud applications. In fact, RDBS products are available in public cloud PaaS offerings: Microsoft Azure SQL Database (SQL Server in the Microsoft Cloud), Amazon RDS (supporting MySQL, Oracle, Microsoft SQL Server, or PostgreSQL systems), or offerings from IBM, Oracle, and others. These products are similar to on-premises servers, however, managed by the cloud provider. This reduces administrative work for users. RDBSs as well as other resources can be provisioned in short time, and resources are principally virtually unlimited (Armbrust, 2010). Due to a pay-

as-you-go principle, users have to pay only for those resources they are actively using, on a timely basis. Hence, it is easy to rent several RDBS instances in order to scale out and to overcome size limitations. There is also no need to take care of licenses since they are already part of the PaaS offerings and captured by the cost model. However, some limitations such as a 150GB limit for Azure SQL databases and a couple of functional restrictions with regard to on-premises products have to be taken into account.

The idea of database sharding is principally well investigated, for instance, elaborating on sharding strategies (Obasanjo, 2009) or discussing challenges such as load balancing (Louis-Rodríguez, 2013) and scaling (Kharchenko, 2012). A lot of work tries to compare distributed relational databases with NoSQL (Cattell, 2011). Anyway, comparisons between RDBSs and NoSQL sometimes appear as ideological discussions rather than technical argumentations, as pointed out by (Kavis, 2010) (Moran, 2010). Sharding is also partially supported by cloud providers. In the Azure federation approach, for instance, special operations are available to split and merge shards in a dynamic manner with no downtime; client applications can continue accessing data during repartitioning operations with no interruption in service.

In this paper, we tackle an important industrial aspect: the operational costs of database sharding in the cloud. In general, existing work on economical aspects of cloud computing is very few (cf. Section 2). Most publications and white papers rather relate to a Total Cost of Ownership (TCO) comparison between on premise and cloud deployments. But there is only little work such as (Hohenstein, 2012) on reducing costs in the cloud – if the decision for the cloud has been taken. Hence, we focus on cost aspects of database sharding in the cloud in detail.

The remainder of this paper is structured as follows. Section 2 provides an overview of related work and detects the lack of research. Afterwards, cost considerations for a simple, but common pricing model are made in Section 3. In particular, we discuss the difficulties of how to reduce storage costs by sizing and splitting partitions appropriately. Section 4 tackles another aspect of sharding that must not be neglected – performance issues. Indeed, cost and performance must be balanced. Further architectural challenges of database sharding are presented in Section 5 work before Section 6 concludes and discusses some future work.

2 RELATED WORK

Although (Armbrust, 2010) identifies short-term billing as one of the novel features of cloud computing and (Khajeh-Hosseini, 2010) considers costs as one important research challenge for cloud computing, only a few researchers have investigated the economic issues around cloud computing from a consumer and provider perspective. Even the best practices of cloud vendors, for instance, (Microsoft, 2013) and (Kharchenko, 2012), do not particularly indicate how to reduce costs.

Most work focuses on cost comparisons between cloud and on-premises and lease-or-buy decisions. (Walker, 2009) compares the costs of a CPU hour when it is purchased as part of a server cluster, with when it is leased for two scenarios. It turned out that buying is cheaper than leasing when CPU utilization is very high (over 90%) and electricity is cheap. To widen the space, further costs such as housing the infrastructure, installation and maintenance, staff, storage and networking must be taken into account.

(Klems, 2009) provides a framework that can be used to compare the costs of using a cloud with an in-house IT infrastructure. Klems also discusses some economic and technical issues that need to be considered when deciding whether deploying systems in a cloud makes economic sense.

(Assuncao, 2009) concentrates on a scenario of using a cloud to extend the capacity of locally maintained computers when their in-house resources are over-utilized. They simulated the costs of using various strategies when borrowing resources from a cloud provider, and evaluated the benefits by using the Average Weighted Response Time (AWRT) metrics (Grimme, 2008).

(Kondo, 2009) examines the performance trade-offs and monetary cost benefits of Amazon AWS for volunteered computing applications of different size and storage.

(Palankar, 2008) uses the Amazon data storage service S3 for scientific intensive applications. The conclusion is that monetary costs are high because the service covers scalability, durability, and performance – services to be paid but often not required by data-intensive applications. In addition, (Garfinkel, 2007) conducts a general cost-benefit analysis of clouds without any specific application.

(Deelman, 2008) highlights the potentials of using cloud computing as a cost-effective deployment option for data-intensive scientific applications. They simulate an astronomic application and run it on Amazon AWS to investigate the performance-cost tradeoffs of different internal

execution plans by measuring execution times, amounts of data transferred to and from AWS, and the amount of storage used. They found the cost of running instances to be the dominant figure in the total cost of running their application. Another study on Montage (Berriman, 2010) concludes that the high costs of data storage, data transfer and I/O in case of an I/O bound application like Montage makes AWS much less attractive than a local service.

(Kossmann, 2010) performs the TPC-W benchmark for a Web application with a backend database and compares the costs for operating the web application on major cloud providers, using existing relational cloud databases or building a database on top of table or blob storages.

(Hohenstein, 2012) showcases with concrete examples how architectures impact the operational costs, once the decision to work in the cloud has been taken. Various architectures using different components such as queues and table storage are compared for two scenarios implemented with the Windows Azure platform. The results show that the costs can vary dramatically depending on the architecture and the applied components.

Concerning sharding, (Biyikoglu, 2011) tackles the question "how do you cost optimize federations?" for Azure SQL federations. The paper recommends consolidating storage to fewer members for cost conscious systems, thus saving on cost but risk higher latency for queries. For mission critical workloads, more money should be invested in order to spread too many smaller members for better parallelism and performance. Since every application's workload characteristics are different, there is no declared balance-point. Thus, Biyikoglu recommends testing the workload and measure the query performance and cost under various setups.

To our knowledge, trying to reduce costs within a certain cloud is not well investigated. We tackle this deficit by optimizing database sharding cost.

3 COST CONSIDERATIONS

This paper is mainly concerned with an economical perspective of database sharding in the cloud since costs are relevant for industrial applications. In fact, pay-as-you-go is one important characteristic of cloud computing (Armbrust, 2010). As (Hohenstein, 2012) pointed out, there is a strong need for cost saving strategies in a public cloud. One important question we are investigating in this context is: how to size shards for achieving optimal costs?

3.1 A Sample Pricing Model

The pricing models of public cloud providers are mainly based upon certain factors such as storage, data transfer in and out etc. Most have in common that the more resources one occupies or consumes, the less expensive each resource becomes.

In the following, we use the pricing model of a popular public cloud provider. We keep the name anonymous because we do not want to promote one specific provider. Anyway, our main statements can be transferred to other providers analogously even if the price models and the relevant factors differ. Our main intention is to illustrate the challenges with pricing models in the public cloud.

In the pricing model, each database in use has to be paid depending on the storage consumption, i.e., the size of the database:

- 0 to 100 MB: $4.995 (fix price)
- 100 MB to 1 GB: $9.99 (fix price)
- 1 GB to 10 GB: $9.99 for the first GB, $3.996 for each additional GB
- 10 GB to 50 GB: $45.96 for the first 10 GB, $1.996 for each additional GB
- 50 GB to 150 GB: $125.88 for the first 50 GB, $0.999 for each additional GB

Hence, a 20GB is charged with $65.92; $45.96 for the first 10GB and 10*$1.996 for the next 10GB. Similarly, an 80GB database costs $155.85 ($125.88 for the first 50GB and 30*0.999 for the next 30GB), while 150GB sum up to $225.78. These prices are on a monthly basis. The maximal amount of data is measured every day, and each day is charged according to the monthly fee. Hence, Using 10GB for the first ten days and 60 GB for the next 5 days is charged with $37.96 ($45.96*10/30 + $135.87*5/30) for 15 days. Also note that pricing occurs in increments of 1GB. Hence, 1.1GB is charged as 2GB.

3.2 Simple Considerations

In case of database sharding, each database shard is charged that way. Unfortunately, reducing costs for sharding is not that simple as it seems to be.

We start with some simple considerations for the above pricing model. Let us first compare ten 10GB databases (a) with one 100GB database (b). Both are able to store 100GB, but the prices differ a lot:

a) 10 * 10GB à $45.96 each = $459.60
b) 1 * 100GB = $175.83

That is, one large partition is 3.8 times cheaper in a month than 10 smaller ones making up a difference

of $284. The monthly difference is even larger in the following case of storing 1,200GB:

a) 120 * 10GB = 120 * $45.96 = $5,515.20

b) 8 * 150GB = 8 * $225.78 = $1,806.24

Here, the factor is about 3 or absolutely $3,700 a month! These examples show clearly that it is more economic to use a few databases of larger sizes than more small-size databases.

Unfortunately, smaller databases provide a better query performance the effect of which is even stronger in the cloud since databases reside on different servers having their own resources. Both setups have access to the same storage capacity, however clearly 120x10GB databases have access to 15 times more cores, memory and IOPS capacity as well as temporary space and log files. Thus, the costs must be balanced with performance requirements to be achieved.

Let us assume that performance measurements detect that 100GB is a reasonable partition size regarding given performance requirements.

The previous analysis referred to a rather static view. For dynamic data, the question arises how to split a partition if the 100GB limit is exceeded. The most intuitive split will certainly be balanced with 50/50%. But is this really the best choice?

a) 50 + 50 GB: 2 * $125.88 = $251.76

b) 80 + 20 GB: $155.85 + $65.92 = $221.77

The split into 80 and 20 GB (b) is $30 cheaper, with the effect that the first partition will again flow over after 20 additional GB: this is not really a cost problem but the performance might be affected by too many splits and unbalanced partitions.

If we now add further 30GB to both partitions in this scenario, we obtain the following costs: In case of (a), 15 GB are put into both partitions equally:

a) 65GB + 65GB: 2 * $140.865 = $281.73

In case of (b), we add 24GB to the first and 6GB to the second partition, assuming an equal distribution of the new 30GB according to the previous split ratio 80/20%. This means another split becomes necessary for the 80+24=104GB database resulting in 83,2GB and 20,8GB partitions. Now, (b) turns out to be $24 more expensive:

b) $159.846 (83.2GB + $67.916 (20.8GB) + $77.896 (26GB) = $305.658

3.3 Long-term Comparison

Obviously, it is necessary to calculate the cumulated costs over a longer period of time. As a more elaborated example, we start with 100GB (exactly the partition limit), and assume a daily increase of 1 GB, equally distributed over the partition key.

Table 1 summarizes the database sizes and the costs for the first 50 days for a 50/50% split ratio.

Table 1: Costs for 50/50% ratio.

day	databases	costs for day x	cumulated costs
1	2 * 50.5	$253.758 / 30	$8.45
2	2 * 51	$253.758 / 30	$16.91
…			
10	2 * 55	$261.750 / 30	$85.91
…			
20	2 * 60	$271.740 / 30	$175.16
…			
25	2 * 62.5	$277.7340 / 30	$221.05
26	2 * 63	$277.7340 / 30	$230.31
…			
30	2 * 65	$281.730 / 30	$267.74
…			
40	2 * 70	$291.720 / 30	$363.65
…			
50	2 * 75	$301.710 / 30	$462.89

Please note that the costs for the "price for day x" column should be divided by 30 ("/ 30") in order to obtain the daily costs. The cumulated costs sum up the daily costs until day x. Keep also in mind that database sizes are rounded up to full GBs, i.e., 50.5GB are taken as 51GB and charged with $253.758 per month.

Table 2: Costs for 80/20% ratio.

day	databases	costs for day x	cumulated costs
1	80.8 + 20.2	$224.765 / 30	$7.49
2	81.6 + 20.4	$225.764 / 30	$15.01
…			
10	88 + 22	$233.754 / 30	$76.51
…			
20	96 + 24	$245.738 / 30	$157.03
…			
25	100 + 25	$251.73 / 30	$198.78
26	80.64 + 20.16 (split!) + 25.2	$302.661 / 30	$208.87
…			
30	83.20 + 20.80 + 26	$305.658 / 30	$249.46
…			
40	89.60 + 22.40 + 28	$319.636 / 30	$354.21
…			
50	96.00 + 24.00 + 30	$331.618 / 30	$463.42

Table 2 summarizes the database sizes and the costs for the first 50 days for an 80/20% ratio, being applied recursively: Each partition is again split with

this ratio. The total costs are lower for the 80/20% ratio during the first days: There is a benefit of $0.96 for the first day, $18.13 for the first 20 days, and $18.28 for the first 30 days. But we notice a change at day 50: the 50/50% ratio becomes cheaper.

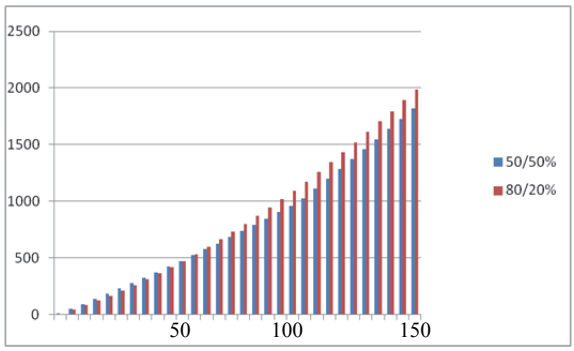


Figure 1: Long-term comparison of cumulated costs.

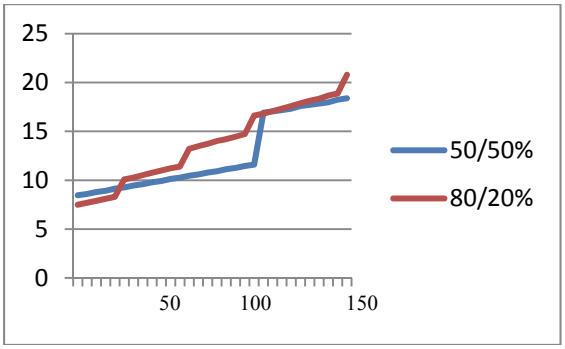


Figure 2: Long-term comparison of daily prices.

Figure 1 compares the total costs for the first 150 days. It shows that 50/50% becomes cheaper and stays cheaper after 50 days; the difference is even increasing day by day.

As Figure 2 shows, the reason is that the price for each day increase a lot from day 25 to 26 (cf. also Table 1 and 2). After 26 days, the daily price of the 50/50% ratio becomes cheaper for each successive day; after day 56, the gap gets even larger due to an additional increase at that day.

If we dive into the details, we notice that the number of smaller partitions has increased at those days. In fact, a split occurs at day 26 for the 80/20% ratio: A partition with 100.8GB is split into 80.64GB and 20.16GB. This means three partitions of 80.64GB, 20.16GB, and 25.2GB occur at day 26, with a daily price of $302.661 / 30 in contrast to $251.73 / 30 the day before (cf. Table 2). This is an increase of $1.69. We now pay $10.08 for that day compared to $9.25 for the 50/50% ratio. Indeed, the small partitions are dominating the costs, and at day 50 the total costs for 50/50% start to become cheaper. Later on, further splits occur and affect the comparison more negatively.

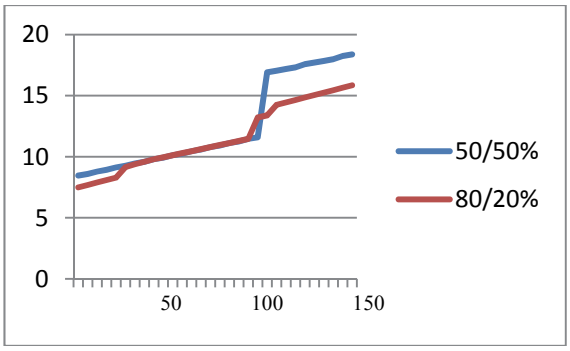


Figure 3: Long-term comparison of cumulated costs (with merge strategy).

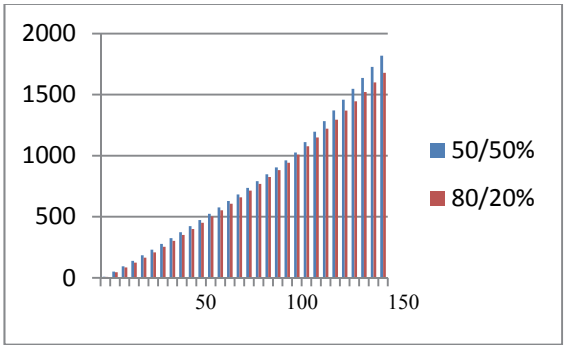


Figure 4: Long-term comparison of daily prices (with merge strategy).

3.4 Improvement

Anyway, there are alternatives to improve costs. The basic idea to reduce costs is to get rid of smaller partitions by means of merging them. In the previous scenario, we can merge the two smaller partitions of 20.16GB and 25.2GB at day 26 right after the split:

- partitions with 20.16GB and 25.2GB: $145.812
- one merged partition with 45.36GB: $117.816

Thus, merging partitions reduces the costs at day 26 by $27.996/30 = 93ct. Using this merge strategy for the evaluation, we can improve the daily price dramatically (cf. Figure 3). The costs for 80/20% remain below 50/50% with a few exceptions. This leads to the cost comparison in Figure 4. Using the 80/20% ratio becomes always cheaper and the savings to 50/50% increases day by day. Now, we benefit from the 80%/20% ratio even in the long run.

We conclude this section with demonstrating what monetary benefit can be achieved by choosing the best ratio compared to a 50/50% split:

- $13.98 savings for 100 days
- $1239.48 savings for 500 days
- $5090.39 savings for 1000 days

One important issue remains open: What is the optimal split ratio? We implemented an algorithm to find out optimal ratios for scenarios given by initial size, daily increase, and number of days. It turned out that a ratio between 0.8 and 0.9 will be most cost-efficient for 100GB shards and 1GB increments. The optimal ratio varies from day to day, but it mostly stays in this interval. Even if the split ratio does not strike the optimum, costs can be reduced compared to a 50/50% split.

Other partition sizes and/or increments certainly lead to other results. But most of our tested scenarios do not let the 50/50% ratio be the best solution.

In principal, it is necessary to make an analytical investigation for finding a formula and making a lim $1 \rightarrow \infty$ calculation; this is subject to future research.

4 PERFORMANCE

Saving costs is certainly one important driver in industrial use cases. Anyway, the cost reduction must also be seen in the context of performance. Performance requirements will affect the size of partitions. Thus, this section presents some basic performance results. We essentially compare using ten (smaller) 1GB databases with using one (larger) 10GB database.

4.1 Test Setup for Ten 1GB Databases

We use the following table structure for each of the shard members: `ShardedTable(k int, id int, t10 int, t100 int, tf500 float, v100 varchar(100), v200 varchar(200), v300 varchar(300), doc xml).`

Each of the 10 shard members contains 1,000,000 records, summing up to 10GB including indexes for all 10 shards. Column `k` is the primary key, which is globally unique over all partitions. Each partition has different ranges of keys, i.e., the first partition k=0..999,999, the second one with k=1,000,000..1,999,999, and so on. In contrast, `id` is a successive number that is only locally unique within each shard. Column `doc` contains an XML document representing the complete record as XML.

We implemented a REST service that executes arbitrary queries with parameterized values. The REST service runs in a VM of its own in a public cloud, in the same data centre as the database shards. The service URL has the following structure: `http://<MyServer>/QueryService/query ?txt=<query>&param=@k&value=80000`

We can perform a parameterized <query> such as `SELECT * FROM ShardedTable WHERE k=@k` from any Web browser; value for parameter `@k` is here 80000. There is an optional request parameter `db` that allows for specifying a single database (as in case of the 10 GB database). Without, a thread pool is spanned for executing the same query on all shards in parallel. We certainly measure only the execution time within the REST server, i.e., without the latency to the query service. The execution times are returned as part of the REST response.

4.2 Test Setup for One 10GB Database

To keep the same data volume as in 4.1, the 10GB database is set up with 10,000,000 records enumerated from 0 to 9,999,999. The table structure is the same as before, however, `k` is the primary key for all the records while each `id` value refers to 10 records.

4.3 Test Scenarios

We investigated and compared the performance of three scenarios:

a) Access to all ten 1GB shards in parallel
b) Access to one 10GB database
c) Access to one single 1GB shard

The last scenario is relevant if the shard to be queried is known; then no parallel query has to be executed.

The measurements are taken ten times each at different times of a day. This explains the broader variance in the results.

We first present the very first execution times of requests, i.e., no caching will take place. Table 3 summarizes the results for searching a primary key value (column `k`). In any case, the result will be a single record, i.e., the record will be found in one of the shards. As expected, searching a single shard (if the partition is known) is performing best. But it is surprising that an (index-based) search in 10GB is faster than parallelizing ten requests to 1GB shards and combining the results.

Table 4 presents the execution times for accessing records with a given `id` value. The `id` column is *not* indexed! In case of sharding, each of the ten shards returns a single record; the 10GB database returns 10 records analogously. Now, sharding produces 4 times better results than a 10GB database. Again, searching one record in a single 1GB database is outstanding, but irrelevant since

each shard contributes to the overall result.

Table 3: Global key search.

Scenario	Average elapsed time [ms]	Standard deviation
10 * 1GB	622.9915	241.6767746
1 * 10GB	437.4642	786.1302572
1 * 1GB	166.0001	255.7591507

Table 4: Full table scan.

Scenario	Average elapsed time [ms]	Standard deviation
10 * 1GB	5950.63548	2402.86658
1 * 10GB	22161.99186	6534.50578
1 * 1GB	1050.68536	708.21466

Table 5 presents the results for searching an *indexed* value (column t10) in all partitions. Each sharded query returns 10 records, i.e., 100 in total; the 10GB database returns 100 records as well. It is again surprising, that the 10GB database beats the sharded query. Even the difference to a single 1GB query is small.

Table 5: Index-based search.

Scenario	Average elapsed time [ms]	Standard deviation
10 * 1GB	558.54096	684.1157257
1 * 10GB	189.43505	82.4324732
1 * 1GB	107.41177	64.3837701

To complete the analysis, we measured the average execution times for each query, being executed 5 times successively and excluding the very first execution. Table 6 summarizes the results for those "hot" queries, essentially confirming the previous results.

Table 6: Successive accesses (in ms).

Scenario	Global Key Search	Full Table Scan	Index-based Search
10 * 1GB	50.2883937	1859.204	68.542061
1 * 10GB	16.6000000	5691.369	30.169731
1 * 1GB	12.2059406	671.324	8.788243

Thus, sharded queries might be expensive due to the overhead of multithreading and consolidating results. This means that partitions might be chosen larger than expected in order to save costs. However, we do not want to conclude here that using parallel queries are less performing in general. Of course, there are further aspects such as the size of the VM to run the sharding layer, the thread pool, the PaaS offering, means for co-location etc. Rather we want to encourage developers to perform performance tests by their own. As we stated in (Hohenstein, 1997), it is absolutely recommended to perform application-specific benchmarks for meaningful results taking into account the applications' characteristics. Here, we simply wanted to get a first impression about the performance behaviour.

5 FURTHER CHALLENGES

From an architectural point o view, it is a good idea to introduce a sharding layer the purpose of which is to distribute queries over member shards and to consolidate responses. There are a couple of further issues we want to mention briefly.

The sharding layer has to take care of connection handling. In fact, each database has a connection string of its own, which usually means that each shard obtains a pool of its own. Thus, the sharded connections are often not pooled in a shared pool.

There is a need for a *shared* meta database directory that contains all configuration information, in particular, about the physical shards, their connection string etc.

The distribution scheme is important. If it is not well-designed for the major use cases, then global transactions and distributed joins across shards cannot be avoided and the sharding layer has to take care. Similarly, cross-shard sorting and aggregation is another challenge for results from different shards.

Another important issue is certainly schema evolution, i.e., changes to the table structures. Any schema upgrade must be rolled out to all members.

Rebalancing of shards (removing and adding shards) is a performance issue. Usually, databases have to be created or deleted and data has to be moved between databases.

Finally, we want to mention certain technical issues such as policies for key generation, shard selection, construction of queries and operations.

6 CONCLUSIONS

Sharding of RDBSs is relevant if applications have to stick to relational database technology, but scalability or Big Data issues arise at the same time. Sharding enables one to add additional database servers to handle growing data volumes and to increase scalability. This is particularly of interest for cloud computing environments because of the pay-as-you-go principle and the ease to provision

new database servers in short time.

In this paper, we investigated database sharding of relational database systems (RDBS) in the cloud from the perspective of cost and performance. We obtained some surprising results.

At first, splitting shards into two equally sized shards is not always advantageous from a cost perspective. Other split factors such as 80/20%, combined with a merge operation, yield better results in our scenarios. Anyway, we demonstrated that achieving optimal costs is difficult in general.

Furthermore, performance measurements show that parallelizing queries to several shards is not always better than querying a single database of the same total size.

In the future, we intend to further elaborate on strategies to split optimally according to incoming load. In particular, cost/performance considerations require further investigations. And finally, we want to apply our ideas to multi-tenancy.

REFERENCES

Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I. and Zaharia, M. (2010): A View of Cloud Computing. *CACM*, 53(4), April 2010.

Assuncao, M., Costanzo, A. and Buyya, R. (2009). *Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters*. In *HPDC '09: Proc. of 18th ACM int. symposium on High performance distributed computing*, Munich, Germany, June 2009.

Berriman, B., Juve, G., Deelman, E., Regelson, M. and Plavchan, P. (2010). The Application of Cloud Computing to Astronomy: A Study of Cost and Performance. *6th IEEE Int. Conf. on e-Science*.

Biyikoglu, C. (2011): *Pricing and Billing Model for Federations in SQL Azure Explained!* http://blogs.msdn.com/b/cbiyikoglu/archive/2011/12/12/billing-model-for-federations-in-sql-azure-explained.aspx

Cattell, R. (2011): Scalable SQL and NoSQL Data Stores. *ACM SIGMOD Record*, Vol. 39(4).

Deelman, E., Singh, G., Livny, M., Berriman, B. and Good, J. (2008). The cost of doing science on the cloud: the Montage example. In *Proc. of 2008 ACM/ IEEE conf. on Supercomputing*, Oregon, USA, 2008.

Garfinkel, S. (2007). Commodity Grid Computing with Amazon S3 and EC2. In *login 2007*.

Greenberg, A., Hamilton, J., Maltz, D. and Patel, P. (2009). The Cost of a Cloud: Research Problems in Data Center Networks. *ACM SIGCOMM Computer Communication Review,* 39, 1.

Grimme, C., Lepping, J. and Papaspyrou, A. (2008). Prospects of Collaboration between Compute Providers by means of Job Interchange. In *Proc. of 13th Job Scheduling Strategies for Parallel Processing,* April 2008, LNCS 4942.

Hamdaqa, M., Liviogiannis, L. and Tavildari, L. (2011): A Reference Model for Developing Cloud Applications. *Int. Conf. on Cloud Computing and Service Science (CLOSER) 2011.*

Hohenstein, U., Krummenacher, R., Mittermeier, L. and Dippl, S. (2012*): Choosing the Right Cloud Architecture - A Cost Perspective. CLOSER'2012.*

Hohenstein, U., Plesser, V., Heller, R. (1997): Evaluating the Performance of Object-Oriented Database Systems by Means of a Concrete Application. *DEXA 1997.*

Kavis, M. (2010): *NoSQL vs. RDBMS: Apples and Oranges.* http://www.kavistechnology.com/blog/nosql-vs-rdbms-apples-and-oranges.

Khajeh-Hosseini, A., Sommerville, I. and Sriram, I. (2011). Research Challenges for Enterprise Cloud Computing. *1st ACM Symposium on Cloud Computing, SOCC 2010*, Indianapolis.

Kharchenko, M. (2012): *The Art of Database Sharding.* http://intermediatesql.com/wp-content/uploads/2012/04 /2012_369_Kharchenko_ppr.doc

Klems, M., Nimis, J. and Tai, S. (2009). Do Clouds Compute? A Framework for Estimating the Value of Cloud Computing. *Designing E-Business Systems. Markets, Services, and Networks*, Lecture Notes in Business Information Processing, 22.

Kondo, D., Javadi, B., Malecot, P., Cappello, F. and Anderson, D. P. (2009). Cost-Benefit Analysis of Cloud Computing versus Desktop Grids. In *Proc. of the 2009 IEEE Int. Symp. on Parallel&Distributed Processing*, May 2009.

Kossmann, D., Kraska, T. and Loesing, S. (2010). An Evaluation of Alternative Architectures for Transaction Processing in the Cloud. *ACM SIGMOD 2010*

Louis-Rodríguez, M., Navarro, J., Arrieta-Salinas, I., Azqueta-Alzuaz, A. Sancho-Asensio, A. and Armendáriz-Iñigo, J. E.: Workload Management for Dynamic Partitioning Schemes in Replicated Databases. *CLOSER'2013*.

Moran, B. (2010): *RDBMS vs. NoSQL: And the Winner is…* http://sqlmag.com/sql-server/rdbms-vs-nosql-and-winner.

Microsoft (2013): *Windows Azure .Net Developer Center - Best Practices*. http://www.windowsazure.com/en-us/develop/net/best-practices.

NoSQL (2013): *List of NoSQL Databases.* http://nosql-database.org

Obasanjo, D. (2009): *Building Scalable Databases: Pros and Cons of Various Database Sharding Schemes*. http://www.25hoursaday.com/weblog/2009/01/16/BuildingScalableDatabasesProsAndConsOfVariousDatabaseShardingSchemes.aspx.

Palankar, M., Iamnitchi, A., Ripeanu, M. and Garfinkel, S. (2008). Amazon S3 for Science Grids: A Viable Solution? In: *Data-Aware Distributed Computing Workship (DADC),* 2008.

Walker, E. (2009). The Real Cost of a CPU Hour. *Computer,* 42, 4.

Towards a Composition-based APIaaS Layer

Claudio Guidi, Saverio Giallorenzo and Maurizio Gabbrielli
Department of Computer Science and Engineering - DISI, University of Bologna, Bologna, Italy
{cguidi, sgiallor, gabbri}@cs.unibo.it

Keywords: Service-oriented Computing, Cloud Computing, API, Composition, SaaS, PaaS, APIaaS.

Abstract: Application Programming Interfaces (APIs) are a standard feature of any application that exposes its functionalities to external invokers. APIs can be composed thus obtaining new programs with new functionalities. However API composition easily becomes a frustrating and time-costly task that hinders API reuse. The issue derives from technology-dependent features of API composition such as the need of extensive documentation, protocol integration, security issues, etc.. In this paper we introduce the perspective of the API-as-a-Service (APIaaS) layer as tool to ease the development and deployment of applications based on API composition, abstracting communication protocols and message formats. We elicit the desirable features of such a layer and provide a proof-of-concept prototype implemented using a service-oriented language.

1 INTRODUCTION

Application Programming Interfaces (APIs) are a standard feature of any application that exposes its functionalities to external invokers. APIs can be composed to create new applications at client or server side. In the former case composition takes the form of a *mashup* (Liu et al., 2007), whereas in the latter case it is usually realized within a specific application. Depending on the desired result, developers choose for a server- or client-side composition, although there are some cases in which only a server-side application can comply with the requirements of the project, e.g., : (i) the developer needs to publish the composition as a new API; (ii) the composition has some strategic value, therefore the developer does not want to publish it as a readable script at client-side; (iii) APIs could use API keys and other identification data that must remain private and/or pass only through certified channels like VPNs or SSH; (iv) privacy policies and national agreements might enforce a known, certified, and geographically restricted route for data; (v) API contracts require the user to be part of a specific federated network. For these reasons and since our interest regards API composition and publication, in this work we focus on server-side API composition.

Composition of APIs implies the choice of a set of different technologies that span from the implementation language to the communication and serialisation protocols used by the application. The choice on these technologies it is very important since it strongly affects development, deployment, and maintenance of the project. In addition, it is not clear what long-term implications could derive from the choice of a specific set of technologies at design-time. In order to clarify how technology could affect a server-side API composition project, let us consider the example of Fig. 1 which reports the architectural view of a realistic application, dubbed PhotoBlog. PhotoBlog manages the publication of photos and their captions, translated in several languages, on various blogging platforms. PhotoBlog derives from the composition of three APIs: (i) *image*, for image handling (resize, cut, etc.) and storage; (ii) *translation*, for the automatic translation of captions; (iii) *blogging*, for publishing images and captions as posts. Moreover, PhotoBlog itself exposes its own APIs to client interfaces.

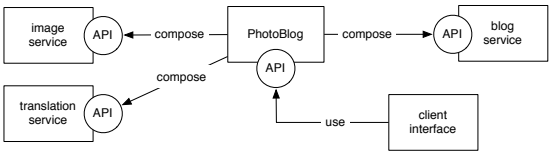


Figure 1: An example of composition of APIs.

Assuming T the technology chosen for the development of the project, the typical implementation of such a composition passes by the following phases:

- *documentation analysis*: the developer must understand the parameters of interaction of each API with T. Documentation analysis usually consists of reading the companion documentation issued

by the API provider;

- *API connection*: the developer must include, or develop by her own, specific libraries for T to handle communication and serialisation protocols with the composed APIs. Notably, each API could require the support of a different protocol;
- *development*: the developer implements the composition in T, employing the specific patterns of composition provided by T;
- *deployment*: the administrator of PhotoBlog deploys the application on a platform of a provider which supports T. Convenience, diffusion, and support for T is another strong concern which influences its choice;
- *maintenance and evolution*: the developer has to maintain the project up-to-date wrt new releases of APIs. API updates could require upgrades of T by either the administrator or the provider of the platform, depending on the service-level agreement.

Even in such a simple example as PhotoBlog, composing APIs requires a lot of "boilerplate" work to enable communication with the composed applications. This diverts the focus of the developer from the core composition functionality to the creation of a suitable environment for composition. In addition such an environment requires an exhausting work of maintenance.

What emerges from our brief analysis is that APIs are a commodity of any application, especially when deployed on the cloud, but the heterogeneity of protocols makes interacting with several applications an unaffordable task, if compared to its yield. The costs of setting a suitable environment easily overcome the potential value of the composition and represent an important drawback of programming web and cloud applications. This fact highlights the strong need for tools that enable efficient development of heterogeneous APIs, either in the cloud or on the Internet in general. However, to the best of our knowledge, currently there is no structured programming patterns nor formal theories which deal with API server-side composition in a cloud environment.

In this paper we try to attack this problem by introducing the notion of an API-as-a-Service (APIaaS) layer as a tool which eases the development and deployment of applications based on API compositions. By doing this, we also elicit the desirable general features of such a layer. Moreover, we also provide a proof-of-concept APIaaS prototype based on a service-oriented approach. In our vision, the APIaaS layer is a PaaS layer which provides all the tools for facilitating API composition, in particular by abstracting communication protocols and message formats. Service-oriented orchestration programming could be adopted as a programming reference for abstracting technological details and focusing on composition.

In § 2 we enhance the PhotoBlog example in order to list all the problems related on API composition. In § 3 we list all the features of an APIaaS layer should provide, whereas in § 4 we present our proof-of-concept APIaaS prototype which exploits a service-oriented approach both for the programming and the architectural level. Finally, in § 5 we discuss some related work, drawing our conclusions in § 6.

2 PROBLEMS WITH API COMPOSITION

In this section we list and discuss the main difficulties related to the development of applications that use and publish APIs. We extend the example of PhotoBlog, assuming T the technology selected for its implementation.

Documentation. Documentation regards the retrieval and integration of documents that describe APIs for the given technology T. In our example such a step must be performed for the three composed APIs, namely image, translation and blogging. Usually the documentation is described for RESTful JSON messages so, for each service, the developer has first to check the documentation in order to extract information on the identifier, the type, and the semantics of the values of request-response operations (if any). Then the documentation of specific libraries for T must be analysed, when supplied.

This process alone can take an important amount of time (Erl, 2004) when compared to that one of the development. Moreover, when the documentation is partial or missing the developer could end in a time-costly trial-and-error phase to acquire the information on how to interact with an API.

Interoperability. Interoperability concerns the mechanisms and technologies needed for requesting services and exchanging data between different APIs. In our example let us suppose that T natively supports interoperability with the translation and blogging APIs, but needs specific libraries for interacting with the image one. In this situation the developer faces two scenarios:

1. **Including Libraries**. The developer integrates libraries of T which offer ready-to-use functionalities and reduce the workload of the developer.

However library inclusion causes the following drawbacks:

- strong *dependencies* in the project. Once included, dependencies need to be kept up-to-date and changes between versions may lead to major changes in the implementation of the applications;
- *conflicts* among libraries, source of bugs and unwanted side-effects;
- *limited knowledge* of the internal mechanisms of libraries. This constitutes a "grey area" of the project on which the developer has little to no knowledge, adding complexity to the debugging process.

2. **Ad-hoc Solutions**. The developer could spend a significant amount of time in implementing *ad-hoc solutions* to support interoperability with the image API in T. In this case, she would need a comprehensive knowledge on the communication and serialisation protocols employed by the image API.

Publication. Published APIs allow interaction with client interfaces and other server-side applications. Publication concerns both APIs deployment and documentation, which represent two concerns:

- the problems addressed for interoperability hold also for API deployment. High levels of interoperability bring strong dependencies on several libraries, whilst supporting only a few protocols reduces the cost of implementation to the detriment of interoperability;
- publishing an API requires also the publication of its companion documentation. Such documentation is produced ad-hoc and requires a steady commitment as it must be up-to-date wrt the implementation it describes.

Security. Security regards secure transition, storage, and retrieval of data and it is a major issue of any application available on any open network. Implementing and administrating security requires specific knowledge, experience, and skills. Indeed the developer can easily include in T cryptographic libraries for secure communication like TSL and SSL and encryption/decryption libraries for data storage/retrieval. However security still requires a tremendous effort to ensure a specific level of reliability: a developer that chooses to implement and deploy the security of data on its own must be aware of the latest best practices (McGraw, 2004) in security. Moreover, she must invest a lot of effort to check the chain of components that handle sensible data to prevent security exploits (Barber, 2001).

In our example, PhotoBlog must provide APIs that enable users to set persistent configurations on the service. Users have to register to the service, link one or more blogs to their account, and identify themselves before publishing a post. This implies two APIs concerning security: registration/access of users and registration/access to blogs, depicted in Fig. 2.

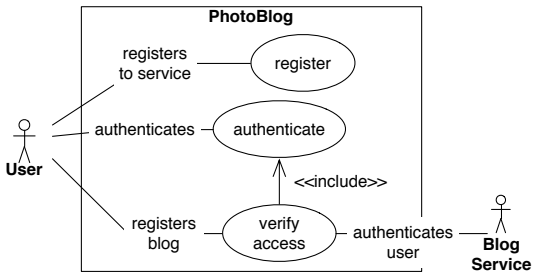


Figure 2: Users and blogs registration.

Monitoring. After PhotoBlog has been deployed, its administrator wants to check the performances of her application. An important step in project development is therefore *monitoring*. Monitorability (Skene et al., 2007) is a major concern when Service-Level Agreements (SLAs) define performances and availability of services with legal consequences such as penalties and compensations. As for security, T probably benefits from libraries to administer monitoring. In PhotoBlog the monitoring function, provided as an API, can serve both clients and its administrator. The administrator can monitor the activities on its application, checking for availability, performances, and resource requirements. Clients can also take advantage of the monitoring feature to assess the availability of the service. In addition, third parties can offer impartial monitoring of services to assess compliance with SLAs.

Portability. Portability is the degree of freedom of an application wrt its dependencies. *Internal* dependencies enforce the usage of libraries whose changes potentially lead to major rewrites of the application. *External* dependencies imply the presence of a specific elements in the running environment of the application. It could be the running environment itself, e.g., a specific interpreter, or other applications that must coexist in the same platform to enable a particular behaviour. In the previous paragraphs we extensively addressed the problems linked to internal dependency, however also external dependencies are a major issue which may lead to a *locked-in* situation. To illustrate this point let us say that the technology T cho-

sen for PhotoBlog is Java, relying on Spring (Johnson et al., 2005) as supporting framework. Once released and deployed by the developer, the administrator of PhotoBlog is forced to choose a running environment that offers Java with Spring. In case the administrator wants to switch to another provider, she must choose one that offers a similar running environment. Otherwise she could spend a significant amount of time in setting the necessary running environment on a generic platform i.e., installing the JVM, the Spring framework, setting it properly, etc..

3 FEATURES OF APIaaS

In this section we discuss the basic features which should characterize a composition-based APIaaS layer in a cloud framework. Notably, APIs can be divided into two groups, based on their implementation: *core* and *composed* APIs. Core APIs belong to applications that directly implement their functionalities without relying on external APIs (e.g., image, translation, and blogging APIs of PhotoBlog). Composed APIs derive from a behavioural composition of other APIs like the PhotBlog API. In our perspective an APIaaS layer should make no distinction in supporting core or composed APIs.

In this work we propose an APIaaS layer as a Platform-as-a-Service layer tailored to cope with the issues of API composition (3*b*). Developers can access such layer by means of private and secure credentials which give access to a private area that supports design of new API compositions starting from other existing core APIs. Developers can search existing APIs in the API registry of the APIaaS layer. APIs come with a formal interface which defines the message types and the behaviour of the interaction. They are also enriched with formal SLA documentation which provides all the necessary information for using/buying them. Each developer can decide to publish her APIs as *private*, *inner* or *public*. Private APIs are visible only by the author, inner APIs are visible only within the same APIaaS platform, whilst public APIs are visible from outside. Notably, inner and public visibility does not imply also accessibility and inner and public APIs can be released for free or under payment. Deployment costs and revenues depend on the business model of the APIaaS provider. Finally each deployed API should be monitorable and its performances visible by the administrator of the API and its clients, guaranteeing impartial SLA conformance.

Coherently with this scenario we propose to divide the APIaaS layer into two main sub-layers: the

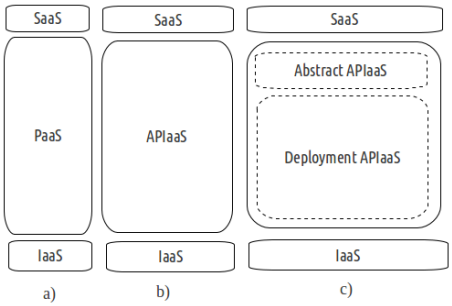


Figure 3: APIaaS layer in the common cloud stack.

Abstract APIAaaS layer and the *Deployment APIaaS* one, as shown in Fig.3*c*. The former sub-layer supports the abstract composition of APIs. The latter provides all the necessary tools for their management. Such a division allows to separate programming issues, considered in the abstract layer, from the architectural ones, addressed by the deployment layer.

In the following, we list and discuss the main desirable features for these two sub-layers.

Abstract APIaaS

- *Composability*: is the availability of the linguistic tools that provide a suitable abstraction for achieving composition;
- *Formal documentability*: is the possibility of documenting and describing each API by means of formal and abstract interface specifications. Such specifications can be used to achieve composition abstracting the details of real communication protocols and message formats. Formal specifications enables the automatic extraction of stubs of descriptive documentation, thus helping developers in API documentation. In addition, semi-automatic production of documentation uniforms the overall documentation easing the interpretation of API documentation by clients. This feature is currently missing in RESTful APIs where there exist no formal nor standard language for defining API interfaces.
- *Portability*: deals with the possibility of moving an API composition from an APIaaS platform to another one without affecting its code. Portability is important to avoid lock-ins on APIaaS providers. Lock-in avoidance is one of the key to guarantee a competitive API market. Moreover, it also allows to separate functionality issues from the one related to deployment and performances.

Deployment APIaaS

- *Protection*: deals with the possibility of composing APIs within the APIaaS layer for inner usage, without publishing them thought a public Software-as-a-Service layer. This feature allows the creation of inner utility APIs used by more complex APIs.
- *Interoperability*: allows the APIaaS layer to connect with different applications and APIs by supporting several communication protocols and message formats (e.g., Http/JSON, Http/SOAP, Java RMI, XML RPC, etc.). This feature allows to connect the formal interfaces of the abstract layer to concrete communication protocols and interoperate with external APIs.
- *Publishability*: deals with the possibility to select and publish a subset of the developed APIs on the public SaaS layer. Such feature allows mashups and applications commonly developed with other technologies to employ the developed APIs.
- *Measurability*: concerns the possibility to measure the performances of APIs in terms of response time, memory usage, served sessions per time unit, etc.. All these information serve for the definition of SLA contracts both between APIaaS providers and API developers and between API developers and their clients. Since APIs can be internally composed, it should be possible to derive minimal SLAs starting from the composition of the SLAs of the composed APIs. As an example consider the case described in § 2. Let us suppose that the image blurring process takes at maximum 500msec whilst the process of image posting takes at maximum 200msec. Provided these data, a developer can guarantee in PhotoBlog's SLA that the response time of its image posting API has a upper bound of 700msec.
- *Sandboxing*: relates to the *protection* feature and extends it. In particular sandboxing of code enables each developer to access a different environment which runs and contains all her code. In the sandboxing environment the developer defines the visibility of its compositions i.e., which should be visible internally or which have public visibility for external usage.
- *Security*: deals with different aspects of security of the APIaaS framework, i.e., : *i*) *access security*; *ii*) *private data consistency*; *iii*) *malware prevention*. Access security deals with well-known security issues like authentication and access token release to allow clients access APIs; private data consistency deals with issues related to corruption of private data performed by malicious activities from the outside (e.g., hacking attacks that alter private data); malware prevention deals with issues related to malicious behaviour of the deployed composed APIs.
- *Scalability*: deals with the possibility to scale the APIaaS architecture depending on the number of subscribed developers and the number of developed APIs.
- *Federability*: deals with the possibility of federating different APIaaS layers in order to share the inner developed APIs among different providers. Such a feature allows for the creation of a horizontal layer among different cloud infrastructures and provides the abstraction of a unique and common development framework for APIs. Such federation could be useful for the creation of groups of specialised APIaaS providers which maximise variety, efficiency, and security of offered APIs. As an example, let us consider the case of a Smart City that providers the federation of three different APIaaS to citizens and companies: the municipality APIaaS, the main hospital APIaaS, and the public transportation company APIaaS. In this case, developers can easily access one of the three APIaaS and immediately exploit also all the APIs offered by the other two with the ease, efficiency, and security of the inner composition.

4 A COMPOSITION-BASED APIaaS PROTOTYPE

In this section we describe our composition-based APIaaS prototype, called JSOA, which is built following a service-oriented paradigm. The prototype represents an important enhancement wrt the PaaSSOA project (Guidi et al., 2012). In particular, JSOA introduces formal documentability, a web GUI designer, monitoring enhancements, publishability, and a prototype of the SaaS layer.

4.1 Abstract APIaaS

In the Abstract APIaaS layer we follow service orchestration principles to supply a composition language for APIs. In particular we use Jolie (Jolie, 2013) as orchestration language because it combines all the basic mechanisms of the well-known web services standards WSDL (W3C, 2001) and WS-BPEL (OASIS, 2007) in the same linguistic domain. In the following we discuss how composability, formal documentability, and portability are achieved by the JSOA Abstract APIaaS layer.

```
1  main {
2  setIntoBlog( r )( p ) {
3    getImgWithFaceBlur@
4      ImageAPI( r.image )( s.image )
5    |
6    getEngTranslation@
7      TranslationAPI( r.text )( s.text );
8    s.username = r.username;
9    s.password = r.password;
10   setPost@BlogAPI( s )( p )
11   }
12 }
```

Listing 1: Composition behaviour of PhotoBlog 2 in Jolie.

Composability. JSOA addresses composability in a natural way using the linguistic primitives supplied by an orchestration language (e.g., Jolie) which easily permit to coordinate concurrent service calls, thus focusing the value on composition and not on other aspects of programming. In particular these primitives cover the following aspects: *i*) *workflow behaviour*: the behaviour is expressed in terms of a workflow that provides both sequence and parallel operators, therefore no thread programming is required; *ii*) *synchronous invocation primitive*: APIs can be invoked by exploiting blocking request-response primitives which can be easily composed within the behaviour; *iii*) *statefulness primitives*: orchestration languages provide specific language primitives for addressing statefulness, e.g., such as correlation sets (Mauro et al., 2011) which ease session and state handling at application level; *iv*) *process spawn*: process spawn is automatically managed by the orchestration engines, no linguistic primitives are required; *v*) *endpoint abstraction*: endpoints allows to invoke APIs abstracting away from communication protocols and message formats. List. 1 reports a snippet of code corresponding to the behavioural composition of PhotoBlog in § 2. Notably, List. 1 does not report the declaration part which we excluded for reasons of presentation. In List. 1 we suppose that `getImgWithFaceBlur@ImageAPI` (Lines 3-4), `getEngTranslation@TranslationAPI` (Lines 6-7), and `setPost` (Line 10) are the names of the API for the image application, the translation application, and PhotoBlog respectively. Moreover, we define the name of the new composed API as `setIntoBlog` (Line 2). When the API is invoked (Line 2), the layer spawns an instance of its behaviour, executing it. In particular List. 1 composes the image and translation APIs in parallel (Line 5) and when they both return a response it commands the sequential execution of the subsequent instructions (Lines 8-9).

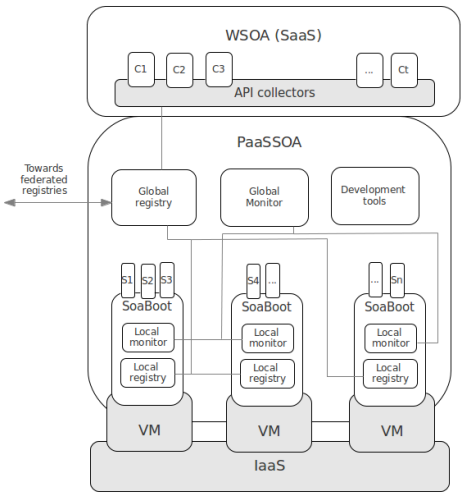


Figure 4: The JSOA Architecture.

Formal Documentability. In Jolie, each input and output endpoints of a service is joined with an interface which defines all the available APIs for the given port and the message types[1]. Each Jolie interface can be enriched with comments and automatically transformed into an human-readable HTML document.

Portability. Portability differs if considered within the same technology or across different technologies. In the former case, JSOA Abstract APIaaS layer completely cope with portability issues. Since the Jolie interpreter is the same, compositions developed into the APIaaS can seamlessly execute without any significant modification of the code (except for endpoint rebinding which consists in a mere url location renaming) on any environment that provides said interpreter. On the contrary, in the latter case, portability is not directly achieved but since Jolie has been developed upon a formal semantics (Guidi et al., 2006) ad-hoc encoding translators can be developed.

4.2 Deployment APIaaS

Fig. 4 reports a sketch of the architecture of JSOA which implements the Deployment APIaaS layer. The main components are:

- *SOABoot*: the basic component which enables service execution in the cloud. It must be available in each computational resource provided by the IaaS and it takes the form of a service itself. SOABoot provides three main functionalities: (*i*) local

[1]Here we do not report the syntax of Jolie interfaces. The interested reader may refer to (Jolie, 2013)

service registration; (*ii*) local service monitor logs storage; (*iii*) local service execution.

- *PaaSSOA*: the component in charge of providing the PaaS abstraction to the developer. It equips a central registry that lists all running orchestrators in any SOABoot available. Registered orchestrators can be exploited for creating new service APIs. Moreover PaaSSOA manages a central monitoring service which collects all the logs from the running services.
- *WSOA*: the component that provides the SaaS abstraction and that is in charge of publishing service APIs outside JSOA.

At the present, JSOA does not respect all the features discussed in § 3. In the following, we briefly discuss the implemented ones whilst we address the missing ones in § 6 as future work.

Protection. is inherently provided by JSOA since all deployed orchestrators and core APIs can be easily invoked by new API compositions within the Abstract APIaaS layer.

Interoperability. is addressed by managing the deployment of the Jolie endpoints depending on the required protocols. Jolie provides both input and output endpoints used by an orchestrator to receive and send messages. As a test, we developed two core API wrappers that invoke WordReference (WordReference, 2013) and CKAN APIs (CKAN, 2013).

Publishability. is implemented by the WSOA component which allows to select a set of already deployed APIs and to export them under different protocols (e.g., Http/JSON, Http/SOAP, Http/POST/XML, etc.). As a test, we developed a plugin that enables the composition of already published APIs within the editor of the SATIN project (Satin, 2013).

Measurability. is provided by the monitoring architecture. A built-in monitor is attached to each deployed orchestrator. The monitor sends all the logged events to the SOABoot local monitor. In turn SOABoot local monitors send collected local data to the central monitor which stores them. The monitoring architecture also provides analysis of data via the Drools engine (Drools, 2013). The current implementation monitors start and end of sessions and start and end of operations. Although simple these data allows for the calculation of important statistics like API response time, memory usage, etc..

Scalability. is addressed by SOABoot and the registry architecture. Depending on the load of the available computational resources new SOABoots can be added. In the same way, computational resources can be released when the corresponding SOABoots do not execute any orchestrator.

5 RELATED WORK

Recently the communities behind the main languages used for web development like JavaScript, Ruby, Java, and PHP produced several RESTful (Richardson and Ruby, 2007) libraries to ease the task of designing, writing, and composing RESTful APIs (some examples are, respectively, (Backbone.js, 2013), (Ruby on Rails, 2013), (Java Community Process, 2013), and (Lavarel, 2013)). RESTful frameworks ease the development of server-side API composition and deployment yet the implementation is part of the basic core of an application which translates into high costs of creation and maintenance.

An inspiring work in the direction of a PaaS middleware for application deployment is (Wettinger et al., 2013) where the authors address the problem of composition (and dependencies) with a middleware-oriented approach. In particular, we notice the focus on decoupling the application logic from the necessary middleware and infrastructure resources. On the same topic, other two interesting works are (Caporuscio et al., 2010; Caporuscio et al., 2014) where the authors introduce the concept of a middleware to leverage integration of heterogeneous services for ubiquitous networking. Such middleware is designed and implemented following a service-oriented approach supporting, respectively, SOAP and RESTful Services. This flexible and dynamic vision is also proposed in (Caporuscio et al., 2011) in which third-party applications discovered at run time should be integrated in applications to offer more sophisticated functionalities.

6 CONCLUSIONS

In this paper we discussed a new proposal for an composition-based APIaaS layer which facilitates API programming and deployment on the cloud by enabling API composition. In our perspective the APIaaS providers should supply large repositories of core APIs which developers could compose. Composed APIs could be available on the same repository of core APIs, enabling the publication of new

and more sophisticated services, thus fostering a liquid market for APIs and the reuse of services.

We also discuss a list of basic features which should be supported by an APIaaS layer and present a proof-of-concept prototype, dubbed JSOA, which implements a subset of such features. We based JSOA on a service-oriented paradigm and exploited primitives offered by its implementation language (Jolie) to orchestrate APIs and, more in general, to implement said features.

The aim of this work is to start a critical discussion towards a shared definition of the concepts and the foundational architecture of an open APIaaS layer.

Future Work. We intend to continue the development of our prototype. Our experience with JSOA proved that prototyping constitutes a relevant resource for testing and analysing benefits and drawbacks of our approach. Namely, the next steps of JSOA development count the implementation of features like sandboxing, security, and federability.

In addition, we also plan to analyse how different economic models affect a federated market of APIaaS providers (as represented in the example regarding the *federability* feature in § 3). What we expect is that the features of our composition-based APIaaS layer would avoid technology lock-in, lower dependencies on specific technologies, and enact a beneficial competition among different providers.

REFERENCES

Backbone.js (2013). Backbone.js javascript framework. http://backbonejs.org/.

Barber, R. (2001). Hacking techniques: The tools that hackers use, and how they are evolving to become more sophisticated. *Computer Fraud & Security*, (3):9–12.

Caporuscio, M., Funaro, M., and Ghezzi, C. (2011). Restful service architectures for pervasive networking environments. In Wilde, E. and Pautasso, C., editors, *REST: From Research to Practice*, pages 401–422. Springer New York.

Caporuscio, M., Funaro, M., Ghezzi, C., and Issarny, V. (2014). *ubi*rest: A restful service-oriented middleware for ubiquitous networking. In *Advanced Web Services*, pages 475–500. Springer.

Caporuscio, M., Raverdy, P.-G., and Issarny, V. (2010). ubiSOAP: A Service Oriented Middleware for Ubiquitous Networking. *IEEE Transactions on Services Computing*.

CKAN (2013). Comprehensive knowledge archive network. http://ckan.org/.

Drools (2013). Drools - the business logic integration platform. http://www.jboss.org/drools/.

Erl, T. (2004). *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Guidi, C., Anedda, P., and Vardanega, T. (2012). Paassoa: An open paas architecture for service oriented applications. In *ESOCC 2012*, pages 208–209. Springer.

Guidi, C., Lucchi, R., Gorrieri, R., Busi, N., and Zavattaro, G. (2006). Sock: A calculus for service oriented computing. In *ICSOC 2006*, pages 327–338.

Java Community Process (2013). The java api for restful web services. https://jcp.org/en/jsr/detail?id=339.

Johnson, R., Hoeller, J., Arendsen, A., Risberg, T., and Kopylenko, D. (2005). *Professional Java Development with the Spring Framework*. Wrox Press Ltd., Birmingham, UK, UK.

Jolie (2013). Java Orchestration Language Interpreter Engine. http://www.jolie-lang.org.

Lavarel (2013). The lavarel php framework. http://laravel.com/.

Liu, X., Hui, Y., Sun, W., and Liang, H. (2007). Towards service composition based on mashup. In *Services, 2007 IEEE Congress on*, pages 332–339. IEEE.

Mauro, J., Gabbrielli, M., Guidi, C., and Montesi, F. (2011). An efficient management of correlation sets with broadcast. In *COORDINATION*, pages 80–94. Springer.

McGraw, G. (2004). Software security. *Security Privacy, IEEE*, 2(2):80–83.

OASIS (2007). Web services business process execution language version 2.0. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html.

Richardson, L. and Ruby, S. (2007). *Restful Web Services*. O'Reilly, first edition.

Ruby on Rails (2013). Ruby on rails. http://rubyonrails.org/.

Satin (2013). Satin project. http://www.satinproject.eu/.

Skene, J., Place, M., and Crampton, J. (2007). The monitorability of service-level agreements for application-service provision. In *WOSP '07*, pages 3–14. ACM Press.

W3C (2001). Web service definition language (wsdl) 1.1. http://www.w3.org/TR/wsdl.

Wettinger, J., Andrikopoulos, V., Strauch, S., and Leymann, F. (2013). Enabling dynamic deployment of cloud applications using a modular and extensible paas environment. In *IEEE CLOUD*, pages 478–485. IEEE.

WordReference (2013). Wordreference. http://www.wordreference.com/.

Comparison of Request Admission Based Performance Isolation Approaches in Multi-tenant SaaS Applications

Rouven Kreb[1] and Manuel Loesch[2]
[1]SAP AG, Walldorf, Germany
[2] FZI Research Center for Information Technology, Karlsruhe, Germany
rouven.krebs@sap.com, loesch@fzi.de

Keywords: Multi-tenancy, Performance, Isolation, SaaS, QoS.

Abstract: In the Software-as-a-Service model one single application instance is usually shared between different tenants to decrease operational costs. However, sharing at this level may lead to undesired influence from one tenant onto the performance observed by the others. Intentionally, the application does not manage hardware resources and the responsible OS is not aware of application level entities like tenants. Consequently, it is difficult to control the performance of individual tenants to make sure they are isolated. In this paper we present an overview and classification of methods to ensure performance isolation based on request admission control. Further, informational requirements of these methods are discussed.

1 INTRODUCTION

In the cloud computing scenario the providers leverage economies of scale and resource sharing to reduce costs of their service offerings. Multi-tenancy enables one application instance to be shared between different tenants, including all underneath layers. This architectural style is widely used in Software-as-a-Service (SaaS) to decrease costs. A tenant is defined as a group of users sharing the same view on an application. A view includes the data they access, the configuration, the user management, particular functionality, and non-functional properties (Krebs et al. 2012). Typically, a tenant represents a legal entity like a company. Thus, multi-tenancy provides every tenant a dedicated view and share of the instance which is isolated from other shares.

Since Multi-tenant Applications (MTA) share the hardware, operating system, and application instance there are potential performance influences between different tenants and unreliable performance is one of the major concerns for potential cloud users. Consequently, one of the cloud provider's goals is to provide the best possible isolation, with regards to the performance the tenants observe.

Performance isolation exists if for customers working within their quotas, the performance is not affected when aggressive customers exceed their quotas (Krebs et al. 2012).

Besides performance isolation, product diversification is an important economic factor. Diversification could be achieved by additional functionality or tenant specific Quality-of-Service (QoS), since customers have a divergent willingness to pay for performance. At the application level the guaranteed performance usually refers to response time and throughput as long as a defined request rate (quota) is not exceeded.

In MTAs where different tenants share one single application instance, the intended abstraction between the operating system that provides resource management and the application that serves multiple tenants makes performance isolation harder to achieve. This is due to the fact that the operating system is not aware of entities like tenants and the application has no resource control, which is essential for controlling performance.

Various approaches to ensure performance isolation in MTAs were already discussed in the literature (Li et al. 2008; Krebs et al. 2012). In this paper we focus on approaches which apply a request based admission control. These methods delay, or reject requests from a certain tenant before they are processed by the application server. Thus, it is possible to influence the performance for each tenant and to reduce the impact of one tenant's requests onto the others as the amount of competing requests in the application server can be limited.

In this paper we present an overview of 5 generic

methods which can be used to establish such an admission control and estimate their capabilities to ensure isolation. Further, we describe the concrete information requirements that have to be fulfilled to realize the different approaches. This helps developers who aim for performance isolation to find the most suitable approach for their scenario.

The reminder of this paper is structured as follows. In Section 2 we provide an overview and classification of potential methods to ensure performance isolation in MT environments based on admission control, and discuss their pros and cons. Section 3 summarizes the informational requirements of the introduced classes. An overview about related work in the area of performance isolation is presented in Section 4. Finally, Section 5 concludes with a summary and an outlook on future work.

2 PERFORMANCE ISOLATION METHODS

In this chapter, five different classes of admission control measures for performance isolation and QoS differentiation are presented. The separation into classes is done based on the kind of required information to make a particular method work.

2.1 Static Approaches

Static approaches make decisions without considering further runtime information such as perceived response times. Usually these approaches have a constant behavior over time. The basic idea for static approaches is illustrated in Figure 1.

A Parameterized Admission Control mechanism uses static rules with constant priorities per tenant for the decision of which request is allowed to pass through to the *Multi-Tenant Application*. Thus, the requests can be accepted, refused, or delayed. The information used by such approaches are the SLA guarantees given to a tenant which are used to derive a constant priority for each tenant. In order to provide QoS differentiation, static priorities can be assigned to the tenants based on the SLA-guaranteed response time and quota. The benefit of such approaches is the rather simple implementation. However, they do not work well without feedback about the perceived performance and quota used by the tenants due to the combination of random load and a difficultly predictable system behavior, combined with the aim of sharing idle resources.

Different kinds of static isolation approaches and there pros and cons have already been discussed (Krebs et al. 2012). One concrete example is a Round Robin with an own FIFO-queue per tenant. In contrast to traditional, non-MTAs such an implementation does not only use one single queue for requests from all tenants, but one dedicated queue for each tenant. The queues are queried one after each other every time when a new request can be processed.

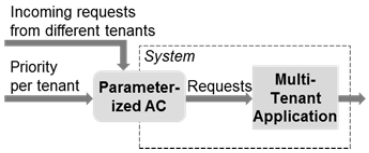


Figure 1: Static Approaches.

The approaches provide a very good isolation as long as the system is not over-committed; however, it is possible that one queue holds plenty requests and the respective tenant's average response time cannot be reached although it could be. Such a scenario is, e.g., given when a tenant has a small amount of requests in his queue which results in a low average response time for this tenant while other tenants may have SLA violations due to longer queues.

2.2 Response Time/Feedback Based Approaches

Static approaches come along with the discussed disadvantages in over-committed and high load scenarios. It is possible that a tenant with a high request rate perceives SLA violations although other tenants could have an average response time significantly below their SLA-guaranteed. In such a case, the average response time of the tenant with a high request rate could be reduced by degrading the tenants that are significantly below their SLA-guaranteed response times. To solve this issue runtime information have to be considered to dynamically adjust the priorities. According to our SLA definition, the perceived response times and the quota are of interest.

Hence, response time based approaches like (Lin et al. 2009) introduce a control loop using feedback about response times and the throughput as depicted in Figure 2.

A *Monitor* offers the perceived response times per tenant as well as the throughput per tenant. This allows the *AC Controller* to dynamically adjust priorities per tenant. They are enforced by the *Parameterized Admission Control*. Since SLA

guarantees are known by the *AC Controller*, the knowledge of the perceived response times allows a more efficient operation by solving the above-described issues.

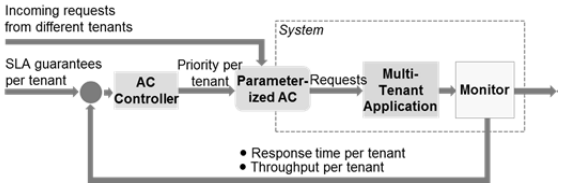


Figure 2: Response Time/Feedback Based approaches.

2.3 Resource Demand Based Approaches

Approaches like (Wang et al. 2012) (Krebs et al. 2014) try to directly control the resource consumption of a tenant to ensure application level performance guarantees. The entitled amount of resources is derived from the SLA guarantees. It can be expected that a higher quality of performance isolation can be achieved since resource demand approaches consider the root cause (consumption of required resources) for the perceived performance, rather than the implication (response times). The mapping of high-level SLAs such as guaranteed response times from the application layer to low-level resource requirements on the resource layer is a research topic that currently attracts a lot of attention. The idea of these approaches is illustrated in Figure 3.

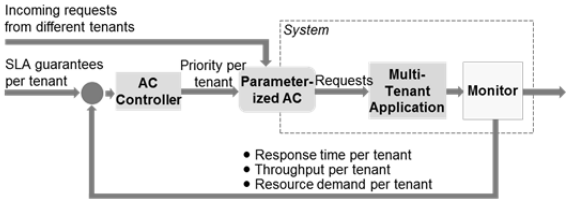


Figure 3: Resource Demand Based Approaches.

The *AC Controller* gives a priority per tenant based on the static SLA guarantees, the current resource demand per tenant as well as the response time and throughput per tenant. The *AC Controller* has two major tasks. First, the mapping of a tenant's SLA guarantee, its perceived response time and its perceived throughput to a fair resources demand per tenant. Second, to set priorities to enforce the tenant's calculated fair resource demands based on the deviation to his current resource demand. Besides more accurate performance isolation, such approaches allow alternative billing options in which tenants pay for the resources they are allowed to consume.

It has to be considered that MTAs are typically composed of three tiers (client, application, database) (Koziolek 2011). Each tier could consist of several instances (Loesch & Krebs 2013). This has to be considered by resource demand based approaches whereas in the previous approaches these internal details could be ignored. Further, different kinds of resources for both the application and the database tier exist. It also is worth mentioning that some resources such as RAM allow determining a current demand represented in Byte whereas resources like CPU only know the two states busy and idle. For them, a well-defined time frame has to be considered for specifying their utilization.

Finally, resource demand based approaches have to consider that the resource demand per request type can vary between different tenants due to tenant-specific customization of the application. The resource demand is also not static over time. For example, increasing database sizes or changing configurations of the application might have an impact.

2.3.1 Direct and Indirect Resource Demand Estimation

For estimating a tenant's resource demand, two fundamental different approaches can be separated: Direct and Indirect Resource Demand Estimation (RDE).

Direct RDE leverages operating system / platform functionality that allows monitoring the resource demand on a tenant's basis at the application-layer. It is the approach depicted in Figure 3. The tenant-specific resource demand is then directly reported to the *AC Controller* which in turn is responsible for adjusting the admission control parameters. This is based on the system's capability to measure the resource demand on a fine-grained level that delivers the amount of required resources per user request (e.g., the service time spent at CPU and required Bytes of RAM). If such monitoring functionality is available it often comes along with a significant overhead (Kuperberg et al. 2011) or lacks in monitoring capabilities among different processes.

Indirect RDE uses algorithms to determine the resource demand per tenant without internal knowledge about tenants by using information that can be measured from outside the tenant-aware application. To estimate the resource demand,

indirect RDE approaches mostly make use of the resource utilization, throughput and response time. Since we are interested in the resource demand per tenant, the throughput must be reported per tenant. For implementing indirect RDE different techniques can be applied such as linear regression or Kalman filter as realized by (Wang et al. 2012). The control loop for indirect RDE approaches is depicted in Figure 4.

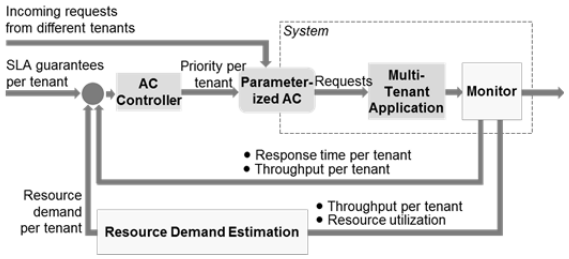


Figure 4: Indirect Resource Demand Estimation.

2.4 Enhanced Resource Demand Approaches

It is possible that the perceived response time is bad due to a high utilization of a single resource that has to be used in order to complete a request. We assume it is possible to build a system where the resource demand of a certain request type is known (Krebs et al. 2014). Together with information about the resource utilization it may allow improving admission decisions. This approach is illustrated in Figure 5.

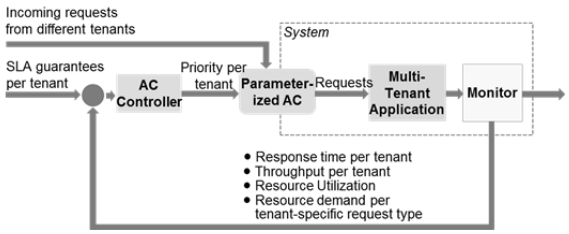


Figure 5: Control Loop for Enhanced Resource Demand based Approaches using direct RDE.

In the previous presented Resource Demand Approaches the resource utilization is only used in case of indirect Resource Demand Estimation. In this kind of approaches, the resource utilization is always delivered to the *AC Controller*. Furthermore, in the previous presented approach, the resource demand was considered on a tenant-base (directly, or indirectly via the *Resource Demand Estimation* component). In this approach, the resource demand can be gathered directly or indirectly as well, however it has to be delivered per tenant-specific request type. Using this information, the *AC Controller* can make better admission decisions. By knowing the resource demand per request type as well as the current utilization of the available resources, requests can be allowed to pass through, even if other resources are highly utilized.

The subsequent scenario illustrates the benefit of these kinds of approaches. For example, a resource demand estimation mechanism may determine that a certain request type requires no resources on the database server. If the database server's resources are high utilized and the application server has a low utilization, the controller can adjust the admission control parameters in a way that this request type is accepted since enough of the crucial resource is available. In contrast to all previous approaches, this requires the Admission Control to be not only tenant- but also request type-specific. Depending on the variety of resource requirements of different requests, this may significantly increase the throughput but again increases complexity.

2.5 Model-based Approaches

Model-based approaches make use of performance prediction models which are based on workload forecasts. Foreseeing performance problems has two major advantages. (1) Adaptation decision can be made pro-actively before problems occur. (2) The impact of different adaptation decisions can be simulated which allows choosing the best adaptation decisions.

Respective performance predictions are driven by the transition towards Cloud Computing platforms and are especially addressed by the research area of self-aware systems engineering (Kounev et al. 2010). Performance prediction models are usually based on the detection of workload patterns and Hidden Markov Models that allow predicting variations in such patterns. Based on such performance prediction models, Model-based Approaches work as depicted in Figure 6. The Performance Prediction Model allows the prediction of response times based on the current AC parameters.

This way, the AC Controller can make adaptation decisions before performance problems occur. Further, it is possible to simulate the impact of different adaptation decisions. Therefore, the *AC Controller* could create multiple adaptation decisions (e.g., multiple sets of tenant priorities) that are expected to optimize the performance isolation.

By using the *Performance Prediction Model*, the

influence of different options onto the performance could then be simulated. The decision that comes along with the best performance isolation can be chosen based on a fitness function for the overall SLA compliance and maybe resource utilization (cf. Section 2.3 and Section 2.4).

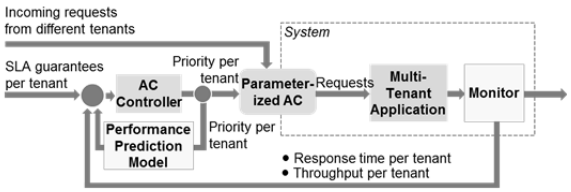


Figure 6: Control Loop for Model-Based Approaches.

3 OVERVIEW

Table 1 depicts an overview about the elaborated informational requirements of the introduced classes of performance isolations approaches. The specific advantages that are reasoned by the increased amount of information used have been discussed in the previous sections as well as the increased effort to gather this information and to implement the presented classes.

Table 1: Informational Requirements.

	1	2	3	4	5
Informational Requirement					
SLA guarantees, per tenant	x	x	x	x	x
Response time, per tenant		x	x	x	x
Throughput, per tenant		x	x	x	x
Resource demand - per tenant - per request type			x	 x	 opt.
Resource utilization				x	opt.
Model for performance predictions					x
Admission Control Parameters					
Specific to tenant	x	x	x	x	x
Specific to request type				x	opt.

It can be recognized that *Static Approaches (1)* only use static SLA information without considering runtime information at all. To overcome the problem of these approaches, we introduced *Response Time based Approaches (2)* that use a control loop leveraging response times and throughputs as runtime information. *Resource Demand based Approaches (3)* try to take advantage by making admission decisions based on the tenant's resource demands which are the root cause for their perceived performance. The table shows the informational requirements for direct RDE approaches where the resource demand is gathered through direct measurements. If the resource demand should not be measured directly, it can be estimated based on the tenant's throughput and the resource's utilization (cf. Section 2.3). The *Enhanced Resource Demand based Approaches (4)* enable a performance isolation that allows preferring requests that can be processed by free resources. This will increase the overall utilization and hence the economic efficiency. Therefore, the resource utilization is required and the resource demands have to be specific for the tenant and the tenant-specific request-type. It is worth to mention again, if the resource demands should be estimated indirectly, this can be done based on the request types' throughputs, response times and the resource's utilization. *Model-based Approaches (5)* are able to anticipate performance issues and simulate different adaptation decisions. Therefore, they require a model for performance predictions, usually based on workload forecasts. For them, it is optional to consider resource demands. When considering resource utilization and the resource demands on basis of request types, free resources can be utilized more efficiently by giving precedence to respective request types as in the case of Enhanced Resource Demand based Approaches.

4 RELATED WORK

Lin et al. (Lin et al. 2009) focus on providing different QoS for different tenants by regulating response times. They make use of a regulator which is based on feedback control theory to achieve this. Li et al. (Li et al. 2008) first predict performance anomalies and identify an aggressive tenant who is responsible for the incident. Based on this, their approach applies an adoption strategy reducing the influence of the aggressive tenant on the others to ensure isolation. Wang et al. (Wang et al. 2012) developed a tenant-based resource demand estimation technique using Kalman filters. This approach predicts the current resource usage of a tenant and uses this information to control the admission of incoming requests based in a predefined quota. Four static mechanisms to realize performance isolation were described and evaluated in (Krebs et al. 2012). One approach is based on thread pool management and three other approaches leverage admission control.

These papers focus on the concrete algorithms and do not discuss the general approach nor classify

their approach or evaluate general informational requirements.

Guo (Guo et al. 2007) discuss multiple isolation aspects including performance isolation for MTAs on a conceptual level. They propose Resource Partitioning, Resource Reservation and Request-based Admission Control as mechanisms to maintain a certain QoS for each tenant. In comparison, we provide a classification and potential solutions within Request-based Admission Control which was not covered in detail. In (Loesch & Krebs 2013) the authors describe where a request-based admission control has to be realized in a distributed MTA scenario. However, a classification or concrete measures to achieve performance isolation are not described. Koziolek (Koziolek 2011) evaluated several existing MTAs and derived a common architectural style. However, Koziolek's architectural style does not discuss performance isolation at all.

5 CONCLUSIONS

Multi-tenancy is an architectural approach which shares one single application instance between several customers. This allows increasing the efficiency of SaaS applications. However, due to the abstraction of the application which is tenant aware, from the actual resources, controlled by the non-tenant aware OS, it is complicated to ensure the isolation of the performance observed by different tenants. In this paper we presented five conceptual approaches with increasing capabilities to control performance to the detriment of the complexity and need for detailed information about the system at runtime. All presented approaches overcome the described layer discrepancy by applying a request-based admission control. The simplest approach is based on a static admission control like a Round Robin which successively selects tenants' requests from tenant specific queues. In contrast, the most complex approach uses performance prediction models to find a suitable admission control method by proactively reacting to forecasted performance incidents that might happen in the future. We further discussed specific advantages of these approaches, followed by a list and comparison of their information requirements. This helps developers of multi-tenant applications to find an appropriate method.

REFERENCES

Guo, C. J. et al., 2007. A Framework for Native Multi-Tenancy Application Development and Management. In: *Proc. of the 9th IEEE Int. Conf. on E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services.*

Kounev, S. et al., 2010. Towards Self-Aware Performance and Resource Management in Modern Service-Oriented Systems. In: *Proceedings of the IEEE International Conference on Services Computing.*

Koziolek, H., 2011. The SPOSAD Architectural Style for Multi-tenant Software Applications. In: *Proceedings of the 9th Working IEEE/IFIP Conference on Software Architecture*.

Krebs, R., Momm, C. & Kounev, S., 2012. Metrics and Techniques for Quantifying Performance Isolation in Cloud Environments. In: *Proc. of the 8th international ACM Conference on Quality of Software Architectures*.

Krebs, R. et al., 2014. Resource Usage Control In Multi-Tenant Applications. In *Proceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing.*

Kuperberg, M. et al., 2011. Defining and Quantifying Elasticity of Resources in Cloud Computing and Scalable Platforms. Karlsruhe Institute of Technology.

Li, X. H. et al., 2008. SPIN: Service Performance Isolation Infrastructure in Multi-tenancy Environment. In: *Proceedings of the 6th Int. Conference on Service-Oriented Computing.*

Lin, H. L. H. et al., 2009. Feedback-Control-Based Performance Regulation for Multi-Tenant Applications. In: *Proceedings of the 5th International Conference on Parallel and Distributed Systems (ICPADS).*

Loesch, M. & Krebs, R., 2013. Conceptual Approach for Performance Isolation in Multi-Tenant Systems. In: *Proceedings of the 3rd International Conference on Cloud Computing and Services Science.*

Wang, W. et al., 2012. Application-Level CPU Consumption Estimation: Towards Performance Isolation of Multi-tenancy Web Applications. In: *Proceedings of the IEEE 5th International Conference on Cloud Computing*.

An Open Market of Cloud Data Services

Verena Kantere
Institute of Services Science, University of Geneva, Geneva, Switzerland
verena.kantere@unige.ch

Keywords: Cloud Computing Services, Cloud Data Services.

Abstract: Cloud data services are a very attractive solution for the management of business and large-scale data. The incapability of creating fully functional data services and, moreover, providing them with clear guarantees on technical and business aspects, creates a problematic cloud service-provisioning situation. We propose the development of a framework that will enable the systematic and efficient creation and management of cloud data services. Such a framework is necessary in order to achieve the exchange of cloud data services in any open market, where cloud providers and their customers can advertise the offered and requested data services in a free manner and make contracts for service provisioning. We discuss the skeleton of such a framework, which comprises three parts: first, profiling data services, second, designing service offers and demands, and, search and match of data services.

1 INTRODUCTION

Cloud computing has given rise to a new perspective of data management that adds more optimization dimensions, beyond traditional performance, such as cost, availability, elasticity, scalability etc. In a cloud environment this is realized with the provisioning of *data services*, which includes the transparent management of infrastructure, data and workload. Cloud data services are a very attractive solution for the management of business and large-scale data. Even though the cloud industry offers today data services, these mainly concern key-value stores and lack substantial functionality of a traditional DBMS (e.g., support for transactions, optimization techniques, fully fledged declarative query language).

The incapability of creating thorough data services and, moreover, providing them with clear guarantees on technical and business aspects, creates a problematic cloud service-provisioning situation. On one hand, cloud providers cannot design efficient and appropriate data services and, on the other, consumers are not aware of their data management needs. Moreover, providers cannot describe accurately their offered data services and, thus, consumers do not have enough information on their received data services. This results in misuse of data services for long periods and expensive migration to new data services. Therefore, even though cloud data services are a critical component of cloud-hosted applications, their deficiencies prohibit their wide applicability.

We argue that we need to fill the gap between providers and consumers of data services that exists in today's cloud business, by not only solving the above two issues, but also offering an all-inclusive solution for the provision of efficient and appropriate data services. We propose the development of a framework that will enable the systematic and efficient creation and management of cloud data services. Such a framework is necessary in order to achieve the exchange of cloud data services in any open market, where cloud providers and their customers can advertise the offered and requested data services in a free manner and make contracts for service provisioning. The openness of such a market eliminates the imbalance of the two parties, by allowing everybody to contribute and request services and enabling the near-real time negotiation of data service provision. The proposed framework comprises three parts: first, a novel technique for profiling data services; second, a methodology and methods for designing service offers and demands; and, finally, methods and algorithms for search and match of data services.

Such a framework will directly benefit cloud business and industry by enabling them to: first, design fully functional data services tailored to the needs of the customer, and, second, give visibility

and clarity to offer and demand, which will lead to optimal service provisioning. Potentially, the research results on data service search and match will give tools for new practices around service engineering and trading.

In the following we discuss how the proposed framework relates to and advances the domain of cloud data service provisioning.

2 CLOUD DATA SERVICES

The new trend for service infrastructures in the IT domain, cloud computing, is also a new area of research (Armbrust, M., et al. 2010). Recently, the research community has shown an enormous interest in the area of cloud data management. The reason is that cloud computing seems to be the ideal paradigm for the time- and cost- efficient management of big amounts of data, such as scientific data of large-scale. Beyond scientific data, cloud data management can be applicable to other data, such as personal data, commercial data, and any kind of archives, public or private. Users (persons, companies, organizations) may store their data in the cloud, which provides data management services. The cloud alleviates the burden of data management from the users for remuneration.

Emerged cloud data service provisions such as the Amazon SimpleDB[1], Amazon Relational Database Service[2], Microsoft Azure[3] and Google Cloud SQL[4] offer preliminary data management services, but are a long way from offering full capabilities of a traditional database management system (DBMS). Moreover, cloud systems that support massively distributed data management, such as Amazon Dynamo (DeCandia, G., et al. 2007), Google Bigtable (Chang, F. et al. 2006), MicroSoft SQL Server for cloud (Bernstein, P. A. et al. 2011), Yahoo PNUTS[5], Cassandra[6] and HBase[7] are concerned with data consistency and availability issues for analytical data in key-value formats, leaving transactional management on relational data out of their scope.

Yet, the business and enterprise world is rapidly turning towards the cloud data-serving systems to cover their data management needs in a cheap and easy, but also, efficient and reliable, fashion. Such data management applications require high levels of functionality, i.e. the functionality of a traditional DBMS, seamlessly offered through a cloud provider. Therefore, it is absolutely necessary to evolve the first generation of cloud data-serving systems to complete data management systems. The proposed framework is an output towards this direction, by enabling the formulation and the provision of fully functional data services through the cloud.

Current research projects in the general area of cloud data management deal with issues of data consistency and manipulation (Kraska, T. et al. 2009), and issues of data filtering and aggregation[8]. Before even dealing with such issues, several research groups, (e.g. (Abadi, D. J., 2009, Aboulnaga, A. et al. 2009)), discuss the benefits, drawbacks and challenges from moving data management applications and tools into cloud systems. One important problem is how to adaptively modify the allocation of a workload within a cloud (Paton, N. W. et al. 2009). Other important problems are the configuration of virtual machines (Soror, A. et al. 2008), the automatic performance modeling of virtualized applications (Shivam, P., 2007) and self-tuning data management (Weikum, G. et al. 2002). The proposed framework is complementary to current research projects by working towards a holistic approach of moving data into the cloud and tackling the above problems: our goal is to profile the configuration of data services that can be offered by cloud providers and that are requested by consumers. It enables automation of data management performance, by offering data service composition and, finally, achieves pre-configured self-tuning data management via the realization of an open market of searching and matching offered and requested data services.

3 A FRAMEWORK FOR CLOUD DATA SERVICES

Problem Setting. The cloud computing paradigm is nowadays the answer for the 'easy' management of data, meaning the efficient processing and querying, especially of vast amounts of data, usually referred to as 'big data', but also various other data of many types, which are owned by small and medium enterprises and persons. Beyond the variety of data

[1] http://aws.amazon.com/simpledb/
[2] http://aws.amazon.com/rds/
[3] http://www.windowsazure.com/
[4] https://developers.google.com/cloud-sql/docs/introduction
[5] http://research.yahoo.com/project/212
[6] http://cassandra.apache.org/
[7] http://hbase.apache.org/
[8] http://srt-15.unine.ch

in terms of type and size, there is an enormous variety in the type of query workloads that people want to run on such data; in other words, the way that people want to access the, same even, data may vary tremendously in terms of duration and complexity. A *data service* in the cloud is the transparent management of data on a cloud infrastructure, and this may include data storage, maintenance and query execution.

There are three elements to combine in order to offer and demand data services, i.e. (a) infrastructure (b) data and (c) workload. These three elements need to be combined in an optimal way, in order to achieve the provision of optimal data services. Infrastructure, data and workload may either belong to the same person or organization or they may belong to different ones. In any case, the creation and the provision of a data service include instances of all three elements: a workload that is executed on some data using some part of the infrastructure. A data service may be requested by a user that owns the workload for execution or may be offered by the cloud that owns the infrastructure and/or the data. Data and workloads need services that comply with their management and execution characteristics, and cloud infrastructure can be employed to offer services that comply with its operational characteristics and availability.

We envision a virtual environment that constitutes an open market of cloud data services. In such an environment (Figure 1), cloud providers and their potential customers can advertise the offered and requested data services in a free manner and make contracts for service provisioning. Customers can be persons and organizations, and, even more, other cloud providers, or brokers of meta-services. The openness of the market aims at eliminating the current imbalance between consumers and producers of cloud data services, by allowing everybody to contribute and request services and enabling the near-real time negotiation of data service provision.

Motivation. Let us assume a computing situation with cloud infrastructure (IaaS) providers, data management (DBMS) providers and customers with data and/or workloads. The DBMS providers are the customers of the IaaS providers and data/workload owners are the customers of the data management providers. An open market of data services would allow all three parties to communicate and advertise their needs and availability, enabling efficient such matches. The participants of the market need to know what they are seeking and what they can offer: On one hand, the DBMS providers need information

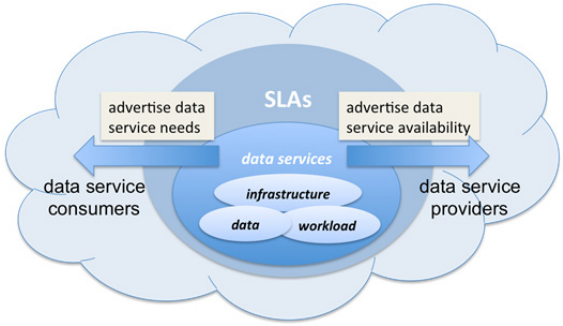


Figure 1: Efficient data service provisioning.

on the available choices for services from IaaS providers in order to form and build their data management services, and the data/workload owners need information on the available choices for data management services. On the other hand, the DBMS providers need to understand how the available infrastructure services can be used to build data management services, therefore, what data services can be built; moreover, data/workload owners need to understand the management and execution needs of their data and workload, respectively, therefore, what data services they are seeking. Also, the participating parties of the open market need to be able to advertise their offer or request for services in a coherent and comprehensive manner. Furthermore, they need to be able to search and match service needs and availability in a timely and efficient manner. Figure 2 shows graphically the described cloud environment.

Problem Definition. The provision of efficient cloud data services necessitates a successful match of offered and demanded services on infrastructure, data and workload. This requires means to (a) formulate, (b) express and (c) advertise offer and demand so that interesting parties can have the information needed for the selection of services.

a) Service Formulation: The basic requirement for the success of an open service market is for all involved parties to be aware of what are the services they can offer or demand, given some conditions. Specifically:

- Given some infrastructure and data, what are the workloads that can be served.
- Given some data and workload, what is the needed infrastructure.
- Given some workload and infrastructure, what should be the data to manage, i.e. the deployment of data and data structures.

b) Service Expression: The service matching requires the systematic and homogeneous expression of services from both ends of service provisioning,

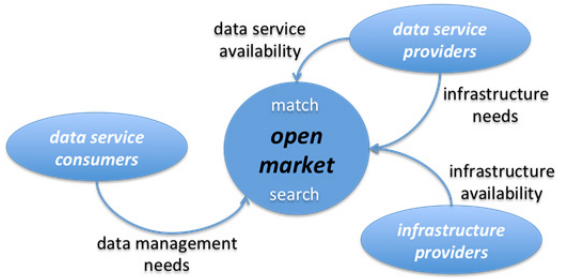


Figure 2: Open market of cloud data services.

i.e. the provider and the consumer, concerning all three elements, infrastructure, data and workload. Specifically, we need to know:
-The characteristics and metrics of possible services.
-The structural relations of services.
-A systematized construction of complex services.

c) Service Advertisement and Selection: Accurately formed and homogeneously expressed services can be advertised as the objects of search and match for service provision in the open market. The success of service match is based on an informative selection of services based on their provisioning characteristics. To achieve this, we need to know:
- How to efficiently and accurately match data service needs and availability.
- How to coordinate service advertisement in order to achieve near-real time service search and match.

Proposed Solution. The solution that we propose in order to achieve the creation and successful operation of an open market for cloud data services consists of an overall framework for the creation and management of data services, which includes: a technique for profiling data services; a methodology and methods for designing service offers and demands, and methods and algorithms for search and match of data services.

a) A Technique to Profile Data Services: Realistic data management needs and infrastructure capabilities can lead to the creation of a data service model. This should be based on the identification and standardization of functional characteristics and metrics for infrastructure, data and workload, as well as their relationships.

b) A Methodology and Methods to Design Data Services: The model enables the design of new services structured on service units. We need to:
- Define the service units as basic meaningful instances of the service model.
- Create a methodology for hierarchical service composition on top of service atoms and define composition directions and optimization objectives.
- Create specific methods for the hierarchical composition of examples of data services types.

c) Methods and Algorithms to Search and Match Data Services: The framework must be completed with the design of a service advertisement broker, a service matcher and a service scheduler.
- The service broker is the meeting point for producers and consumers of services, who are able to advertise their availability and need for services, respectively. Broker instances are able to communicate and form an overlay network for searching data services.
- The service matcher enables the bottom-up matching of offered and demanded services according to the respective instances of the data service model, employing a novel near-optimal matching algorithm for service profiles.
- The service scheduler takes as input the search offers and requests and schedule new, or renewals of, service matches in time frames in which these are valid. The scheduler must be based on novel methods that can monitor inputs of service advertisements. These methods will aim to change the scheduling time frames dynamically, in order to allow for optimal consumption of services through more accurate search and match.

The discussed framework aims at exploring the limits and capabilities of data service formulation and expression, and tailoring these to data service provisioning.

4 PROFILING DATA SERVICES

In order to formulate data services, the notion of a data service needs to become tangible. The first step is to study the data management needs and capabilities of cloud service consumers and providers, and to employ this study to propose a novel model for the description of a wide range of data services. Such a model can be employed to profile the data management needs and capabilities in order to form respective offered and demanded data services in a cloud environment. The goal is to produce a profiling technique that will allow for homogeneous service descriptions across data applications and platforms. Profiling the data services enables the creation of techniques for search and matching of services in an optimal manner.

Towards this direction we need to identify and study the key characteristics of data services. Such a study can give us a concrete idea of what is a realistic data service, what applications it can serve and in what way, what is the extent and the duration

of the service, etc. The key characteristics of data services should be studied for the three elements of infrastructure, data and workload. Examples of basic characteristics that we expect to strongly determine the 'attitude' of services are:
- Concerning Infrastructure: CPU utilization, I/O operations, bandwidth, storage space, degree of shareness, size of virtual machines, etc.
- Concerning Data: replication degree, update rate, security constraints, data structures (views and indexes), partitioning degree and type, data statistics on selectivity estimation, etc.
- Concerning Workload: parallelization of execution, query complexity - translated into execution cost of query plans, data access skewness or similarity, etc.

The identification of the characteristics of data services nees to be accompanied with the definition of respective metrics. The service model should incorporate standard metrics such as response time, throughput and latency, as well as monetary cost. We also need to define new metrics, such as availability and privacy degree, and accompany each metric with an appropriate cost model, depending on the characteristic in hand.

5 DESIGNING DATA SERVICES

The profiling technique is the input to a novel methodology and methods for the design of cloud data services. The goal is to enable the creation of basic and generic data services, i.e. *service units*, and, furthermore, the hierarchical creation of complex data services on top of service units. We envision service units that comprise basic parts, qualitative or quantitative, of all three service elements, i.e. infrastructure, data and workload, profiled appropriately. For example, a service unit can be the combination of: a SPJ (i.e. select-project-join) query with one join, on an attribute with selectivity '10%' on a table of 10M tuples running on 1 CPU with no data transfer via network. Exploring realistic data services can lead to the proposal of example service units.

Furthermore, we need a methodology that facilitates users, meaning service providers and consumers, to create their own complex services along a range of optimization dimensions and ultimate service goals. Such a methodology can indicate paths of hierarchical service composition along such dimensions. We need to consider optimization dimensions respective to the metrics of service characteristics, such as response time and monetary cost. The methodology will take into account the durability or volatility of services, i.e. how persistent they are in time. Moreover, it will take into account any relationship between services. Such relationships can be either dependencies or incompatibilities between services, and may constrain their composition. Service relationships may be found either between instances of the same service element, or different ones. For example, it is incompatible to compose services with different data partitioning type, or a service that parallelizes workload execution depends on a service that updates data on all data replicas, respectively. Using the methodology, we will develop methods for hierarchical composition of services with specific optimization objectives and characteristics.

Since the service design is going to be employed in a dynamic cloud environment with frequent changes of available and requested data services, the goal of the composition methodology and methods will be to enable fast, i.e. near-real-time, composition based on data service patterns and re-usage of composition objects.

6 FINDING DATA SERVICES

To realize an open market of services we need to be able to search and match services in an accurate and efficient manner. Producers and consumers of services will be able to meet on brokers that implement the overall framework and advertise their availability and need for services via the respective instances of the data service model. They will be able to know their availability and need for services by using the profiling scheme and the service design methods. The search and match will target to produce valid combinations of the three service elements: infrastructure, data and queries. Each service to be searched or matched will have two parts, namely the *fixed* part and the *sought* part. The fixed part includes the conditions/constraints of request or availability, and the sought part of the service includes the actual request or availability to be matched. Naturally, we expect that, usually, the profiled infrastructure will be included in the sought part and workload will be included in the fixed part, for demanded services; we expect the opposite for offered services. Data can belong to either the fixed or the sought part, depending on the occasion. For example, if a service consumer needs to execute a workload on a specific database (e.g. workload and data belong to the same owner), then these data are included in the fixed part of the requested service. If a consumer seeks to execute a workload on some

data that are out there, or she requests data services to deploy the data in the cloud, then these data may be included in the sought part of the service, since their deployment (i.e. storage, caching, built of data structures on top of it) in the cloud is flexible.

The overall framework can be implemented on service brokers of the open market. The framework instances can allow communication of the brokers by creating an overlay network (e.g. with centralized or Peer-to-Peer coordination), which can propagate local and received remote advertisements of services. We envision an open market that could take a further step down the road of searching and matching services, by enabling groups of users with semantically related needs for services to advertise their request or availability as a team and match services in a systematic manner.

While we expect search of services to be performed from top to bottom, meaning search performed based on the overall summary of complex hierarchically structured services, matching services in the framework should be enabled by algorithms that aims to match the services from bottom to top, meaning starting from the matching of atoms and moving to complex services built on top of them.

The goal is to enable dynamic and near-real-time search and match of services. Therefore, it is necessary to include methods that take as input search requests and service advertisements and schedule their matching in time frames in which the sought or advertised services are valid. Moreover, since the matched services may not have the same durability in time, the methods need to adapt the scheduling time frame in order to achieve optimal consumption of services through more accurate search and match. Such methods enable the maintenance of multiple dynamic service queues, the number and size of which depends on the variation of the durability of the incoming services.

7 CONCLUSIONS

The provisioning of data services is a new paradigm in data management that represents a very attractive solution for the management of business and large-scale data due to its low cost and high performance capabilities. The industry of cloud computing tries to catch up with fulfilling these data management needs but lacks the appropriate technology for the realization of cloud-hosted database systems, which is a critical component in the software stack of many cloud applications. The proposed framework aims to fill the gap between providers and consumers of data services that exists in today's cloud business, by not only solving the above two issues, but also by offering an all-inclusive solution for the offer of efficient and appropriate data services. Such a solution will enable the successful search and match of data services via the advertisement of service need and availability.

REFERENCES

Moore, R., Lopes, J., 1999. Paper templates. In *TEMPLATE'06, 1st International Conference on Template Production*. SCITEPRESS.

Smith, J., 1998. *The book*, The publishing company. London, 2nd edition.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M., 2010. A view of cloud computing. *In CACM, vol. 53, pp. 50--58.*

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W., 2007. Dynamo: Amazon's Highly Available Key-Value Store. *In Proc. of ACM SIGOPS, vol. 41, pp. 205--220.*

Chang, F., Dean, J., Sanjay, G., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., Gruber, R. E., 2006. Bigtable: A Distributed Storage System for Structured Data. *In Proc. of USENIX OSDI, pp. 5--15.*

Bernstein, P. A., Cseri, I., Dani, N., Ellis, N., Kalhan, A., Kakivaya, G., Lomet, D. B., Manne, R., Novik, L., Talius, T., 2011. Adapting Microsoft SQL Server for Cloud Computing. *In Proc. of ICDE, pp. 1255—1263.*

Kraska, T., Hentschel, M., Alonso, G., Kossmann, D. 2009. Consistency Rationing in the Cloud: Pay only when it matters. *In Proc. of VLDB, pp. 253--264.*

Abadi, D. J., 2009. Data management in the cloud: Limitations and opportunities. *In IEEE Bulletin on Data Eng., vol. 32, pp. 3—12*.

Aboulnaga, A., Salem, K., Soror, A. A., Minhas, U. F., 2009. Deploying database appliance in the cloud. *In IEEE Bulletin on Data Eng., vol. 32, pp. 13--20.*

Paton, N. W., Arago, M. A. T. de, Lee, K.,. Fernandes, A. A, Sakellariou, R., 2009. Optimizing Utility in Cloud Computing through Autonomic Workload Execution. *In IEEE Bulletin on Data Eng., vol. 32, pp. 51--58.*

Soror, A., Minhas, U. F., Aboulnaga, A., Salem, K., Kokosielis, P., Kamath, S., 2008. Automatic virtual machine configuration for database workloads. *In Proc. of ACM SIGMOD, pp. 953—966.*

Shivam, P., Demberel, A., Gunda, P., Irwin, D., Grit, L., Yumerefendi, A., Babu, S., Chase, J., 2007. Automated and On-Demand Provisioning of Virtual Machines for Database Applications. *In Proc. of ACM SIGMOD, pp.1079--1081.*

Weikum, G., Moenkeberg, A., Hasse, C., Zabback, P., 2002. Self-tuning Database Technology and Information Services: from Wishful Thinking to Viable Engineering. *In Proc. of VLDB, pp. 20—31.*

Mjolnirr: A Hybrid Approach to Distributed Computing
Architecture and Implementation

Dmitry Savchenko and Gleb Radchenko
System Programming Department, South Ural State University, Chelyabinsk, Russia
dmitry@mjolnirr.com, radchenkogi@susu.ac.ru

Keywords: Cloud Computing, Cloud Middleware, PaaS, Private Cloud, Mjolnirr.

Abstract: A private PaaS enables enterprise developers to leverage all the benefits of a public PaaS to deploy, manage, and monitor applications, while meeting the security and privacy requirements your enterprise demands. In this paper we propose the design and implementation of a Mjolnirr private cloud platform for development of the private PaaS cloud infrastructure. It provides means for custom applications development which uses resources of distributed computing environment.

1 INTRODUCTION

Cloud computing became very popular during the last few decades (Mell and Grance, 2011). This type of resource provisioning offers convenient access to computing resources, which makes it easy to use them for custom services development.

Before the introduction of the cloud computing concept, there was another distributed computing concept called "grid computing" (Foster and Kesselman, 2003). It is used by the scientific community while solving extra-large and resource-intensive scientific tasks. The name "grid" originated from the metaphor of the power grid. However, unlike electricity, integration of new resources to the grid computing environment is not trivial.

Meanwhile, PaaS (Platform as a Service) solutions provide the ability to create custom cloud applications much easier. But the problem is that the most PaaS solutions are deployed on remote hosting company servers. The owner of the application may not know exactly where his information is stored, as well as he cannot be absolutely sure about the safety and security of the information. These issues might be solved by the private PaaS platform (Fortis, Munteanu and Negru, 2012).

In this paper, we propose the design and implementation of the private cloud platform called "Mjolnirr". It provides a service which is comparable with existing cloud platforms. There are several solutions that provide private PaaS (Yandex Cocaine, AppFog, Stackato) levels of service. Comparing to these solutions, main features of our approach are integrated messaging subsystem, flexible workload management and UNICORE (Streit, 2009) grid environment integration module.

The article contains the following sections. In section 2 we present the main goals of the Mjolnirr cloud platform development. In section 3 we describe the results of the analysis of existing cloud platforms and compare them with our platform. In section 4 we describe the structure of the Mjolnirr platform and the component communication protocol. In section 5 we describe the implementation of the Mjolnirr platform. In section 6 we describe the platform performance evaluation. In section 7 we summarize the results of our research and further research directions.

2 MJOLNIRR PLATFORM CONCEPT

The main goal of the project is to create the cloud platform for development of a private PaaS cloud infrastructure. *Mjölnir* is the name of the hammer of Thor, a major Norse god associated with thunder. Therefore, we have chosen the "Mjolnirr" name as metaphor of a powerful tool linked with thunder and clouds.

Java-based application or library can be implemented as a Mjolnirr-based service. The Mjolnirr platform provides infrastructure for cloud

applications development, including software developer kit, message brokering system and browser support. For a developer, a Mjolnirr application is represented as a collection of independent components communicating by message passing. This approach allows to develop flexible and scalable cloud applications.

Mjolnirr also provides integration with the UNICORE grid environment (Streit, 2009) through the DiVTB (Radchenko and Hudyakova, 2013) platform. The DiVTB (Distributed Virtual Test Bed) platform provides a task-oriented approach for solving specific classes of problems in computer-aided engineering through resources supplied by grid computing environments. Thus, Mjolnirr can be used both to provide infrastructure for scientific projects with the grid systems and in a business infrastructure.

During our research, we should:

1. analyse technologies of private cloud platforms development;
2. develop Mjolnirr platform architecture;
3. implement Mjolnirr platform;
4. evaluate the performance and scalability of the Mjolnirr platform.

3 ANALYSIS OF EXISTING SOLUTIONS

An investigation shows that C-level executives and IT managers in enterprise companies have concerns with integration of cloud computing in their data processing (Avanade.com, 2014). One of the most serious concerns is the possibility of data breaches. A cloud provider can give access to the company's private data (accidentally or intentionally) or may bring harm to the data owner.

It is possible to use the encryption of data stored in the cloud, but this is effective only when cloud is used only for storage. If the data is processed in the cloud, it become available decrypted in the memory of the host, where the processing occurs. In addition to this drawback, the owner of the data does not control the location of his virtual machine, so it can be moved to the physical computer with the virtual machine that contains malware. It means that they will have the same IP address. It may cause the block of the virtual machine or forfeiture of computer containing these virtual machines.

Nowadays, there are two ways to ensure the data security in the cloud. The first way is called "trusted computing". It ensures security of virtual machines in the cloud (Christodorescu et al., 2009). But user data cannot be completely safe. In IaaS clouds, the virtual machine can be moved to the other host, but the concept of trusted computing provides security only for virtual machines running on the same host. Otherwise, the concept of Trusted Cloud Computing Platform (Garfinkel et al., 2003) solves this problem by creating the safe environment for running virtual machines. But neither of these approaches solves the problem of VMs placement on the same host with a malicious VM.

Another way to deal with the security issue is to deploy the cloud infrastructure on the private hardware. But buying and maintaining of the hardware is more expensive than rent of computing resources.

The simplest way is to create a private cloud system, and there are several different platform (PaaS) solutions existing in this area.

Yandex Cocaine and AppFog platforms provide the ability to create private PaaS solutions based on application containers (Api.yandex.com, 2014; AppFog, 2013). These platforms allow creation Heroku-like application hosting. They provide a number of built-in modules and a server infrastructure. Stackato (Activestate.com, 2014) shows all advantages of the already mentioned solutions and provide local application store. However, all of the above solutions consider custom applications as a monolith and ignore its internal structure, that's why it is not possible to automatically and effectively balance the load on the individual subsystems applications. In addition to this drawback, none of these solutions considers end-user workstations as computing resources providers.

We decided to create a Mjolnirr cloud platform, which solves these problems as follows:

1. *Cost reduction* will be provided by using popular open source libraries and programming languages, as well as the opportunity to work not only on the server platform, but also on the personal computers, using idling resources of the computer system.
2. *Ease of application development* will be provided by using popular programming languages, developer tools and SDK.
3. *Integration of new resources ease* will be provided by the application architecture modularity and custom components reusability.

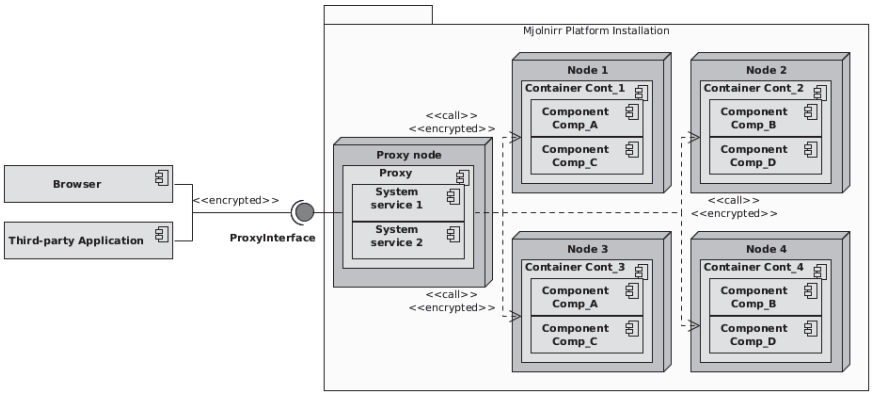


Figure 1: Mjolnirr platform architecture.

4 MJOLNIRR ARCHITECTURE

Mjolnirr platform architecture is shown at the figure 1.

The Mjolnirr platform includes the following components:

1. *Proxy* provides access to the cloud system for the external clients and manages the communication between cloud application components. It also hosts all of the system services (built-in modules for user authentication, distributed file system, database access etc.). Proxy is the only component that accessible from the external network.
2. *Container* is responsible for hosting of cloud applications components and message transmission. It can be deployed both on personal computers and on the computing server nodes.
3. *Components* are custom applications, developed to run in Mjolnirr cloud environment. Each component has a unique name. UI components (applications) are multi-page applications.
4. *Clients*. All client applications use encrypted channel to communicate with the proxy. Each client should use certificate for authentication.

4.1 Proxy

The Proxy (see fig. 2) provides access to Mjolnirr system from the external network. The Proxy performs the following actions:

1) it stores and provides static resources (page layout descriptions, images etc.) of the deployed cloud applications in the *Static Storage*;
2) it provides a *Component Repository* to all of the containers;
3) it provides a *Message Server* for the components of cloud applications;
4) it handles client's requests to the *Client Interface*;
5) it performs the authorization and authentication of users;
6) it hosts all of the system services (users management, distributed file system, database access, etc.). System services act like a custom component and can be called in a standard way (see section 4.3).

The external Proxy interface exposes the following RPC methods:

1) *String getUI (component, page)* - returns the layout description of the page *"page"* of the component *"component"* as a plain text;
2) *byte[] getResourceFile (applicationName, resourceName)* - return the required resource file (image, script file) in binary format;
3) *String processRequest (componentName, methodName, args)* processes a client request, redirecting it to the first free suitable component instance. This method can be called directly from the application page, from script scope. Returns serialized plain object as an execution result;
4) *void uploadComponent (content)* – upload the custom component to the proxy. Component will be added to the *Component Repository* and immediately distributed to the running containers.

4.2 Container

The container provides cloud application component hosting and API for remote components instances method invocation. The Mjolnirr installation can have any number of containers.

Any Mjolnirr-based application consists of independent components, which use a built-in messaging system, implemented in the basis of the Publisher-Subscriber pattern. The Proxy is responsible for message queue maintenance. The Message Server of the Proxy provides publisher-subscriber messaging service for cloud application components. Mjolnirr Containers subscribe to Message Channels that operate as a broadcast delivery – any message sent to the Message Channel will be transmitted to the subscribers of this channel. Each cloud application instance is subscribed on two types of Message Channels:

1) Component Public Channel: every instance of the cloud application component is subscribed on this public channel. This a listener channel – when any message come to this channel, the appropriate component instance will be invoked.
2) Instance Private Channel: provides direct communication between instances.

When container starts, it performs several initialization steps. The order of the container initialization:

1) Container registers on the proxy and receives a list of components to load in response;
2) For each component from the list:
   a. The container checks it's local cache, for each missing package container and downloads it from the proxy's Component Repository;
   b. Container initializes the component instance;
   c. Container subscribes the component instance on the Component Public Chanel and Instance Private Channel.

Container provides messaging API to all the hosted component instances. Typical message transmission sequence looks like shown below:

1) When a component instance calls another component, it sends the call to a Component Public Channel.
2) The first available instance of the component in the cloud system processes the request.
3) The response is returned to an Instance Private Channel of the instance that sent a message.

In addition, a container has an opportunity to work in stand-alone mode. In this mode, the container does not support communication with other containers and acts as a stand-alone computing system (container and proxy at the same time).

4.3 Components

From the developer's perspective, Mjolnirr cloud application is a collection of independent components communicating by message exchange. Components are represented as a package that contains the following information:

1) manifest, that provides the interface of the component, including description of provided methods and their parameters;
2) executable to handle incoming requests;
3) static files, used in pages rendering (images, page layout descriptions and scripts) for UI provision.

Each component can be:

1) *Application Component*: provides the user interface definition, scripts, styles and UI-specific actions. Optionally contains domain logic.
2) *Module Component*: represents a single entity in the domain logic of the application. The Module Component provides data processing and storage, but doesn't provide interface and static files.

4.4 Clients

Mjonirr platform uses HTML and JavaScript to represent end-user interface in common internet browser. But it's possible to use Mjolnirr resources in third-party application using commonly used and standardized protocols.

We've decided to use most common client software – web browser – as a platform default client. Therefore, there is no need to install specialized client software on the client PCs, and it gives the opportunity to use thin clients.

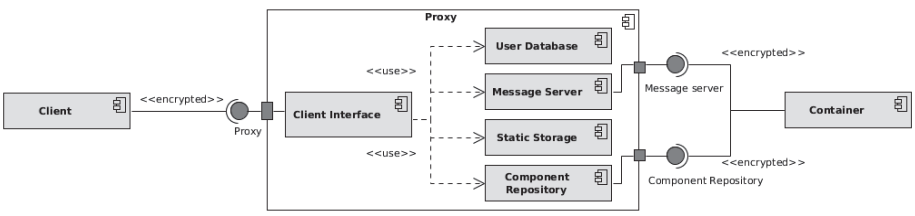


Figure 2: Mjolnirr proxy architecture.

5 IMPLEMENTATION

5.1 Components

Each custom component must have a manifest: an XML file that contains the component's name, the name of the main class, list of methods and a list of parameters for each method. Manifest can be generated automatically using special Maven plugin. Sample manifest for the Calculator service is shown below.

```
<hive>
<application
  name="Calculator"
  classname="org.exmpl.Calculator">
  <methods>
   <method
     name="calculate"
     type="java.lang.String">
    <parameters>
     <parameter
       type="java.lang.String"/>
     </parameters>
    </method>
  </methods>
</application>
</hive>
```

The container parses the manifest for each loaded component.

As stated above, the manifest must have the name of the main class as a fully qualified name of the facade class of described components. Each facade class must have an implementation of the method "*initialize*". The container calls this method immediately after instantiating the component and passes the component *context there*. This context contains information about the working directory of the current application instance and the directory containing the configuration files.

5.2 Messaging Subsystem Implementation

Mjolnirr platform messaging subsystem is implemented on the basis of the queue system called HornetQ (HornetQ project homepage, 2014), which works in accordance with the Publisher-Subscriber pattern. HornetQ can work in embedded mode, it is open source and it provides high performance (Spec.org, 2014). HornetQ server is built into the Proxy. The container provides messaging API to the components. API contains synchronous and asynchronous methods for sending messages.

5.3 UNICORE Integration

The Mjolnirr platform provides a UNICORE 6 integration module. This module uses DiVTB Server for communications with grid environment.

The concept of Distributed Virtual Test Bed (DiVTB) (Radchenko and Hudyakova, 2013) provides distributions of supercomputing simulation in two phases – development of *experiment* and launching *the test stand*. The main advantage of the DiVTB concept is provision of problem-oriented interface to the supercomputer resources.

Mjolnirr UNICORE integration module provides the following methods:

1. *void uploadTestBed (content)* – used for test bed archive upload;
2. *String createExperiment (bedID)* – create experiment from test bed with ID *"bedID",* returns experiment UID;
3. *void startExperiment (experimentID)* – start experiment with specific parameters;
4. *int getStatus (experimentID)* – get experiment status (started, finished or failed);
5. *byte[] getResults (experimentID)* – get experiment results.

6 PERFORMANCE EVALUATION

Mjolnirr Platform was tested in two different experiments on the node with Intel Core i5 M560 and 6 GB of RAM.

6.1 Parallel Execution

We researched the behaviour of a platform during transferring a large number of concurrent messages. The results are shown on the figure 3, left.

6.2 PI Calculation

We also researched the behaviour of the platform during the long calculations. This experiments were conducted on 8 containers running for the N simultaneous requests for N=1..24. The results are shown on figure 3, right.

Experiments have shown that the platform is stable and able to pass the average number of messages; additional transmission costs are about 25 milliseconds per request.

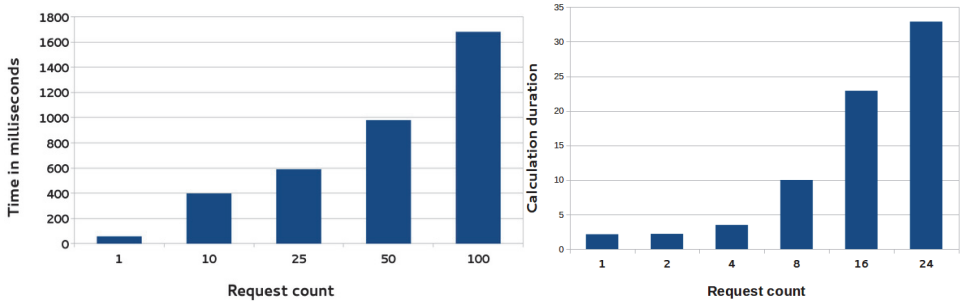


Figure 3: Performance evaluation.

7 CONCLUSIONS

In this article, we described the design and implementation of a private cloud platform called Mjolnirr, which allows development of distributed cloud applications on private computing resources. Main features of the described platform are advanced messaging system and distributed computing support.

As a further development, we will investigate and implement application-level migration support, integration with the advanced resource monitoring systems, flexible adaptation to load changes, advanced system security and application store. The application store will reduce the number of duplicate software products and simplify the creation of individual business infrastructure to meet the needs of a particular company. Also, we will implement a set of standard system services (unified database access, advanced users management and distributed file system access) and improve system security, providing isolation of custom components from the containers to protect the system from accidental or intentional denial of service attacks.

ACKNOWLEDGEMENTS

The reported study was partially supported by Grant Fund for Assistance to Small Innovative Enterprises in Science and Technology research project No. №0000829 and by RFBR, research project No. 14-07-00420-a.

REFERENCES

Activestate.com. 2014. *Stackato: The Platform for the Agile Enterprise | ActiveState*. [online] Available at: http://www.activestate.com/stackato [Accessed: 11 Jan 2014].

Api.yandex.com. 2014. *Cocaine. Cocaine technology description*. [online] Available at: http://api.yandex.com/cocaine/ [Accessed: 11 Jan 2014].

Appfog.com. 2014. *AppFog - PaaS for public and private clouds*. [online] Available at: https://www.appfog.com/ [Accessed: 11 Jan 2014].

Avanade.com. 2014. *Global Study: Cloud computing provides real business benefit | Avanade*. [online] Available at: http://www.avanade.com/us/about/avanade-news/press-releases/Pages/Global-Study-Cloud-Computing-Provides-Real-Business-Benefits-But-Fear-of-Security-and-Control-Slowing-Adoption-page.aspx [Accessed: 11 Jan 2014].

Christodorescu, M., Sailer, R., Schales, D. L., Sg, Urra, D. and Zamboni, D. 2009. Cloud security is not (just) virtualization security: a short paper. pp. 97--102.

Foster, I. and Kesselman, C. 2003. *The Grid 2*. Burlington: Elsevier.

Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M. and Boneh, D. 2003. Terra: A virtual machine-based platform for trusted computing. 37 (5), pp. 193--206.

Mell, P. and Grance, T. 2011. The NIST definition of cloud computing (draft). *NIST special publication*, 800 (145), p. 7.

Peng, J., Zhang, X., Lei, Z., Zhang, B., Zhang, W. and Li, Q. 2009. Comparison of several cloud computing platforms. pp. 23--27.

Pepple, K. 2011. *Deploying OpenStack*. O'Reilly Media.

Radchenko, G. and Hudyakova, E. 2013. Distributed Virtual Test Bed: an Approach to Integration of CAE Systems in UNICORE Grid Environment. *MIPRO 2013 Proceedings of the 36th International Convention*, pp. 183-188.

Santos, N., Gummadi, K. P. and Rodrigues, R. 2009. Towards trusted cloud computing. pp. 3--3.

Streit, A. 2009. UNICORE: Getting to the heart of Grid technologies. *eStrategies*, 3 pp. 8-9.

Spec.org. 2014. *All Published SPEC SPECjms2007 Results*. [online] Available at: http://www.spec.org/jms2007/results/jms2007.html [Accessed: 12 Jan 2014].

HornetQ project page. 2014. *HornetQ – putting buzz in messaging*. [online] Available at: http://www.jboss.org/hornetq [Accessed 12 Jan 2014]

Distributed File System Based on Erasure Coding for I/O-Intensive Applications

Dimitri Pertin[1,2], Sylvain David[2], Pierre Évenou[2], Benoît Parrein[1] and Nicolas Normand[1]
[1]*LUNAM Université, Université de Nantes, IRCCyN UMR CNRS 6597, Nantes, France*
[2]*Fizians SAS, Nantes, France*
{*dimitri.pertin, benoit.parrein, nicolas.normand*}*@univ-nantes.fr,* {*sylvain.david, pierre.evenou*}*@rozofs.com*

Keywords: Distributed File System, RozoFS, Erasure Coding, Mojette Transform, IOzone, Video Editing.

Abstract: Distributed storage systems take advantage of the network, storage and computational resources to provide a scalable infrastructure. But in such large system, failures are frequent and expected. Data replication is the common technique to provide fault-tolerance but suffers from its important storage consumption. Erasure coding is an alternative that offers the same data protection but reduces significantly the storage consumption. As it entails additional workload, current storage providers limit its use for longterm storage. We present the Mojette Transform (MT), an erasure code whose computations rely on fast XOR operations. The MT is part of RozoFS, a distributed file system that provides a global namespace relying on a cluster of storage nodes. This work is part of our ongoing effort to prove that erasure coding is not necessarily a bottleneck for intense I/O applications. In order to validate our approach, we consider a case study involving a storage cluster of RozoFS that supports video editing as an I/O intensive application.

1 INTRODUCTION

Distributed storage systems have been used to provide data reliability. Failures in such large systems are considered as the norm, and can come either from hardware or software considerations. They can result in dramatic data loss and/or the crash of the service. The traditional way to deal with data protection is to replicate the data. Once n copies of data are distributed across multiple network nodes, the system is able to face $n-1$ failures. If a node has a breakdown, its data is not accessible anymore, but other copies are still available on other nodes. On the other hand, this technique is expensive, particularly for huge amounts of data. Replication is the default data protection feature included in distributed storage systems.

Erasure coding is an alternative that provides the same data protection but reduces significantly the storage consumption. Optimal codes, called Maximal Distance Separable (MDS), aim at encoding k data blocks into n parity blocks, with $n \geq k$. The encoded blocks hold enough redundancy to recover the former data from any subset of k parity blocks during the decoding process. Compared to replication whose storage overhead is n, erasure coding's overhead is defined by n/k. However, encoding and decoding require further computations compared to the simple replication technique. Efforts are done today to design efficient codes. The most famous ones are the Reed-Solomon (RS) codes. They rely on Galois field operations. In this arithmetic, addition is fast as it corresponds to exclusive-OR (XOR), but multiplication has many implementations which are much more computational expensive. The storage systems that provide erasure coding by RS suffer from the slowing down of its computations. Many implementations of RS codes exist. Examples of implementation libraries are OpenFEC[1] and Jerasure (Plank, 2007).

We propose the use of the Mojette Transform (MT) (Normand, Kingston, and Évenou, 2006) as an erasure code for distributed storage. The MT relies only on additions and its implementation uses the fast XOR operations. The MT is part of RozoFS[2], an open-source software providing a distributed file system. In RozoFS, when a client wants to store a file, the data is cut into small chunks of 8 KB. Chunks are composed of k blocks of fixed size which depends on the desired protection. These blocks are encoded into n parity blocks, which are discrete projections in the Mojette formalism. These projections are then distributed to storage nodes and the system is able to face

[1]http://openfec.org/
[2]code available at http://www.rozofs.org

$n-k$ failures. Decoding is done client-side after retrieving k blocks out of the n from storage nodes.

The current storage systems provide erasure coding to benefit from the storage capacity saving for longterm storage. When intensive data I/Os are needed, replication is put forward because of the additional workload of erasure codes. Our work differs from this position by exploring the use of an erasure code for intensive applications. We experiment the concept by applying RozoFS to video editing.

The rest of the paper is structured as follows: Section 2 briefly describes some distributed storage systems and their data protection policy. In Section 3, we present the MT and its erasure code characteristics, while Section 4 describes how RozoFS use it to provide a distributed file system. The video editing experiment is presented in section 5 and demonstrates that erasure coding is not a bottleneck for I/O intensive applications. Section 6 concludes the paper with the possible future work.

2 RELATED WORK

Erasure Coding receives significant attention from both the business and the research communities. Scality, Caringo, Isilon or Cleversafe are such examples of companies providing private storage solutions based on erasure codes. Data protection for these systems are partially described in technical white papers.

Scality and Caringo provide replication and erasure coding as a way to protect data. Erasure coding is only used for the longterm storage of massive amounts of data. They both recommend replication for intensive applications. Isilon puts forward the use of erasure coding through a Reed Solomon implementation in OneFS. Replication is only used when erasure coding is not applicable (e.g. too few many nodes). Cleversafe provides exclusively data protection through erasure coding. It relies on the Luby's implementation (Blömer et al.,) of the Reed Solomon algorithm.

Besides these solutions, famous and stable free alternatives exist. One of the most famous free solution is Apache HDFS. This Distributed File System (DFS) aims at archiving and data-intensive computing (i.e. not really an I/O centric DFS). It relies on the MapReduce framework that divides and distributes tasks to storage nodes. Data protection is done by replication, and triplication is the default protection policy. Carnegie Mellon University has developed a version based on erasure codes, called HDFS-RAID (Fan et al., 2009). It is based on Reed-Solomon codes for generic parity computations. HDFS-RAID has been applied in practice (e.g. Facebook Warehouse). However, only a small part of the warehouse's data has been translated by erasure coding. Warm data (i.e. frequently accessed data) is still replicated. Once the data have not been used for a while, the raid-node daemon encodes it and erases the corresponding replicated blocks to make room. GlusterFS[3] and Ceph (Weil et al., 2006) are examples of popular I/O centric DFS. Replication is currently their standard for data protection, but erasure coding is a hot topic on the roadmap.

Erasure coding is already included in private solutions but it is not to be efficient enough for intensive applications. It is exclusively used today for longterm storage and systems benefit from its low data consumption compared to replication. Open-source storage solutions are still actively looking for high speed erasure code implementations as an alternative to replication.

Our contribution, RozoFS, is an open-source software jointly developed by Fizians SAS and the University of Nantes. It provides a POSIX DFS whose file operations are exclusively done by erasure coding. This code is based on the MT whose computations rely entirely on the very fast XOR operations. Thus, we expect RozoFS to be efficient enough for intensive applications.

3 MOJETTE TRANSFORM

The Mojette Transform (MT) is a mathematical tool based on discrete geometry. Long developed at the University of Nantes, its first definition as an erasure code remains in (Normand et al., 1996). A first design for distributed storage relying on the MT was proposed in (Guédon et al., 2001).

3.1 Algorithm

Fundamentally, an erasure code should compute efficiently a desired amount of redundant information from a set of data. This operation is done by a linear operation that should be efficiently inverted. The inverse operation reconstructs the former data from a subset of the computed information.

The Mojette encoding is done as follows. A set of data fills a discrete grid whose elements are identified by (i, j), where i and j are integers. The linear transform consists in summing these elements following different discrete directions. Each direction is defined by a couple of integers (p, q). The elements aligned in

[3]http://www.gluster.org/

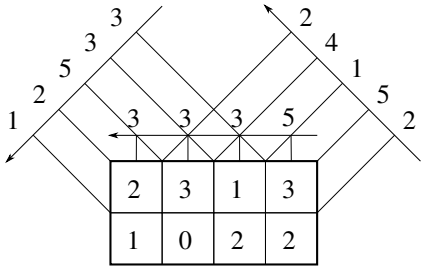


Figure 1: The Mojette Transform, applied to two lines of 4 integers, computes here the following set of 3 projections: $\{(p_i,q_i)=(-1,1),(0,1),(1,1)\}$.

a direction are summed to form a bin. The set of bins defined by the same direction is called a projection. Hence, it is possible to use the Mojette Transform to compute a desired number of projections from a set of lines of data. If k and n are respectively the number of lines of the grid, and the number of projections, and if $n > k$, the set of projections holds redundant information. Figure 1 shows the computation of 3 projections from a set of 2 lines of data.

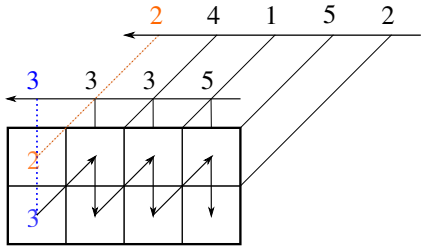


Figure 2: Inverse Mojette Transform as described in (Normand, Kingston, and Évenou, 2006).

The decoding is the inverse operation. It consists in filling the empty grid with the only projection data. Data reconstruction is possible if the Katz's criterion (Normand, Kingston, and Évenou, 2006) holds for the rectangular-shaped grid. This criterion was extended for any convex shape in (Normand et al., 1996). Reconstruction is straightforward for the elements in the corners of the grid. Indeed these elements match entirely a projection bin. (Normand, Kingston, and Évenou, 2006) showed that if a reconstruction path can be found to fill a side of the grid, then it can be applied several time to easily reconstruct the grid. Figure 2 represents such reconstruction considering the example of Figure 1. According to the Katz criterion if the following condition is sufficient to validate the reconstruction:

$$\sum_{i=0}^{r} q_i \geq k \qquad (1)$$

where r is the number of projections that remains, q_i depends on this projection set, and k is the number of lines of the grid. For instance, in Figure 2, $\sum_{i=1}^{2} q_i = 2$ which equals k, so reconstruction is possible. By modifying the number of lines k and the number of projections n, it is possible to set the desired fault-tolerance threshold.

3.2 Implementation

In storage application we cut the input stream into k fixed size blocks. Each line of the grid in the Mojette representation is such a data block. The shape of the grid must be rectangular then. The set of projections is set so that $q_i = 1$. If we suppose that l, the size of blocks (i.e. grid columns) is higher than k, the number of blocks (i.e. grid lines), then the Katz's criterion guarantees that reconstruction is possible from any subset of k projections.

The strength of the Mojette Transform is that encoding and decoding rely entirely on additions. Compared to classical codes like Reed Solomon, there is no need to compute expensive multiplications or divisions in Galois fields. The elements of the grid should fit computer words to improve computation performance. Then, addition is done by efficient exclusive-OR (XOR). Most recent Intel-based processors can perform very fast computations with elements of 128 bits as it fits the dedicated SSE (Streaming SIMD Extensions) registers.

In traditional solutions like Reed Solomon codes, the size of the parity blocks is fixed. The Mojette Transform relaxes this constraint. The size B of each projection varies slightly with the angle of projection, and is given by the following formula:

$$B(k,l,p,q) = (k-1) \mid q \mid +(l-1) \mid p \mid +1 \qquad (2)$$

where k is the number of the grid lines and l the size of blocks. The MT is then considered as $(1+\varepsilon)$ MDS (Parrein et al., 2001), where ε is the quotient of the number of bins required for decoding by the number of element in the grid. The set of projections is taken such as $q = 1$ and p varies around 0 in order to minimize the value of ε.

4 ROZOFS

RozoFS is an open source software solution providing a scalable distributed file system. Once mounted, the file system yields a unique name space (i.e. an hierarchy of directories and files), that relies on clusters of commodity storage nodes. RozoFS manages scalability by adding storage nodes to expand the global resources of the system. For data reliability, it relies on the Mojette Transform as an erasure code.

4.1 Information Dispersal

We consider a network of commodity computers (or nodes) communicating by one or several links. The more the links, the more the paths for packets, and the more reliable and high-performance the system is. Reliability comes from the capacity to communicate even if a link is down. Again, failure probabilities are non negligible and should be considered as the norm. Multiple links induce high-performance because packets can be sent in parallel.

RozoFS considers flows of information of fixed size. A small data chunk of 8 KB fills a Mojette grid as seen in Figure 1. The protection settings, called layouts, define the value of n and k (respectively the number of projections and the number of lines in the Mojette grid).

Table 1: The protection settings provided by RozoFS.

Layout	k	n	storage nodes	fault-tolerance
0	2	3	4	1
1	4	6	8	2
2	8	12	16	4

Currently, three layouts are designed in RozoFS. The table 1 displays the relative information for these configurations. Each layout corresponds to a storage consumption of $n/k = 1.5$ times the size of the input. For instance, we consider the write operation of 1 GB of data in an exported volume of RozoFS, set with the layout 0. It results that 3 projection files of around 500 MB (i.e. plus ε) are distributed across physical disks.

4.2 Stackable Architecture

RozoFS relies on three components:

- *exportd*: the daemon that manages metadata;
- *storaged*: the daemon in charge of the projections;
- *rozofsmount*: used by the clients to mount RozoFS locally.

For the sake of modularity, these components are stackable. For instance, it is either possible to collocate them on a single node, storing projections on different disks, to obtain protection over disk failure (similar to some RAID configurations). In a large network topology, dedicated nodes are much more appreciated for the sake of scalability.

RozoFS Client. Each client can use RozoFS as a standard POSIX file system thanks to the *rozofsmount* process. It relies on FUSE[4] to operates in the user-mode. Clients handle two kinds of operations: (i) the metadata operations (*lookup*, *getattr*, etc), interfacing the export server; (ii) the file I/O operations (read/write the projections), interfacing the storage nodes through the *storcli* subprocesses. The encoding and decoding workload are managed by the clients. The network design should take care of reliability. Services and links must be replicated to provide availability facing failures.

[4]http://www.fuse.sourceforge.net

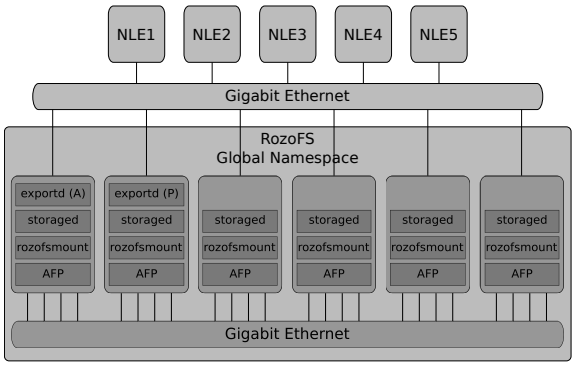


Figure 3: The RozoFS cluster is composed of 6 nodes that hold different services. They provide the DFS to 5 clients running a Non-Linear Editing (NLE) application.

Metadata Server. This node manages the *exportd* process that services the metadata operations. Its configuration file defines the exported file systems. It keeps statistics of storage nodes to provide a data distribution policy based on the storage capacity. The metada server supplies clients with two lists: (i) the list of storage nodes servicing the mounted volume; (ii) the list of storage nodes associate with a regular file (for read/write operations). This service should be replicated to guarantee high availability.

Storage Nodes. These entities hold the *storaged* daemon. Two services are provided by storage nodes: (i) the management services to exportd; (ii) the projection I/O services to the clients, called the *storio* processes. Each storio process can listen to a specific network interface. A storcli groups all storio of a storage node in a load-balancing group to improve availability.

5 EVALUATION

In this section, we explore the capacity of RozoFS to scale as the global workload increases.

Experiment Setup. For our evaluation, we employ a RozoFS cluster of 6 similar servers. Each machine contains an Intel Xeon CPU @ 2.40 GHz, 12 GB of RAM, and 10 disks (7200 rpm SAS) with 2 TB each. These drives are managed by a RAID-5 con-

troller, providing better I/O performances and local reliability. Each node holds 8×1 Gbit Ethernet ports. The exported volume is set to layout 0 providing fault-tolerance against a single failure. The services are distributed as follows:

- the 6 nodes serve as storage servers;
- one among them serves as active metadata server;
- another is a passive/replicated metadata server;
- the 6 nodes mount RozoFS and re-export the file system through the AFP protocol for client access;

Figure 3 displays the platform used here. For high speed communications and reliability, 4 Ethernet ports are reserved for storio processes. RozoFS manages itself the packet scheduling for load-balancing and availability. Because the metadata server is a potential point of failure, it is necessary to set a high-availability strategy. Here, we use DRBD[5] to synchronise the active metadata server with the passive one. Pacemaker[6] is used as a cluster management software that manages the failover mechanisms in case of failure.

The Study Case. We use the previous RozoFS cluster as a storage solution for video editing. Non-Linear Editing (NLE) applications entail intensive workloads for computers. Multiple source editing is particularly file I/O intensive since multiple multimedia contents are accessed simultaneously. For this experimentation, we use the famous NLE software Apple Final Cut Pro[7]. Remote clients, running on Apple Mac OS, attach the RozoFS file system to their own machine using the AFP protocol. Once mounted, the file system provides an easy access to the source media stored on RozoFS. In our case, the system stores video encoded with the lossy compressed format Apple ProRes 422. It requires at least a rate of 200 Mbit/s (i.e. 25 MB/s) to avoid frame drops. We consider that editing involves 5 input video files simultaneously and outputs a single one. The software is both designed for sequential operations (e.g. read the video stream) and direct access to a part of the video.

5.1 IOzone Benchmark

IOzone[8] is a filesystem benchmark that can output file I/O performance according to a set of stressing tests. In particular, the software can perform sequential and random read/write tests, which are in accordance with our study case. We explore the capacity of RozoFS to adapt, facing a growing number of clients accessing the file system simultaneously.

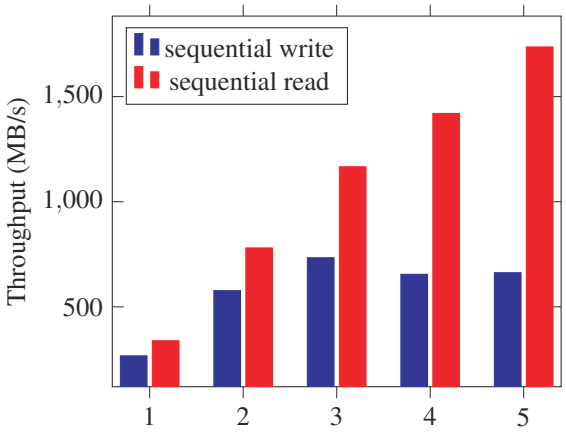


Figure 4: Accumulated throughput (in MB/s) recorded by the threads, depending on the number of clients working in parallel, for sequential access operations.

IOzone takes the following parameters. The file size is set to 1 GB which is larger than the CPU cache, and smaller than the RAM capacity. To fit the multi-source editing, each client involved in the test manages 5 threads. A thread reads or writes the 1 GB file according to the desired access strategy. For instance, in a writing test, each node induces the writing of 5 GB in RozoFS (i.e. which represents 7.5 GB in the physical disks).

We measure the accumulated throughput recorded by the threads as the number of clients grows. Each thread should access data with a rate of at least 25 MB/s to validate the study case. Figure 4 and 5 respectively display the average results from 5 test iterations for sequential and random access.

Benchmarking Sequential I/Os. The write operations are asynchronous and multiple write requests are done simultaneously. Figure 4 shows that as the workload grows, the performance for sequential write scales up to 3 nodes. When more clients are added, the nodes are overwhelmed by the requests and cache misses slow down the performances as data need to be fetch on the disks. The sequential read operations significantly benefit from fast access as data is pre-fetched in the cache. The performance scales up as the number of clients increases. For instance, when 5 clients are involved, the 25 threads record an accumulated throughput of 1800 MB/s. Each client receives $1800/5 = 360$ MB/s and each thread receives $360/5 = 72$ MB/s. In any case in this benchmark, we validate the study case as each thread receive more than 25 MB/s. Mechanical disks are particularly sensitive to the type of access. For sequential access, the

[5] Distributed Replicated Block Device (www.drbd.org)
[6] http://clusterlabs.org/
[7] http://www.apple.com/final-cut-pro/
[8] http://www.iozone.org

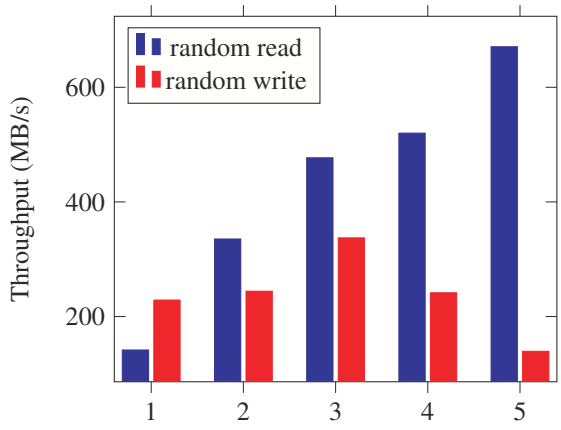


Figure 5: Accumulated throughput (in MB/s) recorded by the threads, depending on the number of clients working in parallel, for random access operations.

disk's arm move the read-and-write head to reach the correct track on the physical disk. Once there, the following accesses are straightforward.

Benchmarking Random I/Os. Random operations are slowed down by intermediate seeks between operations, increasing the latency of disk's head. Figure 5 shows that the throughput suffers from these random accesses. During random read operations, the performance scales as it benefits from small cache effects and each thread receive at least 25 MB/s. However, the random write test is clearly the worst case and shows the limit of hardware as the performances collapse. For 3 clients, each thread receives $350/3/5 = 23$ MB/s. For more clients, performances get worse. We should note that 5 writing threads per client does not fit our study case. For video editing, each editors outputs a single file. It would correspond to a single write thread in our case.

6 CONCLUSIONS

In this work, we have presented the Mojette Transform (MT) as an efficient alternative to classical erasure codes since it relies on fast computations, and a great candidate to handle intensive applications. We have set the MT as the I/O centric process in RozoFS with good expectations for practice efficiency. Finally, we have designed an evaluation based on a RozoFS cluster. The platform is able to handle multiple parallel access from intensive I/O applications (i.e. multiple source video editing). The evaluation has revealed that erasure coding is not a bottleneck for intensive applications and should not be limited to longterm storage.

More specific measures should be done to reveal the real cost of the MT in the computational time. Our evaluation could be extended to more intensive applications like virtualization, which access data over block devices. There is clearly a need for comparisons with other existing solutions. The MT must be compared to erasure code libraries such as OpenFEC and Jerasure. Further erasure code aspects beyond computational considerations should be explored, such as the node repair problem.

ACKNOWLEDGEMENTS

This work has been supported by the French Agence Nationale de la Recherche (ANR) through the project FEC4Cloud (ANR-12-EMMA-0031-01).

REFERENCES

Blömer, J., Kalfane, M., Karpinski, M., Karp, R., Luby, M., and Zuckerman, D. Technical report.

Fan, B., Tantisiriroj, W., Xiao, L., and Gibson, G. (2009). Diskreduce: Raid for data-intensive scalable computing. In *Proc. PDSW 2009*.

Guédon, J. P., Parrein, B., and Normand, N. (2001). Internet distributed image information system. *Integr. Comput.-Aided Eng.*, 8(3):205–214.

Normand, N., Guédon, J., Philippé, O., and Barba, D. Controlled redundancy for image coding and high-speed transmission. In *Proc. VCIP 1996*, Orlando, FL.

Parrein, B., Normand, N., and Guédon, J. P. (2001). Multiple description coding using exact discrete Radon transform. In *Proceedings of the Data Compression Conference*, DCC '01, pages 508–.

Plank, J. S. (2007). Jerasure: A library in C/C++ facilitating erasure coding for storage applications. Technical Report CS-07-603, University of Tennessee.

Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D. E., and Maltzahn, C. (2006). Ceph: A scalable, high-performance distributed file system. In *Proc. OSDI 2006*.

Data Integrity in Cloud Transactions

Sachi Nishida and Yoshiyuki Shinkawa
Department of Media Informatics, Faculties of Science and Technology, Ryukoku University,
1-5 Seta Oe-cho Yokotani, Otsu, Shiga, Japan
t100424@mail.ryukoku.ac.jp, shinkawa@rins.ryukoku.ac.jp

Keywords: Colored Petri Nets, VDM++, Cloud Computing, Transaction Processing, Data Integrity.

Abstract: BASE is a new principle for transaction processing in cloud computing environments, which stands for **B**asically **A**vailable, **S**oft state, and **E**ventually consistent. Unlike the traditional principle of ACID for transactions, BASE pays greater attention to scalability and availability of systems, and less to the data integrity. Therefore, when migrating mission critical transaction systems into cloud environments, we have to carefully evaluate whether the provided data integrity level is acceptable. This paper presents a formal, simulation-based, and model driven approach to evaluating data integrity in BASE transaction systems. We define the data integrity in cloud environments using a new concept for system states, referred to as superposed states. As a formalization tool, VDM++ and Colored Petri Nets (CPN) are used to express and simulate the BASE transaction systems accurately.

1 INTRODUCTION

Transaction processing is one of the most important bases of enterprise information systems, along with database management. Consequently, a seamless migration of it into cloud computing environments is a critical success factor for enterprise-wide usage of cloud computing. Despite the many advantages in scalability, availability, and cost-effectiveness, there are several difficulties and obstacles in this migration. Among them are data integrity, execution performance, security, and so on. Especially, the data integrity is the first and most critical issue in the migration, since it affects the correctness, validity, and soundness of business operations supported by the enterprise information systems.

This difficulty is caused by the different principle for the data integrity from the traditional on-premise transaction processing. This principle is known as BASE (Pritchett, 2008), standing for **B***asically* **A***vailable*, **S***oft state*, and **E***ventually consistent*. Whereas in traditional transaction processing, ACID (Weikum and Vossen, 2001) is the principle for the data integrity, which stands for **A***tomicity*, **C***onsistency*, **I***solation*, and **D***urability*. The fundamental difference between two principles is that while ACID focuses on the data integrity assurance at every time point, BASE focuses on that at some future time point in order to emphasize the scalability and availability of the systems. At the implementation level, this difference is shown as the exclusive control or locking mechanism for the database.

The ACID transactions adopt pessimistic locking for a higher transaction isolation level, whereas the BASE transactions usually use optimistic locking for higher concurrency. This difference could considerably affect the application level data integrity when we migrate transaction processing systems into cloud environments. Therefore, we need a systematic and sophisticated approach to verifying, validating, and evaluating the data integrity for the transaction processing in cloud environments.

This paper presets a formal, simulation-based, and model-driven approach to evaluating data integrity in a BASE transaction system, which would be the common platform in cloud transaction processing. Two formal techniques are used in this approach, namely, the formal specification language VDM++ (Fitzgerald et al., 2004), and Colored Petri Nets (CPN) (Jensen and Kristensen, 2009).

2 SYSTEM STATUS AND DATA INTEGRITY

In this section, we define the concept of data integrity in transaction systems based on the system states.

2.1 The States in Transaction Systems

The main purpose of a transaction is to update the related databases reflecting the "current state" of the system in which it runs. This update changes the attribute values of the accessed database records, and causes a "state transition" in the system, if we regard the system state as a set of data values in the databases (Shinkawa, 2012). Each database in the system is composed of the database schema which determines its structure, and a set of actual database records created through the schema. This set is often referred to as an instance of the database schema[1].

More formally, the state of a system at time t can be regarded as a set of all the database records which determine the attribute values characterizing the system. Assuming the database schemas in the system are $DB_1, \cdots, DB_n$, and the instances of them at the time t are $\overline{DB_i}$, the state of the system at t is defined as

$$\mathrm{S}(t) = \bigcup_i \overline{DB_i} = \bigcup_{i,j} r_{ij}$$

where r_{ij} represents the j-th record in the database $\overline{DB_i}$.

In case of ACID transaction systems, some database records might be locked by the TPM (Transaction Processing Monitor), and they cannot be observed from the outside. Considering this situation, the state of a system is expressed as

$$\mathrm{S}(t) = \bigcup_{i,j} r_j^i - \bigcup_k \mathcal{L}(T_k)$$

where $\mathcal{L}(T_k)$ represents a set of the records locked by the transaction T_k.

On the other hand, in case of BASE transaction systems running in cloud computing environments, all the database records are available to be accessed by any transactions, because of the *Basically Available* property. However some of them could be incompletely updated, inserted, or deleted, which might cause *dirty read*, *phantom read*, or *non-repeatable read*. In addition, the property of *Soft state* allows the different values for the same entity between the different sites or nodes in the networked environments. Since the property of *Eventual consistency* assures the consistency at a future time, the system state must be considered over some duration.

The following steps show how to define the system states in such a situation.

1. Let the transactions running in the system at time t be $T = \{T_1, \cdots, T_n\}$.
2. Let $\mathcal{D}(T_i)$ be the set of database records to be updated, deleted, or inserted by the transaction T_i. By the *Eventually consistent* property, either T_i or T_j will be committed if $D_i \cap D_j \neq \emptyset (i \neq j)$ holds[2].
3. For a set of $P \subseteq \{1, 2, \cdots, n\}$, if
$$\forall i, j \in P[\mathcal{D}(T_i) \cap \mathcal{D}(T_j) = \emptyset]$$
holds, all the transaction $T_i (i \in P)$ will be committed, whereas $T_k (k \notin P)$ will be aborted. Then the system will transit to another consistent state, and each $\mathcal{D}(T_i)$ will be updated to the another set of database records $\tilde{\mathcal{D}}(T_i)$,
4. Since there could be multiple such sets of natural numbers $P_1, P_2, \cdots, P_l$, there are multiple possible states $\tilde{S}_1, \tilde{S}_2, \cdots, \tilde{S}_l$ including
$$\bigcup_{i \in P_1} \tilde{\mathcal{D}}(T_i), \bigcup_{i \in P_2} \tilde{\mathcal{D}}(T_i), \cdots, \bigcup_{i \in P_l} \tilde{\mathcal{D}}(T_i)$$
.
5. From the above considerations, the state of a BASE transaction system is defined as tuple of states
$$S(t) = \langle \tilde{S}_1, \tilde{S}_2, \cdots, \tilde{S}_l \rangle$$
which is a set of possible future states.

Since this state is overlaid or superposed, we refer such a state as "superposed state".

The above set of states can be identified using a binary tree as follows, when there are concurrently executing transactions $T_1, T_2, \cdots, T_n$.

1. Put an arbitrary $T \in \{T_i\}$ on the root node.
2. Let $\mathcal{T}_L$ be the set of the transactions that can be committed together with the transaction T, that is, there are no database records conflicting with T.
3. Let $\mathcal{T}_R$ be the set $\{T_i\} - \mathcal{T}_L$.
4. Put an arbitrary $T_l \in \mathcal{T}_L$ as the left child node of the above T, and an arbitrary $T_r \in \mathcal{T}_R$ as the right child node of T, then let $\mathcal{T}_R$ be $\mathcal{T}_R \cup \mathcal{T}_L$.
5. Repeat the same process on the node T_l, T_r, and all the succeeding child nodes until there is no transaction left except the node transaction. If $\mathcal{T}_L$ or $\mathcal{T}_R$ is an empty set, put *nil* on the node.

The above process yields the binary tree of which nodes consist of the concurrently running transactions.

Each node of the above binary tree is represented as T_λ, where $\lambda = \lambda_1 \cdots \lambda_m (\lambda_i \in \{L, R\})$ shows the depth of the node. For each leaf node transaction $T_\lambda = T_{\lambda_1} \cdots \lambda_m$, a set of committable transactions can be derived as follows.

[1] If there is no confusion, we will not distinguish between the terms "database" and "instance of a database schema".

[2] This functionality is implemented by the optimistic locking in many cloud environments.

1. Add a left leaf transaction $T_{\lambda_1 \cdots \lambda_m}$ and the parent node $T_{\lambda_1 \cdots \lambda_{m-1}}$ to the set. If it is a nil node, skip it.
2. If $\lambda_{m-1} = L$, then continue the process, otherwise move to another left leaf node.
3. Repeat the process until reaching the root node T.

2.2 Data Integrity in BASE Transactions

The term "data integrity is used in contexts with a vague meaning. Before discussing the data integrity in BASE transactions, we have to define the meaning of the data integrity more accurately for rigorous evaluation. One of the possible definition is a set of rules that all the states caused by the appropriate transaction executions must follow (Shinkawa, 2012).

Since a state in a BASE transaction system is determined by a set of values that each database record shows, the above rules can be regarded as the constraints on these values. Therefore, one of the simple ways to express these constraints is to express them in the form of the predicate logic formulae over the values of the database records (Shinkawa and Matsumoto, 2001). In order to express the constraints by the predicate logic, we have to explicitly define the domain of discourse, functions, predicates, variables, and constants, with syntax and semantics.

The domain of discourse, which is a set of entities or objects that are to be dealt with the functions and predicates in the logic formulae, is composed of all the database records in the BASE transaction system currently considered. For practical convenience, each database which is a set of specific database records, and each attribute of the database records, are also regarded as domain elements. Therefore, the domain of discourse is formally defined as

$$\mathcal{D} = (\bigcup_i D_i) \cup (\bigcup_{i,j} r_{ij}) \cup (\bigcup_{i,j,k} a_{i,j}^{(k)})$$

where D_i is the i-th database, r_{ij} is the $j-$th record in the database D_i, and $a_{ij}^{(k)}$ is the k-th attribute of the r_{ij}.

In addition to this domain of discourse, we have to define the functions, predicates, variables, and constants rigorously. Assuming we have defined all these elements of the logic, any constraints on the databases can be represented by a prenex conjunctive normal form (PCNF) as

$$Q_1 \cdots Q_n (\bigvee_j \bigwedge_i P_{ij}(t_1^{(ij)} \cdots t_{m_{ij}}^{(ij)}))$$

where Q_i is a variable with the quantifier "$\forall$" or "$\exists$", e.g. $\forall x_i$ or $\exists x_i$, P_{ij} is a predicate, and $t_k^{(ij)}$ is a term composed of variables, constants, and functions (Schoening, 2008).

However, these logical elements are quite application oriented matters, and a mathematical representation is functionally insufficient to express various complicated constraints. In the next section, we introduce a formal specification language VDM++ to represent these constraints rigorously.

3 SPECIFYING BASE TRANSACTIONS BY VDM++

VDM++ is a set-theoretic and logic based specification language for object orientation, and not for database related systems. However, there are some similarities between databases and objects as follows.

1. A database schema corresponds to a class
2. An attribute corresponds to an instance variable
3. A database record corresponds to an object or a class instance
4. Database manipulations such as insert, update, delete, or select correspond to class or instance methods of a class

In addition, each instance of a database schema, that is, a collection of database records[3], can be regarded as a list of the class instances, and is to be expressed in VDM++ as a sequence with the type of the corresponding class.

From the above similarities, we conclude that any database structures can be expressed as VDM++ specifications. Each constraint for the data integrity, which is expressed as a logic formula in the form of PCNF, can also be expressed as a boolean operation within the corresponding class, assuming the related functions and predicates are expressed by VDM++. The process for expressing these functions and predicates is performed in the same way as the software development using VDM++.

In specifying the functions, predicates, and logic formulae, VDM++ imposes some restrictions on us. For example, while only operations can access the class and instance variables, only functions can use higher order functions and quantifiers ($\forall$, $\exists$). Therefore, we might be required to separate some functionality into the two parts, namely an operation and function parts.

In addition to the databases, the logic of each transaction is to be specified by VDM++, in order to evaluate the data integrity rigorously.

[3]In the relational database model, they are referred te as tuples

4 MODELING BASE TRANSACTIONS USING CPN

As discussed in section 2, each state in BASE transaction system is superposed by multiple future states, therefore the data integrity constraints must be satisfied in all of these multiple states. For such a complicated data integrity evaluation, a simulation approach seems more suitable than a logical approach.

In order to simulate complicated transaction systems, we first have to model the target systems preciously from the data integrity viewpoint, using an appropriate modeling tool. In this paper, we use Colored Petri Nets (CPN) as a modeling and simulation tool, since CPN can model and simulate the target systems from behavioral, functional, and structural viewpoints simultaneously, along with the powerful description language CPN/ML (Jensen and Kristensen, 2009) (Harper et al., 1997).

CPN is an extension of a regular Petri nets, which introduces functionality and data types for more flexible control of the regular Petri nets. CPN is formally defined as a nine-tuple *CPN=(P, T, A, Σ, V, C, G, E, I)* , where

P : a finite set of places.
T : a finite set of transitions.
(a transition represents an event)
A : a finite set of arcs $P \cap T = P \cap A = T \cap A = \emptyset$.
Σ : a finite set of non-empty color sets.
(a color represents a data type)
V : a finite set of typed variables.
C : a color function $P \rightarrow \Sigma$.
G : a guard function $T \rightarrow$ expression.
(a guard controls the execution of a transition)
E : an arc expression function $A \rightarrow$ expression.
I : an initialization function : $P \rightarrow$ closed expression.

In order to evaluate the data integrity in BASE transaction systems, we have to express the behavior of transactions from the following three orthogonal viewpoints. The first is the infrastructure viewpoint, which represents the transaction processing mechanisms including scheduling, queuing, concurrency control, database accesses, and recovery control. The second is the database viewpoint, which represents the structure of the databases to which each transaction accesses. The third is the internal or intra-transaction viewpoint, which represents the application logic in each transaction.

Modeling from the Infrastructure Viewpoint
The detailed mechanism of each cloud computing platform, such as Google App Engine (Sanderson, 2009) or Amazon EC2 (van Vliet and Paganelli, 2011) is not always disclosed, and therefore it seems to be difficult to build an accurate model of it. However, the data integrity mechanism is usually disclosed, otherwise the users cannot expect the reliability of the cloud environment. Therefore, as far as the data integrity is concerned, we can model it.

The basic structure of a transaction system is composed of (1). scheduling, (2). transaction execution, (3). database access, (4). logging, and (5). consistency management components or modules. These components are commonly used in both ACID and BASE transaction systems with the different implementations. For BASE transaction systems, these components are structured into the top-level hierarchical CPN model as shown in Figure 1. Each double-lined rectangular box represents one of the above components, which is referred to as a *substitution transition*. These substitution transitions are to be refined stepwise from abstract levels to concrete levels. During this refinement, actual mechanisms are incorporated in to the model.

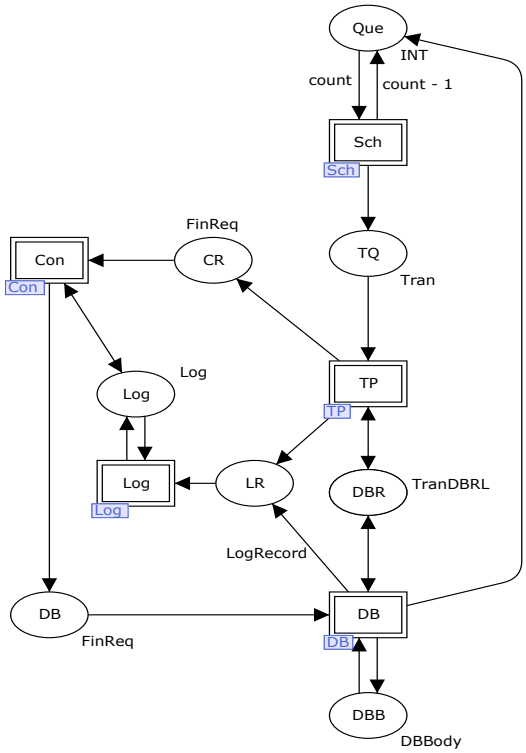


Figure 1: Top Level CPN Model.

Modeling from the Database Viewpoint
The structure and characteristics of each database is specified as a class by VDM++. These specifications include the information on both the conceptual and physical aspects of the databases. On the other hand,

CPN models need only the physical aspects of the databases for simulation and evaluation. Therefore, a class definition is transformed into a CPN model as follows.

1. Each class name, corresponding to a database name, is mapped to an unique integer number defined as the color
   colset DBID = int;
2. Each instance of a class, corresponding to an database record, is mapped to an element of the color set
   colset DBE = product DBID * Key * AttList;
   where “Key and “AttList are the primary key and the list of the attributes reppectively.
3. The sequence that is defined in a class, corresponding to the the instance of the database schema, is mapped to the color set
   colset DBBody = product DBID * DBEList;
   where “DBEList is a list of DBE.

The other operations and functions are transformed into CPN ML codes according to the VDM++ specifications, and assigned to the appropriate arcs or transitions as the arc or guard functions respectively.

Modeling from the Internal Viewpoint

A transaction accesses and updates databases according to the associated application logic. This logic reflects various application requirements and usually includes application domain specific knowledge. However, from the data integrity viewpoint, only the order of database accesses is important. Therefore, one of the simplest way to express the internal logic of a transaction is to use an ordered list of database operations.

The element of this list must represent the information on the accessed database record, including the database id, primary key, and the list of the attribute values in the record. This element is expresses as the color set

colset DBA = product DBID * Key * list AttList;

Each transaction execution yields the instance of this color set, appending the transaction id, which is designated by the color set

colset DBR = product TRID * DBA;

While each transaction implements different logic from each other, a single CPN ML code have to be assigned as an arc function. Therefore, we prepare the transaction specific CPN ML codes to be invoked from a common arc function assigned to the arc.

5 EVALUATING DATA INTEGRITY BY CPN

Once the above three viewpoints are integrated into a single CPN model, we can simulate the transaction system using cpn tools (Jensen and Kristensen, 2009), and can examine the resultant databases whether they satisfy the data integrity constraints. However, unlike ACID transaction systems, BASE transaction systems could hold superposed or overlaid states in concurrent execution environments, as discussed in section 2.

The data integrity in such superposed states becomes complicated, since we have to examine all the possible states reachable from every sets of concurrent transactions. In order to accomplish this complicated evaluation, we need to add the following mechanism to the CPN models previously created.

1. Monitoring the concurrency of transaction executions.
2. Detecting the change in concurrency.
3. Identifying the possible future states based on the database contention.
4. Examining each posible state whether it satisfies the data integrity constraints.
5. Restarting the CPN model after each evaluation.

In addition to the above mechanisms, an initialization mechanism would be needed to perform the evaluation automatically under various situations. This initialization module generates the initial tokens for database records and transactions to be evaluated.

By introducing these mechanisms, the highest level of the CPN model becomes as shown in Figure 2. In this model, the monitoring module, which is represented as a substitution transition “Mon works as follows.

1. Hold the information on the concurrently executing transactions in the token with the color
   colset Hist = product Seq * list TRID;
2. Determine the database contention by examining all the tokens in the place “DBR associated with the “TranDBRL. From these tokens, this module builds the binary tree introduced in section 3, in order to identify the set of committable transactions.
3. When a new transaction is submitted to the place “TQ, add the TRID to the “Hist token with the highest “Seq number. This token with the highest Seq number represents the *current* Hist token.
4. Examine the other Hist tokens whether the current combination of the transactions currently executing is different from the past ones.

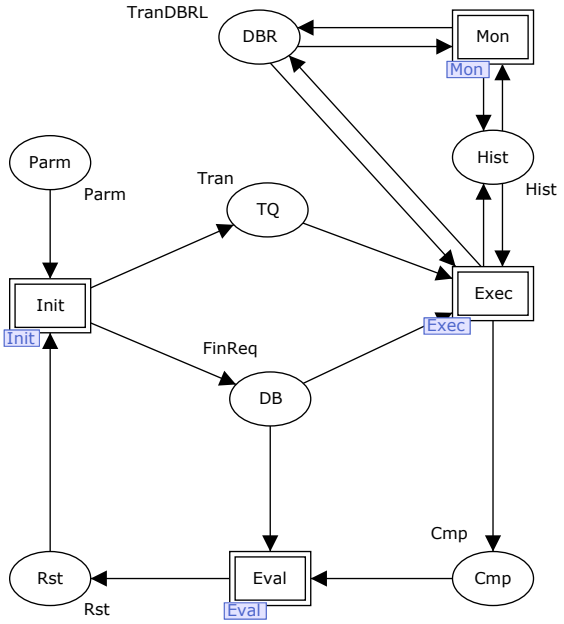


Figure 2: Evaluation CPN Model.

5. If the combination is new, initiate the "Eval module. Otherwise, continue the monitoring.

The "Eval module performs the following functionality for the data integrity evaluation.

1. Receive the above binary tree from the Monitoring module.
2. Construct the lists each of which includes the concurrently committable transactions.
3. Request the commit operation for all the transactions in one of the above lists, and the abort operation for the rest of the currently executing transactions.
4. Examine the resultant database tokens in the "DB place, whether they satisfy the data integrity constraints,
5. Repeat the step 3 to 4 until all the lists are processed.

At the end of the "Evaluation module, it checks the "TQ place whether there are unprocessed transactions left. If there are, it calls the "Initialization module to restart the evaluation.

6 CONCLUSIONS

When we migrate transaction processing from on-premise systems to cloud environments, we have to pay close attention to data integrity, because of the essential differences between ACID and BASE principles. In order to evaluate the data integrity in BASE transaction systems, this paper proposed a set-theoretic and logic based way to define the superposed or overlaid states to evaluate the data integrity in databases with unstable data. These states and data integrity constraints are expressed rigorously using the VDM++ specification language, then are transformed into a Colored Petri Nets (CPN) model. This model represents a BASE transaction system from three orthogonal viewpoints, namely, the transaction processing platform, databases, and the internal logic of each transaction. Adding the monitoring and evaluation functionality to this CPN model, we can simulate the BASE transactions and examine the resultant databases against the data integrity constraints.

REFERENCES

Fitzgerald, J., Larsen, P., Mukherjee, P., Plat, N., and Verhoef, M. (2004). *Validated Designs for Object-oriented Systems*. Springer.

Harper, R., MacQueen, D., Milner, R., and Tofte, M. (1997). *The Definition of Standard ML*. The MIT Press.

Jensen, K. and Kristensen, L. (2009). *Coloured Petri Nets: Modeling and Validation of Concurrent Systems*. Springer-Verlag.

Pritchett, D. (2008). BASE: An ACID alternative. In *ACM QUEUE Volume 6 Issue 3*, pages 48–55. ACM.

Sanderson, D. (2009). *Programming Google App Engine*. Oreilly & Associates Inc.

Schoening, U. (2008). *Logic for Computer Scientists (Modern Birkhaeuser Classics)*. Birkhaeuser Boston.

Shinkawa, Y. (2012). CPN Based Data Integrity Evaluation for Cloud Transactions. In *Proc. 6rd International Conference on Software Paradigm Trends*, pages 267–272.

Shinkawa, Y. and Matsumoto, M. (2001). An Information System View of Consistency and Integrity in Enterprise Operations. In *Proc. 3rd International Conference on Enterprise Information Systems Volume 2*, pages 709–716.

van Vliet, J. and Paganelli, F. (2011). *Programming Amazon EC2*. Oreilly & Associates Inc.

Weikum, G. and Vossen, G. (2001). *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. Morgan Kaufmann.

CLOUD COMPUTING ENABLING TECHNOLOGY

FULL PAPERS

ALPS: The Accountable Cloud Protocol Stack

Wassim Itani[1], Ayman Kayssi[2] and Ali Chehab[2]
[1]*Department of Electrical and Computer Engineering, Beirut Arab University, Beirut, Lebanon*
[2]*Department of Electrical and Computer Engineering, American University of Beirut, Beirut, Lebanon*
w.itani@bau.edu.lb, {ayman, chehab}@aub.edu.lb

Keywords: Privacy in Cloud Computing, Privacy Feedback, Accountable Auditing, Consent and Revocation in the Cloud, Trust and Reputation, Service Routing.

Abstract: The cloud model inherently invites concerns of privacy beyond the traditional worries. Technologies and architectures, physical boundaries, policies and legal constraints all contribute to elevating privacy concerns in the cloud. Existing privacy solutions lack the comprehensiveness required to assure privacy for data in transit and at rest in the cloud. Data privacy has to be protected not just from external entities but also from cloud constituents including cloud providers themselves. We propose in this research an Accountable cLoud Protocol Stack (ALPS) that consists of a set of inter-dependent protocol building blocks, designed in a layer-based architecture, for supporting the development of a wide variety of privacy-aware cloud services and applications. The significance of the stacked architecture of ALPS is that it allows an incremental identification of the different building blocks across the layers, a clean separation of services, ease of development of manageable blocks, and flexibility of evolution. We propose through this research to build this layered cloud computing privacy solution and to implement and test it in real cloud environments relying mostly on the variety of services available from the Amazon, Google, and Microsoft cloud infrastructures.

1 INTRODUCTION

The cloud computing model inherently suffers from a set of challenges that are drastically delaying its pervasive acceptance and implementation. Security and privacy concerns in cloud computing implementations received the chief attention especially after the outbreak of several incidences regarding the leakage of sensitive customer data and records to the outside world (The New York Times, 2011). These security and privacy concerns sprout from the intrinsic nature of the cloud computing model which utilizes a virtual computing and storage infrastructure running on top of a commodity hardware layer that is completely owned and controlled by the cloud provider. This model decreases the user's trust in cloud services and raises a lot of question marks on the security assurances that can be provided to prevent incidents of data leakage, misuse, or loss. Unfortunately, the lack of privacy enforcement mechanisms in cloud platform services is not merely jeopardizing the confidentially and integrity of customer data, but also crippling the implementation of a wide set of cloud applications that rely on basic privacy constructs to support their services.

The goal of this research is therefore to accelerate the adoption of the cloud computing model by addressing security and privacy concerns, through the development of a layered architecture of a set of interrelated protocols targeting privacy protection in the cloud.

We propose to design, analyse, implement and test an Accountable cLoud Protocol Stack (ALPS), composed of a set of inter-dependent protocol layers supporting cloud services that meet stringent security and privacy requirements. Each protocol layer uses the services provided by the lower layer to satisfy the security and privacy requirements of cloud applications adopting the protocol stack model. The advantage of the layered architecture of ALPS is that it allows for the incremental design, analysis, implementation and testing of services. We propose through this research to lay the ground for building a layered cloud computing solution consisting of a set of protocols targeting some of the major security and service manageability challenges currently facing cloud computing.

The primary outcomes of the proposed work are the following: (1) demonstrate how basic privacy

enforcement mechanisms should lie at the crux of a broad range of protocol building blocks tackling several challenges facing cloud computing and its adoption, (2) comprehensively develop and analyse the protocol dependencies and interrelations across the different stack layers and, accordingly, come up with new protocols or additional extensions to existing protocols, and (3) devise a set of general design patterns and platform blueprints that aid in selecting the appropriate privacy mechanisms required by a particular cloud application, and mapping these requirements in a one-to-one fashion or a one-to-many fashion to the respective ALPS layers.

The main challenges facing the accomplishment of these outcomes are represented in: (1) finding the right association between the software and hardware privacy enforcement mechanisms to support the core privacy services at the base of the protocol stack, and to cover the broadest range of cloud application classes, (2) identifying the cloud applications that would benefit from the provided privacy services based on their direct or indirect privacy requirements and (3) establishing a clear and deep understanding of how application requirements map to the set of hierarchical privacy services offered by the cloud stack.

The work proposed here is the first that deals with privacy as a core concept for not just supporting customer data confidentiality and integrity when processed and stored in the cloud, but also for supporting a wide range of applications and services that would go paralyzed without such privacy support. In other words, the application of the ALPS protocol stack provides cloud applications with a rigorous privacy support that can be utilized in building various application domains in the fields of secure audit logging, provider reputation, consent and revocation, and service routing. This is the main contribution that differentiates this work from that existing in the literature.

The rest of this paper is organized as follows: in Section 2 we present some research work related to the privacy building blocks presented in the ALPS protocol stack. Sections 3 and 4 respectively describe the system model and design of the algorithms and cryptographic constructs developed at the various stack layers. Section 5 presents a sample implementation of the ALPS stack in a virtual lab environment as well as in a real cloud computing environment. Conclusions are presented in Section 6.

2 RELATED WORK

The real security and privacy concerns arising from the adoption of the cloud computing model has resulted in several good reports on this topic. Early reports on cloud computing privacy are those presented by Gellman (2009) which shed light on the different risks imposed by cloud computing on the privacy, confidentiality, and legal compliance of customer data and in (Cavoukian, 2008) which focuses on the fact that security and privacy concerns in cloud computing can only be tackled via establishing a solid digital identity infrastructure. In (Pearson, 2009; Pearson & Charlesworth, 2009), Pearson discusses a comprehensive set of guidelines that represent a blueprint for designing privacy-aware cloud services and solutions. In (Pearson, 2009), Pearson summarizes the privacy guidelines in six suggested patterns: "(1) Minimizing the amounts of sensitive customer data sent for storage and processing in the cloud; (2) Protecting sensitive customer data in the cloud; (3) Maximizing user control on the cloud data and services; (4) facilitating user choice; (5) Formally indicating the purpose of data usage and restraining this usage merely to the required purpose; (6) Supporting the customer with an accountable privacy feedback process".

In the last few years the notion of homomorphic encryption (Gentry, 2011; Van Dijket al., 2010; Gentry, 2009) for permitting algebraic operations on encrypted data has gained some recognition and considered one of the solutions for ensuring the privacy of customer data as it is processed and stored in the cloud. However till this day, homomorphic encryption schemes are still theoretical and lack any feasible implementation. According to (Schneier, 2009) a practical fully-homomorphic encryption scheme is yet an open research area.

In (Atallah & Frikken, 2010), the authors present a set of secure and reliable protocols for outsourcing linear algebra operations. The main advantage of their approach is that it does not rely on expensive homomorphic encryption techniques but rather on simple polynomial arithmetic. More research on secure outsourcing protocols of this kind can really aid in better privacy support for data processing in cloud computing. The main challenge here is to be able to securely outsource the different algorithmic blocks that can fully satisfy the requirements of generic computations in cloud computing.

Hardware-based security protocols are believed to be the natural trend that the cloud computing

market will follow in the coming years to resolve the different data privacy and integrity challenges hampering the wide adoption of cloud computing, particularly in healthcare and financial applications. This fact is corroborated by a set of signals received from the IT industry itself:

1. The considerable advancement in physical security mechanisms and packaging technology and the assortment of secure applications that can be implemented on top of physically-secure cryptographic coprocessors. Cryptographic coprocessors (Best, 1980; Tygar & Yee, 1993) represent a prominent trusted computing building block that is believed to play a major role in building secure cloud protocols. A cryptographic coprocessor is a PCI-based card that interfaces into the motherboard of a server system. It represents a dedicated hardware platform and computing system supported with a central processing unit, memory, persistent storage, and networking capabilities. A crypto coprocessor is physically protected by a tamper-responding enclosure that resets the internal state of the unit (volatile and non-volatile storage in addition to any processor registers) once suspecting any wary physical activity on the system.
2. The availability of a set of successful crypto coprocessor implementations meeting the strictest FIPS 140 security standards (Dyer et al., 2001).
3. The emergence of general-purpose open-source cryptographic coprocessor designs that provide competitive performance and higher functionality as compared to commercial products with one to two orders of magnitude reduction in cost.
4. The proposed work of the Trusted Computing Group for developing a set of cloud security services and protocols based on their Trusted Platform Module (TPM) (Bajikar, 2002).

ALPS builds on the privacy guidelines presented in (Pearson, 2009) and on related research in cloud security and privacy (Itani, Kayssi, Chehab, 2009), reputation (Itani et al., 2011) and service routing (Itani et al., 2012). Moreover, we build on the privacy concepts presented in DPaaS (Song et al., 2012) and strive to tackle many of the open questions and problems related to privacy support in cloud computing such as enhancing the auditing protocols and their accountability by elevating the customer confidence in them, reducing their performance cost, extending the privacy support to the broadest range of application classes, and standardizing the privacy services among cloud providers to facilitate customer migration processes. The consent and revocation concepts proposed at layer 4a are based on the ENCORE project (Mont, Sharma, & Pearson, 2012). We aim at enhancing the existing protocols and improving their scalability by decreasing their reliance on trusted third parties in the auditing and key management mechanisms. This will be mainly achieved using the secure execution containers proposed at layer-1 of ALPS.

3 SYSTEM MODEL

The system model assumed in this work consists of three main entities: a cloud customer that consumes the cloud storage and processing services, a cloud service provider that manages and operates the cloud facility, and a trusted third party (*TP*) that configures and distributes tamperproof cryptographic coprocessors for installation in the remote computing cloud. The *TP* is the organizational unit trusted by both, the cloud provider and customer. The trusted hardware building block, represented in the crypto coprocessor, should be installed on every physical server machine executing a privacy-aware virtual machine. A resource-sharing mechanism is implemented to facilitate the allocation of the crypto coprocessor resources among the set of cloud customers registered in the privacy services provided by the ALPS stack. This contributes to the economic feasibility of the system implementation. The secure isolation of the coprocessor's shared resources is cryptographically enforced by employing encryption mechanisms supported by customer-specific keying material. The *TP* is responsible of distributing a separate private/public key pair (PR_{ID}/PU_{ID}) to each cloud customer registered in the privacy services of the ALPS stack. The keying material distribution process is mainly carried out via a secure electronic transaction to share the private/public key pair between the cloud customer and the crypto coprocessor.

The *TP* has the authority to remotely add/delete any private/public key pair stored in the crypto coprocessor persistent storage. This aids in adding and revoking the registration of customers after the crypto coprocessors are installed in the computing cloud.

Moreover, the *TP* is responsible of installing its own private master key, K_M, into the persistent storage of the crypto coprocessor. K_M is required to support remote authentication and execution mechanisms by the *TP* on the crypto coprocessors. Due to the security sensitive responsibilities of the *TP*, serious measures should be employed to fortify

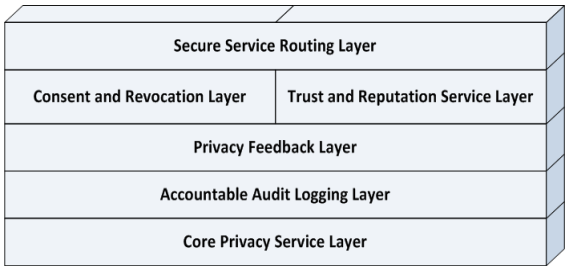


Figure 1: The 5 layers of the ALPS protocol stack.

the *TP* site against the different forms of system and network penetration attacks.

After the cloud client registers with the service it executes an authenticated version of the Diffie-Hellman key management protocol to exchange and share a secret Key K_s with the crypto coprocessor in the cloud.

4 SYSTEM DESIGN

The ALPS stack layers are presented in Figure 1. At layer-1 of the stack lie the core privacy mechanisms, which are designed using a policy-driven association of software and hardware cryptographic constructs to support the secure exchange, processing, and storage of customer data in the cloud. On top of the core privacy layer, three protocol layers, at layers 2, 3, and 4, implement accountable audit logging, a privacy feedback mechanism, and a consent and revocation mechanism, respectively. At layer-4 we also propose the formulation of a secure and accountable reputation-ranking scheme for objectively evaluating the operation of cloud service providers based on a range of ranking criteria including performance, pricing, and quality of service, in addition to security and privacy measures. Layer-5 in ALPS is dedicated to providing secure service routing in collaborative cloud architectures based on constraints that include privacy, quality of service, and pricing. The basic privacy support at the lowest layer in the protocol stack thus transpires through several layers of interdependent protocol building blocks to supplement the services of a large set of applications, some of which might not be evidently reliant on privacy mechanisms (such as in the reputation and service routing protocols).

The following is a preliminary layering for ALPS:

- *The Core Privacy Layer (Layer 1):* At the foundation of the ALPS stack lies the Core Privacy Layer. It consists of a set of security mechanisms for ensuring the privacy, integrity, and legal compliance of cloud data in their various lifecycle stages: when stored and processed in the cloud or when exchanged over Internet links. We propose to leverage state-of-the-art research in the fields of cloud privacy and trusted computing to establish secure execution containers in the computing cloud that are logically and/or physically protected against unauthorized access, including access by the cloud provider itself.

The main challenge when designing the Core Privacy Layer is to support it with economical and performance-efficient software design patterns to reduce the load on the relatively expensive secure execution containers and thus to result in a feasible protocol implementation. The cloud customer is responsible for configuring her software applications to support the security mechanisms enforced by the Core Privacy Layer. The configuration consists of categorizing the functional components of the application into secure and non-secure. The secure components (S_{SID}) should run in the address space of the tamperproof crypto coprocessor. On the other hand the non-secure classification of components (S'_{SID}) indicates that the functional components can be executed in a traditional process on the main server. A specialized daemon process coordinates the secure interaction between the protected and unprotected parts of the same software application. The secure software components are stored in an encrypted format in the cloud. The encryption operations are originally applied on the client side and the decryption takes place when the secure components are loaded into the address space of the tamperproof crypto coprocessor. The classification process is implemented using the concept of privacy tags, which are a set of metadata attributes, cryptographic constructs, and identification information attached to the respective software application. The contents of the privacy tag and the software application structure after applying the division process are sketched in Figure 2.

Before uploading the data to be stored and processed in the remote cloud, the cloud client classifies this data, based on significance and sensitivity, into three privacy classes:

1. Non-Secure (NS): The NS privacy class encompasses non-sensitive user data that can be stored in the cloud storage facility without applying any encryption mechanisms. Network security can be guaranteed by transferring the data to the cloud over a secure SSL session.
2. Compliance-Based Security (*CS*): The *CS* privacy class identifies cloud data that should be stored encrypted in the remote cloud by the cloud

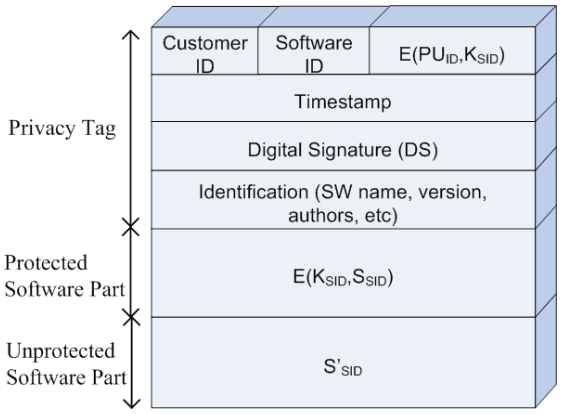


Figure 2: Software division in the Core Privacy Layer (E is the encryption function and K_{SID} is a randomly-generated key for encrypting the protected software part).

service provider. A provider-specific ciphering key supports the encryption mechanism. The *CS* privacy class facilitates the compliance of the ALPS privacy services with regulatory policies and recommendations. The network security of *CS*-marked data is achieved using a secure SSL session. The received network data is decrypted and then re-encrypted using the provider cryptographic key before being stored in the cloud storage facilities.

3. High Security (*HS*): The *HS* privacy class encompasses data with highly-sensitive information that should be encrypted by the data owner (the cloud customer) before being transferred to the cloud. The encryption is done using the key K_s shared with the crypto coprocessor as described in Section 3. Since *HS* data is encrypted using a customer-specific key before network transmission, the cloud provider would not be able to access it. The only entity that can operate on *HS* data in the cloud is represented in the trusted crypto coprocessor which shares with the customer the encryption key K_s.

- *The Accountable Audit Logging Layer (Layer 2)*: Accountable Audit Logging resides on top of the Core Privacy Layer and uses its privacy mechanisms to ensure the confidentiality and integrity of audit logs generated by cloud services and processes on the provider site. The protocol must support a large variety of applications seeking accountable interpretation of the different events taking place within the jurisdiction of the cloud provider.

An abbreviated algorithm for running this layer's logic consists of encrypting audit log records with the key K_s to protect their confidentiality and constructing a hash chain data structure over the

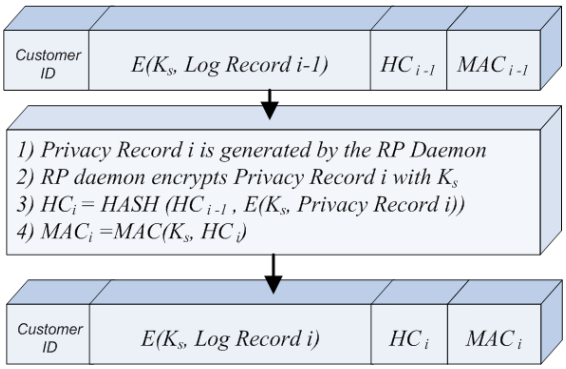


Figure 3: The algorithm of the Privacy Feedback Layer.

encrypted log records (Schneier & Kelsey, 1999). This is illustrated in Figure 3. In this figure, HC_i is the hash chain constructed by hashing the ith log record and the hash chain entry of the previous log record. Note that HC_i contains HC_{i-1}, thus it is feasible to validate the integrity of all previous log records by only verifying HC_i. Originally, HC_{-1} is given a default value to initially lead the hash chain. To authenticate HC_i, a *Message Authentication Code* field (MAC_i) is added to the log record. This field represents the *MAC* of HC_i using K_s.

The resulting secure log record is stored in the cloud storage facility.

- *The Privacy Feedback Layer (Layer 3)*: A privacy feedback mechanism is proposed at layer-3. The feedback uses the services of the lower layers to inform cloud clients of the different privacy operations applied on their data and to make them alert of any suspicious activities that may expose the confidentiality or integrity of their sensitive information. The Privacy Feedback Layer aids in: (1) escalating the customer's confidence in cloud processing and (2) supporting the auditing operations of compliance bodies and certification authorities. All this leads to better adoption of cloud computing services in security-sensitive applications in the financial, governmental, and healthcare fields. The algorithm executed in this layer is described as follows: whenever a privacy-related operation is executed in the address space of the crypto coprocessor, a privacy audit record describing this operation is generated. The type of the privacy operation and the contents of the privacy record are specified by the customer upon registration with the privacy service. Possible fields that may be included in the privacy audit record are: a description of the privacy operation, the application process performing the operation, the cryptographic key strength used in performing the encryption mechanisms, the process providing input data to the

operation, the process receiving the operation output data, the privacy category of the input and output data, the number of consecutive failed signature verifications, the date and time of the operation, etc.
The privacy audit record is then secured and stored based on the services of Layer 2. The protected audit log store is polled by a dedicated customer daemon every preconfigured time slot. The integrity of the privacy audit log is corroborated using the hash chain and its corresponding *MAC* value. Afterwards the daemon decrypts the contents of the privacy records using the key K_s and the customer can analyse the decrypted privacy records and take actions accordingly.

- *The Consent and Revocation Layer (Layer 4a)*: The first protocol at layer-4 is responsible for granting the cloud user more control in managing the various aspects related to the privacy of sensitive data by implementing user-configurable software protection and data privacy categorization mechanisms. The user can specify, via well-defined privacy policies, the conditions and restrictions that govern (1) the release of sensitive data to cloud sites, (2) the processing and sharing of this data by various cloud providers, and (3) the revocation of any previously granted consents on the usage and dissemination of a particular piece of information. The design of the consent and revocation block may use, in addition to the services provided by layers 1, 2, and 3, several security mechanisms such as identity-based encryption and sticky policies to cryptographically bind the customer personal data to the respective privacy enforcement policies. The algorithms at this layer are currently in the final development stages and will be readily available at the final version of this manuscript.

- *The Trust and Reputation Layer (Layer 4b)*: The second protocol at layer-4 relies on all the privacy services provided by the lower layers to provide a secure cloud reputation system for ranking service providers. The results and recommendations obtained can be published as a service and verified by trusted third parties. The reputation service may be based on a range of ranking criteria including performance, pricing, and quality of service, in addition to security and privacy measures. The main objective behind this layer is to provide a valuable IT component in supporting verifiable and accountable compliance with service-level agreements and regulatory policies, and providing new and existing cloud customers with valuable advice for selecting the appropriate cloud service provider(s) that suit their performance, budgeting, and security requirements. Rather than relying on subjective feedback from cloud users, the proposed protocol carries out the reputation calculations based on observable actions extracted securely and independently. The algorithm for calculating the performance reputation of a particular cloud provider by a trusted third-party entity can be described as follows:
The securely generated logs are collected from the different provider sites and analyzed in order to calculate the provider performance reputation scores. The calculation is done as follows: for each log entry, the *TP* utilizes the τ (The data size in Bytes retrieved/processed in the transaction) and β(The time required to retrieve/process the data in the transaction in milliseconds) fields to calculate the transaction data retrieval/processing rates and compares them to the rates promised in the SLA based on the following equations:

$$\Phi_{\mathrm{r,i}}[\%] = \frac{\frac{\beta_\mathrm{i}}{\Psi_\mathrm{r}} - \tau_\mathrm{i}}{\frac{\beta_\mathrm{i}}{\Psi_\mathrm{r}}} \times 100 \quad (1)$$

$$\Phi_{\mathrm{p,i}}[\%] = \frac{\frac{\beta_\mathrm{i}}{\Psi_\mathrm{p}} - \tau_\mathrm{i}}{\frac{\beta_\mathrm{i}}{\Psi_\mathrm{p}}} \times 100 \quad (2)$$

$\Phi_{\mathrm{r},i}$ and $\Phi_{\mathrm{p},i}$ respectively represent the percent improvement over the SLA retrieval rate Ψ_r and processing rate Ψ_p for the ith log record.
By applying averaging and normalization operations on the calculated Φ_r and Φ_p log entry improvement rates, the overall retrieval and processing reputation scores R_r and R_p can be computed. R_r and R_p are functions of the average retrieval/processing rates $\overline{\mathrm{X}}_\mathrm{r}$ and $\overline{\mathrm{X}}_\mathrm{p}$ and their standard deviations σ_r and σ_p, respectively.

$$\overline{\mathrm{X}}_\mathrm{r}[\%] = \frac{1}{N_r} \times \sum_{i=1}^{i=N_r} \Phi_{\mathrm{r},i} \quad (3)$$

$$\overline{\mathrm{X}}_\mathrm{p}[\%] = \frac{1}{N_p} \times \sum_{i=1}^{i=N_p} \Phi_{\mathrm{p},i} \quad (4)$$

$$\sigma_\mathrm{r}[\%] = \sqrt{\frac{1}{N_r} \times \sum_{i=1}^{i=N_r} \left(\Phi_{\mathrm{r},i} - \overline{\mathrm{X}}_\mathrm{r}\right)^2} \quad (5)$$

$$\sigma_\mathrm{p}[\%] = \sqrt{\frac{1}{N_p} \times \sum_{i=1}^{i=N_p} \left(\Phi_{\mathrm{p},i} - \overline{\mathrm{X}}_\mathrm{p}\right)^2} \quad (6)$$

$$R_\mathrm{r} = \left[\lambda + \left(\lambda \times \frac{\overline{\mathrm{X}}_\mathrm{r}[\%]}{100}\right)\right] \pm \left(\lambda \times \frac{\sigma_\mathrm{r}[\%]}{100}\right) \quad (7)$$

$$R_\mathrm{p} = \left[\lambda + \left(\lambda \times \frac{\overline{\mathrm{X}}_\mathrm{p}[\%]}{100}\right)\right] \pm \left(\lambda \times \frac{\sigma_\mathrm{p}[\%]}{100}\right) \quad (8)$$

Where N_r and N_p are the number of retrieval and

processing log entries per provider, respectively, and λ is a normalization constant that represents the middle value of the reputation score range.
The range of the retrieval and processing reputation scores components lies in $]0, 10[$.

- *The Service Routing Layer (Layer 5)*: Service Routing utilizes the reputation and privacy services provided by ALPS to support autonomic service routing in collaborative cloud architectures. The chief contribution of this protocol is in autonomously selecting the optimal set of composite service components among a set of cloud providers for executing a particular user service request. The routing decision engine constructs the service paths based on accountable privacy, reputation, quality of service, and pricing constraints specified in well-defined routing policies. For a comprehensive description of the routing algorithms implemented in this layer the reader is referred to (Itani et al., 2012).

5 ALPS IMPLEMENTATION

A prototype implementation of the ALPS protocol stack is tested in a virtual lab environment. The prototype included a full implementation of the algorithms and cryptographic constructs of the 5 stack layers. The ALPS prototype implementation employs the same virtualization technologies used in real-world cloud infrastructures. On the hardware layer we used an Intel Server running Windows Server 2008. The server comprises two core processors running at a frequency of 3.0 GHz and the system is supported with 8 GB of RAM. The virtualization layer is provided using VMware VSphere 4. The virtual machines operating system (guest) runs Windows Server 2003. To simulate the functionality of a secure cryptographic coprocessor, we assume that one of the server processors is the

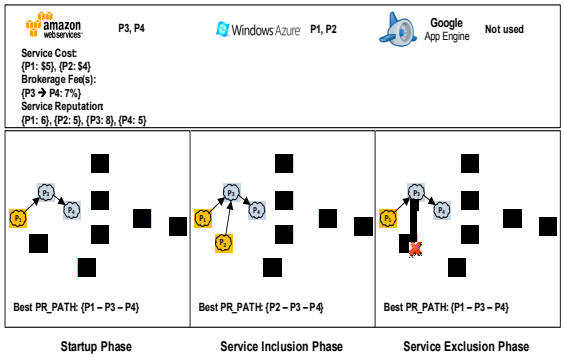


Figure 4: Layer 5 Cloud Configuration 1 Implementation Scenario.

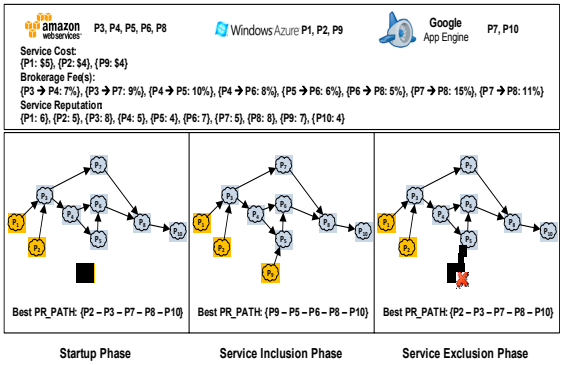


Figure 5: Layer 5 Cloud Configuration 2 Implementation Scenario.

tamper-proof coprocessor, while the other core is that of the main untrusted server system. This configuration is believed to provide an acceptable proof of concept as far as system testing is concerned.

The ALPS blueprint implementation is programmed using the C# programming language which is part of the Microsoft .Net Common Language Runtime (CLR) framework.

The ALPS protocol stack is then implemented from the ground up in a real cloud computing environment using top-notch cloud providers represented in the Amazon EC2, Microsoft Azure, and Google App Engine platforms. We experimentally verified the correctness of the stack algorithms and its scalability using two cloud configurations (see Figures 4 and 5) that increase in complexity from four to ten collaborating cloud providers. To attain a lower bound on the performance of the ALPS protocols in the real cloud platform, we utilized the basic virtual machine specifications on the Amazon EC2 IaaS cloud

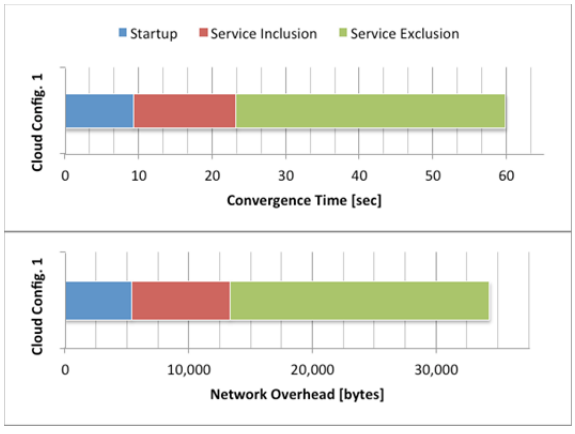


Figure 6: Layer 5 Cloud Configuration 1 – Convergence and Traffic Overhead Results.

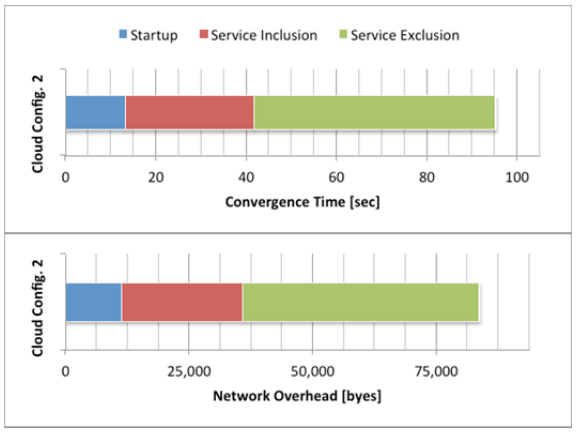


Figure 7: Layer 5 Cloud Configuration 2 – Convergence and Traffic Overhead Results.

infrastructure as provided by the Micro Amazon Machine Image (AMI) profile. The AMIs run a 64-bit version of Ubuntu Linux and the protocol code is developed using the C programming language. On the Microsoft Azure and Google App Engine PaaS platforms, we were satisfied with the default application configuration offered by the provider. No modification, whatsoever, has been carried out on the dynamic Web engines or on the automatic resource scaling and load balancing features provided by Microsoft and Google. The protocol implementation is developed using the C# and Java programming languages on the Microsoft and Google clouds, respectively. Due to the space limitation of this presentation we will discuss the implementation of the upper layer (*The Service Routing Layer*). This choice is due to the fact that this layer abbreviates and utilizes all the layers below it, and thus, provides an adequate performance figure of the entire service stack implementation and operation.

For each cloud configuration (see Figures 4 and 5), we execute a 3-phase scenario for analyzing the two main qualities that characterize the operation of an effective routing protocol:

1. The convergence time of the protocol under different network conditions. The convergence time is the time needed by the routing protocol to populate the routing tables of all the service routers in the cloud configuration.
2. The traffic overhead imposed on the network by the routing protocol. The traffic overhead is measured in Bytes and is determined per convergence time period.

The three phases of each cloud configuration implementation scenario are presented as follows:

1. The Startup Phase: The Startup phase represents the initial state of the cloud configuration before inducing any service adjustment.
2. The Service Inclusion Phase: In the Service Inclusion phase we modify the cloud configuration by adding a new cloud provider that offers the same type of service but with a different cost or reputation score.
3. The Service Exclusion Phase: In the Service Exclusion phase we remove the cloud provider added in the Service Inclusion phase.

Basing the implementation experiments on the above scenario phases is adopted since these phases represent the basic events (service inclusion and exclusion) that could arise in a real cloud environment. The successful implementation of these phases increases the confidence level in the routing protocol to handle an extrapolated and, definitely, more complex set of anomalous events that may be experienced by the cloud configuration.

Protocol Convergence and Traffic Overhead: The protocol convergence time is the time needed by the service routing protocol to populate the routing tables of all the service routers in the cloud configuration. The traffic overhead represents the number of network bytes resulting from exchanged protocol messages in a convergence time interval. The protocol convergence time and traffic overhead are experimentally calculated for all the implementation scenarios in each of the cloud configurations.

The results are presented in Figures 6 and 7. The first observation from the results is that the protocol convergence time and traffic overhead increase as the complexity of the cloud configuration increases with additional cloud providers. This is fully anticipated since as the number of cloud providers in the configuration increases, the routing protocol requires more time and more exchanged network messages to calculate the routing tables and hence to converge. This same fact justifies why the protocol convergence time and the network traffic overhead are directly proportional.

Finally we would like to stress on the fact that, to the best of our knowledge, we could not find any research work dealing with the autonomic service routing problem in cloud computing to compare the obtained experimental results with. This misfortunate fact, together with the timely and practical nature of the problem we are dealing with, makes part of the message we are trying to deliver in this chapter represent a call for action for further research in the field of service management and routing in the cloud. On the other hand, and despite the fact that the protocol we present in this chapter is

inspired by the standard BGPv4 specification, we strongly believe that comparing the implementation results of the experiments in the presented cloud computing environment with existing BGP-based routing solutions in IPv4 networks is infeasible and does not rely on a uniform evaluation standard . This is due to the fact that such IPv4 solutions are designed and developed under different system models and problem definitions which renders the comparison unfair and possibly provoking superfluous distraction to the reader.

6 CONCLUSIONS

In this paper we presented ALPS, an accountable cloud protocol stack for supporting practical solutions to the various security and service manageability challenges facing cloud computing architectures and implementations today. The protocol stack relies on core privacy services and security constructs to support application domains such as secure data storage and processing, accountable audit logging, privacy feedback, consent and revocation, trust and reputation, and autonomic service routing. Further application domains in the fields of energy efficiency and monitoring are currently investigated to extend the application support of the ALPS stack.

ACKNOWLEDGEMENTS

The authors would like to thank and acknowledge the support of the Lebanese National Council for Scientific Research (CNRS).

REFERENCES

Atallah, M. J., & Frikken, K. B. (2010, April). Securely outsourcing linear algebra computations. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security* (pp. 48-59). ACM.

Bajikar, S. (2002). Trusted platform module (tpm) based security on notebook pcs-white paper. *Mobile Platforms Group, Intel Corporation (June 20, 2002),*

Best, R. M. (1980). Preventing software piracy with crypto-microprocessors. *Proceedings of IEEE Spring COMPCON, 80* 466–469.

Cavoukian, A. (2008). Privacy in the clouds. *Identity in the Information Society, 1*(1), 89-108.

Dyer, J. G., Lindemann, M., Perez, R., Sailer, R., Van Doorn, L., Smith, S. W., et al. (2001). Building the IBM 4758 secure coprocessor. *Computer,* 57-66.

Gellman, R. (2009). WPF REPORT: Privacy in the clouds: Risks to privacy and confidentiality from cloud computing. *Released Feb. 23, 2009*.

Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. Proceedings of the 41st Annual ACM Symposium on Theory of Computing, 169-178.

Gentry, C. (2011). Fully Homomorphic Encryption without Bootstrapping.

Itani, W., Kayssi, A., & Chehab, A. (2009). Privacy as a service: Privacy-aware data storage and processing in cloud computing architectures. *2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing,* 711-716.

Itani, W., Ghali, C., Kayssi, A., & Chehab, A. (2011).“Accountable Reputation Ranking Schemes for Service Providers in Cloud Computing”.In Proc. Int. Conference on Cloud Computing and Services Science (CLOSER 2011).

Itani, W., Ghali, C., Bassi, R., Kayssi, A., & Chehab, A. (2012), “BGP-Inspired Autonomic Service Routing for the Cloud”.In Proc. ACM 27th Symposium On Applied Computing (ACM SAC 2012).

Mont, M. C., Sharma, V.& Pearson, S. (2012). *EnCoRe: Dynamic Consent, Policy Enforcement and Accountable Information Sharing within and across Organizations*, HP Laboratories technical Report #: HPL-2012-36, 2012.

Pearson, S. (2009). Taking account of privacy when designing cloud computing services. *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing,* 44-52.

Pearson, S., &Charles worth, A. (2009).Accountability as a way forward for privacy protection in the cloud. *Cloud Computing*, 131-144.

Schneier, B. (2009). http://www.schneier.com/blog/archives/2009/07/homomorphic_enc.html.

Schneier, B., & Kelsey, J. (1999). Secure audit logs to support computer forensics. ACM Transactions on Information and System Security (TISSEC), 2(2), 159-176.

Song, D., Shi, E., Fischer,I, & Shankar, U. (2012).“Cloud Data Protection for the Masses,”. *IEEEComputer*, 45(1):39-45.

The New York Times (2011), “The Cloud Darkens”, http://www.nytimes.com/2011/06/30/opinion/30thu1.html.

Tygar, J., & Yee, B. (1993). Dyad: A system for using physically secure coprocessors. *Proceedings of the Joint Harvard-MIT Workshop on Technological Strategies for the Protection of Intellectual Property in the Network Multimedia Environment, 87*.

Van Dijk, M., Gentry, C., Halevi, S., &Vaikuntanathan, V. (2010). Fully homomorphic encryption over the integers. *Advances in Cryptology–EUROCRYPT 2010*, 24-43.

A User Data Location Control Model for Cloud Services

Kaniz Fatema[1], Philip D. Healy[1], Vincent C. Emeakaroha[1], John P. Morrison[1] and Theo Lynn[2]
[1]*Irish Centre for Cloud Computing & Commerce, University College Cork, Cork, Ireland*
[2]*Irish Centre for Cloud Computing & Commerce, Dublin City University, Dublin, Ireland*
{*k.fatema,p.healy, vc.emeakaroha, j.morrison*}*@cs.ucc.ie, theo.lynn@dcu.ie*

Keywords: Authorization System, Access Control, Data Location, XACML, Cloud Computing.

Abstract: A data location control model for Cloud services is presented that uses an authorization system as its core control element. The model is intended for use by enterprises that collect personal data from end users that can potentially be stored and processed at multiple geographic locations. By adhering to the model's authorization decisions, the enterprise can address end users' concerns about the location of their data by incorporating their preferences about the location of their personal data into an authorization policy. The model also ensures that the end users have visibility into the location of their data and are informed when the location of their data changes. A prototype of the model has been implemented that provides the data owner with an interface that allows their location preferences to be expressed. These preferences are stored internally as XACML policy documents. Thereafter, movements or remote duplications of the data must be authorized by submitting requests to an ISO/IEC 10181-3:1996 compliant policy enforcement point. End users can, at any time, view up-to-date information on the locations where their data is stored via a web interface. Furthermore, XACML obligations are used to ensure that end users are informed whenever the location of their data changes.

1 INTRODUCTION

Cloud Computing offers a new style of computing that allows consumers to pay only for the services used and frees them from the management overhead of the underlying infrastructure. Although Cloud Computing has gained significant traction in recent years, surveys have consistently shown that consumers' concerns around security and loss of control over data are hindering adoption (Subashini and Kavitha, 2011; Chen and Zhao, 2012). Additionally, the physical location of data can have an impact on its vulnerability to disclosure and can have implications for service quality and legal consequences (Albeshri et al., 2012; Gondree and Peterson, 2013). Many regulations such as HIPAA and the EU Data Protection Directive impose restrictions on the movement of data between geographical locations. Data location therefore represents a sensitive issue for enterprises that offer cloud services. Existing and potential customers seek assurance that the enterprise will act as faithful stewards of information entrusted to them, and the enterprise itself wishes to avoid falling foul of data protection rules and other regulations.

However, these concerns must be balanced against the enterprise's desire to maintain redundancy and operational efficiency. There are a number of valid reasons for copying data to geographically dispersed locations. These may include:

- *Risk Mitigation* Copying data to more than one physical location provides a hedge against localised catastrophic events such as fires and natural disasters.
- *Operational Expenditure* Due to the geographically variable nature of overheads, such as energy costs, it may be cheaper to store and/or process data at a location other than where it was stored originally.
- *Storage Capacity* If a data centre has limited capacity it might be helpful to offload some storage to another location.
- *Maintenance* Data may sometimes need to be moved temporarily in order to facilitate data centre maintenance, upgrade or relocation.
- *Localised Caching* Content delivery networks, such as Akamai and Amazon CloudFront, replicate data to edge servers in order to improve users' quality of experience.

Enterprises that offer Cloud services must therefore balance the benefits of migrating data against concerns relating to trustworthiness in the eyes of users and regulatory compliance.

Users' concerns about how their data is disseminated and used can be addressed by allowing them to specify policies about how they wish their data to be used and assuring them of the enforcement of this policy. In the context of data location, this assurance can be provided in two ways (a) with a transparency mechanism that allows users to view the locations where their data is being stored and (b) with a notification mechanism that informs them when there is a change to the *status quo*. If the users' policies are to carry any weight then they must be consulted before operations are performed that would result in movement or remote duplication so that violations can be avoided.

A means of addressing this challenge is to implement an authorization system that inputs queries relating to data relocation and returns results that indicate the permissibility of the proposed actions. Of course, such a system is of little value unless it is appropriately integrated into the service provider's internal processes and mechanisms. As such, there is a residual trustworthiness issue in that the users must trust that this integration has taken place and that the decisions of the authorization system are being honoured. These issues could be addressed through independent verification mechanisms such as trustmarks, audits and assurance services (Lynn et al., 2013).

Although the basic concept of an authorization system (di Vimercati et al., 2005; ISO, 1996) is to protect access to secured resources, the functionality of existing authorization systems can be extended to perform other operations such as enforcing obligations (Park and Sandhu, 2004; Demchenko et al., 2008) and verifying credentials (Chadwick et al., 2008). In this paper, we propose a Cloud data location assurance control model that is based on authorization systems. The ISO/IEC 10181-3:1996 standard access control framework is used as the technological foundation. The following issues are addressed:

1. Capturing users' location preferences as policies that can be automatically consulted
2. Providing an authorization mechanism for data movement decisions that takes user-specific policies into account
3. Providing users with visibility into the location of their data
4. Ensuring that users are informed when the location of their data changes

The remainder of this paper is organized as follows: Section 2 presents an analysis of related work. Section 3 provides background information on authorization systems, XACML policies and their evaluation process and how they can play a role in Cloud data management. Section 4 presents the design of the proposed location control model while Section 5 provides some technical details along with the result of validation tests. Finally, Section 6 concludes the paper and suggests future research directions.

2 RELATED WORK

Geolocation is a technique that can be used *post facto* to determine where data is being served from, and hence whether the data is being hosted from an undesirable location. Most Geolocation techniques make use of network latency as a proxy for distance (Katz-Bassett et al., 2006). The techniques include: *geo ping*, in which a server is pinged from a number of landmarks to identify how close the server is to each landmark and hence determine the location; *constraint-based geolocation*, which uses a triangulation technique to determine the position of a server that is possibly between landmarks; and *topology-based geolocation*, which considers trace route, topology and per hop latency. These techniques are used to estimate the geolocation of a Cloud data centre based on network latency and comparing it using heuristics to expected latencies from known distances (Benson et al., 2011; Ries et al., 2011; Peterson et al., 2011). Ries *et al.* proposed a geolocation approach based on network coordinate systems to identify location policy violations (Ries et al., 2011). They found the accuracy of the geolocation-based approaches to be only 52% at best. Albeshri *et al.* proposed an architecture for data geolocation based on a Proof of Storage (POS) protocol and timing-based distance bounding protocol (Albeshri et al., 2012). The POS protocol is used to provide proof to the client that the server is holding the data and that it has not been modified. The delay in response is used to determine the probable location of the data. A similar approach has been proposed (Peterson et al., 2011; Gondree and Peterson, 2013) that uses a challenge-response protocol based provable data possession technique combined with a geolocation-based solution. However, determining the location of data based on response time has been shown to be inaccurate (Ries et al., 2011). Given the inaccuracy and *post facto* nature of geolocation techniques, we decided against using this approach for our purposes. However, geolocation could have a role as an independent audit mechanism for confirming that users' location preferences are being honoured.

The Data Location Assurance Service proposed by Noman *et al.* is based on an approach where the Cloud service provider sends regular updates to en-

terprises about the location of their data (Noman and Adams, 2012). The enterprise in turn provides location assurance to users by providing information on whether the data are in their preferred location in a simple yes/no format. This approach relies heavily on the trustworthiness of the Cloud service provider. Massonet *et al.* describe how negotiation can be used to integrate service providers' policies with those of infrastructure providers in a federated Cloud environment (Massonet et al., 2011). The infrastucture provider's security policy states the availability of logging facilies, logging type and monitoring, while the servuce provider's policy states its security requirements. A virtual machine is migrated to a site only if its security policy matches with the security requirements of the service provider. An audit log is created from the collaboration of between service provider and infrastructure provider. The location of data is identified through the presence of data centre's certificate in the audit log. Neither of these systems allow for detailed location preferences, such as multiple preferred geographic regions, to be collected from end users and enforced automatically.

Numerous implementations of policy-based authorization have been proposed in order to protect access to sensitive data in the Cloud (Iskander et al., 2011; Basescu et al., 2011; Chadwick and Fatema, 2012). A distributed authorization system architecture for Cloud services was proposed by Almutairi *et al.* where each Cloud layer has a virtual resource manager that protects access to the virtualized resources of its layer using an access control module (Almutairi et al., 2012). Ries *et al.* proposed a policy-based approach to location issues (Ries et al., 2011). However, they did not elaborate on how to incorporate location into policy or how to enforce policies.

A number of policy languages – such as P3P (Cranor, 2003), EPAL (Ashley et al., 2003), PERMIS (Chadwick et al., 2008), FlexDDPL (Spillner and Schill, 2012) and XACML (Godik et al., 2002) – have become available for various purposes. P3P (Cranor, 2003) defines a machine interpretable format for websites to express their privacy practices. The notice and consent model of P3P allows websites to describe their privacy policies and users can read these policies and can choose to interact with the websites and thus provide their consent to the policies. P3P provides: human readable versions of the policies; automated comparison of users preferences with policies; automated reports in the event of a mismatch; and incorporation of users preferences when viewing and changing policies. A drawback of P3P is that it simply checks whether the users preferences match with the organisations stated privacy policies; it does not ensure that the organisations actually enforce their stated privacy policies. FlexDDPL defines a policy language for data distribution which specifies who under what conditions is allowed to store what data, how and where. FlexDDPL policies can be expressed independently from storage controller and gateway implementations and be mapped onto those as part of administration process. However, how these two independent policies can be combined and how the conflicts between those can be resolved is not defined. Moreover their current implementation does not include location.

On the other hand, XACML (an OASIS standard) provides both a policy language and an access control decision request/response language and also specifies a policy evaluation engine. Chadwick *et al.* have modified the XACML model to allow multiple policy languages and multiple policies of independent authorities to be supported by the model, making it suitable for use in Cloud scenarios (Chadwick and Fatema, 2012). The current implementation of it supports both XACML and PERMIS policy languages and is available under an open source license. However it still uses the XACML request and response language and keeps the policy decision points (see below) for each policy language unchanged. We chose to adopt XACML for our location control model due to its standardised format and built-in enforcement mechanism. It is also the most widely used language for defining security policy (Turkmen and Crispo, 2008).

3 BACKGROUND INFORMATION

In this section, we present some background information on authorization systems and how they can be integrated with Cloud deployments. Furthermore, we describe a use case model that highlights the usability of the proposed Cloud data location control mechanism. We also specify the scope and constraints of our solution.

3.1 Authorization Systems

Access control/authorization is a process of determining whether a request to access (*e.g.* read, write, delete, copy, etc.) a resource object (*e.g.* file, database, program, system component) by a subject (*e.g.* user, system or process) should be granted or denied. The access control/authorization system can grant or deny the request based on whether or not certain policy constrains have been satisfied. ISO/IEC

10181- 3:1996 (ISO, 1996) defines a generic Access Control Framework (ACF) as shown in Figure 1. It consists of four components: (i) initiators (subjects), (ii) targets (resource objects), (iii) Access control Enforcement Functions (AEFs) – commonly known as Policy Enforcement Points (PEPs) – and (iv) Access control Decision Functions (ADFs) – commonly known as Policy Decision Points (PDPs). The initiators submit access requests (also known as user requests) that specify the operation to be performed on the target. The AEF transforms the request into one or more decision requests (also known as authorization queries) and sends these to the ADF. The ADF decides whether a decision request should be granted or denied based on the provided policies and sends the decision back to the AEF.

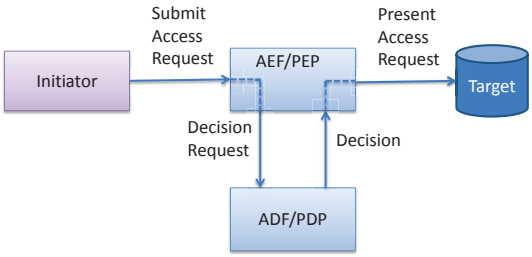


Figure 1: ISO/IEC 10181-3 access control framework (ACF).

An access decision can contain obligations, such as sending e-mail to the data subject or recording the permitted access in a log. These obligations are enforced by the PEP while executing the access decision. Authorization systems may play an important role in Cloud data management since Cloud service delivery can involve significant amounts of data transmission and storage. SaaS providers offering services, such as web applications, may accumulate a large amount of personal data like names, addresses, medical records, purchase history and so on during the execution of the service. The management, processing and storage of such data while considering customers' data location choices is challenging. In this paper, we aim to develop a mechanism for SaaS providers, who might be using services from IaaS and PaaS layers, to incorporate the location choices of their customers while managing and storing data in Clouds.

3.2 XACML Policy Structure and Evaluation

This section provides a brief description of XACML policy structure and its evaluation strategy.

3.2.1 XACML Policy Structure

XACML policies are constructed with the following components: policy set (*PS*), policy (*P*), rule (*R*), target (*t*), policy combining algorithm (PCA) and rule combining algorithm (RCA). A target t is composed of four types of attributes: subjects (S), resources (Rs), actions (A) and environments (En). A rule is defined as a tuple, $r = (id, t, e, c)$, where id is a rule ID, t is a rule target, $e \in \{\text{Permit}, \text{Deny}\}$ is an effect and c is a condition which is an optional element and is evaluated to a Boolean value. A policy is defined as a tuple, $p = (id, t, R, RCA, O)$, where id is a policy ID, t is a policy target, $R = \{r_1, r_2, \ldots, r_n\}$ is the set of rules and RCA is the rule combining algorithm and O is an optional set of obligations.

A policy set is defined as tuple $PS = (id, t, P, PAC, O)$, where id is the policy set ID, t is the policy set target, $P = \{p_1, p_2, \ldots, p_n\}$ is the set of policies, PCA is the policy combining algorithm and O is an optional set of obligations. An access request in XACML v2 is defined as a set of attributes, $R_q = (s, rs, a, en)$, where s is a set of subject attributes, rs is a set of resource attributes, a is a set of action attributes and en is a set of environment or context attributes. Each attribute set describes the corresponding object. The request means that the subject described by s is asking to be perform the action described by a on the resource described by rs in the presence of the context described by en.

3.2.2 Evaluation of a Request

An authorisation decision is performed by the XACML PDP by matching the values of the request with the values in the policies. A *target* element of XACML is evaluated as Match if all the elements specified in the target match with the values presented in the request context. If any one of the elements specified in the *target* is *indeterminate*, then the *target* is evaluated as *indeterminate*. Otherwise, the *target* is evaluated as *no-match*. The *condition* element represents a Boolean expression. When the *target* element of an XACML policy evaluates to Match and the *condition* element evaluates to True, the *rule* evaluates to Effect and when the *target* element of an XACML policy evaluates to Match and the *condition* element evaluates to False the rule evaluates to NotApplicable (Anderson, 2005). The standard policy combining algorithms of XACML v2 (Anderson, 2005) and v3 (Rissanen, 2013) are defined as:

1. Deny-overrides (Ordered and Unordered in v3),
2. Permit-overrides (Ordered and Unordered in v3),
3. First-applicable and

4. Only-one-applicable.

In the case of the deny-overrides algorithm, if a single policy element evaluates to “Deny”, then, regardless of the evaluation result of the other policy elements, the combined result is “Deny”. For ordered deny-overrides the behaviour of the algorithm is the same except that the order in which the collection of policies is evaluated will match the order as listed in the policy set. Similarly, in the case of the permit-overrides algorithm, if a single “Permit” result is encountered, regardless of the evaluation result of the other policy elements, the combined result is “Permit” and for the ordered permit-overrides the behaviour of the algorithm is the same except that the order in which the collection of policies is evaluated will match the order as listed in the policy set. In the case of the “First-applicable” combining algorithm, the first decision encountered by the policy element in the list becomes the final decision accompanied by its obligations (if any). The only-one-applicable policy-combining algorithm ensures that only one policy is applicable by virtue of its target. The result of the combining algorithm is the result of evaluating the single applicable policy. If more than one policy is applicable, then the result is “Indeterminate”. In the XACML v3 specification some other combining algorithms are also defined, such as deny-unless-permit (which returns “Deny” only if no “Permit” result is encountered) and permit-unless-deny (which returns “Permit only” if no “Deny” result is encountered).

3.3 Use Case

To contextualize our approach in this paper, we present a use case that exemplifies the scope of our approach. Figure 2 shows an overview of our system model and how it integrates with Cloud deployments. The model considers interactions among three different groups of stakeholders: (i) Cloud service providers, (ii) enterprises and (iii) end users.

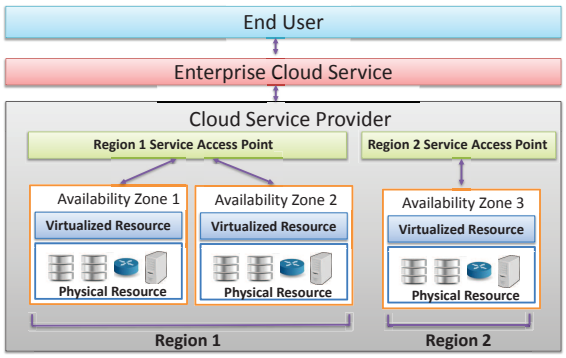


Figure 2: Location aware Cloud usage model.

Cloud Service Providers – For the purposes of this use case, we assume that the Cloud service providers are either IaaS or PaaS providers who offer virtualized resources/environment to the enterprises as services. The services are offered in separate regions, as shown in Figure 2. They deploy resource instances and data according to the instructions of their customers and do not move data without being instructed by the customers. For example, Amazon EC2 provisioning currently works in this manner. Each region is identified by a unique name and is accessed via specified access points that act as a gateways to the service. Each region may consist of a number of availability zones, as shown in Figure 2, where the instances and data can be copied to provide better availability in the event of a failure. Customers can choose one or more availability zones where their data and service instances can be replicated.

Enterprises are the customers of IaaS and PaaS services. They compose their SaaS services over the IaaS or PaaS services to offer them to the end users. These services may be designed to collect and process the end users’ personal data. Depending on their organizational needs, an enterprise can choose to put their services in more than one regions and a number of availability zones in each region. Enterprises manage the customers’ data and are responsible for instructing the Cloud providers correctly about which data is to be stored in which region.

End users/data owners – End users are the persons who are consuming the service. Enterprises may collect personal data from end users which are later stored and processed in the Cloud environment. The end users who are sharing their personal data with the enterprise are referred to as *data owners*.

This model assumes data to be of any format which should have two properties: i) it should be identified by one uniquely identifiable name, *DataID*, and ii) it is expected to be treated as one unit while actions (*e.g.* copy, read, write) are performed on it. The idea of having a unique identifier within a Cloud is not an impractical assumption as each Cloud user has a unique account which can be used or combined to form the unique identifier. In federated scenarios involving multiple participants a number of Cloud and identifier linking mechanisms can be used to create a globally unique identifier.

The granularity of data that is protected by the policy depends on the enterprise’s application scenario. It can be applied to a file or a set of files grouped under one folder, or a database entry. The unique identifier of the data, DataID, and the identifier of the policy, PolicyID, that is applicable for protecting the data is linked together by the model. This linking will allow

the policy that is relevant for that data to be evaluated while a request for performing an action (*e.g.* copy, read, write) on a data is received. As policy is linked to a DataID regardless of the format of data the model will protect the access of any type of data based on the relevant policy.

We use a loyalty card program, as offered by many retail chains, to exemplify this use case. During the sign up process, the customer gives some personal information such as her name, address and contact information (e-mail/telephone number). The customer uses the card while purchasing products from the retailer and may be offered discounts based on the purchase history. Each loyalty card can be represented electronically as an XML infoset containing a unique Loyalty Card number (which can be used as DataID)and other elements, such as a personal information element containing name, address, age; and a purchase history element, which can grow over time based on the number of purchases. To improve availability and quality of experience for their customers, the retailers use multiple Cloud services with different location and regions. The retailer accesses the services and data in these regions via the access points offered by the Cloud Service provider.

The data owners are given the opportunity by the retailer to choose one of the locations as a primary location for storing their data initially. They can also choose other locations where their data can be moved if necessary. They can also activate an option to receive e-mail when the location changes. The retailer ensures that the Cloud providers deploy the data to the correct region according to the preference of the data owner. In case of data movement, the data owner receives a notification if required by her policy.

4 DATA LOCATION CONTROL

The core of our approach is an enterprise Cloud service manager (ECSM) that manages access to all Cloud data and services using an authorization system. The authorization system evaluates two types of policies for making an access decision upon an access request. The first is the organizational administrative policy that specifies the enterprise roles and their attributed actions on services or data (*e.g.*, a marketing officer can execute a service, read the data and so on) or the roles of other external services to perform actions on their protected data. This administrative policy is executed for all access requests. The second type of policy consulted is the data owner's specific policy. All the data owners' policies are stored in the data owners' policy store.

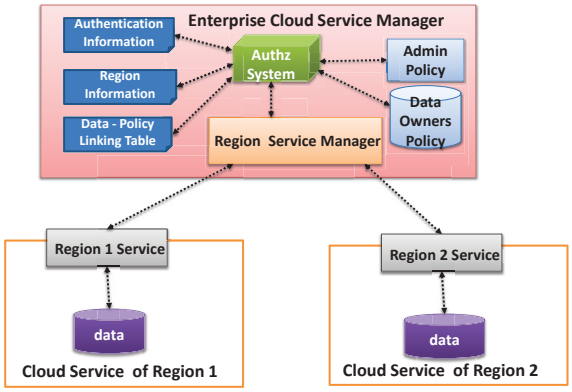


Figure 3: The enterprise Cloud service manager (ECSM) manages services and data in different Cloud regions.

The enterprise service manager also includes a user authentication information table for managing user authentications, a region information table that details the regions where each data set identified by a DataID is stored and a data-policy linking table that correlates each PolicyID to a DataID in order to retrieve the correct policy of the data owner for a specific data set. Figure 3 depicts how the region service manager receives instructions from the authorization system on where to store or retrieve data.

4.1 Intra-service Interaction Strategy

This section shows the interaction strategy to manage data locations within the same enterprise. The discussion is partitioned into four catagories: inputting data, accessing data, updating data location and checking location.

Data Input – To use the enterprise Cloud service, the end users first need to register for it. During the registration process, the user specifies authentication information (typically a user name and password or credentials) that is stored by the service provider to identify the user. The authentication information provided by each user is associated with a *DataID* that identifies the data of this user. In our example use case scenario it would be the loyalty card number. The user provides a policy expressing her preferred primary location for storing the data. It also specifies alternative locations where data can be moved for different reasons, for example, to achieve better performance or cheaper cost. The data owner's policy is provided with a PolicyID, which is also linked to the DataID in the data-policy linking table as shown in Figure 3. Each DataID is linked to a region information table that contains the current locations of the data identified by the DataID. This information is updated whenever the data location changes.

Accessing Data – All the requests for accessing

data (*e.g.* reading the data or processing the data) goes to the ECSM's authorization system. The ECSM first gets the data owner's PolicyID in the data-policy linking table and retrieves the policy from the data owners' policy store. The administrative policy and the data owner's policy are evaluated against the access request. If the policies evaluations are successful, the requested access is granted. The enterprise authorization system can be configured with various static conflict resolution strategies (Godik et al., 2002) or can obtain the conflict resolution strategy dynamically (Mohan and Blough, 2010) to resolve the conflicts between the administrative policy and data owner's policy.

Updating Data Location – The enterprise Cloud service administrator continuously monitors the performance of the services. If there is a need to copy the data to another location, she places a permission request to the ECSM authorization system. The ECSM accesses the data owner's policy and the administrative policy to evaluate them as described previously. If the policy allows for copying to the new location, then the data are copied and the region information table is updated to reflect these changes. If the user requires notification upon data location change, then the authorization system enforces this through the obligation enforcer. If the copy request is rejected then the data location remains unchanged.

Checking Location – When a user wants to query the location of her data, she first logs in to the service with her access credentials so that the service can identify the data she owns. To allow the user access the region information, either the administrator policy or the data owner policy has to allow it as the authorization system evaluates these policies to determine the access right. If any of the policies return "Grant" then the user is given the resquested information.

4.2 Inter-cloud Data Outsourcing

To achieve better efficiency and scalability, enterprises may need to outsource the processing of end users' data. For example, the retailer in our use case scenario might outsource the processing of the purchasing history to an analytics service. To realise this strategy, the enterprise maintains an external cloud information table to store the information of the external Cloud services with copies of the the data for each DataID. This table is linked with the region information table that contains the information of regions where the enterprise host its own service and originally stores the data, as seen in Figure 4.

The enterprise's administrative policy expresses which external services are allowed to access their protected data and each data owner's policy specifies the allowed locations where his/her data can be copied. When an external service requests permission from the ECSM's authorization system to transferring data to a certain location of their Cloud, the administrative policy along with each data owner's policy are evaluated. If the administrative policy or the data owner policy does not allow data transfer to the specified locations, then the permission is denied. If the request is accepted, the data along with the policy of the data owner are sent to the external service. A Service Level Agreement (SLA) negotiation process may take place between the enterprise and the external service provider in order to ensure that the data owner policy is respected upon any access request to the data. The external service provider may use the same PolicyID and DataID or may assign new IDs according to its organizational format. When assigning new IDs, there must be a link to the old ones for traceability. The external service provider maintains the outsourced data location similar to the approach described in Section 4.1. However, in this case access is granted based on the data owner's policy since the external service provider might have a different administrative policy.

A scheme similar to that for viewing location information as mentioned in Section 4.1 is provided to allow data owners to view their data location entries in the external cloud information table. To view the data location information, there are two options. The end user (data owner) can access this information through the default enterprise originally hosting her data. The default enterprise is then responsible for acquiring this information from the external service provider. The external service provider's administrative policy can be written in a way to allow this access. This can be specified in the SLA document with the external service provider. Alternatively, since the external service provider evaluates the data owners policy upon receipt of an access request for a user's data, the data owner's policy can be used to allow the necessary access to the data location information. Figure 4 shows how data location can be controlled by the ECSM's authorization systems for two different service managers.

5 IMPLEMENTATION

This section presents the technical details required for a practical implementation of the proposed location control model. This section also includes a description of a prototype implementation and testbed that used to perform validation tests of the location control model.

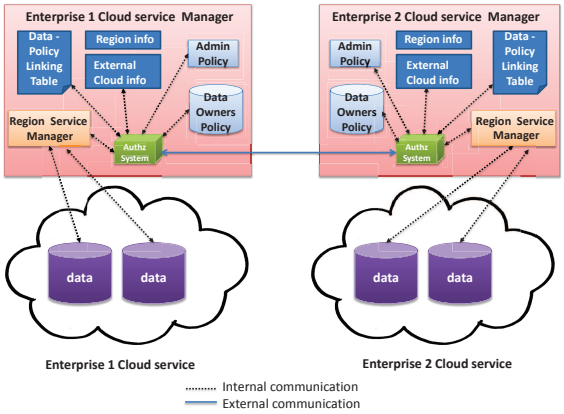


Figure 4: Controlling data location across multiple Clouds.

5.1 Data Presentation

Our model assumes that each end user's data is uniquely identified by a DataID. For our loyalty card use case scenario, XML is chosen as the representation format as it allows the data to be easily structured in separate elements which can be easily identified, modified and extended. However, alternate representation schemes, such as database tables or NoSQL entries could also be used. An example of the loyalty card XML format is given in Listing 1, which consists of two main elements: *PersonalInfo*, describing the personal identification information for a customer, and *PurchaseHistory* which encapsulates a list of the customer's purchases. Additional elements could be added based on the details of the specific loyalty card programme offered by the enterprise.

Listing 1: Loyalty Card Example.

```
<LoyaltyCard number="012345">
 <PersonalInfo>
    <Name>A B</name>
    <Address>IR</Address>
    <Age>30</Age>
    <Sex>Female</Sex>
 </PersonalInfo>
<PurchaseHistory>
  <Purchase>
    <Product>
      <ProductName>Bread</ProductName>
      <Brand>Acme</Brand>
      <Amount>3</Amount>
      <Price>15.50</Price>
      <DateOfPurchase>23.3.2013</
          DateOfPurchase>
      <PurchaseTime>15:00</PurchaseTime>
      <PurchaseStore>Cork</PurchaseStore>
    </Product>
  </Purchase>
</PurchaseHistory>
```

5.2 Policy Creation

Although the policies expressed as XML are human readable to an extent but are not particularly user-friendly, and it is unreasonable to expect end users to be able to write policies with these policy languages. Therefore, some form of user-friendly interface and translation mechanism is required. In our model, the data owner is presented with a web interface (implemented with PHP) for indicating location preferences. The interface works like a template for policy; the policy document is created automatically based on the locations that the user selects. The prototype implementation of the interface can be found in online[1]. After the customer's location preferences have been entered the resulting XACML request context containing user's policy (as PolicyContents element) is generated. This request context is then sent to the authorization system via a SOAP call. Depending on the enterprise's application scenario a default policy can be created when the user does not choose any location. In our prototype implementation the end user has to choose at least one location where she data to be stored initially and if the end user does not choose any optional locations then the created policy contains only the initial location.

The XACML policy language is used for the internal representation of policies as it is an OASIS standard and is the most widely used language for defining security policy (Turkmen and Crispo, 2008). XACML allows various types of attributes to be expressed, such as *Subject*, *Resource*, *Action* and *Environment*. The attributes presented in a policy are matched against the attributes presented in a request context by the PDP (see Section 3) when making an authorization decision. We have defined *Location* as an Environment attribute in our XACML policy which requires the request context to specify an appropriate value to be evaluated by the policy. An example of a policy document derived from a data owner's input is presented in Listing 2, allowing store/copy/transfer to Region 1 and Region 2 and obligating that an e-mail be sent to him/her when the location changes.

Listing 2: Data owner's policy example.

```
<PolicySet  PolicyCombiningAlgId="deny-
    overrides">
<Policy PolicyId="UserAccessPolicy1"
    RuleCombiningAlgId="deny-overrides">
<Target/>
 <Rule RuleId="Rule1" Effect="Permit">
```

[1]http://143.239.71.90:8013/data_location/User_registration.html

```
<Description>My data can be copied/
    stored/transferred in Region 1 and
    Region 2
</Description>
<Target/>
<Condition>
 <Apply FunctionId="and">
 <Apply FunctionId=
   "string-at-least-one-member-of">
 <ActionAttributeDesignator AttributeId=
  "action-id" DataType="string"/>
 <Apply FunctionId="string-bag">
  <AttributeValue DataType="string">
        COPY  </AttributeValue>
  <AttributeValue DataType="string">
        STORE  </AttributeValue>
  <AttributeValue DataType="string">
        TRANSFER  </AttributeValue>
 </Apply> </Apply>
 <Apply FunctionId=
  "string-at-least-one-member-of">
 <EnvironmentAttributeDesignator
     AttributeId="Location"
        DataType="string"/>
 <Apply FunctionId="string-bag">
    <AttributeValue DataType="string">
         Region 1  </AttributeValue>
    <AttributeValue DataType="string">
         Region 2  </AttributeValue>
  </Apply></Apply></Apply>
</Condition>
 </Rule>
<Obligations>
  <Obligation
  ObligationId="Email-to-AB@gmail.com"
  FulfillOn="Permit"/>
</Obligations>
 </Policy>
</PolicySet>
Note: Some prefixes from XACML elements
    are removed for readability.
```

5.3 Visibility for Data Owners

As described in Section 4, the location information for each user's data is referenced in the region information table of the ECSM. As the policies that are evaluated against an access request decide whether the access request should be granted or not, there needs to be a policy to allow the data owner to access the entries in the region information table for his/her data. This can be placed either inside the Cloud administrative policy set or the data owner's policy set. When it is stored in the Cloud Administrator's policy set, it has to be written in a generic way without requiring each individual user's credential to be specified in the policy. An example of such a policy is: If the credential of the requester matches the data owner's credential then allow access to the Region Information of that specific data. The result of such a generic policy is that there will exist only one policy for this purpose in the system. This policy will match the credential of the data owner, which is passed to the authorization system from the Authentication Information table, with the credential presented by the requester while making an access request to the service.

Alternatively, the policy to allow data owners to access the region information of their data can be placed in the data owners' policy sets. The policy would allow the requester presenting the appropriate credential to access region information for his/her data. A drawback with this approach is that the size of policy for each data owner will be increased which may lead to significant growth in the size of the overall policy store. On the other hand, a benefit of this approach is that it would ease the process of distributed enforcement of the data owner's policy in a remote Cloud as the data owner's policy will be passed to the remote Cloud service and no extra mechanism will be needed to identify the data owner.

With the first approach, the user credential and the administrative policy would also need to be passed to the remote service in order to allow the user to access location information from an external Cloud. The remote service would need to make sure that these user credentials are stored correctly and passed to the request contexts and would also need to integrate the received administrative policy with its own administrative policy. We therefore chose the second approach to ease the distributed enforcement of our location control model. Listing 3 shows an example policy which is placed inside the data owner's policy to allow access to the Region Information table. It specifies the UserName and Password that are to be matched to allow the access.

Listing 3: Data owner's policy example to permit access to region information.

```
<Policy PolicyId="UserAccessPolicy2"
    RuleCombiningAlgId="deny-overrides">
<Target/>
<Rule RuleId="Rule2" Effect="Permit">
<Description>Requester presenting the
    specified credential can access the
    region information </Description>
<Target/>
<Condition>
 <Apply FunctionId="and">
  <Apply FunctionId=
   "string-at-least-one-member-of">
   <SubjectAttributeDesignator
    AttributeId="UserName"
    DataType="string"/>
   <Apply FunctionId="string-bag">
    <AttributeValue DataType="string">
       Alice
    </AttributeValue>
  </Apply></Apply>
  <Apply FunctionId=
   "string-at-least-one-member-of">
    <SubjectAttributeDesignator
      AttributeId="Password"
```

```
      DataType="string"/>
    <Apply FunctionId="string-bag">
      <AttributeValue DataType="string">
         AlicePassword
      </AttributeValue>
  </Apply></Apply>
  <Apply FunctionId=
   "string-at-least-one-member-of">
    <ResourceAttributeDesignator
     AttributeId="ResourceType"
     DataType="string"/>
    <Apply FunctionId="string-bag">
      <AttributeValue DataType="string">
         RegionInformation
      </AttributeValue>
    </Apply></Apply>
 </Apply>
</Condition>
</Rule>
</Policy>
```

5.4 ECSM Implementation

The enterprise cloud service manager for the prototype was implemented using PHP. It in turn uses an open source authorization system that is implemented in Java as a web service and is available online[2]. This authorization system receives requests as *XACMLAuthzDecisionQuery* elements of the SAML profile of XACML and returns decisions in the form of *XACMLAuthzDecisionStatementType* elements. The system can store policy that comes with the request context and allows the policy of the data owner to be stored. It maintains the data-policy linking table for the provided DataID and PolicyID. It can return obligations if configured appropriately in the policy and can retrieve policy for a relevant DataID which is passed by the ECSM to remote authorization systems via SOAP calls. The remote authorization system receiving a request based on its policy either accepts the request and stores the policy or rejects the request. The authorization system can be configured to use various conflict resolution rules for combining the decisions of administrative policy and data owners' policy. The details of the conflict resolution rules can be found in (Fatema et al., 2011). In this instance, the deny-overrides conflict resolution rule is used. The rationale behind this choice is that it will ensure that if either the administrative policy or the data owner's policy returns a "Deny" the final decision will be a "Deny". The final decision will be a "Grant" only if one of the policies returns a "Grant" but no other policy has returned a "Deny".

We have implemented the Authentication Information table and Region Information table as MySQL tables. In the prototype implementation the Region Service Manager stores the data in different remote directories based on the specified region using the SSH SCP protocol to copy the data. An example of request context showing a copy request by an enterprise role Admin for a data to a certain location is presented in Listing 4 and the corresponding response from the authorization system is presented in Listing 5. For readability, the prefixes of some elements are removed.

Listing 4: Copy request by Admin.

```
<soapenv:Envelope xmlns:soapenv="http://
    schemas.xmlsoap.org/soap/envelope/">
<soapenv:Header/>
<soapenv:Body>
<XACMLAuthzDecisionQuery>
<xacml-context:Request>
<xacml-context:Subject>
<xacml-context:Attribute
  AttributeId="Role"
DataType="string">
<xacml-context:AttributeValue>
      Admin
</xacml-context:AttributeValue>
</xacml-context:Attribute>
</xacml-context:Subject>
<xacml-context:Resource>
<xacml-context:Attribute
  AttributeId="rid"
DataType="string">
<xacml-context:AttributeValue>
      LC001
</xacml-context:AttributeValue>
</xacml-context:Attribute>
</xacml-context:Resource>
<xacml-context:Action>
<xacml-context:Attribute
   AttributeId="action:action-id"
DataType="string">
<xacml-context:AttributeValue>
        COPY
</xacml-context:AttributeValue>
</xacml-context:Attribute>
</xacml-context:Action>
<xacml-context:Environment>
<xacml-context:Attribute
   AttributeId="Location"
DataType="string">
<xacml-context:AttributeValue>
        Region1
</xacml-context:AttributeValue>
</xacml-context:Attribute>
</xacml-context:Environment>
</xacml-context:Request>
</XACMLAuthzDecisionQuery>
</soapenv:Body>
</soapenv:Envelope>
```

Listing 5: Response to copy request.

```
<soapenv:Envelope xmlns:soapenv="http://
    schemas.xmlsoap.org/soap/envelope/">
<soapenv:Body>
 <xacml-context:Response xmlns:xacml-
     context="urn:oasis:names:tc:xacml:2
```

[2]http://sec.cs.kent.ac.uk/permis/downloads/Level3/standalone.shtml

```
      .0:context:schema:os">
4    <xacml-context:Result
5    ResourceId="ou=some,o=service,c=gb">
6       <xacml-context:Decision>
7             Permit
8       </xacml-context:Decision>
9        <xacml-context:Status>
10          <xacml-context:StatusCode
11          Value="
                urn:oasis:names:tc:xacml:1
                .0:status:ok"/>
12       </xacml-context:Status>
13       <xacml:Obligations xmlns:xacml=
14         "urn:oasis:names:tc:xacml:2.0
               :policy:schema:os">
15       <xacml:Obligation ObligationId=
16         "Email-to-AB@gmail.com"
17          FulfillOn="Permit"/>
18       </xacml:Obligations>
19     </xacml-context:Result>
20    </xacml-context:Response>
21   </soapenv:Body>
22 </soapenv:Envelope>
```

5.5 Validation

Validation testing of the prototype was performed in order to confirm the correctness of the following functionality:

1. Data owners' location preferences are correctly translated into policy.
2. Data and policy are both stored correctly.
3. When a copy request is sent by an administrator to a region that is accepted by the data owner's policy, a "Grant" response is returned and an obligation to e-mail the data owner is created. On receiving a "Grant" response the data is retrieved from its current region and sent to the approved region and the region information table is updated accordingly.
4. When a copy request is sent by an administrator to a region that is *not* permitted by the data owner's policy, a "Deny" response is returned and the data is not copied in that case.
5. Requests by a data owner for region information with valid credentials are accepted and the data owner is given the region information only for her data.
6. When a transfer request is sent by an administrator to a remote Cloud service in a region that is accepted by the data owner's policy, a "Grant" response, along with the data owner's policy is received from the authorization system.
7. Remote services correctly store the data along with the data owner's policy after data has been transferred.

An additional objective was to determine the performance overhead of the system.

The validation work was performed using an instance of the prototype implementation deployed on a virtual machine running in a vSphere private cloud with ESXi host. The virtual machine was configured with one virtual core Intel Xeon E5620 CPU running at 2.4 GHz with 2GB of memory. Another virtual machine with an identical configuration was used to mimic a remote storage location. All of the functionality described above was found to operate correctly. By averaging over the response times of 100 requests in total it was found that it takes approximately 0.04*s* in order to process a transaction that (a) identifies the correct data location and actions to perform on that data; and (b) queries the authorization system, obtains a response and updates the database accordingly; which indicates that at least 25 requests can be handled per sec by the current implementation of the model. In contrast, a 0.2*s* overhead was observed for connecting to the remote storage via SSH and transferring a file containing an XML representation of a loyalty card of size less than 1KB via SCP. Therefore, for this particular scenario the overhead of using the location control model was found to be negligible compared to the overall cost of performing remote data transfers.

6 CONCLUSION AND FUTURE WORK

The data location control model presented here allows enterprises to manage location of end users' data based on their preferences. It also empowers users with the ability to get up-to-date location information for their data and the assurance that they will be notified when their data changes location. A drawback of this approach is that it relies on the trustworthiness of the provider who must be trusted to integrating requests to the authorization system into their software and procedures and honour the resulting output. Hence, end users cannot verify that location information provided to them is actually true. However, these issues could be addressed by trustmarks and other third-party verification methods. Crytographic techniques could also be used to enhance the trustworthiness of the model. For example, the data could be encrypted using a key that can must be obtained through a request to the authorization system.

For convenience, our prototype stored the data to be protected as files in XML format. However, the model could also be used to protect access to collections of files, such as medical records composed of a

variety of files in various formats. Future work will focus on improving the granularity of policies to allow for selective disclosure of data. This granularity could be extended to database entries where the one single row can be identified by DataID and selective disclosure can be provided for various items represented by columns of that row. A private cloud was used as a testbed to perform the validation of the system. Future work will examine the issues around running the authorization system on public clouds. For example, in a federated cloud scenario, how well do the region boundaries of each provider correlate with those of the others?

Future work will also focus on how to integrate location awareness into open cloud platforms such as OpenStack. For OpenStack, obvious integration points include: Horizon, the web-based dashboard that controls all OpenStack components; Keystone, which manages the authentication and authorization of services and users; Swift, which provides object store funtionality; and Cinder, which provides block storage. OpenStack supports the concepts of geographically dispersed regions with separate endpoints (Jackson, 2012), providing a good fit with the data control model described in Section 4. Under this scenario, one Keystone and Horizon is shared between the regions to provide a common access control and dashboard, while distributed Swift and Cinder components allow for complete separation of storage by region. The object storage functionality provided by Swift provides an easier starting point for the integration of location control compared to Cinder as it deals with named, atomic units of data. A first step will to be to allow location preferences to be associated with users and Swift objects, and to provide an API that allows developers to check the permissibility of copying objects from one region to another.

ACKNOWLEDGEMENTS

The research work described in this paper was supported by the Irish Centre for Cloud Computing and Commerce, an Irish national Technology Centre funded by Enterprise Ireland and the Irish Industrial Development Authority.

REFERENCES

Albeshri, A., Boyd, C., and Nieto, J. (2012). Geoproof: Proofs of geographic location for cloud computing environment. In *Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on*, pages 506–514.

Almutairi, A., Sarfraz, M., Basalamah, S., Aref, W., and Ghafoor, A. (2012). A distributed access control architecture for cloud computing. *Software, IEEE*, 29(2):36–44.

Anderson, A. (2005). OASIS extensible access control markup language (XACML) version 2.0. *OASIS Standard*, 1.

Ashley, P., Hada, S., Karjoth, G., Powers, C., and Schunter, M. (2003). Enterprise privacy authorization language (epal 1.2). *Submission to W3C*.

Basescu, C., Carpen-Amarie, A., Leordeanu, C., Costan, A., and Antoniu, G. (2011). Managing data access on clouds: A generic framework for enforcing security policies. In *Advanced Information Networking and Applications (AINA), 2011 IEEE International Conference on*, pages 459–466.

Benson, K., Dowsley, R., and Shacham, H. (2011). Do you know where your cloud files are? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 73–82. ACM.

Chadwick, D., Zhao, G., Otenko, S., Laborde, R., Su, L., and Nguyen, T. A. (2008). PERMIS: a modular authorization infrastructure. *Concurrency and Computation: Practice and Experience*, 20(11):1341–1357.

Chadwick, D. W. and Fatema, K. (2012). A privacy preserving authorisation system for the cloud. *Journal of Computer and System Sciences*, 78(5):1359–1373.

Chen, D. and Zhao, H. (2012). Data security and privacy protection issues in cloud computing. In *Computer Science and Electronics Engineering (ICCSEE), 2012 International Conference on*, volume 1, pages 647–651. IEEE.

Cranor, L. F. (2003). P3P: Making privacy policies more useful. *Security & Privacy, IEEE*, 1(6):50–55.

Demchenko, Y., Koeroo, O., de Laat, C., and Sagehaug, H. (2008). Extending XACML authorisation model to support policy obligations handling in distributed application. In *Proceedings of the 6th international workshop on Middleware for grid computing*, page 5. ACM.

di Vimercati, S. D. C., Samarati, P., and Jajodia, S. (2005). Policies, models, and languages for access control. In *Databases in Networked Information Systems*, pages 225–237. Springer.

Fatema, K., Chadwick, D. W., and Lievens, S. (2011). A multi-privacy policy enforcement system. In *Privacy and Identity Management for Life*, pages 297–310. Springer.

Godik, S., Anderson, A., Parducci, B., Humenn, P., and Vajjhala, S. (2002). Oasis extensible access control 2 markup language (xacml) 3. Technical report, Tech. rep., OASIS.

Gondree, M. and Peterson, Z. N. (2013). Geolocation of data in the cloud. In *Proceedings of the third ACM conference on Data and application security and privacy*, pages 25–36. ACM.

Iskander, M. K., Wilkinson, D. W., Lee, A. J., and Chrysanthis, P. K. (2011). Enforcing policy and data consis-

tency of cloud transactions. In *Distributed Computing Systems Workshops (ICDCSW), 2011 31st International Conference on*, pages 253–262. IEEE.

ISO (1996). Information technology – open systems interconnection – security frameworks for open systems: Access control framework.

Jackson, K. (2012). *OpenStack Cloud Computing Cookbook*. Packt.

Katz-Bassett, E., John, J. P., Krishnamurthy, A., Wetherall, D., Anderson, T., and Chawathe, Y. (2006). Towards IP geolocation using delay and topology measurements. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 71–84. ACM.

Lynn, T., Healy, P., McClatchey, R., Morrison, J., Pahl, C., and Lee, B. (2013). The case for cloud service trustmarks and assurance-as-a-service. In *Intl. Conference on Cloud Computing and Services Science Closer'13*.

Massonet, P., Naqvi, S., Ponsard, C., Latanicki, J., Rochwerger, B., and Villari, M. (2011). A monitoring and audit logging architecture for data location compliance in federated cloud infrastructures. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 1510–1517.

Mohan, A. and Blough, D. M. (2010). An attribute-based authorization policy framework with dynamic conflict resolution. In *Proceedings of the 9th Symposium on Identity and Trust on the Internet*, pages 37–50. ACM.

Noman, A. and Adams, C. (2012). DLAS: Data location assurance service for cloud computing environments. In *Privacy, Security and Trust (PST), 2012 Tenth Annual International Conference on*, pages 225–228. IEEE.

Park, J. and Sandhu, R. (2004). The UCON ABC usage control model. *ACM Transactions on Information and System Security (TISSEC)*, 7(1):128–174.

Peterson, Z. N., Gondree, M., and Beverly, R. (2011). A position paper on data sovereignty: The importance of geolocating data in the cloud. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*.

Ries, T., Fusenig, V., Vilbois, C., and Engel, T. (2011). Verification of data location in cloud networking. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pages 439–444. IEEE.

Rissanen, E. (2013). OASIS extensible access control markup(XACML) version 3.0. *OASIS Standard*, 1.

Spillner, J. and Schill, A. (2012). Flexible data distribution policy language and gateway architecture. In *Cloud Computing and Communications (LATINCLOUD), 2012 IEEE Latin America Conference on*, pages 1–6. IEEE.

Subashini, S. and Kavitha, V. (2011). A survey on security issues in service delivery models of cloud computing. *Journal of Network and Computer Applications*, 34(1):1–11.

Turkmen, F. and Crispo, B. (2008). Performance evaluation of XACML PDP implementations. In *Proceedings of the 2008 ACM workshop on Secure web services*, pages 37–44. ACM.

A Cloud Accountability Policy Representation Framework

Walid Benghabrit[1], Hervé Grall[1], Jean-Claude Royer[1], Mohamed Sellami[1], Monir Azraoui[2], Kaoutar Elkhiyaoui[2], Melek Önen[2], Anderson Santana De Oliveira[3] and Karin Bernsmed[4]

[1]*Mines Nantes, 5 rue A. Kastler, F-44307 Nantes, France*

[2]*EURECOM, Les Templiers, 450 Route des Chappes F - 06410 Biot Sophia Antipolis, France*

[3]*SAP Labs France, 805 avenue du Dr Donat Font de l'Orme F - 06250 Mougins Sophia Antipolis, France*

[4]*SINTEF ICT, Strindveien 4, NO-7465 Trondheim, Norway*

{*firstname.lastname*}*@mines-nantes.fr,* {*firstname.lastname*}*@eurecom.fr, anderson.santana.de.oliveira@sap.com, Karin.Bernsmed@sintef.no*

Keywords: Accountability, Data Protection, Framework, Policy Language, Policy Enforcement.

Abstract: Nowadays we are witnessing the democratization of cloud services. As a result, more and more end-users (individuals and businesses) are using these services for achieving their electronic transactions (shopping, administrative procedures, B2B transactions, etc.). In such scenarios, personal data is generally flowed between several entities and end-users need (i) to be aware of the management, processing, storage and retention of personal data, and (ii) to have necessary means to hold service providers accountable for the usage of their data. In fact, dealing with personal data raises several privacy and accountability issues that must be considered before to promote the use of cloud services. In this paper, we propose a framework for the representation of cloud accountability policies. Such policies offer to end-users a clear view of the privacy and accountability obligations asserted by the entities they interact with, as well as means to represent their preferences. This framework comes with two novel accountability policy languages; an abstract one, which is devoted for the representation of preferences/obligations in an human readable fashion, a concrete one for the mapping to concrete enforceable policies. We motivate our solution with concrete use case scenarios.

1 INTRODUCTION

According to (Pearson et al., 2012), accountability regards the data stewardship regime in which organizations that are entrusted with personal and business confidential data are responsible and liable for processing, sharing, storing and otherwise using the data according to contractual and legal constraints from the time it is collected until when the data is destroyed (including onward transfers to third parties). Obligations associated to such responsibilities can be expressed in an accountability policy, which is a set of rules that defines the conditions under which an accountable entity must operate.

Today, there is neither an established standard for expressing accountability policies nor a well defined way to enforce these policies. Since cloud services often combine infrastructure, platform and software applications to aggregate value and propose new cloud applications to individuals and organizations, it is fundamental for an accountability policy framework to enable "chains of accountability" across cloud services addressing regulatory, contractual, security and privacy concerns.

In the context of the EU FP7 A4Cloud project[1] we are currently working on defining a framework where accountability policies will be enforceable across the cloud service provision chain by means of accountability services and tools. Accountable organizations will make use of these services to ensure that obligations to protect personal data and data subjects' rights[2] are observed by all who store and process the data, irrespective of where that processing occurs. Under the perspective of the concept of accountability, we have elicited the following types of accountability obligations that must be considered while designing our policy framework:

- Access and Usage Control rules - express which rights should be granted or revoked regarding the

[1]The Cloud Accountability Project: http://www.a4cloud.eu/.

[2]This work mainly focus on the European Data Protection directive (Directive, E. U., 1995).

use and the distribution of data in cloud infrastructures, and support the definition of roles as specified in the Data Protection Directive, e.g. data controller and data processor.

- Capturing privacy preferences and consent - to express user preferences about the usage of their personal data, to whom data can be released, and under which conditions.
- Data Retention Periods - to express time constraints about personal data collection.
- Controlling Data Location and Transfer - clear whereabouts of location depending on the type of data stored and on the industry sector processing the data (subject to specific regulations) must be provided. Accountability policies for cloud services need to be able to express rules about data localization, such that accountable services can signal where the data centers hosting them are located. Here we consider strong policy binding mechanisms to attach policies to data.
- Auditability - Policies must describe the clauses in a way that actions taken upon enforcing the policy can be audited in order to ensure that the policy was adhered to. The accountability policy language must specify which events have to be audited and what information related to the audited event have to be considered.
- Reporting and notifications - to allow cloud providers to notify end-users and cloud customers in case of policy violation or incidents for instance.
- Redress - express recommendations for redress in the policy in order to set right what was wrong and what made a failure occur.

In this paper we provide a cloud accountability policy representation framework designed while considering the aforementioned requirements. We define an abstract yet readable language, called AAL, for accountability obligations representation in a human readable fashion. We also define a concrete policy enforcement language, called A-PPL, as an extension of the PPL (Ardagna et al., 2009) language. The proposed framework, offers the means for a translation from abstract obligations expressed in AAL to concrete policies in A-PPL.

The rest of this paper is organized as follows. Section 2 describes related work. Section 3 gives an overview on the main components of our policy representation framework. We present the abstract accountability policy language we propose in Section 4 and the concrete one in Section 5. Section 6 describes a realistic use case as a proof of concept to our work. Section 7 discusses our work and presents directions for future work.

2 RELATED WORK

In the following, we provide an overview of related work in the field. We organize this section along the following categories that relate to our contribution in this paper: accountability in computer science, obligations in legal texts and directives, enforcement and policy languages.

2.1 Accountability

There is a recent interest and active research for accountability which overlap several domains like security (Weitzner et al., 2008; Zhifeng Xiao, 2012; Pearson and Wainwright, 2013), language representation (Métayer, 2009; DeYoung et al., 2010), auditing systems (Feigenbaum et al., 2012; Jagadeesan et al., 2009), evidence collection (Sundareswaran et al., 2012; Haeberlen et al., 2010) and so on. However, only few of them consider an interdisciplinary view of accountability taking into account legal and business aspects. We particularly emphasize the work from (Feigenbaum et al., 2012) and (Pearson and Wainwright, 2013) since they provide a general, concrete view and yet an operational approach.

Regarding tool supports and frameworks we can find several proposals (Wei et al., 2009; Haeberlen et al., 2010; Zou et al., 2010), but none of them provides a holistic approach for accountability in the cloud, from end-user understandable sentences to concrete machine-readable representations. In (Sundareswaran et al., 2012), authors propose an end-to-end decentralized accountability framework to keep track of the usage of the data in the cloud. They suggest an object-centered approach that packs the logging mechanism together with users' data and policies.

2.2 Obligations in Regulations

There is an international trend in protecting data, for instance in Europe with Directive 95/46/EC (Directive, E. U., 1995), the HIPAA rules (US Congress, 2002) in the USA and the FIPPA act (Legislative Assembly of Ontario, 1988) in Canada. As an example, Directive 95/46/EC states rules to protect personal data in case of processing or transferring data to other countries. There exist some attempts to formalize or to give rigorous analyses of this kind of rules. In (Métayer, 2009) the authors present a restricted

natural language SIMPL (SIMple Privacy Language) to express privacy requirements and commitments. In (Breaux and Anton, 2005) the authors describe a general process for developing semantic models from privacy policy goals mined from policy documents. In (Kerrigan and Law, 2003), the authors develop an approach where contracts are represented by XML documents enriched with logic metadata and assistance with a theorem prover. In (DeYoung et al., 2010) the authors provide a formal language to express privacy laws and a real validation on the HIPAA and GLBA (US Congress, 1999) sets. These works either are not end-to-end proposals, only cover data privacy not accountability or are only formal proposals without an enforcement layer.

2.3 Enforcement and Policies

A number of policy languages have been proposed in recent years for machine-readable policy representation. We reviewed several existing policy languages (see (Garaga et al., 2013) for details), defined either as standards or as academic/industrial proposals.

Existing policy languages have focused on specific obligations. The eXtensible Access Control Markup Language (XACML) (OASIS Standard, 2013) aims at providing a declarative language for access control. The XML-based languages P3P (Platform for Privacy Preferences Project) (Marchiori, 2002), PPL (Primelife Policy Language) (Ardagna et al., 2009) and SecPal4P (Becker et al., 2010) are used to describe privacy policies and data collection policies. SLAng (Lamanna et al., 2003) and ConSpec (Aktug and Naliuka, 2008) are designed to automatize contract negotiations and to monitor the agreed contract statements. However, all these proposals fail to provide elements for specifying accountability-specific obligations such as logging, reporting, audits, evidence collection and redress.

Having identified the limitations of existing languages, we analyzed their extensibility and their suitability to express accountability obligations. Extensible languages such as XACML and PPL appear to be the most suitable. In our work we consider PPL since it provides elements that capture the best accountability obligations (see Section 5). In a nutshell, PPL extends XACML, an XML-based language aimed for access control, to provide an automatic means to define and manage privacy policies. PPL allows expressing data handling policies (on the data controller side) and data handling preferences (on the data subject side) that are evaluated and matched to output a sticky policy that travels with the data downstream. Therefore, PPL specifies statements on access control, authorizations and obligations. In addition, the language provides a way to declare some accountability-specific obligations such as logging and notifications. However, they fail to capture these obligations accurately and may be unpractical when directly used within an accountability policy. Besides, auditing is not part of PPL since its focus is on privacy and not accountability.

3 THE CLOUD ACCOUNTABILITY FRAMEWORK

In this section, we provide an overview of our proposed policy representation framework. Such framework must allow end-users to easily express their accountability obligations and preferences and even be complete and rigorous enough to be run by a policy execution engine. Hence we are faced with the following dilemma: the policy must be written by an end-user, which does not necessarily have skills in a certain policy language and the policy must be machine understandable at the same time. Machine understandable means that sentences can be read, understood and executed by a computer.

In this context, we propose a policy representation framework (see Figure 1) that allows a user, step (1) in Figure 1, to express his accountability needs in a human readable fashion and (2) offers the necessary means to translate them (semi-)automatically[3] to a machine understandable format.

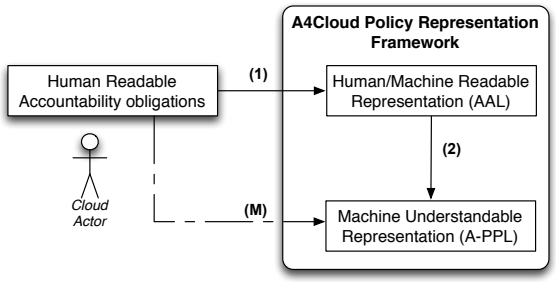


Figure 1: Overview on the accountability policy representation framework.

Accountability as it appears in legal, contractual and normative texts about data privacy make explicit four important roles that we consider in our proposal:

- Data subject: this role represents any end-user which has data privacy concerns, mainly because he outsourced some of its data to a cloud provider.

[3]here "semi" means that sometimes human assistance could be needed.

- Data processor: this role is attributed to any computational agent which processes some personal data. It should act under the control of a data controller.
- Data controller: it is legally responsible to the data subject for any violations of its privacy and to the data protection authority in case of misconduct.
- Auditor: it represents data protection authorities which are in charge of the application of laws and directives.

3.1 Step (1). Human/machine Readable Representation

To express accountability obligations we define an Abstract Accountability Language (AAL), which is devoted to expressing accountability obligations in an unambiguous style and which is close to what the end-user needs and understands. As this is the human readable level, this language should be simple, akin to a natural logic, that is a logic expressed in a subset of a natural language.

For instance, a simple access control obligation to state that "the data d cannot be read by all agents" will be formulated in a human/machine readable fashion using our accountability language as "MUSTNOT ANY:Agent.READ(d:Data)". Details on the AAL syntax are provided in Section 4.

3.2 Step (2). Machine Understandable Representation

In this step (called the mapping), the accountability obligations expressed in AAL are (semi-)automatically translated into a machine understandable policy. We target a policy language that is able to enforce classic security means (like access or usage controls) but also accountability obligations. Such automatic translation may need several passes, due to the high level of abstraction of AAL.

As analyzed in Section 2, the PrimeLife Policy Language (PPL) (Ardagna et al., 2009) seems the most convenient language for privacy policies representation. It can be extended to address specific accountability obligations such as auditability, notification or logging obligations. Hence, we propose an extension to PPL, A-PPL for accountable PPL, which supports such obligations. The details of this extension are described in Section 5.

4 ABSTRACT LANGUAGE

We introduce in this section AAL (Abstract Accountability Language), which is devoted to expressing accountability obligations in an unambiguous human readable style. The AAL concepts are presented in Section 4.1, its syntax in Section 4.2 and we provide an outlook on our approach for a machine understandable representation of AAL policies in Section 4.3.

4.1 AAL Concepts

As explained in (Feigenbaum et al., 2012) an accountable system can be defined with five steps: prevention, detection, evidence collection, judgment and punishment. We follow this line for the foundation of our accountability language. In AAL, usage control expressions represent the preventive description part. Audit expressions encompass the detection, evidence collection and judgment parts. Finally, rectification expressions represent the punishment description part. We use the term rectification since these expressions don't cover only punishment, but also remediation, compensation, sanction and penalty. Thereby, an AAL sentence is a property (more formally a distributed system invariant) expressing usage control, auditing and rectification. The general form of an AAL sentence is: `UsageControl Auditing Rectification` and the informal meaning is: *try to ensure the usage control, in case of an audit, if a violation is observed then the rectification applies*. The reader should also note that there are two flavors of AAL sentences:

- User preferences: expressing the obligations a data subject wants to be satisfied, for instance he does not want its data to be distributed over the network or only used for statistics by a given data processor, and so on.
- Processor obligations (sometimes called obligations or policies): these are the obligations the data processor declares to ensure regarding the data management and processing.

Finally, as many policy representation languages, we consider permission, obligation and prohibition in AAL. They occur in various approaches, like in PPL, or in the ENDORSE project[4]. Permission, obligation and prohibition are respectively expressed in AAL sentences with these keywords: `MAY`, `MUST` and `MUSTNOT`, as advocated by the IETF RFC 2119 (Bradner, 1997).

[4]http://ict-endorse.eu/

4.2 AAL Syntax

Figure 2 shows the syntax of AAL using a Backus-Naur Form (BNF) (Knuth, 1964) like syntax. AAL allows the expression of `Clauses` representing obligations that have to be met either in an accountability policy or preference. A `Clause` has one usage expression and optionally[5] an audit and a rectification expression: `Exp ('AUDITING' Exp)? ('IF_VIOLATED_THEN' Exp)?`. The expression `Exp` of a clause can be either atomic or composite. Composite `Exps` are written in the form `Exp ('OR' | 'AND' | 'ONLYWHEN' | 'THEN') Exp`.

As an example, consider the user preference of a data subject who grants read access to an agent A on its data D. This usage control is a permission, which can be expressed as follows.

```
MAY A.READ(D:Data)
```

But the full accountability sentence does imply that an auditor B will audit the system and, in case of violations, can sanction the data controller C.

```
MAY A.READ(D:Data)
AUDITING MUST B.AUDIT(C.logs)
IF_VIOLATED_THEN MUST B.SANCTION(C)
```

Further examples of user preferences and processor obligations expressed in AAL are provided in Section 6.

4.3 Machine Understandability

Generating machine understandable policies from accountability preferences and obligations written in AAL can be easily done when dealing with usage control clauses. However, this mapping is less obvious for clauses with temporal modalities and with auditing. The main issue for such mapping is the gap between the AAL language, which is property-oriented, and the machine understandable language, which is operational. To fill this gap we need more artifacts, Figure 3 provides an overview on our proposed mapping process.

According to this figure, we can see that going from a human/machine readable representation in AAL to a machine understandable representation of the accountability preferences/obligations (arrow numbered (2) in Figure 3) is done through three steps:

- **(2'.1).** First, a temporal logic is used to make more concrete AAL sentences as temporal logic properties. Indeed, in an accountability policy we should represent the notions of permission, obligation and prohibition. In addition, there is a need to express conditions and various logical combinations. Furthermore, one important thing is to have time, at least logical discrete time, for instance to write: "X writes some data and then stores some logs". Our target is a temporal logic with time, one concrete candidate is mCRL2 (Cranen et al., 2013).
- **(2'.2).** Second, a policy calculus is used to describe the operational semantics associated to the concrete properties defined in (2'.1). This calculus is based on the concept of reference monitor (Schneider, 2000) for both the agents and the data resources. It relies on a previous work for distributed agent communicating via messages (Allam et al., 2012). This operational semantics provides means for abstractly executing the temporal logic expressions. This process is known as "program synthesis", starting from a property it generates a program ensuring/enforcing the property.
- **(2'.3).** Finally, the generated policy using our policy calculus is (semi-)automatically translated to a machine understandable policy based on predefined transformation rules. Our target is the A-PPL extension of PPL which is described in the next section.

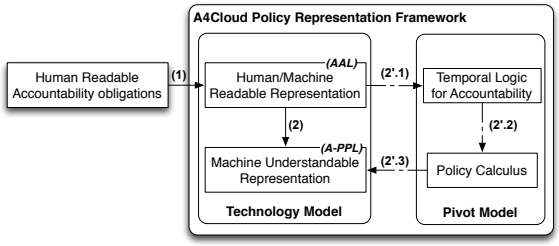


Figure 3: Overview on the machine understandable translation of AAL.

5 CONCRETE LANGUAGE

In this section, we present an enhanced version of PPL with extensions that address the identified limitations of the language to accurately map accountability obligations. We name this accountability policy language (A-PPL).

Our accountability policy representation framework maps AAL clauses to concrete and operational machine understandable policies. As already mentioned in section 2.3, in order not to define yet another completely new language to map accountability obligations to machine-understandable policies, we conducted a preliminary study on existing languages and among all the possible candidates, PPL seems the one

[5] Items followed by ? are optional expressions.

```
(1) Clause ::= (LOGGING? Exp (AUDITING Exp )? (IF_VIOLATED_THEN Exp )?)+
(2) Exp ::= Modal | Exp (OR|AND|ONLYWHEN|THEN) Exp
(3) Modal ::= (MAY|MUST|MUSTNOT) Action
(4) Action ::= Agent . Oper ( Param ) ((BEFORE|AFTER) Time)?
(5) Agent ::= uri
(6) Oper ::= READ|WRITE|LOG|SEND|NOTIFY|service[Agent_provider]
(7) Time ::= date | duration
(8) Param ::= Constant | Variable
(9) Constant ::= stringLiteral
(10) Variable ::= Literal?
```

Figure 2: Excerpt of the AAL Syntax.

that best captures the accountability concepts. Therefore, in this section, we present how A-PPL extends PPL to address accountability obligations.

PPL implicitly identifies three roles: the data subject, data controller and data processor roles. Besides, PPL defines an obligation as a set of triggers and actions. Triggers are events related to the obligation that are filtered by a condition and that trigger the execution of actions. Therefore, PPL defines markups to declare an obligation. Inside the obligation environment, one can specify a set of triggers and their related actions.

5.1 Extension of Roles

To address accountability concerns in a cloud environment, it might be necessary to include in the policy a reference to the role of the subject to which the policy is applied to. For instance, in PPL, it was not possible to identify the Data Controller. We therefore suggest adding to the PPL `<Subject>` element a new attribute, `attribute:role`, for this purpose. Furthermore, in addition to the four roles PPL inherently considers (Data Subject, Data Controller, Downstream Data Controller, Data Processor), A-PPL extends PPL with one additional role. We add the auditor role that is considered as a trusted third party that can conduct independent assessment of cloud services, information systems operations performance and security of the cloud implementation. This new role is important to catch accountability specific obligations such as auditability, reporting notification and remediation.

5.2 Extension of Actions and Triggers

We add to PPL a set of new A-PPL actions and triggers in order to map accountability obligations. In particular, we enhance the action of logging `ActionLog` and notification `ActionNotify` that already exist in PPL. For instance, while PPL currently enables notification thanks to the `ActionNotifyDataSubject`, A-PPL defines a new and more general `ActionNotify` action in which one can define the recipient of the notification thanks to a newly defined parameter `recipient`;moreover, the additional `Notificationtype` parameter defines the purpose of the notification which can be policy violation, evidences or redress for example. On the other hand, the current `ActionLog` action in PPL fails to capture accountability obligations. The new `ActionLog` action in A-PPL introduces many additional parameters to provide more explicit information on the logged event: for example, `timestamp` defines the time of the event, and `Resource location` identifies the resource the action was taken on. We also create two actions related to auditability: `ActionAudit` that creates an evidence request and `ActionEvidenceCollection` that collects requested evidence. In addition, auditability requires the definition of two new triggers related to evidence: `TriggerOnEvidenceRequestReceived` that occurs when an audited receives an evidence request and `TriggerOnEvidenceReceived` that occurs when an auditor receives the requested evidence. Similarly, when an update occurs in a policy or in a user preference, the update may trigger a set of actions to be performed. Thus, we create two additional triggers: `TriggerOnPolicyUpdate` and `TriggerOnPreferenceUpdate`. Finally, to handle complaints that a data subject may file in the context of remediability, we define the trigger `TriggerOnComplaint` that triggers a set of specific actions to be undertaken by an auditor or/and a data controller.

6 VALIDATION

In this section we validate the accountability policy representation framework by extracting obligations from one of the use cases documented in the A4Cloud public deliverable DB3.1 (Bernsmed et al., 2013) and illustrate their representation in AAL and A-PPL.

6.1 The Health Care Use Case

This use case concerns the flow of health care information generated by medical sensors in the cloud. The system, which is illustrated in Figure 4, is used to support diagnosis of patients by the collection and processing of data from wearable sensors. Here, we investigate the case where medical data from the sensors will be exchanged between patients, their families and friends, the hospital, as well as between the different Cloud providers involved in the final service delivery.

In this use case the patients are the data subjects from whom (sensitive) personal data is collected. The hospital is ultimately responsible for the health care services and will hence act as one of the data controllers for the personal data that will be collected. The patients' relatives may also upload personal data about the patients and can therefore be seen as data controllers (as well as data subjects, when personal data about their usage of the system is collected from them). As can be seen in Figure 4, the use case will involve cloud services for sensor data collection and processing (denoted cloud provider "X"), cloud services for data storage (denoted cloud provider "Y") and cloud services for information sharing (denoted cloud provider "M"), which will be operated by a collaboration of different providers. Since the primary service provider M, with whom the users will interface, employs two sub-providers, a chain of service delivery will be created. In this particular case, the M platform provider will be the primary service provider and will act as a data processor with respect to the personal data collected from the patients. Also the sub-providers, X and Y, will act as data processors. The details of the use case is further described in DB3.1 (Bernsmed et al., 2013).

6.2 Obligations for the Use Case

We have identified a number of obligations for this use case, which needs to be handled by the accountability policy framework. Here we list three examples and we explain how they will be expressed in AAL and mapped into A-PPL. Note that the complete list of obligations is much longer, but we have chosen to outline those that illustrate the most important relationships between the involved actors. Due to space limitations we do not include the complete A-PPL policies here; the reader is referred to the project documentation (Garaga et al., 2013) to see the full policy expressions.

Obligation 1: The Data Subject's Right to Access, Correct and Delete Personal Data. According the Data Protection Directive (Directive, E. U., 1995), data subjects have (among others) the right to access, correct and delete personal data that have been collected about them. In this use case it means that the hospital must allow read and write grant access to patients as well as relatives with regard to their personal data that have been collected and stored in the cloud. There must be also means to enforce the deletion of such data.

The AAL expression catching the obligations associated to the patient is:

```
(MAY Patient.READ(D:Data)  OR
 MAY Patient.WRITE(D:Data) OR
 MAY Patient.DELETE(D:Data))
AUDITING
  MUST Auditor.AUDIT(hospital.logs)
IF_VIOLATED_THEN
  MUST Auditor.SANCTION(hospital)
```

The policy includes the `ALWAYS` operator since this property is expected to be true at any instant. The policy also includes the condition `D.subject=Patient` to express that this expression only concerns the personal data of the Patient. This policy also expresses the audit and rectification obligations that have to be ensured by an external Auditor.

Using the accountability policy representation framework, the AAL expression will be mapped into two different A-PPL expressions; one for permitting read and write access to the patients and another one for enforcing the data controller to delete the personal data whenever requested. Read and write access control is achieved through XACML rules. Regarding deletion of data, a patient can express data handling preferences that specify the obligation that the data controller has to enforce to delete the personal data. This obligation can be expressed using the A-PPL obligation action `ActionDeletePersonalData`, which will be used by the patient to delete personal data that has been collected about him.

An explicit audit clause implies that information related to the usage control property are logged (the amount and the nature of this information is not discussed here). Thus the audit clause is translated into an `AuditAction` which is responsible to manage the interaction with the auditor. This runs an exchange protocol with the auditor which ends with two responses: either no violation of the usage has been detected or a violation exists. In the latter, some rectification clauses should be specified.

In the sequel we only consider usage control clauses since the translation process for audit and rectification is similar to the previous example.

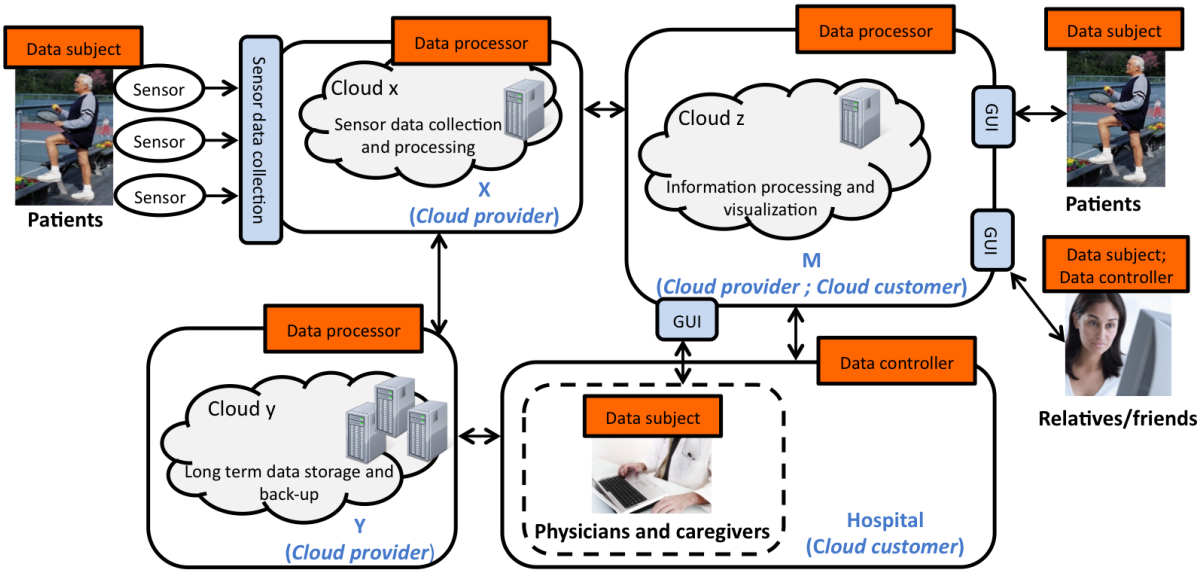


Figure 4: An overview over the main actors involved in the health care use case.

Obligation 2: The Data Controller Must Notify the Data Subjects of Security or Personal Data Breaches. This obligation defines what will happen in case of a security or privacy incident. In AAL it will be expressed by the hospital as:

```
MAY hospital.VIOLATEPOLICY() THEN
 (MUST hospital.NOTIFY[Patient]("incident") AND
  (ANY:Agent in hospital.relatives[Patient] THEN
    MUST hospital.NOTIFY[ANY]("incident")))
```

In A-PPL, such notification is expressed through the obligation action `ActionNotify`. It takes as parameters, the recipient of the notification (here, the data subject) and the type of the notification (here, security breach).

Obligation 3: The Data Processor Must, upon Request, Provide Evidence to the Data Controller on the Correct and Timely Deletion of Personal Data. To express the timely deletion of personal data, which in addition will be logged to be used as evidence, the following AAL expression can be used by the provider M:

```
MUST M.DELETE(D:Data) THEN
MUST M.LOGS("deleted", D, currentDate)
```

In A-PPL, the obligation trigger `TriggerPersonalDataDeleted` combined with the obligation action `ActionNotify` will notify the data subject of the deletion of its data. In addition, if necessary, the obligation action `ActionLog` will allow the provider M to log when personal data have been deleted.

The three examples that we have provided in this section represent a snapshot of the full power of AAL and A-PPL. In (Bernsmed et al., 2013) we outline more examples of obligations for the health care use case, which among other things demonstrate how informed consent can be gathered from the patients before their data is being processed, how the purpose of personal data collection can be specified and controlled, how the data processor M can inform the hospital of the use of sub-processors and how the data processors can facilitate for regulators to review evidence on their data processing practices.

7 CONCLUSION

Dealing with personal data in the cloud raises several accountability and privacy issues that must be considered to promote the safety usage of cloud services. In this paper we tackle the issue related to accountability obligations and preferences representation. We propose a cloud accountability policy representation framework. This framework enables accountability policy expression in a human readable fashion using our abstract accountability language (AAL). Also, it offers the means for their mapping to concrete enforcement policies written using our accountability policy language (A-PPL). Our framework applies the separation of concerns principle by separating the abstract language from the concrete one. This choice makes both contributions, i.e. AAL and A-PPL, self-contained and allows their independent use. The abil-

ity of our framework to represent accountability obligations was validated through a realistic health care use case.

Our future research work will focus on the mapping from AAL to A-PPL. As part of our implementation perspectives, we are currently working on two prototypes. An AAL editor that assists end-users in writing their preferences/obligations and implements the required artifacts to map them to concrete policies in A-PPL. We also started the development of an A-PPL policy execution engine that will be in charge of interpreting and matching A-PPL policies and preferences

ACKNOWLEDGEMENTS

This work was funded by the EU's 7th framework A4Cloud project.

REFERENCES

Aktug, I. and Naliuka, K. (2008). ConSpec – a formal language for policy specification. *In Electronic Notes in Theoretical Computer Science*, volume 197, pages 45–58.

Allam, D., Douence, R., Grall, H., Royer, J.-C., and Südholt, M. (2012). Well-Typed Services Cannot Go Wrong. *Rapport de recherche RR-7899, INRIA*.

Ardagna, C. A., Bussard, L., De Capitani Di Vimercati, S., Neven, G., Paraboschi, S., Pedrini, E., Preiss, S., Raggett, D., Samarati, P., Trabelsi, S., and Verdicchio, M. (2009). Primelife policy language. http://www.w3.org/2009/policy-ws/papers/Trabelisi.pdf.

Becker, M. Y., Malkis, A., and Bussard, L. (2010). S4P: A generic language for specifying privacy preferences and policies. *Microsoft Research*.

Bernsmed, K., Felici, M., Oliveira, A. S. D., Sendor, J., Moe, N. B., Rübsamen, T., Tountopoulos, V., and Hasnain, B. (2013). Use case descriptions. *Deliverable, Cloud Accountability (A4Cloud) Project*.

Bradner, S. (1997). IETF RFC 2119: Key words for use in RFCs to Indicate Requirement Levels. *Technical report*.

Breaux, T. D. and Anton, A. I. (2005). Deriving semantic models from privacy policies. In *Sixth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY '05)*, pages 67–76.

Cranen, S., Groote, J. F., Keiren, J. J. A., Stappers, F. P. M., de Vink, E. P., Wesselink, W., and Willemse, T. A. C. (2013). An overview of the mCRL2 toolset and its recent advances. *TACAS'13*, pages 199–213, Berlin, Heidelberg. Springer-Verlag.

DeYoung, H., Garg, D., Jia, L., Kaynar, D., and Datta, A. (2010). Experiences in the logical specification of the HIPAA and GLBA privacy laws. In *9th Annual ACM Workshop on Privacy in the Electronic Society (WPES '10)*, pages 73–82.

Directive, E. U. (1995). Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data . http://ec.europa.eu/justice/policies/privacy/docs/95-46-ce/dir1995-46_part1_en.pdf.

Feigenbaum, J., Jaggard, A. D., Wright, R. N., and Xiao, H. (2012). Systematizing "accountability" in computer science. *Technical Report YALEU/DCS/TR-1452*, University of Yale.

Garaga, A., de Oliveira, A. S., Sendor, J., Azraoui, M., Elkhiyaoui, K., Molva, R., Önen, M., Cherrueau, R.-A., Douence, R., Grall, H., Royer, J.-C., Sellami, M., Südholt, M., and Bernsmed, K. (2013). Policy Representation Framework. Technical Report D:C-4.1, Accountability for Cloud and Future Internet Services - A4Cloud Project.

Haeberlen, A., Aditya, P., Rodrigues, R., and Druschel, P. (2010). Accountable virtual machines. In OSDI, pages 119–134.

Jagadeesan, R., Jeffrey, A., Pitcher, C., and Riely, J. (2009). Towards a theory of accountability and audit. In *Proceedings of the 14th European conference on Research in computer security*, ESORICS'09, pages 152–167, Berlin, Heidelberg. Springer-Verlag.

Kerrigan, S. and Law, K. H. (2003). Logic-based regulation compliance-assistance. In *International Conference on Artificial Intelligence and Law*, pages 126–135.

Knuth, D. E. (1964). backus normal form vs. backus naur form. Commun. *ACM*, 7(12):735–736.

Lamanna, D. D., Skene, J., and Emmerich, W. (2003). SLAng: A Language for Defining Service Level Agreements. In *Proceedings of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems*, pages 100–, Washington, DC, USA. IEEE Computer Society.

Legislative Assembly of Ontario (1988). Freedom of information and protection of privacy act (r.s.o. 1990, c. f.31).

Marchiori, M. (2002). The platform for privacy preferences 1.0 (P3P1.0) specification. W3C recommendation, W3C. http://www.w3.org/TR/2002/REC-P3P-20020416/.

Métayer, D. L. (2009). A formal privacy management framework. *Formal Aspects in Security and Trust*, pages 1–15.

OASIS Standard (2013). eXtensible Access Control Markup Language (XACML) Version 3.0. 22 January 2013. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html.

Pearson, S., Tountopoulos, V., Catteddu, D., Südholt, M., Molva, R., Reich, C., Fischer-Hübner, S., Millard, C., Lotz, V., Jaatun, M. G., Leenes, R., Rong, C., and Lopez, J. (2012). *Accountability for cloud and other future internet services*. In CloudCom, pages 629–632. IEEE.

Pearson, S. and Wainwright, N. (2013). An interdisciplinary approach to accountability for future internet service provision. *International Journal of Trust Management in Computing and Communications*, 1(1):52–72.

Schneider, F. B. (2000). Enforceable security policies. ACM Transactions on Information and System Security, 3(1):30–50.

Sundareswaran, S., Squicciarini, A., and Lin, D. (2012). Ensuring distributed accountability for data sharing in the cloud. *Dependable and Secure Computing, IEEE Transactions on*, 9(4):556–568.

US Congress (1999). Gramm-leach-bliley act, financial privacy rule. 15 usc 6801-6809. http://www.law.cornell.edu/uscode/usc_sup_01_15_10_94_20_I.html.

US Congress (2002). Health insurance portability and accountability act of 1996, privacy rule. 45 cfr 164. http://www.access.gpo. gov/-nara/cfr/waisidx_07/45cfr164_07.html.

Wei, W., Du, J., Yu, T., and Gu, X. (2009). Securemr: A service integrity assurance framework for mapreduce. In *Proceedings of the 2009 Annual Computer Security Applications Conference*, pages 73–82, Washington, DC, USA. IEEE Computer Society.

Weitzner, D. J., Abelson, H., Berners-Lee, T., Feigenbaum, J., Hendler, J., and Sussman, G. J. (2008). Information accountability. Commun. *ACM*, 51(6):82–87.

Zhifeng Xiao, Nandhakumar Kathiresshan, Y. X. (2012). A survey of accountability in computer networks and distributed systems. Security and Communication Networks, 5(10):1083–1085.

Zou, J., Wang, Y., and Lin, K.-J. (2010). A formal service contract model for accountable SaaS and cloud services. In *International Conference on Services Computing*, pages 73–80. IEEE.

Context-aware Cloud Application Management

Uwe Breitenbücher, Tobias Binz, Oliver Kopp, Frank Leymann and Matthias Wieland
Institute of Architecture of Application Systems, University of Stuttgart, Stuttgart, Germany
{*breitenbuecher, lastname*}*@iaas.uni-stuttgart.de*

Keywords: Application Management, Context, Automation, Cloud Computing.

Abstract: The automation of application management is one of the most important issues in Cloud Computing. However, the steadily increasing number of different services and software components employed in composite Cloud applications leads to a higher risk of unexpected side effects when different technologies work together that bring their own proprietary management APIs. Due to unknown dependencies and the increasing diversity and heterogeneity of employed technologies, even small management tasks on single components may compromise the whole application functionality for reasons that are neither expected nor obvious to non-experts. In this paper, we tackle these issues by introducing a method that enables detecting and correcting unintended effects of management tasks in advance by analyzing the context in which tasks are executed. We validate the method practically and show how context-aware expert management knowledge can be applied fully automatically to running Cloud applications.

1 INTRODUCTION

In recent years, Cloud Computing gained a lot of attention due to its economical and technical benefits. Especially properties such as pay-on-demand computing, self-service, and elasticity enable enterprises to outsource IT (Leymann, 2009). These properties are key success factors for both Cloud providers as well as customers. To exploit these properties reliably for their offerings, Cloud providers have to automate their internal processes for application provisioning, configuration, and management. Therefore, a lot of tools, methods, and technologies have been developed in the past years: Script-centric DevOps communities support automated configuration management, guidelines such as the Information Technology Infrastructure Library (ITIL) (Malone et al., 2008) help applying best practices to IT service management, and standards such as TOSCA (OASIS, 2013) enable application developers to describe applications and their management in a portable way—to name a few examples. However, automating management tasks on complex Cloud applications becomes a more and more difficult challenge: The increasing number and diversity of employed Cloud services on various layers offered by different Cloud providers leads to a higher risk of unexpected side effects when they are combined with each other or integrated with traditional software components and middleware systems. Additional complexity arises from the heterogeneity of management technologies: Cloud providers, middleware systems, and software components often provide proprietary management APIs, security mechanisms, and data formats (Breitenbücher et al., 2013b). Thus, if a task affects multiple different parts of an application, also the management technologies of these parts may need to be integrated. This requires (i) a deep technical insight in each part and management technology as well as (ii) an overall understanding of the whole system and possible impacts to avoid errors. In these scenarios, often only experts of the respective technology are able to execute management tasks correctly. However, also experts reach their limits when a management task has to be executed on a complex application whose exact structure and runtime information are not documented: Unknown relations and dependencies between components that directly influence each other's functionality lead to a serious management challenge. Even if only a small operation has to be executed on a single component, its impact on other components is not predictable seriously if the application's structure is not known exactly. Thus, if the *context*, in which a management task is executed, is not known, understood, and considered, there is a high risk of unexpected side effects that may compromise the application's functionality. In addition, as executing management task manually is slow, costly, and error prone, Cloud application management must be automated to enable Cloud properties (Fehling et al., 2012; Oppenheimer et al., 2003).

As a result, in case multiple heterogeneous Cloud services, software components, and middleware systems are involved and different management technologies have to be orchestrated, it's of vital importance to (i) apply expert management knowledge, (ii) consider the context in which tasks are executed, and (iii) automate management processes to avoid manual errors.

In this paper, we tackle these issues. We present an approach that enables applying expert management knowledge for tasks in a certain context automatically to running Cloud applications. Therefore, we introduce an abstract (i) *Context-Aware Application Management Method* and (ii) present a fully automated realization of this method for Cloud Computing to validate its practical feasibility. The method introduces *Declarative Management Description Models (DMDM)* to describe management tasks declaratively including their context in a formal model. This enables experts to detect unintended impacts and side effects of management tasks through analyzing them in the context in which they are executed. We show that an individual context analysis is required for many management tasks on Cloud applications due to the heterogeneous nature of the involved components and management technologies—which is not possible having traditional imperative models such as workflows or scripts as these models describe only the task-specific information but not the context in which they are executed. The presented automated realization of the method for Cloud application management validates the method's practical feasibility. It enables Cloud providers to offer a variety of different Cloud applications consisting of heterogeneous components without the need to employ or educate specialized experts that have the required technical management knowledge.

The remainder of this paper is structured as follows: In Section 2, we describe a motivating scenario to explain the method, which is introduced in Section 3. Section 4 presents the automated realization of the method to validate its practical feasibility for Cloud application management. In Section 5 we give an overview on related work. Section 6 concludes the paper and gives an outlook on future work.

2 MOTIVATION

In this section, we describe the problem tackled by our approach in detail and introduce a motivating scenario that is used throughout the paper to explain the presented method and its realization.

2.1 Problem Statement

In this section, we describe the major issues of Cloud application management and why context consideration is of vital importance in this area. Due to different functional as well as non-function requirements on services, Cloud offerings of different providers often have to be integrated as using a single Cloud provider for the complete application landscape of a company is often not possible (Fehling et al., 2012). The focus of this paper lies, therefore, on the management of *Complex Composite Cloud Applications*, which consist of multiple heterogeneous Cloud services, software components, and middleware systems of different vendors and Cloud providers.

The management of such applications is quite difficult as it requires a deep technical understanding of the involved components, the dependencies between them, and their management technologies—which are most often proprietary implementations providing heterogeneous management interfaces (Leonhardt, 2013). As a consequence, human-induced failures account for a significant number of outages as human errors accompany even the simplest system operation and management tasks (Brown and Patterson, 2001). The management flexibility recently introduced by Cloud Computing has additionally increased this effect as management tasks, such as the provisioning of new virtual machines, are literally at the systems manager's fingertips and often handled completely via proprietary management interfaces offered by Cloud providers (Fehling et al., 2012). Due to this management evolution, human errors are more likely to occur since the physical infrastructure and thereon deployed software systems are often virtualized and, therefore, obfuscating the actual structure of an application. Thus, it becomes more and more difficult to consider the *context*, in which management tasks are executed, correctly as application structures, system boundaries, and dependencies are becoming increasingly blurred by the Cloud. In addition, due to the lack of interoperability between Cloud services of different providers and traditional software components, most often complex configuration management is needed to integrate heterogeneous systems. As a result, executing management tasks which affect multiple parts of an application at once requires special management expertise. These problems impede automating Cloud application management, which is a major requirement for efficient use of Cloud resources and to avoid manual human errors (Fehling et al., 2012; Oppenheimer et al., 2003). Therefore, we need a means to capture and apply context-aware management expertise fully automatically to running Cloud applications.

2.2 Management Automation

To automate application management, the execution of management tasks is often described imperatively using executable processes implemented as workflows (Leymann and Roller, 2000) in languages such as BPEL (OASIS, 2007) or BPMN (OMG, 2011), scripts such as bash, or programs implemented in a computer programming language such as Java. Especially the more and more emerging script-centric configuration management technologies such as Chef (Nelson-Smith, 2013) are prime examples for this kind of imperative management descriptions. However, if an application is crucial to the business of an enterprise, errors that possibly result in system downtime are not acceptable. Therefore, before executing such management processes, they must be verified by experts to ensure correctness. Unfortunately, one of the most important problems of this approach is that the context, in which the management tasks are executed, is not explicitly described and, thus, not visible in such processes. As a result, management processes cannot be analyzed by experts in consideration of the management tasks' context as only operation calls, service invocations, or script executions on the *directly* affected components are described, but not the surrounding environment: Experts see only the directly affected part of the application, not the whole application structure. Thus, other application components that may be affected indirectly, too, cannot be considered in this analysis. For example, if the database of a Web-based application shall be replaced by a database from a different vendor, the application's Web server may require a certain database connector to be installed for connecting to the new database. If this dependency is not considered and handled by the management process replacing the database through installing the required new connector, too, the application cannot connect to its database anymore. This quickly results in system downtimes caused by errors that are neither easy to find nor to fix. Thus, the most important requirement to enable context-aware management is a formal model that describes both the tasks and the context.

2.3 Motivating Scenario

In this section, we describe the motivating scenario that is used throughout the paper to explain the proposed method and its realization. The scenario describes a business application that consists of a PHP-based Web frontend and a PostgreSQL database. The frontend should be migrated from one Cloud to another. Because the application evolved over time, it is currently distributed over two Clouds: The PHP frontend is hosted on Microsoft's public Cloud offering "Windows Azure"[1], the PostgreSQL database on Amazon's PaaS offering "Relational Database Service (RDS)"[2], which enables hosting databases directly as a service. The PHP frontend runs on an Apache HTTP Server (including the PHP-module) which is installed on an Ubuntu Linux operating system that runs in a virtual machine hosted on Azure. The management task that has to be executed is migrating the PHP frontend to Amazon's IaaS offering "Elastic Compute Cloud (EC2)"[3] in order to reduce the number of employed Cloud providers. This migration results in two issues that compromise the application's functionality if they are not considered by experts having the knowledge to recognize the following two problems in advance: (i) missing database driver and (ii) missing configuration of the database service. To migrate the PHP frontend, we have to create a new virtual machine on Amazon EC2, install the Apache HTTP Server and the PHP-module, and deploy the corresponding PHP files on it. This works without further configuration issues. However, connecting the PHP application to the database is not as easy as it seems to be: Simply defining the database configuration of the PHP frontend by setting the database's endpoint, username, and password is not sufficient. Here, a technical detail of the underlying infrastructure needs to be considered: The PHP-module of the Apache HTTP Server needs different database drivers to connect to different types of databases. Thus, if the PostgreSQL driver gets not installed explicitly on the server, the Moodle PHP frontend is not able to connect to the database. However, this is not easy to recognize as applications often employ MySQL databases whose drivers are typically installed together with the PHP-module. Thus, installing the required driver for PostgreSQL might be forgotten. The second issue is even more difficult to foresee if the administrator is not an expert on Amazon RDS: Databases running on Amazon RDS are per default not accessible from external components. To allow connections, a so-called "Security Group" must be defined to configure the firewall. This group specifies the IP-addresses which are allowed to connect to the database. Both issues result in breaking the application's functionality as the frontend cannot connect to the database. The reason for both problems lies in ignoring the context in which the tasks are executed: First, if an application shall connect to a certain type of database, the runtime environment hosting the

[1] http://www.windowsazure.com/

[2] http://aws.amazon.com/rds/

[3] http://aws.amazon.com/ec2/

application must support connecting to this kind of database. It is not enough to simply set the configuration as the underlying infrastructure, i. e., the context, needs to be considered, too. Second, accessing a database hosted on Amazon RDS requires also more than simply writing endpoint information into a configuration file as also the firewall of the service has to be configured. Thus, also for this task, the context in the form of the infrastructure that hosts the database has to be considered to recognize the problem.

However, both problems cannot be detected by experts if the migration is implemented using traditional approaches such as management workflows or scripts: The processes would only model the steps for (i) shutting down the old virtual machine on Azure, (ii) creating the new virtual machine on Amazon EC2, (iii) installing the Apache Web Server and the PHP-module, (iv) deploying the frontend, and (v) setting the database's IP-address, username, and password in the frontend's configuration. The process neither provides information about the database's type nor which infrastructure is used to host the database—the context is not described. In addition, if administrators execute this migration manually, they require a lot of technical expertise and background information about the different APIs and employed technologies. Thus, we need a means to (i) capture expert management knowledge for a certain context and (ii) make it applicable fully automatically in order to help administrators avoiding unintended mistakes.

3 CONTEXT-AWARE MANAGEMENT METHOD

In this section, we introduce the *Context-Aware Application Management Method* that provides a means to consider the context in which management tasks on application components or relationships are executed. The method separates between a declarative description of the management tasks to be executed and the final executable management process. It consists of six steps which are shown in Figure 1: First, (1) we capture all runtime information and the structure of the application to be managed in a formal model which is extended in the second step (2) by declaring tasks on components and relations. In the third step, (3) experts analyze this model for unintended side effects and (4) adapt the model afterwards if necessary to resolve the found problems. In the fifth step, (5) this declarative model is translated into an imperative process model that is executed in the last step (6) to perform the tasks on the running application. We explain each step in the following subsections and start each description with a short summary of the step in italic text to provide a compact overview that is detailed afterwards.

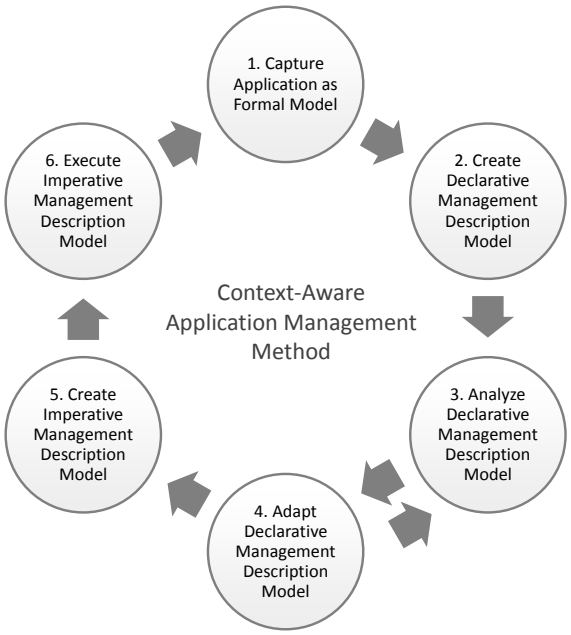


Figure 1: Context-Aware Application Management Method.

3.1 Step 1: Capture Application as Formal Model

Describe all components of the application and their relations in a formal model that specifies their types and current runtime information.

First, the application to be managed is described as a formal model. This model captures the application structure and its state, i. e., all *components* such as Web servers, virtual machines, or installed applications, the *relations* between them, e. g., database connections, and their *runtime information*. The semantics of components and relations (we call both *model elements* in the following) are described using types, e. g., a component may be of type "ApacheHTTPServer", a relationship of type "SQLConnection". To enable a precise definition of the model elements, types can be inherited: the "ApacheHTTPServer" type is a subtype of "HTTPServer". Runtime information are described as model element properties, e. g., the "ApacheHTTPServer" has properties "IP-Address" and "Port" that specify the server's endpoint. The property schema is defined by the type of the model element. This formalization of the running applica-

tion provides a detailed, structured, and machine readable means to document a current snapshot of the application structure and all runtime information.

3.2 Step 2: Create Declarative Management Description Model

Create a Declarative Management Description Model that declaratively describes the management tasks to be executed on components and relations.

In the second step, the desired management tasks are described based on the formal model. Therefore, we introduce the *Declarative Management Description Model (DMDM)* that extends the formal model captured in Step 1 by a declarative description of the management tasks to be executed on components and relations of the application. This model declares management tasks in an abstract manner without technical implementation details and specifies the target component or relation of each task. A DMDM is not executable as it describes only *what* has to be done, but not *how*—all technical details are missing. For example, a DMDM may declare a "Create" task on a relation of type "SQLConnection" between a PHP application and an SQL database, which means that the connection has to be established. However, it provides neither technical implementation details nor specifies the control flow between multiple different tasks.

3.3 Step 3: Analyze Declarative Management Description Model

Analyze the impact of tasks in consideration of the context defined by the components, relations, and runtime information described by the DMDM.

The DMDM created in the previous step captures a snapshot of the application and the abstract management tasks to be executed. The model describes the whole *context* in which tasks are executed by modelling all components and adjacent relations of the application that are possibly affected. In the third step, management tasks are analyzed in their context by experts of different domains to detect unexpected impacts leading to unintended side effects. DMDMs enable cooperation between different experts and separate concerns based on a uniform, structured, and formal model: Experts on the "ApacheHTTPServer" are able to detect that the installation of a certain database connector is required, experts on the Amazon Cloud are able to configure the Security Group in order to allow connections from the external PHP frontend of the application. Thus, DMDMs can be analyzed by multiple experts of different domains in a cooperative manner.

3.4 Step 4: Adapt Declarative Management Description Model

Adapt the analyzed Declarative Management Description Model if necessary to solve the analyzed problems.

After the expert analysis, found problems have to be resolved in order to achieve the desired management goals. Therefore, the DMDM is adapted in this step by the respective experts to enable correct execution of the management tasks. Therefore, components, relations, and tasks of the DMDM may be added or reconfigured. For example, the missing database connector found in the analysis of the previous step is resolved by adding the task to install the required connector on the Web Server. Thus, each task was verified in its respective context in the previous step and gets corrected if necessary in this step. However, if tasks are added or reconfigured, all tasks have to be analyzed again for correctness as the context changed by this adaptation. This may lead to new problems and unintended side effects on other components or relations that have to be found. Therefore, Step 3 and Step 4 are repeated until no new problems are found and all tasks were considered in the final context. This ensures that also the adaptations are checked by management experts.

3.5 Step 5: Create Imperative Management Description Model

Create an Imperative Management Description Model in the form of an executable process to perform the management tasks declared in the DMDM.

The verified and adapted Declarative Management Description Model resulting from the previous step describes the tasks to be performed declaratively in an abstract manner—only *what* management tasks have to be performed, but not *how*. Thus, the model is not executable as the technical realization is not described. Therefore, an executable process model that implements the management tasks declared in the DMDM must be created. As this process model describes how the tasks have to be executed *imperatively*, we call these management processes *Imperative Management Description Models (IMDM)*. An IMDM can be executed using an appropriate process engine and describes also the control flow and data

handling between the management tasks. The IMDM has to implement exactly the semantics of the management tasks described by the adapted Declarative Management Description Model resulting from the previous step.

3.6 Step 6: Execute Imperative Management Description Model

Execute the created Imperative Management Description Model to manage the running application.

In the last step, the IMDM is executed to perform the desired management tasks on the real running application. Therefore, a process engine is employed to run the process. As a result, the changes described by the tasks are applied to the running application in consideration of the context. Afterwards, the method may start from the beginning to execute further tasks.

4 VALIDATION

The presented method enables combining *declarative* management descriptions, which include all relevant context information to verify the management tasks, and *imperative* processes, which are employed to actually perform the tasks on running applications. Thus, it combines two different types of Management Description Models which enables benefiting from advantages of both worlds. Therefore, the presented method provides a *theoretic* basis for enabling automated context-aware application management. In this section, we validate the proposed method by showing a fully automated *practical* implementation using existing frameworks. We describe the realization of the prototype for all steps of the method in the following.

4.1 Formalizing Applications Using Enterprise Topology Graphs

In Step 1, the application's structure and runtime information needs to be captured in a formal model. We use *Enterprise Topology Graphs (ETG)* (Binz et al., 2012) as model language as they are a common way to capture such information formally. ETGs are directed, possibly acyclic, graphs that describe the application's structure as topology model that contains each component as typed node and each relationship as typed edge between the nodes. Runtime information of nodes and relations are captured as properties of the respective element. Thus, ETGs can be used to model the context in which a management task is executed. As ETGs support the XML-format, they are machine readable. Figure 2 shows the motivating scenario as ETG example, graphically rendered using the visual topology notation "Vino4TOSCA" (Breitenbücher et al., 2012). Each component is depicted as rounded rectangle, relations as arrows between these rectangles, and runtime information as key-value properties. Element IDs are undecorated text, element types are enclosed by parentheses. Binz et al. showed that ETGs of running applications can be discovered fully automatically using the *ETG Discovery Framework* (Binz et al., 2013). Thus, the first step of formalizing the application to be managed can be automated by using this framework.

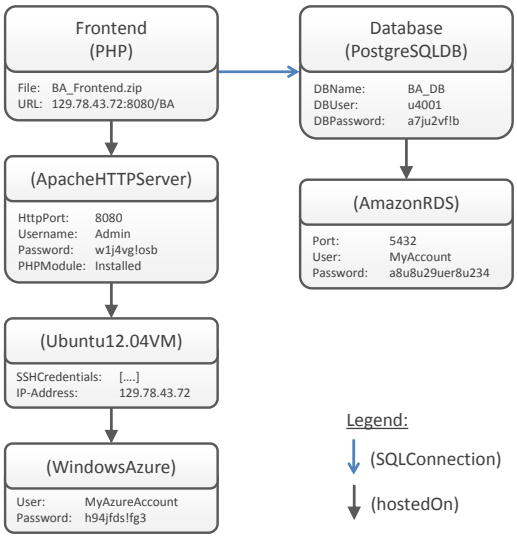


Figure 2: ETG of the running application.

4.2 Automating DMDM Creation Using Automated Management Patterns

Capturing running application snapshots in such formal ETG models provides a structured means to describe the context in which a management task is executed. Therefore, to create the DMDM in the second step, we use the discovered ETG and annotate the management tasks to be executed directly at the affected components and relations of the ETG. In Breitenbücher et al. (Breitenbücher et al., 2013a), we introduced so-called *Desired Application State Models*, which provide exactly this type of model for describing tasks to be executed declaratively in the context in which they have to be executed based on ETGs.

Figure 3 shows the Desired Application State Model that describes our migration motivating scenario. The colored circles with the symbols inside represent the management tasks to be executed in the

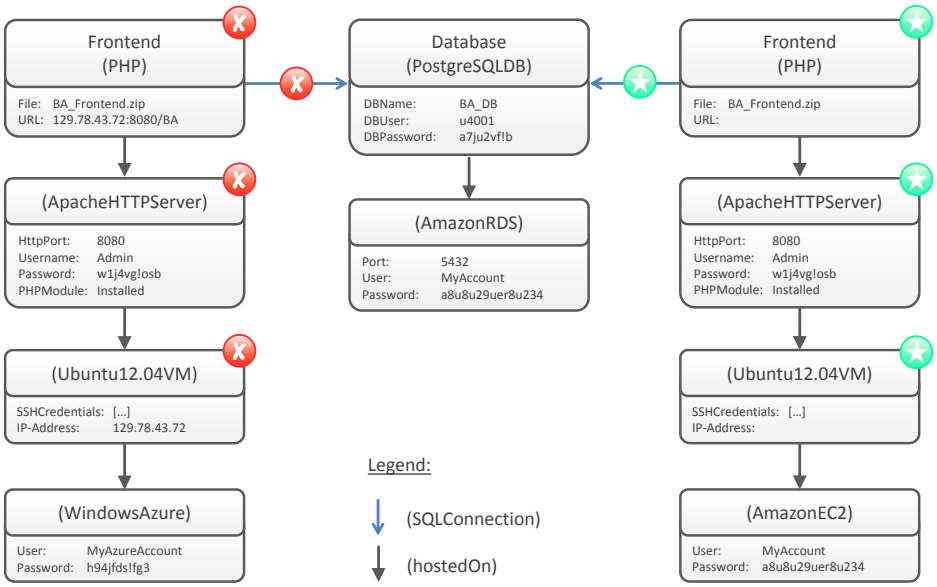


Figure 3: Desired Application State Model resulting from applying the Automated Management Pattern to the discovered ETG.

form of so-called *Management Annotations* (Breitenbücher et al., 2013a). A Management Annotation describes a task to be performed in a declarative way: It defines only the type of the task and possible configuration properties, but not how to execute it. The green colored "Create-Annotations" with the star inside declare that the corresponding element it is attached to has to be created, while the red colored "Destroy-Annotations" with the "x" inside declare that the model element has to be destroyed. Management Annotations can be also bound declaratively to non-functional requirements in the form of policies that must be fulfilled when executing the task (Breitenbücher et al., 2013c).

Annotating management tasks to ETGs, i. e., creating a DMDM, can be automated, too: Desired Application State Models can be created automatically by applying so-called *Automated Management Patterns* to ETGs (Breitenbücher et al., 2013a). An Automated Management Pattern captures management expertise through implementing a transformation that annotates management tasks in the form of Management Annotations fully automatically to an input ETG. This transformation attaches, configures, or removes nodes, relations, and Management Annotations. In addition, Automated Management Patterns can be configured via input parameters. For example, the Desired Application State Model shown in Figure 3 is the result of applying the "Migrate PHP-based Application to Amazon Pattern", which was configured to use Amazon's IaaS offering EC2 for hosting the PHP frontend. The only manual step is selecting and configuring the pattern. Thus, Step 2 can be automated, too.

4.3 Context-aware Task Analyzer

After the Desired Application State Model was created automatically by applying an Automated Management Pattern, it has to be analyzed by experts in Step 3 and adapted if necessary in Step 4. As we aim at automating the whole method realization, also these two steps need to be automated. Therefore, we introduce the concept of *Context-Aware Management Task Analyzers (CAMTA)* that provides a means to capture context-aware expert management knowledge in a form that enables a fully automated application to the Desired Application State Model resulting out of the previous step. The notion of CAMTAs is detecting and correcting problems by analyzing the tasks in their context and adapting the model if necessary fully automatically without manual interaction. Therefore, a CAMTA consists of two parts: (i) An *Annotated Topology Fragment* and a (ii) *Transformation*, similarly to Automated Management Patterns. The Annotated Topology Fragment is a small topology that specifies the management tasks in a certain context for which the CAMTA is able to (i) analyze correctness and (ii) may provide expert management knowledge required to adapt the model if necessary. The fragment is used for matchmaking of CAMTAs and Desired State Models: If all elements in a CAMTA's

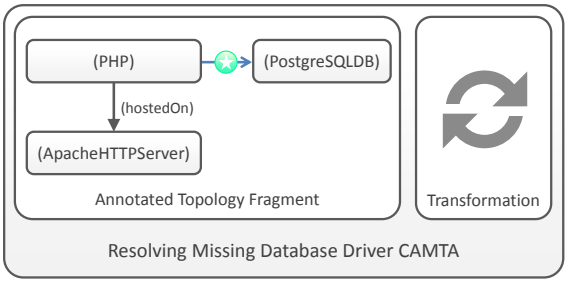


Figure 4: CAMTA that recognizes the problem of missing PostgreSQL database connector driver.

fragment match elements in the model, the Context-Aware Management Task Analyzer is able to analyze exactly that part. Thus, the Annotated Topology Fragment is used to select the CAMTAs that have to be applied to analyze the DMDM in Step 3 fully automatically. For adapting the model in Step 4, each CAMTA implements a context-aware transformation that transforms the input Desired State Model fully automatically if necessary. Therefore, the transformation checks if the tasks specified in the CAMTA's Topology Fragment can be executed safely: If yes, the transformation returns the unmodified model. If not, the transformation adds or configures components, relationships, or tasks for correcting the Desired State Model. Figure 4 shows a CAMTA that analyzes the tasks of establishing a SQL connection from a PHP application hosted on an Apache HTTP Server to a PostgreSQL database. The shown CAMTA is able to analyze if establishing a SQL-Connection in the context of a PHP Application running on the Apache HTTP Server to a PostgreSQL database is possible. This is expressed by its Topology Fragment on the left. The transformation shown on the right analyzes the Desired State Model and finds out that the PostgreSQL connector driver is missing. Therefore, it adds the corresponding elements and tasks to the model. Thus, based on two CAMTAs, the Desired State Model, which results from applying the automated migration pattern in Step 2, gets adapted fully automatically for resolving the issues of the missing database connector and Security Group configuration. Therefore, the respective CAMTAs insert two different Management Annotations into the Desired State Model: (i) a "ConfigureSecurityGroup-Annotation" that is attached to the AmazonRDS node and an "InstallDriver-Annotation" attached to the Apache HTTP Server node. The ConfigureSecurityGroup-Annotation configures the AmazonRDS instance in a way that the database is accessible by the Apache HTTP Server, the InstallDriver-Annotation declares that the required connector for PostgreSQL databases must be installed.

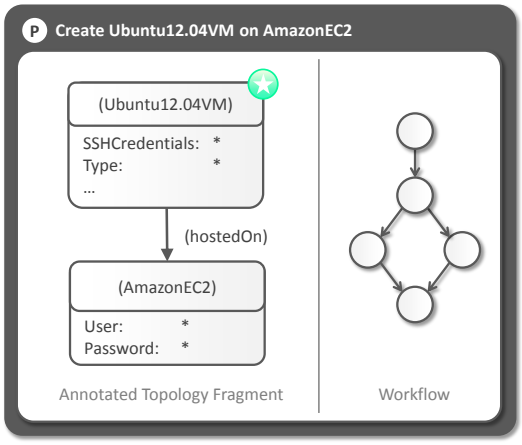


Figure 5: Management Planlet that creates an Ubuntu12.04 virtual machine on Amazon EC2.

As Desired State Models typically specify multiple management tasks to be executed in the form of Management Annotations that need to be analyzed in their context, multiple different CAMTAs are needed to check the correctness of the whole model. As they may add or configure elements, all CAMTAs need to be applied every time after one transformed the model to ensure that all tasks are validated in the current context that may be changed by formerly applied CAMTAs. As soon as input and output model do not change anymore after applying all matching CAMTAs, Step 4 is finished. The approach is similar to Automated Management Patterns (Breitenbücher et al., 2013a) that also use transformations to modify models.

4.4 Management Plan Generation

After the DMDM was analyzed and adapted for correctness in Step 3, the resulting model is not executable directly as it describes the tasks to be performed only declaratively, i. e., without implementation or control flow: The Declarative Management Description Model has to be transformed into an executable Imperative Management Description Model in Step 5. Therefore, we employ the management framework presented by Breitenbücher et al. (Breitenbücher et al., 2013a) that employs *Management Planlets* to translate Desired Application State Models fully automatically into executable BPEL workflows. Management Planlets provide the low-level *imperative* management logic to execute the *declarative* Management Annotations used in Desired Application State Models and support defining functional as well as non-functional requirements (Breitenbücher et al., 2013c). They serve as generic man-

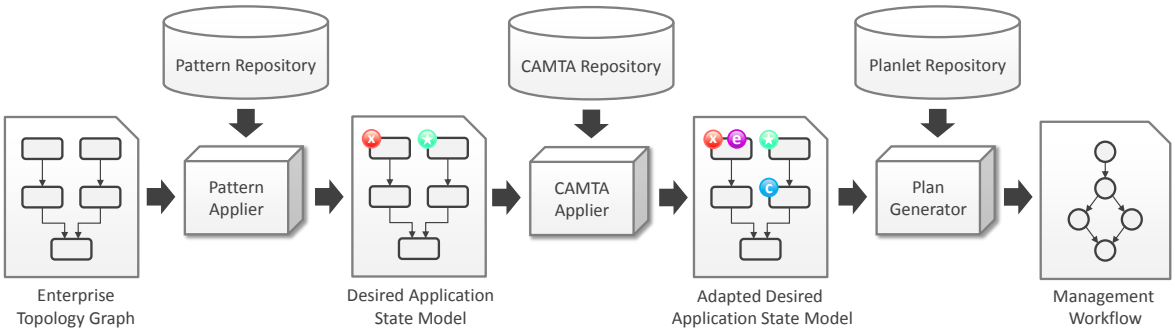


Figure 6: Conceptual Architecture of the method's realization.

agement building blocks that can be orchestrated to implement a higher-level management task. A Management Planlet consists of two parts: (i) *Annotated Topology Fragment* and (ii) a *workflow*. The fragment exposes the Planlet's functionality and is used to find Planlets that are capable of executing the specified management tasks in the specified context. For example, the Planlet shown in Figure 5 is capable of executing the Create-Annotation attached to an Ubuntu12.04VM node if this node has to be hosted on AmazonEC2. The Planlet's executable workflow implements exactly the management logic required to create this virtual machine on EC2. Based on these fragments, different Planlets can be orchestrated to implement an overall management plan that performs all management tasks defined in the Desired Application State Model. Therefore, the framework employs a Plan Generator that transforms Desired Application State Models into executable management workflows. The whole method realization is shown in Figure 6: A discovered ETG is transformed into a Desired Application State Model by applying an Automated Management Pattern. This model is analyzed and possibly adapted by several CAMTAs. The Plan Generator gets the resulting model as input, searches Planlets that are able to execute the specified Management Annotations, and orchestrates the matching Planlets' workflows to an overall management workflow. This workflow implements all management tasks declared by the Desired Application State Model and can be executed in Step 6 to perform the tasks on the real running application.

4.5 Evaluation of the Realization

The introduced declarative model layer describing tasks in its formal context that encompasses all application components, their relations, and runtime information enables experts to detect dependencies that are not visible in traditional imperative management processes such as workflows. The reason is that these traditional processes describe only logic for the directly affected components, not the surrounding environment context: Details of the application structure, i. e., the context, are not contained in such process models and can, thus, not be analyzed. However, especially workflow technology and script-centric technologies such as Chef are well-established technologies used in systems management due to the provided features and properties: Workflow technology provides features, such as recoverability and compensation mechanisms, that enable correct and complete execution of long running processes, which is often a major requirement (Leymann and Roller, 2000). On the other side, the script-centric DevOps communities provide a lot of templates and implementations that can be used for configuration management without further modifications. The method enables using these technologies for the actual execution of the management tasks in the form of IMDMs. Thus, it enables combining both types of management description models to support both consideration of context and profiting from features provided by common technologies.

The presented realization is limited in terms of completeness: Management tasks in a Desired Application State Model are analyzed and possibly adapted only if there are CAMTAs that provide the required knowledge for analyzing and adapting the tasks in the actual context. The approach could be extended by marking all verified management tasks, i. e., all Management Annotations that were considered by one or more CAMTAs. In addition, a description of the task analysis could be added by CAMTAs to provide information about the analyzed issues and determined problems. This may help users to understand the performed analysis and enables drawing their attention to the tasks that were not considered by the system. These tasks could be then analyzed manually by experts before generating the final management workflow.

5 RELATED WORK

Context-aware systems adapt their functionality and behaviour using context information about the environment. An often used definition for context was given by Dey (Dey et al., 2000): "Context is any information that can be used to characterize the situation of an entity, where an entity can be a person, place, physical or computational object". An important type of context information, which is often neglected, is the state and structure of an application to be managed. In this paper, we use this type of context information to verify, configure, and execute management tasks on applications and their infrastructure. The automated realization of the presented management method provides, therefore, the basis to implement Context-aware Cloud Application Management Systems.

To model and manage context information, many frameworks have been developed in the past years. There are simple, widget-like frameworks for sensor information such as the Context Toolkit (Dey et al., 2001) and systems that support smart environments like Aura (Judd and Steenkiste, 2003) or Gaia (Roman and Campbell, 2000). Different types of development frameworks, e. g., the framework of Henricksen and Indulska (Henricksen and Indulska, 2004), and context management platforms, e. g., the Nexus Platform (Großmann et al., 2005), were developed that aim at efficient provisioning of context information within a global scope. These frameworks use *Context Models* as an abstraction layer between applications and the technical infrastructure that gathers the context data. However, there is no framework that manages context information for application management in the form of topology models. The Declarative Management Description Model introduced in this paper provides a kind of Context Model that (i) enables to capture the environment in which management tasks are executed and (ii) the management tasks themselves described in a declarative fashion. The context is captured in a domain-specific data structure in the form of ETGs. Furthermore, no sensors integration has to be achieved because the context is detected on the fly using the ETG Discovery Framework (Binz et al., 2013). Thus, the context is always up to date and does not have to be stored or managed using additional tooling.

There are several approaches that enable describing application topologies including runtime information and dependencies. Scheibenberger and Pansa (Scheibenberger and Pansa, 2008) present a generic meta model to describe resource dependencies among IT infrastructure components. They separate the static view, which captures functional and structural aspects, from the dynamic operational view, which captures runtime information. In contrast to the employed concept of ETGs in the validation, their approach enables to model dependencies between component properties. The method's realization may be extended to capture also such fine-grained dependencies if necessary that may help experts to analyze possible impacts of a certain management task. The Common Information Model (CIM) (Distributed Management Task Force, 2010) is a standard that provides an extensible, object-oriented data model used to capture information about different parts of an enterprise. It also provides a specification to describe application structures including dependencies. However, all these works may be used to formalize the application structure, dependencies, and runtime information, but they provide no means to model also the management tasks to be executed as required to implement a DMDM.

There are several frameworks that employ declarative descriptions to generate workflows such as Eilam et al. (Eilam et al., 2011), Maghraoui et al. (Maghraoui et al., 2006), and Keller et al. (Keller et al., 2004). The first two focus mainly on provisioning of applications whereas the third also considers application management. In general, the proposed method can be applied to approaches and frameworks that transform declarative descriptions into imperative processes. However, it must be ensured that the declarative descriptions (i) provide the whole context and (ii) that the management tasks to be executed are described by this model somehow. In a former work (Breitenbücher et al., 2014), we showed how declarative provisioning descriptions can be transformed automatically into imperative provisioning workflows based on the TOSCA standard (OASIS, 2013). The application to be provisioned is described as a topology, which models all components and relations of the application. As the tasks to be executed are clear (create each application component and instantiate the relations between the components) and the whole context of the provisioning is provided by this model in the form of the application topology, the presented method can be applied to this standards-based provisioning approach, too.

6 CONCLUSIONS

In this paper, we introduced an abstract Context-Aware Application Management Method that enables applying context-aware management expertise to running Cloud applications. We showed that separat-

ing models for context-aware analysis and management execution provides a powerful means to benefit from advantages of both worlds. Therefore, we employed abstract Declarative Management Description Models for describing the context as well as the management tasks to be executed themselves that are transformed into Imperative Management Description Models. The presented method is validated by an automated prototypical realization for Cloud Application Management using existing frameworks and technologies. In future, we plan to integrate non-functional requirements into the method and its realization and to apply both to the OASIS standard TOSCA.

ACKNOWLEDGEMENTS

This work was partially funded by the BMWi project CloudCycle (01MD11023).

REFERENCES

Binz, T., Breitenbücher, U., Kopp, O., and Leymann, F. (2013). Automated Discovery and Maintenance of Enterprise Topology Graphs. In *SOCA*, pages 126–134. IEEE.

Binz, T., Fehling, C., Leymann, F., Nowak, A., and Schumm, D. (2012). Formalizing the Cloud through Enterprise Topology Graphs. In *CLOUD*, pages 742–749. IEEE.

Breitenbücher, U., Binz, T., Képes, K., Kopp, O., Leymann, F., and Wettinger, J. (2014). Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA. In *IC2E*. IEEE.

Breitenbücher, U., Binz, T., Kopp, O., and Leymann, F. (2013a). Pattern-based Runtime Management of Composite Cloud Applications. In *CLOSER*. SciTePress Digital Library.

Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., and Wettinger, J. (2013b). Integrated cloud application provisioning: Interconnecting service-centric and script-centric management technologies. In *CoopIS*, pages 130–148. Springer.

Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., and Wieland, M. (2013c). Policy-Aware Provisioning of Cloud Applications. In *SECURWARE*, pages 86–95. Xpert Publishing Services.

Breitenbücher, U. et al. (2012). Vino4TOSCA: A Visual Notation for Application Topologies based on TOSCA. In *CoopIS*, pages 416–424. Springer.

Brown, A. B. and Patterson, D. A. (2001). To err is human. In *EASY*, page 5.

Dey, A. K., Abowd, G., and Salber, D. (2000). A context-based infrastructure for smart environments. In *Managing Interactions in Smart Environments*, pages 114–128. Springer.

Dey, A. K., Abowd, G. D., and Salber, D. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Hum.-Comput. Interact.*, 16:97–166.

Distributed Management Task Force (2010). Common Information Model.

Eilam, T. et al. (2011). Pattern-based composite application deployment. In *Integrated Network Management*. IEEE.

Fehling, C., Leymann, F., Rütschlin, J., and Schumm, D. (2012). Pattern-based development and management of cloud applications. *Future Internet*, 4:110–141.

Großmann, M., Bauer, M., Hönle, N., Käppeler, U.-P., Nicklas, D., and Schwarz, T. (2005). Efficiently Managing Context Information for Large-Scale Scenarios. In *PerCom*. IEEE.

Henricksen, K. and Indulska, J. (2004). A Software Engineering Framework for Context-Aware Pervasive Computing. In *PerCom*. IEEE.

Judd, G. and Steenkiste, P. (2003). Providing contextual information to pervasive computing applications. In *PerCom*. IEEE.

Keller, A., Hellerstein, J. L., Wolf, J. L., Wu, K.-L., and Krishnan, V. (2004). The champs system: change management with planning and scheduling. In *NOMS*, pages 395–408. IEEE.

Leonhardt, S. (2013). A generic artifact-driven approach for provisioning, configuring, and managing infrastructure resources in the cloud. Diploma thesis, University of Stuttgart, Germany.

Leymann, F. (2009). Cloud Computing: The Next Revolution in IT. In *The Photogrammetric Record*, pages 3–12.

Leymann, F. and Roller, D. (2000). *Production workflow: concepts and techniques*. Prentice Hall PTR.

Maghraoui, K. E. et al. (2006). Model driven provisioning: Bridging the gap between declarative object models and procedural provisioning tools. In *Middleware*, pages 404–423. Springer.

Malone, T., Blokdijk, G., and Wedemeyer, M. (2008). *ITIL V3 Foundation Complete Certification Kit*. Art of Service Pty Limited.

Nelson-Smith, S. (2013). *Test-Driven Infrastructure with Chef*. O'Reilly Media, Inc.

OASIS (2007). Web services business process execution language (WS-BPEL) version 2.0.

OASIS (2013). Topology and Orchestration Specification for Cloud Applications Version 1.0.

OMG (2011). Business Process Model and Notation (BPMN), Version 2.0.

Oppenheimer, D., Ganapathi, A., and Patterson, D. A. (2003). Why do internet services fail, and what can be done about it? In *USITS*. USENIX Association.

Roman, M. and Campbell, R. H. (2000). Gaia: Enabling active spaces. In *SIGOPS*, pages 229–234. ACM.

Scheibenberger, K. and Pansa, I. (2008). Modelling dependencies of it infrastructure elements. In *BDIM*, pages 112–113. IEEE.

SHORT PAPERS

Evaluating the Impact of Virtualization on Performance and Power Dissipation

Francisco J. Clemente-Castelló, Sonia Cervera, Rafael Mayo and Enrique S. Quintana-Ortí
Department of Engineering and Computer Science, Jaume I University (UJI), Castellón, Spain
{*fclement, cerveras, mayo, quintana*}*@uji.es*

Keywords: Virtualization, Cloud Computing, KVM, Energy Consumption, Server Consolidation.

Abstract: In this paper we assess the impact of virtualization in both performance-oriented environments, like high performance computing facilities, and throughput-oriented systems, like data processing centers, e.g., for web search and data serving. In particular, our work-in-progress analyzes the power consumption required to dynamically migrate virtual machines at runtime, a technique that is crucial to consolidate underutilized servers, reducing energy costs while maintaining service level agreements. Preliminary experimental results are reported for two different applications, using the KVM virtualization solution for Linux, on an Intel Xeon-based cluster.

1 INTRODUCTION

Cloud computing is now a mainstream approach to accelerate application deployment and execution, with lower maintenance and total costs of ownership (TCO), increased manageability, and higher flexibility to rapidly adjust resources to a fluctuating demand. Cloud computing was originally conceived for applications and setups very different from those usually found in high-performance scientific computing. However, due to the recent advances in virtualization design and its smooth unfolding, cloud solutions are being also adopted for resource management in the high performance computing (HPC) arena.

Virtualization is the essential technology underlying cloud computing, which has resulted in the renaissance of a research line that was initiated in the 70s (Goldberg, 1974). Two key aspects that have further nourished the spur of virtualization are the development of production-level tools (e.g., KVM (Kivity et al., 2007) and Xen (Barham et al., 2003)) and the accommodation of hardware to support this technology (hardware-assisted virtualization) in a significant fraction of current processors from large architecture companies such as AMD, IBM or Intel.

Most virtualization efforts address the consolidation of underutilized dedicated physical servers and the implementation of tools for cloud computing. On the other hand, research & development projects that adopt virtualization as a tool for HPC are significantly more scarce. While there exist sound reasons for these primary studies, a needful prerequisite before embracing virtualization in HPC environments is an analysis of the impact of this technology on performance.

In this paper we present an experimental study of a virtual machine (VM) hypervisor, from the points of view of performance, productivity and, especially, power consumption. Specifically, our study examines the problems arising from live relocation of VMs, inspecting the actual effect on power of such strategy, for two well-known applications running on a three-node Intel Xeon-based cluster. For our study, we selected KVM (Kernel-based Virtual Machine), due to its low overhead, fair adoption, and its integration with the open source cloud computing platform OpenStack (Ope, 2013).

The rest of the paper is structured as follows. In section 2 we review several basic concepts, advantages and benefits of virtualization, and its connection with energy saving strategies via consolidation. In section 3 we briefly discuss KVM, the technology employed in our experimental evaluation. The main contribution of this paper is in section 4, where we present the experimental setup and preliminary results from our work-in-progress. Finally, in section 5 we close the paper with a few concluding remarks and a discussion of future work.

2 VIRTUALIZATION AND CONSOLIDATION

System (or hardware) virtualization creates an abstraction layer on top of the physical machine (host) which allows one or more VMs (guests) to run separately from the underlying hardware resources. Current virtualization solutions rely on the concept of virtual machine monitor (VMM), or *hypervisor*, that is in charge of virtualizing the hardware and executing the VMs, acting as a firewall between these two components.

There exist two alternatives for virtualization (Goldberg, 1974). In one of these, the hypervisors interact directly with the hardware. The hypervisor is in this case a kernel-mode driver (or module) of the host operating system (OS). In the alternative one, both the hypervisor and the VMs execute on a top of a standard OS which offers access to the host resources, including its devices. The advantage of the first option is clear: since it provides direct access to the hardware, avoiding interaction through multiple software layers, in principle its peak performance can be close to that attained from a native execution. Xen, KVM and VMWare (Nieh and Leonard, 2007) are examples of the first type of virtualization, while QEMU (qem, 2013) and VirtualBox (vir, 2013) follow the nonnative alternative.

There are several important aspects that have to be considered when applying virtualization (Younge et al., 2011; Younge et al., 2010), independently of whether the target is a data processing center, where throughput is the fundamental driving force, or an HPC facility, which may be willing to trade-off workload productivity for application performance. One particular aspect asks for an assessment of the balance between the costs (i.e., negative performance impact) and the benefits of accommodating virtualization. Fortunately, many processor architectures nowadays feature hardware support to amend possible penalties resulting from virtualization. Furthermore, there is a continuous research to address these overheads also from the software viewpoint (e.g., paravirtualization that offers virtualized memory addresses; previrtualization software to adopt a certain hypervisor while, simultaneously, maintaining compatibility with the physical hardware; etc.)

On the other hand, applications can also benefit from embracing virtualization. For example, an OS can be tuned to improve application performance by letting the hypervisor control allocation of resources among applications such as a fixed memory space, a certain fraction of the processor cycles or, in VMs running with real-time constraints, the maximum latency for interrupt handling. In most of these situations, as a single VM runs in a virtual environment isolated from those of other VMs, an application failure will only affect the causing VM. Thus, in case the VM cannot recover, all its resources can be reallocated by the hypervisor to a different VM.

An additional advantage of virtualization is derived from the possibility of running a collection of the logical nodes of a virtual *cluster* concurrently on a smaller number of physical nodes of an actual cluster. Under certain conditions (workload characteristics, service level agreements, etc.), this in turn enables the deployment of a virtualization-aware energy-saving strategy where servers are consolidated on a reduced number of physical machines, which may render a reduction of energy consumption, both by the processing equipment and the infrastructure (e.g., cooling, UPS, etc.). A step further in this line is to adopt a dynamic strategy for consolidation that adaptively selects the number of active physical servers depending on the workload, migrating VMs to allow consolidation, and turning on/off unused nodes (Kusic et al., 2009).

Live migration of VMs is crucial to leverage the energy savings potentially yielded by server consolidation. In this process, a VM that is running on a physical server A is migrated to an alternative server B, transparently to the user that is running his applications in the VM. For this purpose, i) all the memory in use by the VM has to be migrated from A to B; next ii) those memory pages that were modified by the VM on A since the migration started are copied to B; and finally iii) the process is completed with the transfer of the current processor state for the VM from A to B.

3 KVM

KVM is an open source software for full virtualization of x86 hardware. From the implementation point of view, it is a Linux kernel module that operates as a type-I hypervisor, providing the functionality that is needed to run VMs on the host platform. The integration of KVM into Linux offers two major advantages: first, all enhancements to Linux can be automatically leveraged from KVM; and second, KVM developers only need to tackle with the optimization of the applications running on the VMs being thus isolated from the underlying software layer (OS).

The main characteristics of KVM are:

Scheduling, Resource Control, and Memory Management. VMs run in KVM as regular Linux processes. Therefore, all the kernel-level man-

agement tools in the OS are also applicable to VMs. This includes scheduling, resource control and memory management, among others. Thus, for example, improvements to Linux such as process priority, CFS (*Completely Fair Scheduler*) and group control, which allow a fine-grain control of scheduling, can be easily leveraged to guarantee a certain quality of service (QoS) in VM management.

Storage. Images of VMs are treated in KVM as any other Linux regular file. VMs are thus stored using the standard mechanisms and media available in Linux, which includes local disks as well as high performance network storage systems.

Support for New Hardware Features. KVM inherits the entire Linux system so that all devices supported by Linux are supported by KVM as well. This permits the interoperation with systems that comprise a large number of CPUs and large amounts of RAM.

Security. VMs run as Linux system processes in KVM. Therefore, any single VM is protected from malicious execution of other VMs. Even more important is that the hypervisor is protected from such malicious behaviour.

4 EXPERIMENTS

In this section we present the experiments that were performed to assess the impact that the adoption of virtualization exerts on the performance and power dissipation of two applications with quite different properties. One particular aspect that is analyzed is the overhead incurred during live migration of VMs since, as argued at the end of section 2, this is crucial to consolidate physical servers and thus attain energy savings.

4.1 Hardware Setup

All the experiments were performed on a cluster composed of three HP Proliant DL120 G6 servers, with a single Intel Xeon X3430 processor (four cores operating at 2.67 GHz with a power dissipation of 95 Watts), 4 GB of RAM, and two Fast Ethernet cards per server. All nodes operated under 64-bit Linux Ubuntu 10.04LTS, and were connected via a Fast Ethernet switch.

The hypervisor was KVM version 0.12.3, with support for live migration of VM. Created instances of VMs featured a single-core `x86_64` CPU with 512 MB of RAM, running 64-bit Linux Ubuntu 11.10.

Power was measured using an external powermeter *Watts up? .Net* with a sampling rate of 1 Hz. The measures were collected for the full execution of the test, and the average power was multiplied by the execution time to obtain the energy consumption in Watts-hour.

4.2 Application Benchmarks

We employed the following two benchmarks for the evaluation:

AB. A significant part of today's servers perform data and web searches. We therefore selected an Apache server (Fielding and Kaiser, 1997) to evaluate the effect of virtualization on this type of services. Performance was measured for this test in terms of average time to serve requests (response time) from the clients.

MVB. Many scientific applications running on HPC platforms occupy a relevant fraction of CPU and memory resources of the target system. Performance, in terms of time-to-solution, is the key factor for these applications. Therefore, our goal here is to measure the impact of virtualization and live migration on the execution time of this type of computations. For that purpose, we chose a simple linear algebra operation, the matrix-vector product, which is representative of many memory-bounded computations that underlie more complex scientific computations. The actual performance is reported as the rate of GFLOPS (billions of floating-point arithmetic operations per second), which is inversely proportional to the time-to-solution.

When reporting results from both benchmarks, we also indicate the size of the VM that needs to be migrated.

4.3 Evaluation of Benchmark AB

In the experiments in this subsection, we evaluate the impact of live migration on performance by comparing two different scenarios. In one of them, (labelled as "VM" in the following four figures,) the Apache server runs in a VM placed on node A of the system, serving the web queries sent by a number of clients (users) running on node B. In the alternative scenario, (labelled as "VM+Migration",) the Apache server is migrated from node A to node C, while requests from the users on node B are simultaneously received. To avoid interferences, in this second scenario the migration occurs over a network different from that used to send queries and responses between

server and clients. Thus, the differences observed between both scenarios, for the duration of the VM relocation, are a clear indicator of the overhead introduced by the migration process.

Figure 1 reports the results with the benchmark **AB** with 20 and 120 users, that concurrently send requests to the server, for the period of time that comprises the VM migration plus some additional margins before the relocation commences and after it is completed. The traces show several relevant details. When there are only 20 users, the average response time is 0.889 s, while this value grows to 5.375 s for 120 users. It is also worth pointing out the consequences that the workload intensity has on the time that is necessary to migrate the VM. With 20 users (left-hand side plot), the VM that runs the server employs only 316 MB of memory, which results in 25 s to transfer it over the network and a total migration time of 31 s. The difference in this case, 6 s, is due to the time required to send the actual state of the VM. When the number of users grows to 120 (right-hand side plot), the size of the VM becomes larger, 441 MB, but the impact on the migration time is even more acute: 35 s for the network transfer and a total migration time of 64 s. One more aspect to note from the results in both figures is that, during the period of time in which the state of the VM is being migrated, there appears a much longer delay in the average response time. The reason is that, during this period, certain parts of the VM memory are modified while, simultaneously, others are being transferred and, as a consequence, the synchronizations necessary to maintain the consistency of the information have a relevant negative impact on the productivity.

Figure 2 shows the results from the same tests, but now with the option `keep alive` active that enables to use persistent connections to the server. Although the average response times are similar to those reported in the previous experiment, we note the increase of the memory utilized by the VM, for example, 508 MB for 120 users. The net outcome is a remarkable growth of the time required for the migration, which now requires 155 s.

4.4 Evaluation of Benchmark MVB

We have performed three types of experiments with benchmark **MVB**, to analyze the impact on both performance and power dissipation. In the first experiment, we simply evaluate the impact that the usage of VMs and live migration exerts on the GFLOPS rate of the operation. For that purpose, we consider three scenarios, depending on how/where the operation is run: i) directly executed on (one single core) of a node (label "Node"); ii) on a VM running on (one core) of a node (label "VM"); and iii) on a VM while it is migrated between two nodes (label "VM+Migration").

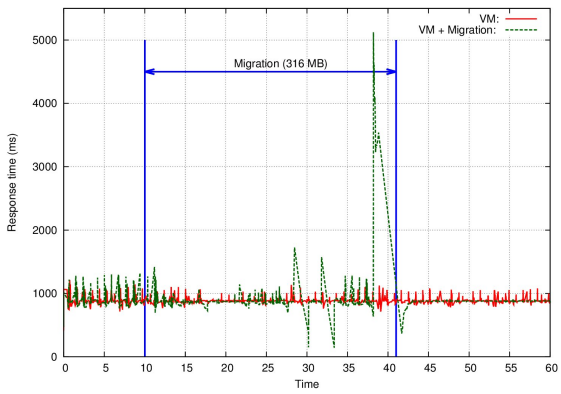
Figure 3 compares the results for this first experiment. As the matrix-vector product is a memory-bounded operation, the differences observed between the performances attained for the first two scenarios are quite small: 1.75 GFLOPS when the operation is run directly on the node vs. 1.70 GFLOPS if instead is executed by the VM. Furthermore, the migration only exerts a visible impact on the GFLOPS rate while the control is being transferred between the nodes, while the overhead is negligible for the rest of the relocation period.

The next experiments examine on the energy costs of virtualization and relocation. For reference, we initially analyze the power dissipated by a node that executes benchmark **MVB** using 1, 2, 3 and 4 cores as well as when the node is in different states/phases: shut off waiting for a Wake-on-LAN (WoL) signal, starting up, idle, and shutting down; see Figure 4. The corresponding values are captured numerically in Table 1. These results reveal that the power that is dissipated per core is gradually diminished as the the number of cores executing the workload is increased. For instance, the execution of the benchmark by a single core yields an increase of 22 W with respect to the idle state; on the other hand, the increase from 3 to 4 cores executing the workload results in an increment of power of only 10 W. A rapid conclusion from this study is that consolidation, when possible, can actually yield the desired energy-savings for this particular benchmark.

Table 1: Power dissipation and energy consumption of different states.

State	Duration	Power or Energy
Shut off/WoL	–	9 W
Starting up	75 s	1.42 Wh
Idle	–	50 W
Running **MVB** - 1 core	–	72 W
Running **MVB** - 2 core	–	90 W
Running **MVB** - 3 core	–	100 W
Running **MVB** - 4 core	–	110 W
Shutting down	35 s	0.48 Wh

The following experiment reports the power overhead incurred by virtualization of benchmark **MVB**. Specifically, in Figure5 we show two scenarios: one with the benchmark being executed directly on the node (label "Node") and an alternative with the benchmark running on a VM (label "VM"). In both cases, up to 4 cores/VMs are involved in the experi-



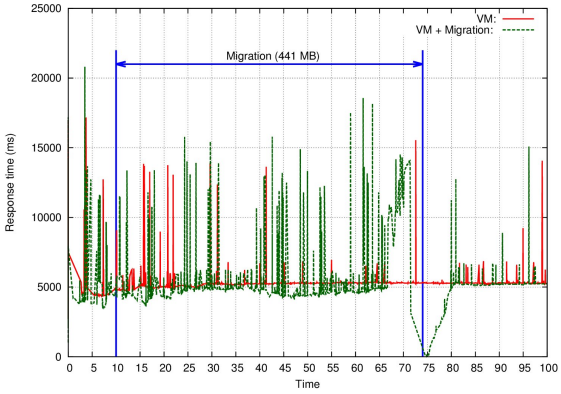
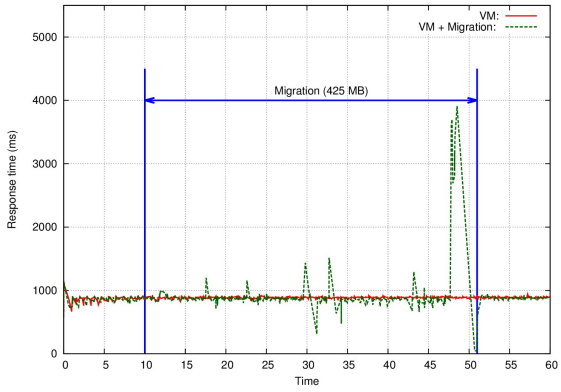


Figure 1: Response time for benchmark **AB** with 20 and 120 users (left and right, respectively).



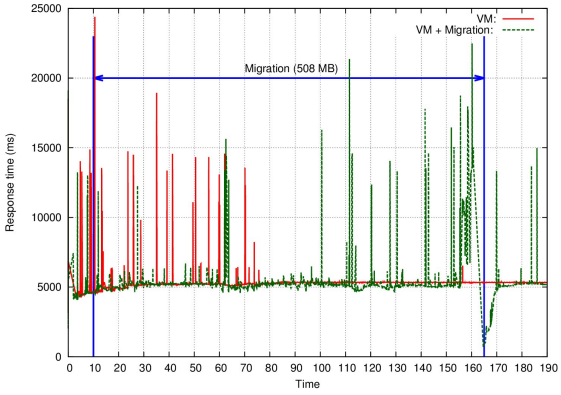


Figure 2: Response time for benchmark `AB` with 20 and 120 users (left and right, respectively) and *keep alive* active.

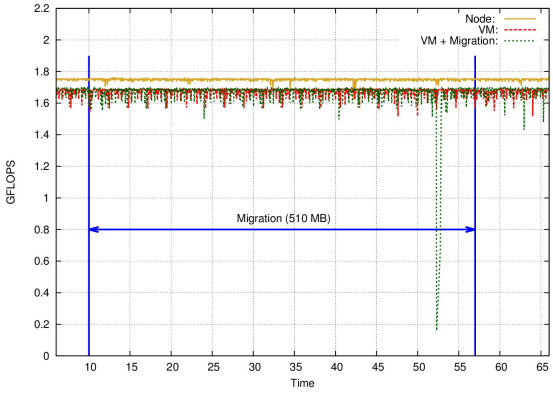


Figure 3: GFLOPS rate for benchmark **MVB**.

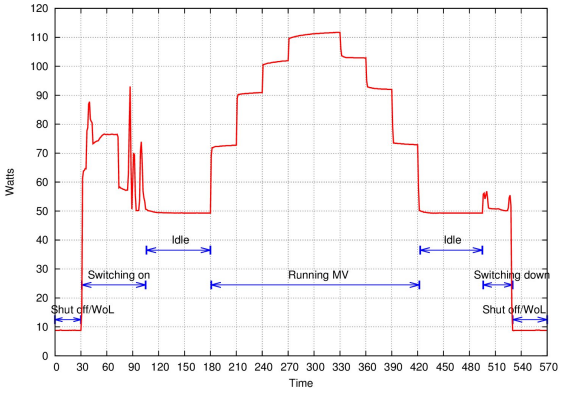


Figure 4: Power dissipation and energy consumption of different states.

ment. The trace reveals that the power differences between both scenarios are small, decreasing the number of active cores, and being in all cases lower than 2.5 W.

The final two experiments illustrate the overhead of VM relocation on power consumption. In the first one, we report the power dissipated by a node A during the relocation of an idle VM to a different node B, and (after 300 s) back from B to A; see the left-hand side plot of Figure 6. In the second test we repeat the experiment, this time while the VM is executing one instance of benchmark **MVB**; see the right-hand side plot of the same figure. Theses two traces clearly expose the overhead in both performance (time) and

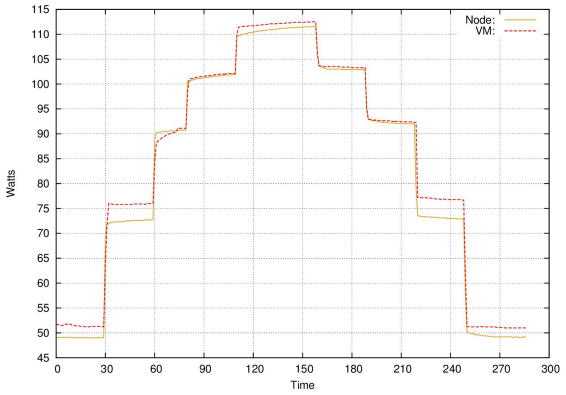


Figure 5: Power dissipation of a node running benchmark **MVB**.

power that the second case exhibits due to the presence of the workload.

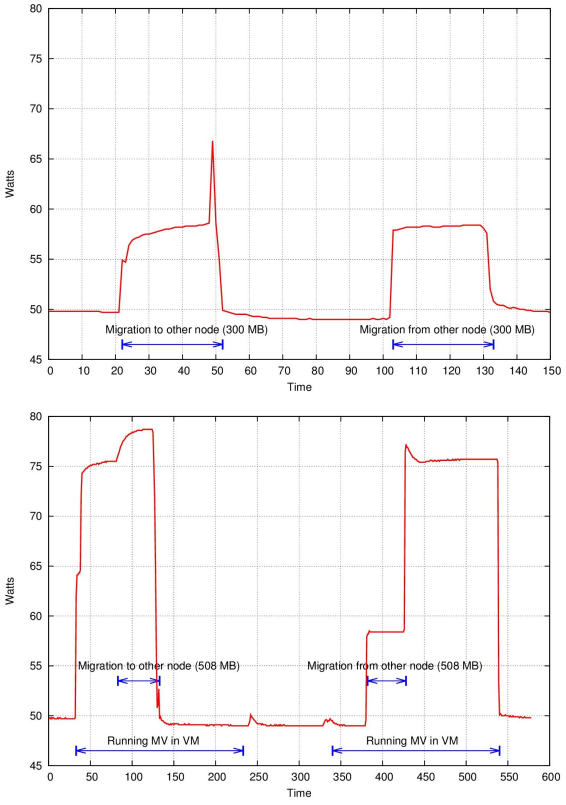


Figure 6: Power dissipation during the relocation of an idle VM and a VM running benchmark **MVB** on a single core (top and bottom, respectively).

5 CONCLUSIONS AND FUTURE WORK

In this paper we have investigated the benefits and costs of virtualization for two different applications, related to the common workloads running on datacenters and HPC facilities. The results from this work-in-progress link the response time and performance with the potential energy savings that consolidation of physical servers can yield. Further research is necessary to formally characterize the effects of VM relocation into the power dissipation and performance of the applications.

ACKNOWLEDGEMENTS

This work was supported by project TIN2011-23283 of the *Ministerio de Economía y Competitividad* and FEDER, and project P11B2013-21 of the Universidad Jaume I.

REFERENCES

(2013). Openstack open source cloud computing software. http://www.openstack.org.

(2013). QEMU. http://wiki.qemu.org.

(2013). VirtualBox. https://www.virtualbox.org/.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. (2003). Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5):164–177.

Fielding, R. and Kaiser, G. (1997). The Apache HTTP server project. *IEEE Internet Computing*, 1(4):88–90.

Goldberg, R. (1974). Survey of virtual machine research. *IEEE Computer*, 7(6):34–45.

Kivity, A., Lublin, U., and Liguori, A. (2007). KVM: the Linux virtual machine monitor. *Reading and Writing*, 1(1465-6914):225–230.

Kusic, D., Kephart, J. O., Hanson, J. E., Kandasamy, N., and Jiang, G. (2009). Power and performance management of virtualized computing environments via lookahead control. *Cluster Computing*, 12(1):1–15.

Nieh, J. and Leonard, O. C. (Aug. 2007). Examining VMware. *Dr. Dobb's Journal*.

Younge, A., Henschel, R., Brown, J., von Laszewski, G., Qiu, J., and Fox, G. (2011). Analysis of virtualization technologies for high performance computing environments. In *IEEE Int. Conf. on Cloud Computing 2011*, pages 9–16.

Younge, A. J., Brown, J., Henschel, R., Qiu, J., and Fox, G. (2010). Performance analysis of HPC virtualization technologies within FutureGrid. In *Conference CloudCom 2010*. poster.

Energy-aware VM Scheduling in IaaS Clouds using Pliant Logic

Attila Benyi[1], Jozsef Daniel Dombi[1] and Attila Kertesz[2,1]
[1]*Software Engineering Department, University of Szeged, 6720 Szeged, Dugonics ter 13, Hungary*
[2]*Institute for Computer Science and Control, MTA SZTAKI, H-1111 Budapest, Kende u. 13-17, Hungary*
benyi.attila@stud.u-szeged.hu, dombijd@inf.u-szeged.hu, kertesz.attila@sztaki.mta.hu

Keywords: Cloud Computing, VM Scheduling, Pliant System, Simulation.

Abstract: Cloud Computing is facing an increasing attention nowadays as it is present in many consumer appliances by advertising the illusion of infinite resources towards its customers. Nevertheless it raises severe issues with energy consumption: the higher levels of quality and availability require irrational energy expenditures. This paper proposes a Pliant system based virtual machine scheduling approach for reducing energy consumption of IaaS datacenters. In order to evaluate our proposed solution, we have designed a CloudSim-based simulation environment, and applied real-world traces for the experiments. We show that significant savings can be achieved in energy consumption with our proposed Pliant-based algorithms, and by fine-tuning the parameters of the proposed Pliant strategy, a beneficial trade-off can be set between energy consumption and execution time.

1 INTRODUCTION

Cloud computing incorporates many aspects of sharing software and hardware solutions, including computing and storage resources, application runtimes or complex application functionalities. The cloud paradigm changed the way people look at computing infrastructures. First, one does not need to be expert in infrastructure administration, operation and maintenance even if large scale systems are utilized. Second, the elasticity of Infrastructure as a Service clouds allows these systems to better follow the users' actual demands. However, there is also an adversary effect: the virtualized nature of these systems detaches users from several operational issues like energy efficient usage, that has been addressed previously in the context of parallel and distributed systems, and largely remains unnoticed (Buyya et al., 2009; Schubert and Jeffery, 2012).

The Cloud computing technology made a qualitative breakthrough as it is present in many consumer appliances including various mobile devices. They advertise the illusion of infinite resources towards the consumers, meanwhile it also raises severe issues with energy consumption: the higher levels of quality and availability require irrational energy expenditures, according to some experts the consumed energy of resources spent for idling represent a considerable amount (Lefèvre and Orgerie, 2009). Current trends are claimed to be clearly unsustainable with respect to resource utilisation, CO_2 footprint and overall energy efficiency. It is anticipated that further growth is limited by energy consumption, furthermore competitiveness of companies are and will be strongly tied to these issues.

As cloud services become more and more popular, small- and medium-sized cloud service providers will soon face increasing user demands that cannot be met with their current infrastructures. These user demands range from occasional needs for extreme amount of resources (compared to the provider's current infrastructure) to the need for multi-site virtual machine deployment options that enable enhanced services such as disaster recovery. Thus these providers need to increase the size of their infrastructure by introducing multiple datacenters covering various locations, and offering unprecedented amount of resources. Current IaaS solutions provide the opportunity for service providers to satisfy these needs by focusing their attention to non-technical issues like the increased operating cost of their datacenters. Despite energy consumption is a major component of these operating costs, current IaaS solutions barely handle the infrastructure with energy aware solutions. Therefore providers were restricted to reduce their consumption on the hardware level so far, independently from the applied IaaS solution. Energy costs are also increasing, and datacenter equipment is stressing power and

cooling infrastructures, thus the main issue is not the current amount of data center emissions but the fact that these emissions are raising faster than any other carbon emission (Berral et al., 2010). Although these improvements in hardware are crucial, we believe that the energy consumption could also be significantly reduced with software in over-provisioned IaaS systems. Over-provisioning is a key behaviour at smaller sized providers, who offer services for users with occasional peaks in resource demands.

Reducing the carbon footprint of European countries is also a must and expected by the European Commission, as well as to increase the number and size of European Cloud providers (Schubert and Jeffery, 2012). By federating these providers, more competitive initiatives can be founded, that can be sophistically managed to meet these expectations. The general goal of the management layer in a Cloud federation is to distribute load among the participating cloud providers, to enhance user satisfaction by filtering out underperforming providers, and schedule and execute service calls with minimized energy consumption within the selected IaaS system. To achieve this, we have already proposed an architecture called Federated Cloud Management (FCM – as introduced in (Kecskeméti et al., 2012)). In this holistic approach a two-level brokering solution is used: a meta-brokering component is used to direct service calls to providers, and then a cloud-brokering component to map these calls onto an optimized number of virtual machines.

In this paper we target the later, cloud-brokering layer, and we focus on the energy-aware management of datacenters of single cloud providers specialized for provisioning task-based cloud applications. In order to enable experimentation in this field, we have developed a CloudSim-based simulation environment. To cope with the high uncertainty and unpredictable load present in these heterogeneous, virtualized large-scale systems, we apply a Pliant system based approach (Dombi, 2012) to the management of these systems, which is similar to a fuzzy system (Dombi, 1982).

Therefore the main contributions of this paper are: (i) the development of a cloud simulation environment for task-based cloud applications, (ii) the design of an energy-aware and Pliant-based VM scheduling algorithm for VM management Clouds, and (iii) the evaluation of the proposed algorithms in the extended simulation environment with real-world traces.

The remainder of this paper is as follows: Section 2 presents the related VM management approaches in datacenters; Section 3 introduces our extended simulation architecture; Section 4 introduces the advanced scheduling algorithms using the Pliant method for VM scheduling; and Section 5 describes the evaluation methodology and the simulation results. Finally, Section 6 summarizes the main contributions of the paper.

2 RELATED WORK

Regarding energy efficiency in a single cloud, Cioara et al. (Cioara et al., 2011) introduced an energy aware scheduling policy to consolidate power management by using reinforcement learning techniques to restore a service center to an energy efficient state. Feller et al. proposed a dynamic cluster manager called Snooze (Feller et al., 2010), which is able to dynamically consolidate the workload of a heterogeneous large-scale cluster composed of resources using virtualization. In a later work (Feller et al., 2012), they use power meters to monitor energy usage of cloud resources, and estimate the resource usage of VMs. Their mechanisms address VM placement, relocation and migration by keeping VMs on as few nodes as possible.

Cardosa et al. (Cardosa et al., 2009) presented a novel suite of techniques for placement and power consolidation of VMs in datacentres taking advantage of the min-max and shares features inherent in virtualization technologies, like VMware and Xen. These features allow to specify the minimum and maximum amount of resources that can be allocated to a VM, and provide a shares based mechanism for the hypervisor to distribute spare resources among contending VMs. Lee et al. (Lee et al., 2010) discuss service request scheduling in Clouds based on achievable profits. They propose a pricing model using processor sharing for composite services in Clouds.

Lucas-Simarro et al. (Lucas-Simarro et al., 2013) proposed different scheduling strategies for optimal deployment of services across multiple clouds based on various optimization criteria. The examined scheduling policies include budget, performance, load balancing and other dynamic conditions, but they neglected energy efficiency, which is the aim of our work.

Regarding fuzzy approaches, Salleh et al. (Salleh et al., 1999) have shown how to set up and use fuzzy logic in a traditional way for dynamic task scheduling in multiprocessor systems. We have already published a paper (Dombi and Kertész, 2011) on applying the Pliant approach to job scheduling in Grids. In this current paper we would like to show that it is also possible to use Pliant system for scheduling, with only a few rules. The novelty of this contribution lies in the

way we apply the Pliant system to Clouds: the way we select cloud-specific properties as parameters of the Pliant system.

Concerning cloud simulations, Berge et al. (vor dem Berge et al., 2012) have designed a simulator called SVD within the CoolEmAll project for investigating energy consumption in datacenters. It is an extended version of the GSSIM simulator, and they are planning to support application modeling and profiling through benchmarks. Regarding federation-wide simulations, Sotiriadis et al. (Sotiriadis et al., 2013) investigated inter-cloud simulations by developing the SimIC simulation toolkit that is able to mimic the inter-cloud service formation to enable the investigation of service-oriented cloud utilization, but they neglect energy efficiency.

Listing 1: Pseudo code of the default OptUtil algorithm

```
lowestVm = first VM with the same
    queue type as the cloudlet;
FOREACH (vmlist as vm)
        IF (vm.utilization() <
            lowestVm.utilization()
         AND vm.queueType ==
             lowestVm.queueType)
                lowestVm=vm;
IF (lowestVm.utilization >  100)
        IF (try to create a new vm)
                lowestVm = new vm;
cloudlet.setVm = lowestVm;
```

3 SIMULATION OF CLOUDS

We have used the CloudSim simulator (Calheiros et al., 2011) to develop a simulation environment for our research. Beloglazov and Buyya (Beloglazov and Buyya, 2012) have already started to examine how energy efficiency could be investigated within this simulator. Datacenters consume huge amounts of energy resulting in high operating costs and increased carbon dioxide emissions. The dynamic consolidation of VMs using live migration and switching off idle nodes can be used to optimize resource usage and reduce energy consumption, but they argue that aggressive consolidation may lead to performance degradation. They proposed adaptive heuristics for dynamic consolidation of VMs based on an analysis of historical data from the resource usage by VMs, while ensuring a high level of adherence to the Service Level Agreements (SLA). They used PlanetLab trace files (Park and Pai, 2006) workload logs to simulate load changes of continuously running services in VMs. These traces contain records of each VM's periodic utilization, thus the simulation assumes each VM is going to process only one task (called as cloudlet in CloudSim) at a time as a service.

In this work our goal was to investigate task-based (HPC/HTC) cloud applications executed by a single cloud provider possibly having more than one datacenter. Since CloudSim is tailored to the evaluation of continuously running web-based applications (Beloglazov and Buyya, 2012), we decided to extend this simulation environment to suite our needs.

Our approach is slightly different to the one used by the original version of CloudSim, as we tried sending cloudlets with varying parameters, such as start time and length at random intervals. For that purpose we used the log files provided by Prezi Inc. (Prezi, 2013) (discussed in detail in Section 5). These log files contain detailed data on each cloudlet received, such as its start time, length and queue type. To adapt CloudSim to the new features, several changes had to be made. One of the crucial changes was in the *CloudletScheduler* component, so each VM could handle multiple cloudlets at the same time. As long as the VM's utilization is below 100%, it can process new cloudlets, and once a VM reaches its full utilization, further cloudlets get queued. Once a VM has no cloudlets left to process, it is shut down, and if a host has no remaining VMs, it is shut down as well. Each host's power consumption is based on a power model, which is based on a benchmark result provided by SPEC (SPEC, 2013). We used 5 different power models to make the difference between varying algorithms more obvious. Each datacenter sums up the power consumed by its hosts for every timeframe a cloudlet is being processed, giving us a close approximation of the amount of power and time needed to complete all the requested cloudlets. For each cloudlet a VM is chosen by our default VM scheduling algorithm called 'OptUtil' shown in Listing 1. The hosts (physical machines) created during the simulations differ in their characteristics, altogether 5 types of hosts were used. However, while there are different hosts, only one type of VM was used in all simulations.

In case every VMs utilization is over 100%, the algorithm will try to create a new one, thus ensuring the lowest process time. For each new VM the host is chosen based on its power model, and we are assuming that every host will be fully utilized, so the host with the lowest power consumption on 100% utilization will be submitted, ensuring the lowest power consumption. In the following section we discuss the Pliant-based VM scheduling solution.

4 PLIANT SCHEDULING APPROACH

Fuzzy sets were introduced by Lofti Zadeh in 1965 with the aim of reconciling mathematical modeling and human knowledge in the engineering sciences. Most of the building blocks of the theory of fuzzy sets were proposed by him, especially fuzzy extensions of classical basic mathematical notions like logical connectives, rules, relations and quantifiers. The Pliant system (Dombi, 2012) is a kind of fuzzy theory that is similar to a traditional fuzzy system (Dombi, 1982). The difference between the two systems lies in the choice of operators. The Pliant system has a strict, monotonously increasing t-norm and t-conorm, and the following expression is valid for the generator function:

$$f_c(x)f_d(x) = 1, \tag{1}$$

where $f_c(x)$ and $f_d(x)$ are the generator functions for the conjunctive and disjunctive logical operators, respectively. This system is defined in the [0,1] interval. In our previous paper (Dombi and Kertész, 2011), we developed a scheduling component that uses the Pliant system to select a good performing Grid broker for a user's job even under conditions of high uncertainty. The algorithm we developed calculates a score for each cloudlet using the cloud's properties. The calculation step includes a normalization step, where we apply a special Sigmoid function. In the normalization step it should be mentioned that if the normalized value is close to one, it means it is a more valuable property, and if the normalized value is close to zero, it means it is a less prioritized property. For example, if the counter of power consumption is high, the normalization algorithm should give a value close to zero. In our previous work (Dombi and Kertész, 2011) we found that if we use the aggregation operator to calculate the score number, we can achieve better results.

Here, we created two scheduling algorithms in order to handle the energy aware management case with a similar approach. One considers time and the other considers energy for optimization. There are hosts in the simulated datacenters, and each host can run several VMs. This environment can be described with the same three properties, namely a power usage counter (PUC), the power consumption counter (PCC) and the number of processors (PROC):

- The power usage counter gives performance of the CPU usages of the given simulation time. The value can be larger than 100, which means that there are more cloudlets in the VM's queue.

Table 1: Parameters of the Sigmoid function.

Property	Time		Energy	
Property	Alpha	Lambda	Alpha	Lambda
PUC	0.5	-4.0	0.5	-4.0
PCC	85.0	-0.08	75.0	-0.08
PROC	1.0	0.8	1.0	0.8

- The power consumption counter gives the energy usage of the given host at a given time. The value is generally between 40 and 120 MIPS, but it depends on the actual physical processor.
- The number of processors gives the available number of processors of a host.

We have developed two Pliant decision making algorithms that take into account the above-mentioned properties and decide to which VM a cloudlet should be submitted: one optimizes cloudlet executions for time, and the other one for energy. We use different normalization for the two strategies. First we start with a normalization step and we apply different kinds of Sigmoid functions to normalize the environment's property value. We examine the environment's variable and define the value of the Sigmoid's parameter. Table 1 shows the predefined values of the parameters of the normalization functions.

In this environment every host has 4 processors, so after the normalization the normalized property value is the same for all instances. We would like to emphasize that it is better if we use less power, therefore we created two different parameter sets: one for time-aware and one for energy-aware scheduling. As we can see in Figure 1, the minimum energy in this environment is around 40 and the maximum is around 120. Here we can see that if the number of power consumption is increasing then the value of the normalized function is decreasing. The opposite is true for the number of processors.

After the normalization step we modify the normalized value to emphasize the importance of the result. To achieve this we will modify the normalized value by using the Kappa function with $\nu = 0.4$ and $\lambda = 3.0$ parameters:

$$\kappa_\nu^\lambda(x) = \frac{1}{1 + \left(\frac{\nu}{1-\nu}\frac{1-x}{x}\right)^\lambda} \tag{2}$$

Finally to calculate a VM's score number for the given cloudlet, we use the aggregation operator:

$$a_{\nu,\nu_0}(x_1,\cdots,x_n) = \frac{1}{1 + \frac{1-\nu_0}{\nu_0}\frac{\nu}{1-\nu}\prod_{i=1}^n \frac{1-x_i}{x_i}}, \tag{3}$$

where ν is the neutral value and ν_0 is the threshold value of the corresponding negation. Here we don't

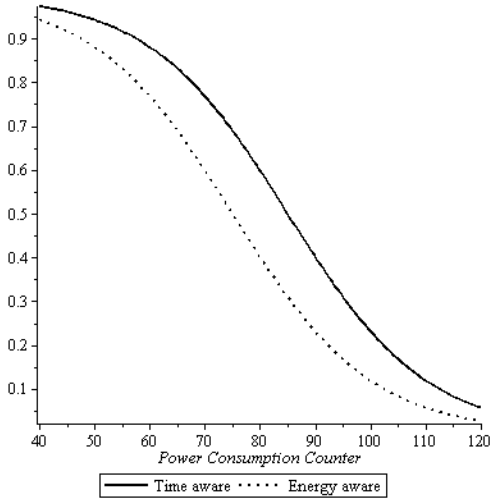


Figure 1: Utilized normalized function for the power consumption (PCC).

want to threshold the result so both parameters have the same value 0.5. The result of the calculation is always a real number that lies in the [0,1] interval. So we calculate the score for all VM to find which VM is the most suitable for our strategy. If the best score value is very low (the value depends on the strategy), then we try to create a new VM.

5 EVALUATION

In order to investigate the energy consumption of cloud providers in our extended simulation environment, we have used real-world trace files of an international company called Prezi Inc, who offers a presentation editing service, which is available on multiple platforms, therefore they have to convert some of the uploaded media files to other formats before they can display them on all devices. In April 2013, they launched a competition titled "Scale Contest" (Prezi, 2013) for university students to test their knowledge of control and queueing theories on real-life problems. Their conversion processes are carried out on virtual machines: at peak times, they need to launch more instances of these VMs, but over the weekend they can stop most of them. This campaign was initiated in order to find a suitable algorithm that launches the exact number of VMs for a given workload. They published log files on their website containing workload traces for two weeks of utilization, which serves as a basis for algorithmic experimentations.

They operate three queues in their system for the jobs participating in the conversion processes:

- export: contains jobs which result in downloadable zipped prezis.
- url: these jobs download an image from a URL and insert them into a prezi.
- general: all other conversion jobs (audio, video, pdf, ppt, etc).

The lines of the published workload traces have the following format: "2012-12-14 21:35:12 237 general 9.134963". This means that at the given time, a job enters the *general* queue with the id 237, and the job will take 9.134963 seconds to run. These logs had to be used as input by the competitors. They contain three weeks of actual data accumulated by Prezis conversion system, and the first two weeks of logs are publicly available. They planned to use unpublished logs from the third and fourth week to evaluate your submissions to the competition.

Table 2: Evaluation results for RoundRobin.

Hosts	**Cloud-lets**	**VMs**	**Energy (kWh)**	**Time (sec)**
100	10000	1<	63.20	25200
100	50000	1<	104.66	39000
500	50000	1<	143.62	48600
500	100000	1<	381.37	70200

Table 3: Evaluation results for OptUtil.

Hosts	**Cloud-lets**	**VMs**	**Energy (kWh)**	**Time (sec)**
100	10000	1<	18.90	7500
100	50000	1<	87.12	32400
500	50000	1<	90.41	7200
500	100000	1<	197.26	15000

For a preliminary evaluation phase we used the trace file of the first week. We have performed experiments with datacenters having 100 to 500 hosts, and submitted 10000 to 100000 jobs from the log. By default we used a round robin strategy to schedule the logs to the available VMs (1 at the beginning), and if no more available VM was present in the system (that could execute the job without any delay) at a given time, we have deployed another one continuously. The results of this evaluation can be seen in Table 2. We have also executed similar simulations by applying our proposed optimized utilization strategy called 'OptUtil', that deploys another VM, if the available ones are at least 80% loaded. The results of this second evaluation can be seen in Table 3.

From these preliminary evaluation we can see that our proposed algorithm performed better than the round robin, both in energy consumption and execution time.

To develop Pliant-based algorithms, we created three initial strategies: the first one uses only one VM to execute all submitted jobs (MINIMUM), the second deploys a new VM for all jobs (MAXIMUM), and the third uses randomized VM selection from the available VMs (smartly prioritizing the less loaded ones), and deploys a new one, if no free VM is found (SMARTRANDOM). Tables 5, 4 and 6 summarize the results of evaluating these algorithms. From these results we can see that utilizing the lowest number of VMs results in the lowest energy consumption, but of course on the expense of the execution time, which is the highest in this case.

Table 4: Evaluation results for MAXIMUM.

Hosts	Cloud-lets	VMs	Energy (kWh)	Time (sec)
100	1000	241	7.64	759
100	10000	241	76.35	4088
100	50000	241	365.35	14220
100	100000	241	934.22	39224

Table 5: Evaluation results for MINIMUM.

Hosts	Cloud-lets	VMs	Energy (kWh)	Time (sec)
100	1000	3	0.19	8179
100	10000	3	1.91	81008
100	50000	3	6.54	240940
100	100000	3	13.87	461724

Table 6: Evaluation results for SMARTRANDOM.

Hosts	Cloud-lets	VMs	Energy (kWh)	Time (sec)
100	1000	3	0.20	8619
100	10000	3	1.53	60298
100	50000	3	5.77	198060
100	100000	3	12.50	386074

Based on these preliminary evaluations we have created a Pliant-based strategy (PLIANTDEFAULT), first focusing on execution time reduction with some energy savings. For its default algorithm Table 7 shows the results of the simulation. This table shows that it could achieved significant performance gains in terms of execution time as expected, but it also had much higher energy consumption than the MINIMUM and SMARTRANDOM initial strategy.

Therefore we have modified the parameters of the applied Pliant system, and created more focused algorithms. In Table 8 we used a Pliant version that is more focused on execution time savings (PLIANTTIME), while in Table 9 we modified a Pliant parameter to focus on energy savings (PLIANTENERGY).

Table 7: Evaluation results for PLIANTDEFAULT.

Hosts	Cloud-lets	VMs	Energy (kWh)	Time (sec)
100	1000	14	0.26	749
100	10000	16	2.87	3768
100	50000	24	17.26	14240
100	100000	25	53.21	39304

Table 8: Evaluation results for PLIANTTIME.

Hosts	Cloud-lets	VMs	Energy (kWh)	Time (sec)
100	1000	13	0.21	629
100	10000	16	2.77	4128
100	50000	21	15.20	14380
100	100000	21	43.55	39274

Figure 2 shows comparison diagrams concerning the last rows of the tables.

As a result of the evaluations we can state that for minimal energy consumption the least amount of VMs should be used with smartly randomized VM selection. Nevertheless, when there is a need for execution time optimizations, we have to find a trade-off between energy consumption and execution time. With our proposed Pliant-based VM scheduling algorithms we have shown that significant savings can be achieved in energy consumption with moderate execution time reductions.

6 CONCLUSION

Cloud Computing is facing an increasing attention nowadays and it raises severe issues with energy consumption: the higher levels of quality and availability require irrational energy expenditures. Reducing the carbon footprint of European countries is also a must and expected by the European Commission, as well as to increase the number and size of European Cloud providers.

In this paper we have proposed a Pliant system based virtual machine scheduling approach for reducing energy consumption of IaaS datacenters. We have designed a CloudSim-based simulation environment for task-based cloud applications, and applied real-world traces for the performed experiments. We have shown that significant savings can be achieved in energy consumption with our proposed Pliant-based algorithms, and by fine-tuning the parameters of the proposed Pliant strategy, a beneficial trade-off can be set between energy consumption and execution time.

Our future work aims at automating the parameter selection in different IaaS systems, and adapting

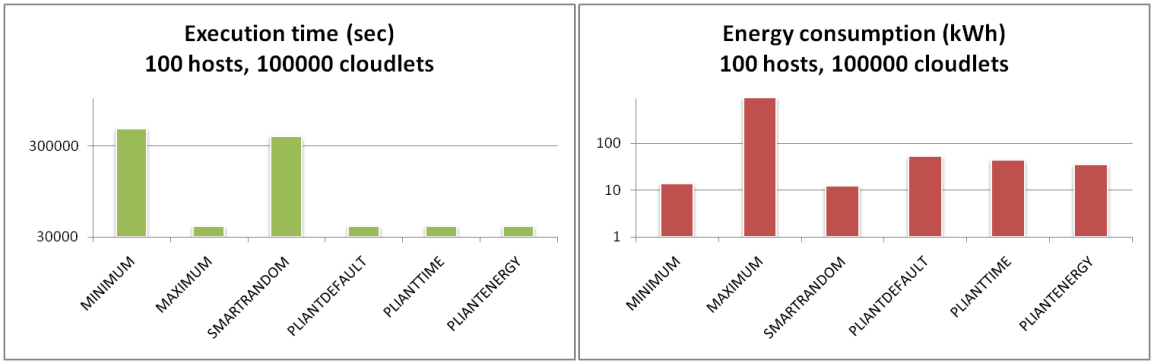


Figure 2: Comparison diagrams for 100000 cloudlets.

Table 9: Evaluation results for PLIANTENERGY.

Hosts	Cloud-lets	VMs	Energy (kWh)	Time (sec)
100	1000	12	0.18	669
100	10000	16	2.34	3788
100	50000	18	12.99	14380
100	100000	18	34.55	39274

the proposed approach in production-level academic Clouds.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the EU FP7 IDGF-SP project under grant agreement 312297, and it was supported by the European Union and the State of Hungary, co-financed by the European Social Fund in the framework of TAMOP 4.2.4. A/2-11-1-2012-0001 'National Excellence Program'.

REFERENCES

Beloglazov, A. and Buyya, R. (2012). Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience*, 24(13):1397–1420.

Berral, J. L., Goiri, I., Nou, R., Julià, F., Guitart, J., Gavaldà, R., and Torres, J. (2010). Towards energy-aware scheduling in data centers using machine learning. In de Meer, H., Singh, S., and Braun, T., editors, *e-Engery*, pages 215–224. ACM.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Comp. Syst*, 25(6):599–616.

Calheiros, R. N., Ranjan, R., Beloglazov, A., Rose, C. A. F. D., and Buyya, R. (2011). Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw, Pract. Exper*, 41(1):23–50.

Cardosa, M., Korupolu, M. R., and Singh, A. (2009). Shares and utilities based power consolidation in virtualized server environments. In *Integrated Network Management*, pages 327–334. IEEE.

Cioara, T., Anghel, I., Salomie, I., Copil, G., Moldovan, D., and Kipp, A. (2011). Energy aware dynamic resource consolidation algorithm for virtualized service centers based on reinforcement learning. In *ISPDC*, pages 163–169. IEEE.

Dombi, J. (1982). A general class of fuzzy operators, the De Morgan class of fuzzy operators and fuzziness measures induced by fuzzy operators. *Fuzzy Sets and Systems*, 8:149–163.

Dombi, J. (2012). Pliant operator system. In Fodor, J. C., Klempous, R., and Araujo, C. P. S., editors, *Recent Advances in Intelligent Engineering Systems*, volume 378 of *Studies in Computational Intelligence*, pages 31–58. Springer.

Dombi, J. D. and Kertész, A. (2011). Advanced scheduling techniques with the pliant system for high-level grid brokering. In Filipe, J., Fred, A., and Sharp, B., editors, *Agents and Artificial Intelligence*, volume 129 of *Communications in Computer and Information Science*, pages 173–185. Springer Berlin Heidelberg.

Feller, E., Rilling, L., Morin, C., Lottiaux, R., and Leprince, D. (2010). Snooze: A scalable, fault-tolerant and distributed consolidation manager for large-scale clusters.

Feller, E., Rohr, C., Margery, D., and Morin, C. (2012). Energy management in iaaS clouds: A holistic approach. In *Proc. 2012 IEEE Fifth International Conference on Cloud Computing (5th IEEE CLOUD'12)*, pages 204–212, Honolulu, HI, USA. IEEE Computer Society.

Kecskeméti, G., Kertész, A., Marosi, A., and Kacsuk, P. (2012). Interoperable resource management for establishing federated clouds. In Villari, M., Brandic, I., and Tusa, F., editors, *Achieving Federated and Self-Manageable Cloud Infrastructures: Theory and Practice*, pages 18–35. IGI Global, Hershey.

Lee, Y. C., Wang, C., Zomaya, A. Y., and Zhou, B. B. (2010). Profit-driven service request scheduling in clouds. In *CCGRID*, pages 15–24. IEEE.

Lefèvre, L. and Orgerie, A.-C. (2009). Towards energy aware reservation infrastructure for large-scale experimental distributed systems. *Parallel Processing Letters*, 19(3):419–433.

Lucas-Simarro, J. L., Moreno-Vozmediano, R., Montero, R. S., and Llorente, I. M. (2013). Scheduling strategies for optimal service deployment across multiple clouds. *Future Generation Comp. Syst*, 29(6):1431–1441.

Park, K. and Pai, V. S. (2006). Comon: a mostly-scalable monitoring system for planetlab. *Operating Systems Review*, 40(1):65–74.

Prezi (2013). Prezi Inc. "scale contest" website, http://prezi.com/scale/, april 2013.

Salleh, S., Sanugi, B., and Jamaluddin, H. (1999). Fuzzy logic model for dynamic multiprocessor scheduling.

Schubert, L. and Jeffery, K. (2012). Advances in clouds - research in future cloud computing.

Sotiriadis, S., Bessis, N., and Antonopoulos, N. (2013). Towards inter-cloud simulation performance analysis: Exploring service-oriented benchmarks of clouds in simIC. In Barolli, L., Xhafa, F., Takizawa, M., Enokido, T., and Hsu, H.-H., editors, *AINA Workshops*, pages 765–771. IEEE Computer Society.

SPEC (2013). Spec website, http://www.spec.org, august 2013.

vor dem Berge, M., Costa, G. D., Kopecki, A., Oleksiak, A., Pierson, J.-M., Piontek, T., Volk, E., and Wesner, S. (2012). Modeling and simulation of data center energy-efficiency in coolemall. In Huusko, J., de Meer, H., Klingert, S., and Somov, A., editors, *E2DC*, volume 7396 of *Lecture Notes in Computer Science*, pages 25–36. Springer.

Improving Resource Utilization in Cloud Environments using Application Placement Heuristics

Atakan Aral and Tolga Ovatman
Department of Computer Engineering, Istanbul Technical University, Istanbul, Turkey
{*aralat, ovatman*}*@itu.edu.tr*

Keywords: Application Placement, Resource Allocation, Cloud Computing, Software as a Service.

Abstract: Application placement is an important concept when providing software as a service in cloud environments. Because of the potential downtime cost of application migration, most of the time additional resource acquisition is preferred over migrating the applications residing in the virtual machines (VMs). This situation results in under-utilized resources. To overcome this problem static/dynamic estimations on the resource requirements of VMs and/or applications can be performed. A simpler strategy is using heuristics during application placement process instead of naively applying greedy strategies like round-robin. In this paper, we propose a number of novel heuristics and compare them with round robin placement strategy and a few proposed placement heuristics in the literature to explore the performance of heuristics in application placement problem. Our focus is to better utilize the resources offered by the cloud environment and at the same time minimize the number of application migrations. Our results indicate that an application heuristic that relies on the difference between the maximum and minimum utilization rates of the resources not only outperforms other application placement approaches but also significantly improves the conventional approaches present in the literature.

1 INTRODUCTION

In addition to software delivered as services over the Internet, hardware and software systems that make the delivery possible are referred as cloud computing (Armbrust et al., 2010). In cloud computing, resources such as CPU, memory, bandwidth and storage are treated as utilities that can scale up and down on demand. It also allows per-usage metering and billing of these resources. By means of cloud computing, cloud users can handle unexpected high demands without over-provisioning and they do not need to invest for abundant hardware resources initially. Cloud providers, on the other hand, have the opportunity of reallocating idle resources for other cloud users.

Cloud computing services are categorized into three layers according to the level of abstraction they provide. These are, in ascending order of abstraction: (1) Infrastructure as a Service (IaaS) that provide virtual raw resources, (2) Platform as a Service (PaaS) that provide virtual development environments, and (3) Software as a Service (SaaS) that provide online applications on demand (Buyya et al., 2011).

One general research challenge in cloud computing is the efficient allocation of cloud resources to users since cloud providers should satisfy quality of service (QoS) objectives while minimizing their operational cost (Zhang et al., 2010). In this paper, we approach this challenge from the perspective of a SaaS provider. Provider receives user applications with heterogeneous resource requests and assigns these applications to the virtual machines (VMs) with finite capacities. Multiple requests can be assigned to a single VM so the SaaS provider would prefer to fully utilize the VM resources at hand before hiring new ones. It is also possible to move applications between VMs, but this would cause a temporary halt of the migrating application and would reduce QoS. Consequently, the objective is to maximize the number of applications assigned to VMs and at the same time, minimize the number of application migrations.

For the sake of simplicity, we make the following four assumptions regarding the characteristics of the applications and VMs.

1. Cloud provider possesses constant number of VMs with homogeneous capacities. These capacities are four dimensional (namely CPU, memory, bandwidth and storage). It is not possible to add new VMs or resize existing ones.

2. Similarly, applications have the same four constant consumptions (requests).

3. Requests of the applications are foreknown.
4. Cloud provider receives one application at a time and after this application is assigned to a VM, it runs infinitely (i.e., it does not have a life span). However, it may temporarily suspend, migrate to another VM and continue its execution there.

These assumptions may seem too restrictive at first but our approach can be easily generalized to real world cloud scenarios. We use static values (VM capacities, counts etc.) because we aim to use current resources at hand in most efficient way. When these resources are almost fully utilized, SaaS provider possibly increases them, and our algorithm immediately adapts and starts to make decisions according to the new configuration. Similarly, when resource consumption changes over time or even terminates, new decisions will be made considering the current utilization. Third assumption of foreknown resource consumptions is based on the Service Level Agreements (SLAs) made between the cloud provider and cloud user where expected performance criteria is defined.

Fundamentally, the problem of optimally placing applications to VMs can be formulated as an NP-hard multidimensional bin packing problem and solved via optimization techniques such as linear programming (Zhang et al., 2010). However, it would be computationally expensive to run the optimization algorithm at each application arrival especially for more than a few VMs. Moreover, optimization would incur large number of application migrations. Our solution is to distribute applications to VMs using an intelligent heuristic until the point that a migration is inevitable (i.e. when no VM has enough capacity to accept incoming application). After this point, an optimization technique can be employed to rearrange the applications.

Contributions in this paper may be summarized as follows.

1. A Mixed Integer Programming (MIP) formulation of the application placement problem that minimizes the number of migrations
2. Novel heuristics to assign applications to VMs efficiently and intelligently
3. Comparison of the heuristics to the existing ones
4. Analysis of the heuristic approach on various VM counts and capacities

The following section gives an overview of previous work in resource allocation and application placement literature. Section 3 explains the details of our method including MIP formulation and heuristics, while Section 4 contains the experiments, their results and discussion. Finally, we conclude the paper with final remarks in Section 5.

2 RELATED WORK

Allocation of a cloud provider's resources to its customers is entitled as resource allocation, resource selection or application placement depending on the context and what is meant by the term resource. In this section, we review different approaches to the problem in the cloud computing literature. It should be noted that similar problems are encountered and extensively studied before the arrival of cloud computing (Urgaonkar et al., 2005). However cloud computing brings some unique challenges to the problem such as inclusion of network as a resource and as a request, geo-diversity and location limitations (jurisdiction), multi-tenancy, more heterogeneous resources and less predictable demands.

2.1 VM Assignment Strategies

Selection of physical machines to host virtual machines while fulfilling the requirements and optimizing resource usage is a problem that can be solved via optimization and approximation algorithms. Some examples use heuristics, linear programming, artificial intelligence, nature inspired computing and game theory (Endo et al., 2011).

In a recent study (Papagianni et al., 2013), the problem is formulated as two phases that must be optimized together: node mapping and link mapping. User request come in the form of VMs with defined capacities and their connectivity as a graph. The aim is to optimally assign these capacity request to physical machines (node mapping) and connectivity requests to network entities (link mapping). They apply a relaxed MIP approximation for the node mapping and shortest path (or minimum cost flow) algorithm for the link mapping in a coordinated way.

Another approach (Xiao et al., 2013), introduced the skewness metric to measure unevenness in the utilizations of various resources within a server. They aim to find a trade-off between overload avoidance and green computing concepts. Using a set of threshold based heuristics and a prediction algorithm they dynamically create a list of migrations that relieves overloaded servers to ensure QoS and evacuates underloaded ones to exploit green computing.

Following the improvements of the cloud systems, VM migration is being used to increase utilization of the virtual machines on the cloud infrastructure. (Yang et al., 2011) considers the different VM migration strategies adopted by the host machines that have different load states by taking into account four different resource types. In a more recent study (Wang et al., 2013), migration is performed regarding the

queue model of the time of application deployment requests from the clients. A centralized control management mechanism has been defined to inform the client which server is available at a time.

2.2 Application Assignment Strategies

Literature in the area of allocation within a VM is not as extensive as the one in allocation within a server, as the most studies consider resource allocation for the benefit of IaaS providers. A detailed model of SLAs is given in the study (Wu et al., 2011) which falls into the application placement category. SLAs between cloud providers and users are taken into consideration to quantitatively measure the service quality. They also suggest a simple heuristic: 'assign the request to the VM with the lowest (or the highest in other variation) utilization'.

In another study (Tang et al., 2007) optimal load balancing for dynamically changing demands is considered. Their approximation method aim to maximize satisfied demand and minimize application migrations. Proposed application placement controller estimates future demand and migrates applications accordingly to avoid overloading.

A recent work (Espadas et al., 2013) offers a resource allocation mechanism that exploits multitenancy and provides tenant isolation, load balancing and VM allocation. They also formally measure over and underprovisioning of VM resources.

3 APPLICATION PLACEMENT ALGORITHM

3.1 Overview

A high level pseudo code of both parts is given in Algorithm 1. In part I (lines 4 to 12), migrations are not allowed, and each arriving application is assigned to the VM determined by an heuristic of evenness. Goal of this part is to keep the utilization of four resources in a VM approximately even. By increasing the resource evenness of all VMs, overall utilization of the resources are maximized (Xiao et al., 2013).

To decide which VM is the best for a given application, it is transiently considered to be assigned to the VMs that has enough remaining capacity one by one and evenness is calculated. Then, the application is actually assigned to the VM with the best evenness value (line 12). At one point of the part I, incoming application's resource request will not fit into the remaining capacity of any VM (the application is not directly assignable and condition in line 11 is not satisfied). However, rejected application can still be assigned to a VM if some applications are re-assigned.

Part II (lines 13 to 16) migrates some of the applications and tries to make room for the new application. If part II succeeds to assign the application (line 16), part I of the algorithm continues to receive new applications; otherwise, it is certain that there is no placement that can assign received applications to the VMs and the algorithm terminates (line 17).

Algorithm 1: Pseudo code for the application placement strategy.

```
1  rejected ← false
2  while rejected = false do
3      receive application A
4      assignable ← false
5      foreach virtual machine VM do
6          if VM has enough capacity for A then
7              assignable ← true
8              assign A to VM
9              calculate unevenness of VM
10             remove A from VM
11     if assignable = true then
12         assign A to VM with min unevenness
13     else
14         run optimization algorithm
15         if optimization succeeds then
16             assign A and migrate others
17         else rejected ← true
```

One variation of part I can be obtained by calculating unevenness twice, i.e. before and after tentatively assigning the application. In that case, actual assignment is made to the VM whose unevenness decreases the most, instead of the one whose unevenness becomes the minimum.

3.2 Heuristics

We propose four heuristics to approximate the unevenness in the utilization of the resources on a VM. We also adapt the skewness heuristic originally suggested for assigning VMs to servers (Xiao et al., 2013). Finally we implement the greedy approximation algorithm, round-robin, to use as the baseline.

3.2.1 Standard Deviation (*SD*)

Standard deviation is a natural choice to find the unevenness of data points since it shows the amount of dispersion from the average. *SD* of the resource utilizations on a VM *v* can be calculated as follows.

$$SD(v) = \sqrt{\sum_{i=1}^{m} (r_i - \bar{r})^2} \quad (1)$$

Here, r_i denotes the utilization of the ith resource, while $\bar{r}$ is the average utilization of all resources of v. m is the number of resources.

3.2.2 Span (*SP*)

For the trivial case of only four instances, a simpler heuristic may be used. Since we are interested more in outliers than inliers, difference between the maximum and minimum utilization rates can be a good candidate for an unevenness heuristic.

$$SP(v) = \max_{i \in [1,m]} (r_i) - \min_{i \in [1,m]} (r_i) \quad (2)$$

3.2.3 Cumulative Difference (*CD*)

Another candidate is the total difference of the utilization rates of all resource pairs on v, as given by the formula below. This heuristic also considers the inliers similar to *SD*, but it is simpler.

$$CD(v) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{m} |r_i - r_j|}{2} \quad (3)$$

3.2.4 Cumulative Difference from Minimum (*DM*)

A similar heuristic to *CD* is to calculate differences from the minimum utilization rate. This approach tries to ensure that no resource is under-utilized (low outlier) relative to others, while previous approaches are trying to prevent both low and high outliers.

$$DM(v) = \sum_{i=1}^{n} \left(r_i - \min_{j \in [1,m]} (r_j) \right) \quad (4)$$

3.2.5 Skewness (*SK*)

Formulation of *SK* is given in (Xiao et al., 2013).

$$SK(v) = \sqrt{\sum_{i=1}^{m} \left(\frac{r_i}{\bar{r}} - 1 \right)^2} \quad (5)$$

3.2.6 Round-Robin (*RR*)

Round-robin algorithm is a very straightforward solution to the application placement problem. For each application, the greedy algorithm attempts to assign it to the next VM. If that VM is not available, consecutive ones are tested in order. In the ideal case where all VMs are always available, it assigns equal number of applications to each of them. Naturally, it does not consider the evenness of resource types within a VM as opposed to the heuristics mentioned above.

3.3 MIP Formulation

To determine an optimal application placement scheme, we use MIP techniques. We minimize the number of migrations when determining the place (VM) of each application from the pool of VMs. In achieving this, for each solution we compare the place of each application with its former place and try to maximize the number of applications that don't change their place.

We have used the native library lp_solve and Java ILP API to solve the MIP problem introduced earlier. We have simulated four VMs residing in a host machine which let us to run our simulations on a decent personal computer. Heavier analysis on a large number of VMs residing in federated cloud environments can also be simulated, however MIP performance exponentially decrease for such cases.

3.3.1 Variables

When using MIP, it is required to build formulae that represents the constraints of the system and an objective function to be optimized during the process. In constructing such formulae variables that represent the overall properties of the system shall be used. Below, the variables we used in our model and their meanings are explained briefly:

- #VMs: Number of VMs present in the cloud environment.
- #Apps: Number of applications to be placed upon the present VMs.
- oldAsgn: A boolean matrix that holds the present assignment of each application upon a VM.
- newAsgn: A boolean matrix that holds the resulting assignment by MIP.
- resNeed: An integer matrix that holds the amount of resource needed by each application for the four different type of resources mentioned before.
- resAv: An integer matrix that holds the amount of resource available for each VM and each resource.

3.3.2 Objective Function

In modeling resource consumption for a cloud environment, we provide the MIP solver with Equation 6 to maximize. We try to maximize the sum of old and new application-VM assignment products which produce a value of 1 if the assignment didn't change and 0 if a migration is present.

$$\sum_{i=0}^{\#VMs} \sum_{j=0}^{\#Apps} (\text{newAsgn}[i][j] \times \text{oldAsgn}[i][j]) \quad (6)$$

3.3.3 Constraints

Naturally, we also apply a number of constraints in order to drive the MIP to produce meaningful results. In Equation 7 we guarantee that each application has been assigned to exactly one VM. Additionally in Equation 8 resource needs of each application that has been assigned to a specific VM for each type of resources are summed up to be less than the available resource assigned to the VM.

$$\bigwedge_{i=0}^{\#VMs} \left(\sum_{j=0}^{\#Apps} \text{newAsgn}[i][j] = 1 \right) \quad (7)$$

$$\bigwedge_{i=0}^{\#VMs} \bigwedge_{j=0}^{\#Res} \left(\left(\sum_{k=0}^{\#Apps} \text{newAsgn}[i][k] \times \text{resNeed}[j][k] \right) \leq \text{resAv}[i][j] \right) \quad (8)$$

4 EXPERIMENTS

A simple simulator is implemented to conduct the experiments explained in this section. Simulator generates applications with uniformly random resource requests within a given range and manages VMs with user defined count and capacity. For all experiments, applications are generated with resource requests within the range of $[0,30]$ units while VM count and capacities varied.

4.1 Postponement of Migration

This first experiment aims to compare the power of heuristics to assign maximum number applications before a migration is necessary. As mentioned before, migrations should be avoided since they may cause temporary downtimes for an application. So, a good heuristic should place applications in a way that migrations become compulsory as late as possible.

4.1.1 Experimental Setup

Part II of the algorithm is not required for this experiment since we are only interested in the performance of the heuristics before the optimization starts. Excluding the computationally expensive part of the algorithm allowed to test heuristics for a great number of configurations and repetitions. Two variants (minimum and most decreasing) of five heuristics as well as round-robin are compared on 100 different VM configurations. This consists of 10 VM counts ranging from 3 to 12 and 10 VM capacities ranging from 75 to 300 units per resource.

All 11 strategies are run on each configuration 30.000 times to avoid randomness of applications affect the results. Execution of a strategy on a configuration is stopped when no VM is available for the received application. After that, the total number of applications assigned to all VMs are logged and the average of 30.000 runs are calculated.

4.1.2 Results and Discussion

Due to the great number of configurations tested in this experiment, it is not possible to include all results here. We instead present five configurations with a fixed VM count (8 VMs) on the left half of the Table 1 and five configurations with a fixed VM capacity (200 units per resource) on the right half. Remaining 90 configurations follow the same pattern as them.

For each configuration, Table 1 contain average number of applications assigned by 11 strategies, expected number of applications to fully utilize VMs and the rate of improvement by the best performing strategy in comparison to the baseline (*RR*). Subscripts *min* and *dec* stand for the minimum and most decreasing variants of the strategies, respectively.

Our results indicate that in 477 of 500 cases (95,4%), *min* variant outperformed the *dec* variant of the same strategy. In most of the 23 cases where *min* variant performed worse, tiny VM capacities and the *DM* strategy are used. This clearly demonstrates that arriving application should be assigned to the VM that becomes the most even with the assignment, instead of the one whose evenness increases the most.

It is also evident that intelligent heuristics always perform better than the greedy assignment. In all 100 configurations, all 10 strategies manage to assign more applications than the round-robin. The worst improvement rate is $4,3\%$ (3 VMs with 300 units of capacity per resource) while the best one is $12,1\%$ (12 VMs with 75 units of capacity per resource). Utilization rate of round-robin is between $73,8\%$ and $90,2\%$ while utilization rate of the heuristics is between $80,5\%$ and $96,4\%$.

Improvement rate gets better as the number of VMs increase (as seen in the right half of the final row in Table 1) but their capacities decrease (as seen left half of the same row). With high capacities and small number of VMs, randomness of resource requests tend to cover imperfect assignments while with low capacities and large number of VMs, decisions are more critical and should be made with more care. In cases that are ideal for cloud computing benefits (i.e. many distributed VMs with relatively low capacities), heuristics show their real strength.

Table 1: Partial result of the application count experiment.

VM Capacity	**100**	**150**	**200**	**250**	**300**	**200**	**200**	**200**	**200**	**200**
VM Count	**8**	**8**	**8**	**8**	**8**	**4**	**6**	**8**	**10**	**12**
RR	42,0	67,2	92,7	118,4	144,3	46,0	69,3	92,7	116,2	139,7
SK_{min}	45,8	72,3	98,8	125,3	151,9	48,1	73,4	98,8	124,3	149,9
SK_{dec}	45,5	72,0	98,6	125,2	151,9	48,1	73,2	98,6	124,1	149,7
SP_{min}	**46,2**	**73,2**	**100,2**	**127,2**	**154,3**	**48,7**	**74,4**	**100,2**	**126,2**	**152,3**
SP_{dec}	46,0	72,6	99,3	126,0	153,2	48,4	73,8	99,3	124,9	150,7
SD_{min}	**46,2**	**73,2**	**100,2**	**127,3**	**154,3**	**48,7**	**74,4**	**100,2**	**126,2**	**152,3**
SD_{dec}	45,8	72,3	98,9	125,5	152,1	48,2	73,4	98,9	124,4	150,1
CD_{min}	**46,2**	**73,2**	**100,2**	**127,3**	**154,3**	**48,8**	**74,4**	**100,2**	**126,2**	**152,3**
CD_{dec}	45,9	72,5	99,1	125,9	152,7	48,3	73,7	99,1	124,8	150,5
DM_{min}	45,7	72,6	99,6	126,5	153,6	48,3	73,8	99,6	125,4	151,5
DM_{dec}	45,8	72,5	99,2	126,2	153,1	48,3	73,6	99,2	125,0	150,8
Expected	53,3	80,0	106,7	133,3	160,0	53,3	80,0	106,7	133,3	160,0
Improvement	**10,0%**	**8,9%**	**8,1%**	**7,5%**	**6,9%**	**6,1%**	**7,4%**	**8,1%**	**8,6%**	**9,0%**

Since it is already demonstrated that heuristics make better assignments than the greedy algorithm and the *min* variants are better than *dec* variants, we can finally make a comparison among them. In trivial cases where capacity per resource is less than 100 units, all heuristics perform more or less the same, while in all other cases *SP*, *SD* and *CD* perform significantly better than *SK* and *DM*.

Although being still much better than greedy, skewness algorithm is outperformed by the best three heuristics by $1,2\%$ on average. Our observations show that both variants of *SK* are sensitive to average resource utilization of the VMs. As a result, they tend to assign new applications to the VMs that contain more applications although better alternatives in terms of evenness are available. This causes their placements resemble greedy first-fit algorithm.

4.2 Minimization of Migration Count

Goal of this experiment is to assess the quality of application placement of the strategies. A good placement is expected to allow assignment of a new application after a small number of migrations, or in the ideal case, fully utilize VMs without any migrations. It should be noted that the objective of the optimization algorithm is to minimize the number of migrations so the migration counts are guaranteed to be the smallest possible values.

4.2.1 Experimental Setup

Both part I and II are included in this experiment and that forces us to decrease repetition count. That's why, only the best three heuristics of the first experiment, round-robin and skewness algorithms are executed for a single configuration: three VMs with capacities of 300 units per resource. Greater number of VMs causes a significant increase in running time.

An execution is stopped on the following two conditions: (1) In part I, when remaining capacities of all VMs are less than the expected value of the four resource requests (i.e., 15 units), (2) In part II, when the optimization algorithm runs once. We call the first condition a perfect placement since all VMs are almost fully utilized without any migrations. Second condition is only considered when a perfect placement is not achieved and the algorithm continues to part II. In that case, if the algorithm succeeds to assign the application to a VM, number of migrations are logged. Otherwise, if the application is rejected, it is also considered as a perfect placement because MIP proved that it was impossible also for the heuristic to make that placement.

Each strategy is executed 1.000 times calculating the average number of migrations and the number of perfect placements. Number of migrations for a perfect placement is regarded zero.

4.2.2 Results and Discussion

Table 2 gives the average number of migrations introduced by the optimization after each five strategies (*RR*, SK_{min} and three best performers of experiment 1) failed to assign any more applications. The table also contains the number of cases (among 1.000 executions) where heuristics made a perfect placement so an optimization was not required.

Results demonstrate that placements made by the heuristics allow assignment of new applications with less changes since SD_{min}, SP_{min} and CD_{min} strategies required $34,5\%$, $33,3\%$ and $31,0\%$ less migrations than *RR*, respectively. Similar to the first experi-

Table 2: Results of the migration count experiment.

Strategy	Avg. Migr. Count	Perfect Count
RR	8,4	26
SD_{min}	5,5	108
SP_{min}	5,6	100
CD_{min}	5,8	86
SK_{min}	7,2	88

ment SK_{min} performed somewhere in the middle with $14,3\%$ better than *RR* but $27,8\%$ worse than the others in average. SD_{min} and SP_{min} made perfect placements in $10,8\%$ and $10,0\%$ of executions, followed by SK_{min} with $8,8\%$ and CD_{min} with $8,6\%$. Coincidental perfect placements are made by *RR* in only $2,6\%$ of cases.

It is clear that intelligent heuristics make better placements that can be optimized with significantly less migrations and that are already optimum in roughly four times more incidents than the greedy approach. Suggested heuristics also outperform SK_{min} clearly with the average number of migrations. As a result, they allow applications to run with less suspension as well as decreasing the need for time consuming optimization algorithm.

Due to its simplicity, we choose SP_{min} heuristic to represent heuristic approach in the future performance analysis.

4.3 Analysis on Constant Total Capacity

As explained before, partitioning total hardware capacity to higher number of virtual machines provides benefits of cloud computing but reduces overall utilization. This reduction can be especially significant when different application types are not combined appropriately and a certain resource of a VM runs much shorter than the others. In this experiment, we aim to observe utilization performance of our algorithm in comparison to greedy approach given various number of VMs sharing a fixed hardware capacity.

4.3.1 Experimental Setup

SP_{min} heuristic which is one of the best performing and simplest strategies, is compared to the greedy round-robin algorithm using 3 to 12 VMs. The VMs share a total capacity of 1.000 units per resource. Similar to the first experiment, part I of the algorithm is executed 30.000 times for both strategies to calculate average count of applications that are assigned. Executions are stopped when a migration is necessary.

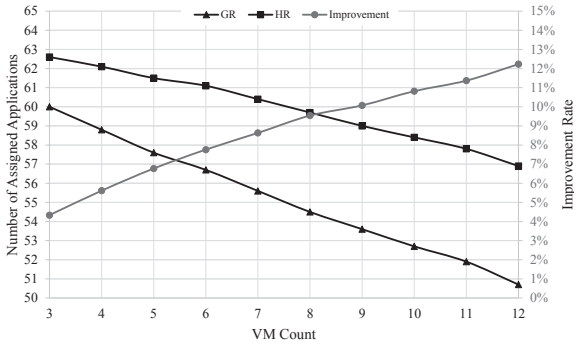


Figure 1: Results of the constant capacity analysis of the heuristic approaches (*HR*) in comparison to gready (*GR*).

4.3.2 Results and Discussion

Chart in Figure 1 displays the number of applications that are assigned by heuristic (SP_{min}) and greedy (*RR*) approaches to the same constant total capacity that is partitioned to varying number of VMs. It also shows the rate of improvement provided by the heuristic strategy on its secondary y-axis.

As expected, both strategies assign less applications as the number of VMs increase, however the rate of decrease is different. *RR* starts with $60,0$ applications and encounters a decrease to $50,7$ applications with a slope of $-1,03$. SP_{min}, on the other hand, starts with slightly more applications, $62,6$, and ends around $56,9$ with a less steep slope around $-0,63$.

Other heuristics are also tested but the results are omitted since their performance is nearly as good as SP_{min}. That's why we can consider this experiment as a comparison of heuristic and greedy approaches rather than SP_{min} and *RR*. Experimental results indicate that using a more intelligent heuristic instead of the greedy algorithm slows down the decrease in resource utilization while partitioning increases. Accordingly, improvement provided by heuristics gets more significant. This is especially important when we consider that benefits of cloud computing becomes more evident with higher number of VMs.

5 CONCLUSION

In this paper we propose a method for assigning applications with multi-dimensional resource requests to virtual machines with finite capacity. The method employs heuristics and mixed integer programming to optimally solve the application placement problem. Our goal is to allow SaaS providers to host maximum number of applications with their resources at the same time increasing their service quality.

While optimization algorithms such as our MIP

formulation guarantee to assign maximum number of applications, they decrease QoS in two ways: (1) They are extremely time consuming for high number of instances so they delay the decision making process while the request is pending, (2) Their decisions cause migrations of applications which require temporary downtimes.

Suggested heuristics approximate to the optimal placement before the optimization is inevitable. They do so by combining workloads together to keep utilization of resources as even as possible. Previously suggested unevenness metric, skewness, fails to yield good combinations while greedy algorithms such as round-robin do not consider evenness at all.

Heuristics are evaluated via a simple simulator that supports random generation of demands and maintains VM capacities. Our experiments demonstrate that heuristics provide the following improvements to QoS in comparison to greedy approximation.

1. They make the optimal placement and almost fully utilize VMs up to $10,8\%$ of cases. This is four times more than the rate provided by the greedy algorithm. In such cases, service quality is preserved since the execution of MIP algorithm and application migrations are not required.

2. In the rest of the cases, they delay the requirement for MIP algorithm up to $12,1\%$ applications. This means more applications can be accepted without migrations and any effect on service quality.

3. When the execution of MIP algorithm is mandatory, their placements require up to $34,5\%$ less application migrations causing less applications to suspend and less harm to service quality.

As future work, we aim to reproduce our results on a state of the art cloud computing simulator and to test our heuristics on other scenarios. These include other random distributions of resource requests and dynamic resource demands.

ACKNOWLEDGEMENTS

This work is performed in joint with “mCloud” project of Simternet Iletisim Sistemleri Reklam San. ve Tic. Ltd. Sti. “mCloud” project is supported by The Scientific and Technological Research Council of Turkey (TUBITAK) – TEYDEB project number 7130115.

REFERENCES

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. (2010). A view of cloud computing. *Commun. ACM*, 53(4):50–58.

Buyya, R., Broberg, J., and Goscinski, A. M. (2011). *Cloud computing: Principles and paradigms*, volume 87. Wiley. com.

Endo, P., de Almeida Palhares, A., Pereira, N., Goncalves, G., Sadok, D., Kelner, J., Melander, B., and Mangs, J.-E. (2011). Resource allocation for distributed cloud: concepts and research challenges. *Network, IEEE*, 25(4):42–46.

Espadas, J., Molina, A., JiméNez, G., Molina, M., RamíRez, R., and Concha, D. (2013). A tenant-based resource allocation model for scaling software-as-a-service applications over cloud computing infrastructures. *Future Gener. Comput. Syst.*, 29(1):273–286.

Papagianni, C., Leivadeas, A., Papavassiliou, S., Maglaris, V., Cervello-Pastor, C., and Monje, A. (2013). On the optimal allocation of virtual resources in cloud computing networks. *Computers, IEEE Transactions on*, 62(6):1060–1071.

Tang, C., Steinder, M., Spreitzer, M., and Pacifici, G. (2007). A scalable application placement controller for enterprise data centers. In *Proceedings of the 16th international conference on World Wide Web*, WWW ’07, pages 331–340, New York, NY, USA. ACM.

Urgaonkar, B., Shenoy, P., Chandra, A., and Goyal, P. (2005). Dynamic provisioning of multi-tier internet applications. In *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, pages 217–228.

Wang, Y., Chen, S., and Pedram, M. (2013). Service level agreement-based joint application environment assignment and resource allocation in cloud computing systems. In *Green Technologies Conference, 2013 IEEE*, pages 167–174.

Wu, L., Garg, S., and Buyya, R. (2011). Sla-based resource allocation for software as a service provider (saas) in cloud computing environments. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 195–204.

Xiao, Z., Song, W., and Chen, Q. (2013). Dynamic resource allocation using virtual machines for cloud computing environment. *Parallel and Distributed Systems, IEEE Transactions on*, 24(6):1107–1117.

Yang, K., Gu, J., Zhao, T., and Sun, G. (2011). An optimized control strategy for load balancing based on live migration of virtual machine. In *Chinagrid Conference (ChinaGrid), 2011 Sixth Annual*, pages 141–146.

Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18.

On Optimizing Resource Allocation and Application Placement Costs in Cloud Systems

Cihan Seçinti and Tolga Ovatman
Department of Computer Engineering, Istanbul Technical University, Maslak, Istanbul, Turkey
secintic@itu.edu.tr, ovatman@itu.edu.tr

Keywords: Virtual Machine Resource Allocation, Application Live-migration, Utilization Optimization.

Abstract: Resource utilization problem has been widely studied for cloud systems where a number of virtualized resources are shared among applications hosted as services. Resource utilization optimization can be confronted at two levels: allocating resources to virtual machines(VM) across physical machines or assigning applications to virtual machines present in the host. With the improving capabilities on virtualization technologies, realizing resource allocation at both levels is becoming more viable using VM reconfiguration and live-migration of applications. In this paper we investigate applying resource allocation optimization at these two levels and the emerging trade-off in deciding the appropriate technique to be used. We first analyze the effect of gradually increasing the amount of resources assigned to a virtual machine using VM reconfiguration and compare our results with fully assigning host's resources without reconfiguration. Later, we investigate the amount of utilization revenue when application live-migration is used for applications having smaller/larger performance needs. Finally, we compare the host utilization for different amounts of cost rates between live-migration and reconfiguration. Consequently our analysis results identify the cost rate and application granularity levels where it is optimal to apply live-migration or VM reconfiguration.

1 INTRODUCTION

The tremendous increase in employment of cloud systems through the recent years triggered the evolution of software towards cloud services. As the technological advancement continue improving the capabilities of virtualization and application hosting, it became viable to develop more efficient application placement strategies that can be used in provisioning cloud systems. Latest developments in virtual machine(VM) technology and application live-migration techniques added new dimensions to the problem of managing hosts' resources effectively.

Virtual machine(VM) reconfiguration can be described as the process of changing the amount of resources(e.g. CPU, memory, bandwidth, storage) assigned to the VM by the host(Verma et al., 2010). VM reconfiguration can simply be performed by shutting down the VM and tuning the amount of resources allocated for the VM. However some resources, like bandwidth and storage, are reconfigurable without shutting down VM and additionally recent advances are enabling "hot add" features for CPU and memory reconfiguration. This situation reveals on-the-fly resource allocation problem between the host and VM layers in a cloud system.

On the other hand, application live-migration is another development supported by most of the VM managers which enables moving a running application from a virtual machine to another without disconnecting the client or application(Clark et al., 2005). Application live-migration includes warm-up, copying and memory migration phases and requires a substantial amount of VM down-time during live-migration process. Being able to migrate applications with different levels of SLA requirements also reveals the problem of resource allocation by moving applications between VMs to utilize the host more effectively.

Although the resource management problems described above are encountered at different layers in the cloud system, both situations directly effect the resource utilization of the host. Moreover they both have their own costs(e.g., downtime, processing) which may change according to the technology and/or VM type being used.

In this paper, we investigated the resource allocation problem in both layers and reveal the situations where it is meaningful to apply VM reconfiguration or application live-migration. For VM reconfigura-

tion we compared the case where available resources are assigned equally among the available VMs and the resources needed by a VM is gradually assigned to the VM as the applications continue to be deployed. We also investigated the effect of amount of initial resources assigned to VMs over the amount of final utilization of the host. We found out an initial resource allocation threshold can be identified where the final utilization starts to decrease as the initial resources assigned to each of the VMs start to increase.

We also analyzed the effect of application live-migration over the final utilization of the host with respect to the granularity of the applications being deployed to the VMs. We used round robin placement and mixed integer linear programming(MILP) in deciding the placement of applications having resource requirements inside certain intervals. This way we analyze if the effect of live-migrating smaller/bigger applications make any difference in utilization of the host.

Figure 1 summarizes the approach being used throughout the paper. The provisioning engine is responsible of resource allocation to virtual machines and carrying on application placement decisions. As increasing number of applications are being submitted to the host machine provisioning engine may migrate applications across VMs to generate additional resources. We use MILP to check for a possible applcation migration scheme that can discover necessary amount of resources. If this solution doesn't work additional resources can be assigned from the host to VMs if present. Our further analysis focus on making the decision of application migration versus ressource addition to VMs by reconfiguration. The tradeoff is analyzed regarding different rates of costs between migration and reconfiguration processes.

As mentioned above both of the resource allocation activities(migration and reconfiguration) described above have their own costs when applied. In our last stage of experiments we investigated the effect of different cost rates between these activities and discussed the thresholds when it becomes meaningful to apply each of these activities. Our results reveal the certain situations that can be used in provisioning process during the decision of using VM reconfiguration or live-migration technologies. Also by analyzing the trade-off between applying these technologies under certain cost rates we were able to explore conditions that can aid in deciding in which layer(host-VM, VM-application) it is meaningful to apply on-the-fly resource allocation.

In the next section, we continue by summarizing the related work in the literature. In Section 3 we explain the trade-off between VM resource allocation versus application live-migration in more detail and present the MILP model used in decision of application migrations in more detail. Section 4 presents the details of the experiments performed and a discussion of the results. Finally in Section 5 we conclude the study and present possible future directions.

2 RELATED WORK

Resource allocation is an important concept on systems that exhibits frequent application deployment. Nowadays, with development of cloud systems resource allocation regain importance on the problem of increasing utilization of the servers that are being run on the cloud infrastructure. Even before the advances in cloud computing resource allocation is vastly studied for clusters and data centers(Bennani and Menasc, 2005; Zhu and Singhal, 2001).

Migration algorithms can be applied to increase utilization of systems having a diverse range of characteristics. There were several studies based on increasing live-migration performance. In a recent study(Kikuchi and Matsumoto, 2012), live-migration is considered to be a serious factor in causing the response time degradations for applications. Thus, live-migration operations should be controlled to minimize the effect on the performance of applications running on the cloud system. In the study, the impact of transmission control protocol's behavior on live-migration is experimented per se to estimate the best live-migration strategy to be used. In our study instead on performing complicated estimation computations we have focused on detecting thresholds based on relative ratios which allows to perform decisions before the deployment process in a simple way.

Following the improvements of the cloud systems, live-migration is being used to increase utilization of the virtual machines on the cloud infrastructure and different algorithms are developed to increase usability of live-migration method. Yang(Yang et al., 2011) considers the different VM(and hence application) migration strategies adopted by the host machines that have different load states by taking into account four different resource types. The migration is done considering the amount of resource allocated to each VM, however the amount of resource utilization has not been taken into account. In a more recent study(Wang et al., 2013), migration is performed regarding the queue model of the time of application deployment requests from the clients. A centralized control management mechanism has been defined to inform the client which server is available at a time. However, Wang et al. assumed each request to be asigned to a

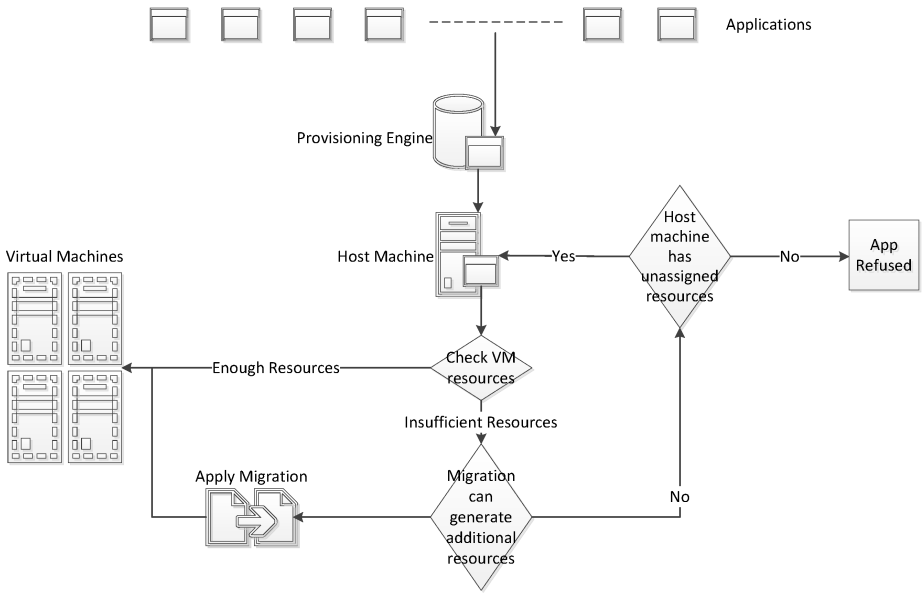


Figure 1: Overall MILP based provisioning process.

new virtual machine which ignores multiple applications deployed to a common virtual machine.

On the other hand, cloud reconfiguration research is mostly based on physical machine reconfiguration. In this particular area, the challange is more focused on figuring out how to place virtual machines to physical machines. In a recent study by He et al. it is claimed that utilization of the physical machines would be increased when the reconfiguration is applied to physical machines(He et al., 2012). In their study they focused on virtual machines placement on physical hosts rather than taking into account application placement as well.

There are a few studies which examine the tradeoffs between reconfiguration and migration technologies. A study by Chen et al. proposes a framework that uses live-migration to reduce cost of resource reconfiguration(Chen et al., 2012). They conduct experiments to illustrate the performance of the framework in maximizing utilization and reducing cost of the runtime reconfiguration. Also they propose a two-level runtime reconfiguration strategy to reduce virtual machine live-migration and shorten the total live-migration time. Earlier studies by Verla et al. examine the performance impact of reconfiguration in cloud systems and build a model to predict the duration and performance effects of such activities (Verma et al., 2010; Verma et al., 2011).

In our study we also focus on maximizing performance by considering the effect of migration and reconfiguration under different VM and application resource consumption characteristics. We have adopted the strategy to determine resource requirements of applications in a random way limited inside certain intervals. In the study by Hwang et al. it is considered that the exact resource requirements of any application could not be found easily(Hwang and Pedram, 2012). Therefore, calculating resource requirements in a randomized way is not a trivial approach. They also argue that by determining application resources randomly, error calculation can be done depending on standard deviation of randomized variables. We adopt this strategy by limiting our random resource determination process under certain limits which we use to represent the characteristics of the application size. This way we were able to fuzzily quantize the application sizes in our experiments.

As a final case to mention, we have used MILP techniques in deciding the optimal placement of application among VMs in our experiments. Utilization of ILP techniques is a known approach in resource optimization of cloud systems. A study by Borovskiy et al. treats the workload distribution issue as a set partitioning problem and devise a formal approach based on integer programming techniques(Borovskiy et al., 2011). In an earlier study a similar ILP based approach was applied on workload scheduling (Van den Bossche et al., 2010). Later, Alvarez et al. applied ILP techniques in data allocation problem by taking into account stroage and computational costs (Ruiz-Alvarez and Alvarez, 2012). Although we haven't dig into a single type of resource allocation problem deeply, we have built a simple model regarding four different types of resources altogether.

3 RESOURCE ALLOCATION AND APPLICATION LIVE-MIGRATION

An obvious problem in assigning a host's resources to a number of VMs is fragmentation caused by varying resource requirements of applications that are going to be placed in VMs. A straightforward strategy might be allocating all the resources present in the host equally among the VMs once and for all which requires no more reconfiguration in the future. A more sensible approach can be assigning a certain amount of host resources equally to the VMs and allocating remaining resources in appropriate amounts as the VMs require more resources in order to accept incoming applications. However this solution requires a reconfiguration cost, which may include shutting down a VM for a certain amount of time.

Another solution for more effective resource allocation is to use application live-migration. In this case, a reallocation of the present applications residing in VMs needs to be computed which contains two distinct challenges. The first challenge is leveraging the cost of VM down-time during the migration process where the second one is exploring a new placement for applications that contain minimal number of application migrations. Live migration of an application can be achieved in two different ways. Each VM in the system can be affiliated with a single application so that the application migration is achieved by migrating VMs across hosts. Another possibility is to redeploy the application to another VM which alsomakes it possible for a VM to host multiple applications. Our analysis consider the second case however the first case can also be supported by minor modifications by slightly transforming the problem from application migration among VMs to VM migration among hosts.

It can be crucial to decide which strategy to be used in resource allocation by considering different aspects of the system. These aspects include resource needs of the applications at hand as well as the cost rate between VM reconfiguration and application live-migration. In our experiments we use cost ratios in detecting the optimal results since each type of cost can change over time and VM type being used in the host system. By using profiling techniques the cost rate can be determined and appropriate strategy can be chosen accordingly. Moreover a priori knowledge on the resource consumption behavior of the applications can be used in deciding any of those resource allocation approaches as well. This a priori knowledge do not need to be very detailed; identifying applications as heavily resource consuming or lighter should be enough.

Before continuing to experiments performed to analyze the mentioned trade-offs we would like to explain an important step that takes a crucial role in determining migration costs. As mentioned above, selecting the migrating applications in determining a better placement for applications that reside in the host plays a key role in determining live-migration costs. The determined placement should include minimum number of application migrations in order to achieve a reordering process with minimal cost. In our experiments we mainly focus on cost rates between migration and reconfiguration, thus we have chosen to use symbolic costs for each of the concepts and perform our analysis based on different cost rates instead of specific cost values.

In order to determine an optimal application placement scheme we have used MILP techniques. By using MILP we minimize the number of migrations when determining the place of each application to the pool of VMs. In achieving this, for each solution we compare the place(VM) of each application with its former place and try to maximize the number of applications that doesn't change their place. To perform this, we provide MILP with Equation 1 to maximize. In Equation 1 i index is used to select among VMs and j index is used to select among applications. Since an application is either assigned or not to a VM we used assignment matrices of boolean variables. We try to maximize the sum of old and new application-VM assignment products which produces a value of 1 if the assignment didn't change and 0 otherwise if a migration is present.

$$\sum_{i=0}^{\#VMs}\sum_{k=0}^{\#Apps}(\text{newAssign}[i][k]\times\text{oldAssign}[i][k]) \quad (1)$$

In addition we also do apply a number of constraints in order to drive the MILP to produce meaningful results. In Equation 2 we guarantee that each application has been assigned to only one VM. Additionally in Equation 3 resource needs of each application that has been assigned to a specific VM for each type of resources are summed up to be less than the available resource assigned to the VM.

$$\bigwedge_{i=0}^{\#VMs}\left(\sum_{k=0}^{\#Apps}\text{newAssign}[i][k]=1\right) \quad (2)$$

$$\bigwedge_{i=0}^{\#VMs}\bigwedge_{j=0}^{\#Res}\left(\left(\sum_{k=0}^{\#Apps}\text{newAsgn}[i][k]\times\text{resNeed}[j][k]\right)\leq\text{resAv}[i][j]\right) \quad (3)$$

In our experiments we have used four different types of resources which are CPU, memory, bandwidth and storage requirements. We also use a total

of four VMs which greatly reduces the computational complexity of the MILP. A more detailed discussion of the experimental environment is presented in the following section.

4 EXPERIMENTS ON OPTIMIZING PROVISIONING TRADEOFF

4.1 Experimental Setup

Our experiments on VM reconfiguration and application live-migration initially focus on separate aspects about applying resource allocation using each technique respectively. Initially, we focus on the difference in the host utilization when application migration is used for various application resource consumption characteristics. Secondly, we try to pinpoint the amount of initial resources need to be assigned to VMs to increase the final utilization of the host. And finally we combine both of our findings and analyze when it is meaningful to use each technique regarding the cost rate between migration and reconfiguration.

During our experiments we have used some assumptions which can be listed as follows:

- We have used four kinds of resources in our simulations: CPU, memory, bandwidth and storage. Each of our applications have four constant values indicating their resource need for a specific kind of resource. These values may point to SLA requirements of the applications need to be hosted by the cloud system.
- Each application's resource needs are set independently using a uniform random distribution. On the other hand in our experiments we produce those random values inside a number of intervals resembling cloud systems with applications in different granularity levels. For instance in Figure 2, 10 different granularity levels are used where randomly produced application resource needs vary around 5, 10, 15, ... units respectively.
- An important assumption is an application can be deployed on the cloud or migrated but it is not undeployed. A more detailed simulation can be performed by producing the incoming and leaving applications using a predefined distribution. In our simulations we mainly focus on fully utilizing the host so we only handle the case where applications never leave the host machine.

Our simulations are developed using JAVA programming language. We have used the native library lp_solve and Java ILP as API to solve the MILP problem introduced earlier. We have simulated four VMs residing in a host machine which let us to run our simulations on a decent laptop. In our simulations we have used POJOs to resemble hosts and virtual machines involving integer variables(resource variables) that represent the resources being used in simulations. Applications are added to system as discrete events causing the resource variables to be decreased according to the resource consumption values of the applications. The resource consumption values are determined randomly inside specific intervals. These intervals are indicated in the result charts presented later in the paper. Heavier analysis on a large number of VMs residing in federated cloud environments can also be simulated however MILP performance exponentially decrease for such cases.

4.2 Experiment Results

Our first experiment consists of assigning applications having different resource needs to the equally sized VMs in a round robin manner. When using round robin placement the applications are placed in VMs following a predefined order. This order can be determined using priorities but in our case all the virtual machines are assumed to have the same priority. When placing an application, if the next VM determined by the round robin is out of resources the application will be placed to the next VM in line. This process is carried on until an appropriate VM is found, otherwise the application is rejected due to the lack of resources.

In Figure 2 we consider using MILP to select a set of applications to be migrated across VMs when round robin algorithm is unable to find enough resource for any of the VMs to satisfy SLA needs of the submitted application. X-axis represents the intervals which is used to produce random values for application resource needs and y-axis shows the improvement of the host utilization before an application is rejected because of the unavailability of resources. The values in the y-axis are the ratio between pre-migration and post-migration host utilization. The intervals in the x-axis is determined resembling a sliding window interval. In each interval the median of the produced values is shifted 5 units, which we believe is granular enough to analyze the variations in the utilization.

For each set of experiments applications are produced with random resource needs inside the specific interval and assigned to an available VM using round robin. In one set of experiments we have also applied MILP and application migration after round robin is

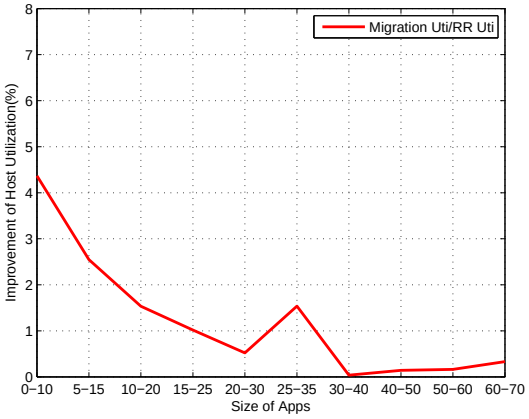


Figure 2: The improvement in host utilization when migration is used for different sized applications.

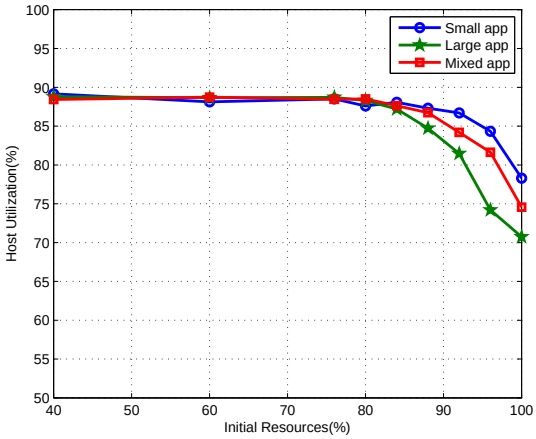


Figure 3: The impact of initial VM resource assignment over host utilization.

unable to find a VM with necessary amount of resources available. In our experiments we have used VMs having 125 units of resources available, so it can be seen that migration is applicable when the application sizes being deployed to the system is less than about 25% of VM sizes. The peak in 2535 point results from the uniform distribution around 30 unit application size being a factor of the 120 unit resource reserved for the host.

Our second experiment is to reveal the amount of initial resources that needs to be assigned to each VM in order to perform a better resource allocation. In this experiment instead of allocating all the resources equally among the VMs we start by allocating a predetermined amount of resource equally among VMs. Later when any of the VMs becomes full, instead of rejecting an application required amount of resources is granted to the VM so that the application can be deployed. This gradual increase continues until the host runs out of resources and reject incoming application. In Figure 3 we compare the amount of utilization of the host when the first application is rejected for applications with different resource needs. The applications used in the experiment can be either small(requiring 0%-25% of each resource), large(requiring 25%-50% of each resource) or randomly mixed applications of both. We can see that independent of the application size the host utilization starts to decrease after the case where 84% of initial resources are assigned.

Our final experiment compares the costs of application migration and VM reconfiguration. We have used only small applications(requiring 0%-25% of each resource) and compare the total cost for different cost rates assigned to reconfiguration and migration cases. Instead of using real cost values we rather chose to iterate over cost rates because cost values can vary significantly based on the hardware and software configurations of the system and technologied neing used. However cost rates can be used independently of these concepts because there should always be a rate between migration cost and reconfiguration cost.

In Figure 4 we indicate two thresholds which constrains the use of each concept during resource allocation process. The first threshold is the vertical initial resource allocation threshold which is the boundary when the total utilization starts to decrease, in other words it becomes less meaningful to apply reconfiguration. The horizontal threshold is the cost induced by solely using migration without reconfiguration, which marks the boundary where it becomes less feasible to apply migration and reconfiguration at the same time. A logical decision is to use the lower left region in the figure to use both concepts collaboratively.

Finally we would like to compare the performance of MILP in terms of host utilization and execution time for different number of VMs andusing small scale applications. In Figure 5 utilization achieved by applying MILP based replacement of the applications is compared with the execution time based performance of MILP. As expected, MILP performance is degraded as the number of VMs increase since there exist a larger number of possibilities to be checked during optimization process. However because of the same reason the optimal placement of applications on the VMs increases the host utilization further as the number of VMs increases. There exist a larger number of possible VMs to place the applications upon causing a better application scheme to be found by the MILP.

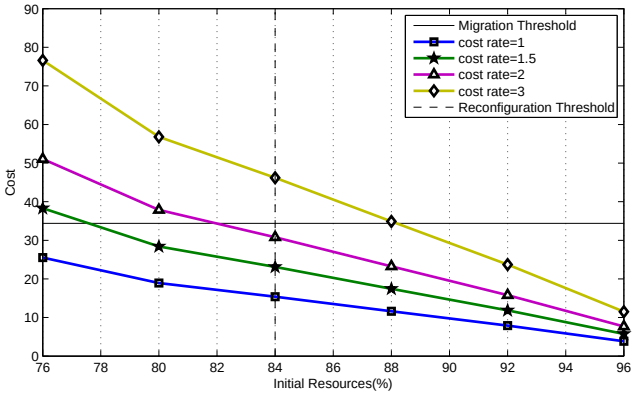


Figure 4: The trade-off between cumulative migration-reconfiguration cost and initial resource assignment for different cost rates between migration and reconfiguration.

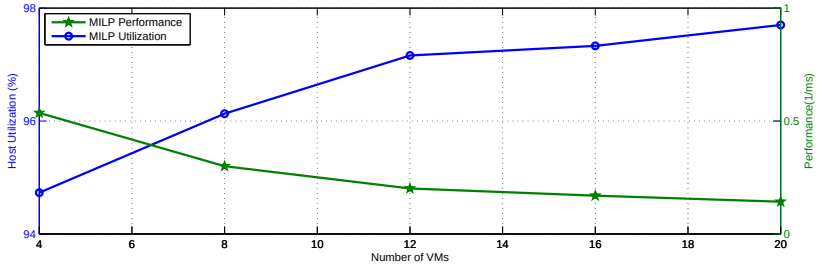


Figure 5: Utilization rate versus performance of MILP optimization for different number of VMs.

4.3 Evaluation and Analysis

By the experiment results explained above we have provided guidelines to be used in VM reconfiguation and application live-migration in cloud systems. By analyzing our first set of results in Figure 2 we have revealed an interval of 0% to 25% for application resource needs where using live-migration increases the total utilization of the host. This result makes sense because as the size of the application becomes larger it is less likely to allocate a large enough resource space from the leftovers in the VMs. By profiling or estimating resource need characteristics of the applications to be deployed to cloud system it is possible to decide on live-migration use strategy where such a strategy makes difference where smaller applications are dominant.

On the other hand a similar analysis can be made for reconfiguration purposes by examining Figure 3 as well. It is clear that instead of fully assigning all the resources equally to the VMs at hand, it is more preferable to assign around 80% of the resources and gradually reconfigure each VM according to needs. An interesting fact can be the stable utilization rate until 80%'s which may be tuned by taken the reconfiguration cost into account since assigning smaller amount of resources initially will clearly require a larger amount of reconfiguration.

Taking into account the costs of reconfiguration and live-migration techniques Figure 4 identifies a region where it is possible to leverage the trade-off between applying these two techniques. Since the cost of each operation can change from system to system we have chosen to use different cost rates in our analysis. We can interprete the regions formed by the thresehold values in the figure as follows: the upper left quadrant contains the cases where it is advantageous to perform configuration but the cost of using reconfiguration and live-migration together exceeds the cost of using migration solely. In this case it is meaningful to chose and use only one of the techniques at all. The lower left quadrant is the ideal case where both of the techniques can be used collaboratively. For the lower right quadrant using live-migration is preferrable since gains by reconfiguration starts to decrease. However, it can still be meaningful to use reconfiguration as well for lower percentages of initial resource allocation regarding the cost rates. The upper-right quadrant is the area where using solely live-migration should be preferred.

5 CONCLUSIONS AND FUTURE WORK

Advances in the cloud resource allocation technologies led to apply resource allocation strategies at different layers of a cloud system. In this paper we investigated resource allocation challenges at two different levels, more specifically between host and VM layer and between VM and application layer. Our analyses show that when the resource consumption of applications gets larger it becomes less meaningful to apply application live-migration at VM layer. Moreover it also becomes less feasible to use VM reconfiguration as the amount of initial resource assigned to each VM start to go beyond 84%. In deciding the level to apply resource allocation strategy VM reconfiguration is preferable when the amount of cost rate between reconfiguration and live migration is below 2. Moreover, the optimal cost is observed when an initial resource assigned to each VM is inside 80%-84% area. As a future study we plan to realize the cloud system environment in an actual testbed and replicate our results in actual workloads. Additionally we plan to extend our approach to support horizontal scaling and multiple hosts which require more advanced(even non-linear) optimization techniques.

ACKNOWLEDGEMENTS

This work is performed in joint with “mCloud” project of Simternet Iletisim Sistemleri Reklam San. ve Tic. Ltd. Sti. “mCloud” project is supported by The Scientific and Technological Research Council of Turkey(TUBITAK)-TEYDEB project number 7130115. This work is also partly supported by ITU HP Cloud Computing Center.

REFERENCES

Bennani, M. N. and Menasc, D. A. (2005). Resource allocation for autonomic data centers using analytic performance models. In *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, pages 229–240.

Borovskiy, V., Wust, J., Schwarz, C., Koch, W., and Zeier, A. (2011). A linear programming approach for optimizing workload distribution in a cloud. In *CLOUD COMPUTING 2011, The Second International Conference on Cloud Computing, GRIDs, and Virtualization*, pages 127–132.

Chen, W., Qiao, X., Wei, J., and Huang, T. (2012). A profit-aware virtual machine deployment optimization framework for cloud platform providers. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 17–24.

Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A. (2005). Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI’05, pages 273–286, Berkeley, CA, USA. USENIX Association.

He, S., Guo, L., Ghanem, M., and Guo, Y. (2012). Improving resource utilisation in the cloud environment using multivariate probabilistic models. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 574–581.

Hwang, I. and Pedram, M. (2012). Portfolio theory-based resource assignment in a cloud computing system. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 582–589.

Kikuchi, S. and Matsumoto, Y. (2012). Impact of live migration on multi-tier application performance in clouds. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 261–268.

Ruiz-Alvarez, A. and Alvarez, M. (2012). A model and decision procedure for data storage in cloud computing. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 572–579. IEEE.

Van den Bossche, R., Vanmechelen, K., and Broeckhove, J. (2010). Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 228–235. IEEE.

Verma, A., Kumar, G., and Koller, R. (2010). The cost of reconfiguration in a cloud. In *Proceedings of the 11th International Middleware Conference Industrial track*, Middleware Industrial Track ’10, pages 11–16, New York, NY, USA. ACM.

Verma, A., Kumar, G., Koller, R., and Sen, A. (2011). Cosmig: Modeling the impact of reconfiguration in a cloud. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*, pages 3–11.

Wang, Y., Chen, S., and Pedram, M. (2013). Service level agreement-based joint application environment assignment and resource allocation in cloud computing systems. In *Green Technologies Conference, 2013 IEEE*, pages 167–174.

Yang, K., Gu, J., Zhao, T., and Sun, G. (2011). An optimized control strategy for load balancing based on live migration of virtual machine. In *Chinagrid Conference (ChinaGrid), 2011 Sixth Annual*, pages 141–146.

Zhu, X. and Singhal, S. (2001). Optimal resource assignment in internet data centers. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings. Ninth International Symposium on*, pages 61–69.

Challenges of Capacity Modelling in Complex IT Architectures

Andrea Kő, Péter Fehér and Zoltán Szabó
Institute of Information Technology, Corvinus University of Budapest, Budapest, Hungary
{andrea.ko, peter.feher, zoltan.szabo}@uni-corvinus.hu

Keywords: Internal Cloud, Capacity Management, Modelling, Neural Networks, Multivariate Statistics.

Abstract: As internal cloud, and cloud technologies widespread among companies, the responsibility of providing reliable IT infrastructure and adequate capacities became the top priority for companies. While internal clouds and related technologies creates the flexibility for customer, limited IT resources arise problems for providing capacities, that has impact on IT service quality. The presented research addressing this problem, and seeks creating models describing the relationship between IT service quality and background infrastructure capacity usage with two distinct methodologies, in a complex cloud-like environment of a financial institution. The research was analysed a pilot area of a widely used electronic banking service. As multivariate statistical modelling and hypothesis testing had limited results in phase 1, but in phase 2 further modelling opportunities were explored, a model based neural networks were developed. The research analyses the limitations of pure statistical analysis in cloud-like environments, but concludes to the usability of alternative methods.

1 INTRODUCTION

As the popularity of internal and hybrid clouds is advancing, companies apply cloud technology like solutions in their internal environments. As companies implement virtualization, dynamic resource scaling and load balancing, adequate IT architectures and infrastructures became more and more important. As opposite of external clouds, where capacities are seemingly infinite for users, in case of internal clouds resource capacities are limited, and require careful planning. In this case internal cloud architectures offer flexibility, but also limit of capacity planning, because dependences of infrastructure elements and IT services are difficult to clarify. Exploration of the relation between IT service quality and capacity management is vital in order to understand and improve operational processes and supports identification problems faster and more reliable way.

This paper aims to explore the relationship between front-end quality of IT service in internal cloud environment, and the infrastructure capacity usage, analysing methods to predict required capacities and service levels. The research used the environment of a financial institution (Central-European branch of a multinational bank) that is developing its IT architecture via cloud technologies.

IT service quality prediction allows enterprises to support the need of adaptation and customization of IT infrastructure and thus it helps to prevent the actual occurrence of failures or to mitigate upcoming failures. Such proactive adaptation capabilities are increasingly relevant especially for future service-oriented systems (Metzger et al., 2012) and in internal cloud environments (Rountree and Castrillo, 2013). Investigation of the relation between capacity management and IT service quality support to understand and improve operational processes and enhance problems identification quickly and systematically. It facilitates establishing valid and reliable service performance measures. Measuring customer satisfaction and other performance outcomes can benefit from the solution as well.

2 LITERATURE REVIEW

IT services underpins business operations, IT has business critical role as it has strong impact on business operations. IT service management (ITSM) methodologies and standards, especially ITIL (IT Infrastructure Library) have recently become a very popular approach and a widely used methodology to improve this organizational activity (Taylor, 2007).

The IT Infrastructure Library, as a comprehensive approach is a set of processes and functions that define best practices for managing, optimizing IT services, ensuring the responsiveness, cost-effectiveness of IT. ITIL helps to make IT a strategic asset for the company. The key ITSM procedures can be grouped into 5 major categories, integrating several processes into a life-cycle ranging from service Strategy to Continual Service Improvement.

Our research topic concentrates on a core process of ITIL: capacity management. This is a business-critical activity that is responsible for the smooth operations of IT supported business operations, that safeguards the quality of services and ensures the efficient processing of large volumes of business transactions. The goal of capacity management is to ensure that the capacities underpinning the business services through IT services meets the agreed requirements cost-effectively and in a timely manner, by understanding and satisfying current and future capacity and performance demand of the business. Capacity management, through the alignment of IT objectives with business priorities on short and long terms reduces the total cost of ownership of IT infrastructure and operations, and has central part of the warranty of the IT services (OGC, 2007). According to Broussard (Broussard, 2008) measuring and management of the performance of IT services is among the top issues for IT managers. Microsoft operations framework (MOF) has capacity management process too (Bagley et.al. 2002).

Capacity management has proactive and reactive tasks to enable the synchronization of IT and business performance. It ensures current operations by optimizing resource utilization and fine-tuning the infrastructure, monitoring the operations and supporting capacity-related incident management. Capacity management has also long term responsibilities that enable organizations to balance long term costs against the IT resources needed, and balancing supply and demand of IT services (development plans, budgeting, procurement, etc.). It has responsibilities and procedures on business, services and component levels (OGC, 2011).

Although the concept has long history and integrates experiences and best practices, designing and establishing comprehensive capacity management activities in a complex IT environment is a demanding task. Capacity management is closely related to application sizing, but it follows a service-oriented and comprehensive approach, covering the whole life cycle of IT services. Another related concept is the performance management. Performance management traditionally monitor and manage IT performance, optimize infrastructure utilization, and support component-level capacity planning and service-level reporting. Capacity management utilizes several methods and techniques (many of them discussed in Kant and Srinivasan, 1992), integrated into a holistic approach of Service Design processes. Implementation issues of capacity management discussed in details by Higday-Kalmanowitz and Simpson (2004).

The main challenge relies on the management of complexity. Recently business services are supported by several multiple interdependent and multilayered IT services. As more automation, virtualization is embedded in the underpinning IT systems, the dynamic nature of the resources requires special attention, careful design and strict control (HP, 2008). The IT organization is responsible for guaranteeing the agreed level of performance and quality to business users. In order to maintain control over the IT environment and to ensure the services meet business requirements, IT must have the explicit knowledge of the dependencies and performance of the entire system, both in real time and in historical terms. A key aspect of capacity management is the detailed understanding of business patterns, user profiles, dependencies of business processes and IT services. This requires extensive and well-targeted, objective monitoring of IT activities and resource utilization. The exploration and documentation of complex interdependencies of IT services and the supporting infrastructure components is also a requirement. Planning and forward looking processes also require sophisticated tools and techniques to enable modelling, analysis and reporting (OGC 2011; EMA 2012).

Capacity management integrates many activities, collects many inputs from several data resources (business, service, component and financial data) and enables several processes that have central role in the management of the IT as a business. It monitors utilization of individual components and response times of services. It collects and transforms business needs and utilization patterns into IT requirements for services. By analysing data it identifies trends, bottlenecks and enables tuning of components, applications or services. On short term the management of thresholds and capacity related incidents is closely integrated to operational activities (scheduling, workload management, virtualization, etc.). On long term, demand management and capacity planning, application sizing etc. requires modelling and trending tools and

techniques: trend analysis, analytical modelling and simulations (OGC 2011).

There are similar initiatives in capacity planning and performance optimization. Lee and Asanovic implemented a multiprocessor system simulator (METERG QoS system) to provide a method for the estimation of the lower bound on end-to-end performance for a given configuration of resource reservations (Lee and Asanovic, 2006). A comprehensive high level modeling framework was developed by Heckmann to analyse economical and technical hypothetical questions of service providers throughout a service's life cycle (Heckmann, 2012). HP experts created a closed multi-tier multi-station queueing network model that combines performance modeling with performance profiling (Chen et.al. 2007). Their experiments show that the model is appropriate in a typical 3-tier e-commerce application in a virtualized environment to translate SLAs to lower-level resource requirements for each involved system in providing the service. Kousiouris (Kousiouris, et.al. 2011) used a generic black box approach, based on Artificial Neural Networks, appropriate for SLA negotiations. Our model, based on a similar modelling approach, is focusing on the runtime of a complex group of applications.

3 RESEARCH OVERVIEW

The presented research in this paper is aimed to explore the relationship between front-end quality of the provided IT service, and the background infrastructure capacity usage in an internal cloud environment.

Although the research, as a whole is considered to be an explorative, in order to create a model to describe this relationship, the researchers have strong hypotheses on the related factors. During the development of the relationship model, these hypotheses are codified and tested in a pilot environment with multidimensional statistical methods.

The research was conducted in a Central-European branch of a multinational financial institution (named as "Bank" in the following). The Bank serves both private and business customers with mainly commercial banking products, with the addition of investment possibilities. Products can be reached via several channels, including local offices, telebanking, Internet banking services, and dedicated business terminals for companies.

Information technology is a vital part of this company, as in every financial institution. Information technology is expected to provide cost savings, and improved internal efficiency (Fung, 2008; Ehikhamenor, 2003), but it is also expected to create new opportunities for expanding business, and interacting with customers (Liao and Wong, 2007, Vatanasombut et al 2008; Lee, 2008).

By today, the banking industry heavily depends on the information technology services, therefore it is vital to provide these services at least on a minimum acceptable level. Failing this condition results decreasing quality in business products, lower customer satisfaction, and eventually losing customers.

With the increase of technological complexity, it is more and more difficult to monitor and relate component performances and capacities (Metzler, 2003). The question is what is required to provide quality services, what are the factors that have impact on service quality?

In case of complex IT architectures, with the application of cloud technologies, IT infrastructure resources are often shared (virtualised) among logical elements. In results a complex load on the physical level by logically separate servers and services. Companies usually measure the physical level and neglect end-user quality questions. Moreover even physical level measures are the results of ad-hoc decisions, resulting incomplete datasets for analysis. But the main challenge is the lack of definite relationship between infrastructure performance and end user service quality.

To explore the above mentioned research questions the services of the Internet Banking channel of the institution were selected. The research idea was considered adequate for research purposes, because of the following reasons:

- Service quality is directly perceptible by customers
- The background IT architecture is complex, and covers the relevant banking products and services
- Provides sample data in adequate size for statistical analysis that is representative for the research services.

Although the main conditions are adequate for analysis, the research had to deal with limited and incomplete datasets on the physical and logical level of the IT architecture.

The Internet Banking channel covers 8 main banking products and services, with 4 other administrative services. While the administrative services cover over one third of the overall transaction numbers, these services use only load on the direct web server, while the business services use

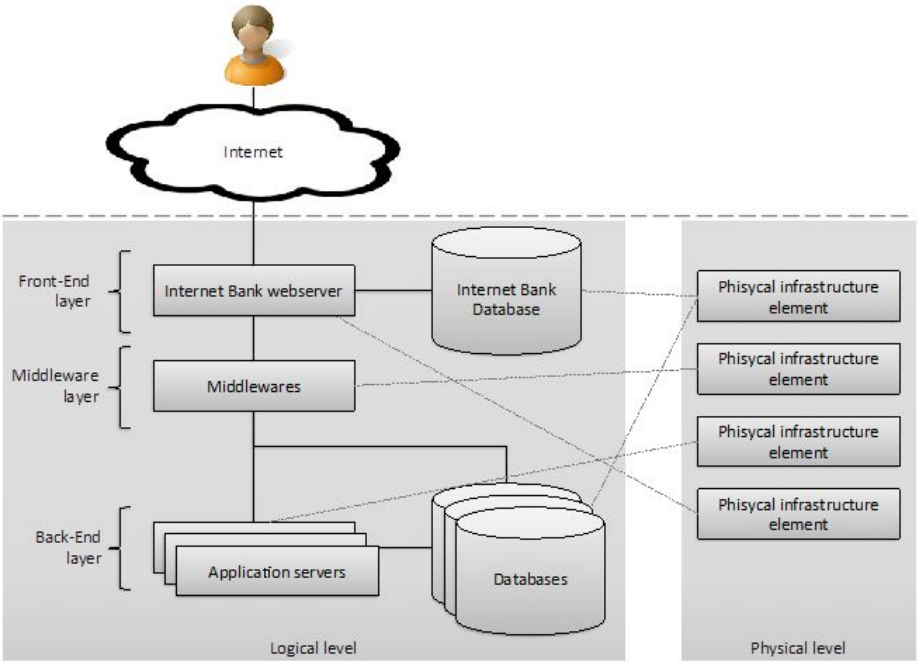


Figure 1: Architecture overview of the researched Internet Bank.

a wide range of the IT architecture. This paper analyses the problems of the most popular transaction type: domestic money transfer, that covers almost two third of business service transactions.

3.1 Architecture Structure of Services

In order to understand the relationship between components, the architecture behind the visible services should be explored. Generally IT architecture of every provided service consists of a front end (client software) and a back-end (servers and resources) layer (Figure 1). The two layers are connected via a middleware layer (Stephens, 2010).

In case of an infrastructure cloud (Sitaram and Manjunath, 2011) the capacity of the same physical resource (a server, a storage, CPU, memory, etc.) is shared between several virtualised elements (application servers, database servers, etc). Moreover, the same virtual server can be used by several other applications and services. Therefore a certain IT service competes for the same resources against other IT services.

Based on the used technology and policy settings virtual infrastructure elements, or applications can have dedicated resources, or a service can use resources in a flexible way, based on its current needs. The presented research concentrates on the latter environment, because in this case the quality of a service does not depend only on the used capacities, but also influenced by other competing resources.

In case of the presented research, the Internet Bank is accessed by the customers of the bank via an online user interface that can be used in any browser. This interface is served directly in the front-end layer with an application server, and its database that contains front-end related data. In case of business services the access of the back-end infrastructure elements (applications, databases, via the middleware layer) is required to record transactions, to check account balances, or to filter fraudulent activities. Each virtual server of database runs on different physical infrastructure elements that are shared among several logical infrastructure elements, even outside of the Internet Banking service group.

In this architecture quality of the money transfer service is measured as response times within the boundaries of the Bank, in the context, where business services are defined as the available functions, or transaction types of the user interface. Because the bank have no influence neither on the quality of the Internet network, nor on the quality of provided internet service for the customer, this part of the response time was not used during analysis.

Measurement on transaction numbers, and transaction time periods were recorded for the researched service (money transfer), and the used

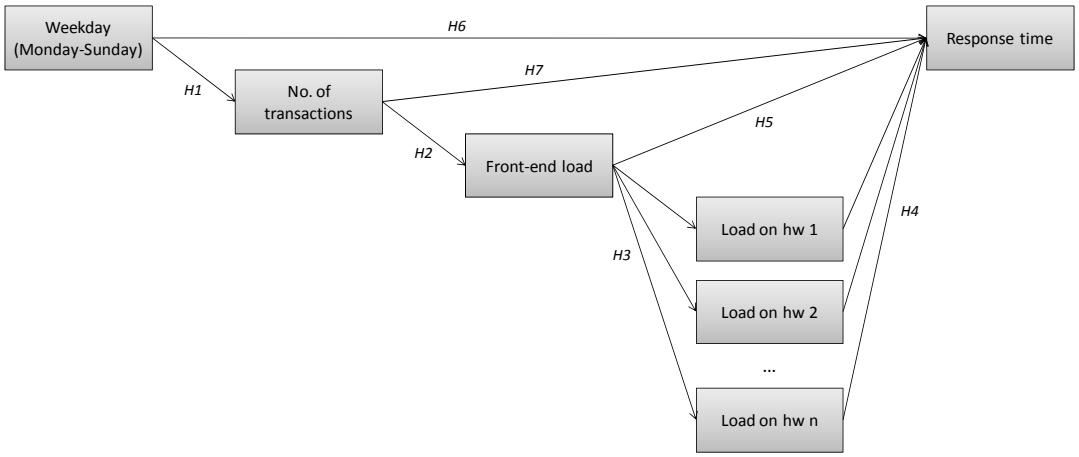


Figure 2: Basic research model and hypotheses.

logical and physical infrastructure elements through the data flow and processing were identified. Because logical infrastructure elements use– even in case of dedicated pools – the capacities of the physical infrastructure, measures were applied on the physical level.

4 ANALYSIS

In Phase 1 the research started to prepare a prediction model based on multivariate statistical analysis. In order to form the model series of hypotheses was formed and tested. In phase 2 the limitations of the research model were addressed and alternative methods were used.

4.1 Phase 1: Research Model and Hypotheses

Figure 2 summarizes the expectations of the model. It is expected, that the service quality of money transfer – in this case the response time – is impacted by four main factors:

- Capacity usage of the used physical infrastructure (H4)
- Capacity usage of the front-end application (H5)
- The day of the week (H6)
- Number of transaction in the measured timeframe (H7)

As background hypothesis, the research analyzed the relationship, how the capacity usage is impacted by the behaviour of the users during a week that revealed in the number of transactions. For this relationship we analyzed the relationship between the day of the week and the number of transactions (H1), the number of transaction and the capacity usage of the front-end application (H2), and the front-end load and capacity usage of back-end applications (H3). For every relationship (H1-H3 and H4-H7) the research expected positive impact that makes available prediction model for service quality.

4.2 Data Preparation

In many researches data are analysed for identifying extraordinary values. In this research extraordinary values (in every parameter) are the most vital parts of the research problems. If there are particular high values in response time that does not fit into the patterns of the data, researching the reasons is the most valuable for companies, because these response time causes them the problems in operations.

These cases are not uncommon for the research service in this shared and virtualized environment, and these kinds of cases can have serious impact on customer satisfaction and company reputation (Johnson and Peppas, 2003, Johnston 2004). Response time above accepted in the Service Level Agreement represent a 24,31% share of the total transactions.

Because of these reasons, and despite only 3,7% of the data are considered as extraordinary, no data were excluded at this phase of the research, nor were the sample split into normal and extraordinary cases.

4.3 Hypotheses Testing

To test the hypothesis 1 (H1), that specific days has influence on the number of transactions the analysis of variance were executed. To test the homogeneity

of variances Levene Statistics was applied and was significant (0,000), while F test also justifies the difference of transaction number means of different weekdays.

Moreover there are also differences in the patterns of capacity usage in the Front End layer (H2) and Processing layer components. Capacity usage averages tend to follow transaction numbers (strong correlations >0,9 on significance level < 0,01), but response times (H7) follow a different pattern (no significant correlation). Moreover the research could not find significant correlation between service quality and the front-end load or the processing layers (H4 and H5).

The filter the cross-impact of each component, partial correlation analysis supported hypotheses testing. Even with this refines analysis, the research cannot confirm the relationship between the capacity usage of the physical infrastructure and service quality (H4 and H5).

The research can only confirm, that there is a relationship between the special days and response times, but this relationship alone is not complete to build a predictive model for service quality.

4.4 Phase 2: Neural Network Analysis of Capacity Data

To overcome the unsatisfying results of statistical models in capacity analyses, we applied another promising modelling approach, artificial neural networks (ANN). Neural networks are inspired by the working mechanism of human brain and they are known as promising solutions in forecasting and business classification applications, because of their beneficial characteristics (Turban, 2011):

- They are able to deal with highly nonlinear relationships,
- Not prone to restricting normality and/or independence assumptions
- Can handle variety of problem types
- Usually provides better results (prediction and/or clustering) compared to its statistical counterparts
- Handles both numerical and categorical variables.

Neural networks are able to “learn”, they are often called universal approximators (Sifaoui, 2008). Learning algorithms specify the process by which neural network learns the underlying relationship between input and outputs or between inputs. Learning algorithms can be supervised or unsupervised (Mohri et al., 2012), (Barlow, 1989). Supervised learning is often applied in prediction area, while unsupervised learning is frequently used in clustering problems. Pattern recognition, forecasting, prediction, and classification are the typical tasks for ANN. Application areas are various, like finance, marketing, manufacturing, operations, information systems, and so on.

In our case neural networks were utilized for those services where the statistical investigation had no satisfying result. According our task type, which is a prediction, one of the most promising ANN model, multilayer perceptron network was selected. Multilayer perceptron is a feedforward, supervised learning model (Rumelhart, 1986).

Several software tools are available for modelling neural networks, amongst other commercial and open source data mining software suits, like SAS Enterprise Miner, Statistica Data Miner, PASW, Rapidminer, R and standalone solutions, like NeuroSolutions, BrainMaker, NeuralWare and NeuroShell. Rapidminer was selected as a modelling environment, because nowadays it is a leading open source data mining solution with rich functionality (Rexer, 2009). It has more than 600 operators and user friendly graphical user interface.

ANN network topology for HUF transfer service consisted of one input layer, one output layer and one hidden layer. Input variables included date, like weekday, hour and minute and infrastructure measurements, like CPU. Output variable was PI runtime. Training data set of HUF transfer model included 11807 records of HUF transfer service data of a monthly period. This data set was applied to build ANN model. Model testing was done on the 13622 records of HUF transfer service data from next month. Results included predicted response time, actual response time are shown on the figure 3.

In model evaluation the following error functions were applied: correlation, root_mean_squared_error, absolute_error, squared_error, prediction_average, spearman_rho, kendall_tau. Correlation was 0.208 for the whole data set, which was not satisfactory result. This bad value is caused by the outliers (8% of the whole data set). In a modified data set which doesn’t contain outliers; correlation was 0.89, which is an appropriate outcome. The ANN modelling environment was applied afterwards for capacity investigation in the bank by consultants. The ANN model was regularly updated and maintained according the changing capacity environment.

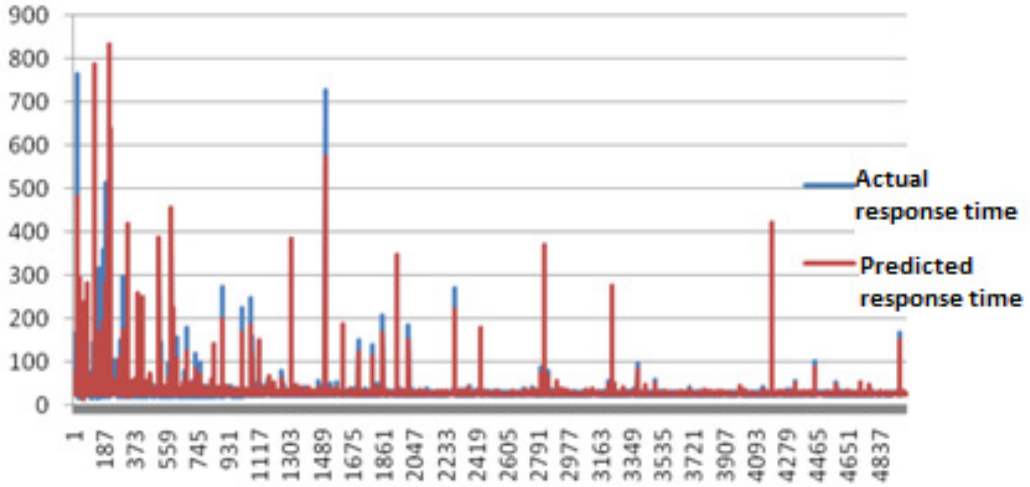


Figure 3: Predicted and actual response time for HUF transfer service.

5 CONCLUSION

Our results shows that traditional multivariate statistic-based models and techniques have only limited explanation power in today's complex virtualized, cloud-like IT infrastructure, as popularly labelled as internal clouds. Seemingly trivial hypotheses were failed, suggesting that there are hidden interdependencies and unrecognized relationships between IT infrastructure elements and the business services. Although realizing the necessity and usefulness of traditional statistical analysis, to develop a comprehensive and easy-to-use model more sophisticated tools should be used, underpinned with the knowledge crystallized in the corporate architecture models.

This paper analyzed the relation between capacity management and IT service quality. We set up a research model and hypothesis, which was tested in a Bank. We could accept four hypotheses, but two were declined. These two services required another research approach, beyond the traditional statistical models. Therefore an artificial neural network for capacity modelling was selected, as alternate modelling method. ANN has several promising characteristics; the most important ones are its non-linear feature and learning ability. One of the main limitations of our result is that neural networks are not so popular amongst business decision makers, because they are deemed to be black-box solutions; we have no explanation about the results. In spite of the sometimes negative attitude of decision makers towards ANN, this model was applied after the research in the Bank for capacity modelling in IT operations.

Capacity and service level management can be supported with ANN based prediction models. Predicted changing numbers of transactions (based on a marketing campaign or policy issues) is a good input parameter for running the model. Based on the ANN approach changed capacity usage and internal response time can be predicted, and unacceptable response time periods can be identified. Based on the predicted response times, capacity and service level managers can decide on a) investing into new infrastructure elements to provide the required capacity for peak times, b) change server capacities to restructure resources for more demanding applications or c) influence customer behaviour to smooth overused capacities into underused time periods. Capacity planning allows organisations to prepare their future quality of service for predicted user behaviours.

Further developments include fine-tuning of the ANN modelling environment and testing other ANN topologies, and additional architectural environments.

REFERENCES

Bagley, W.; Hanna, A.; Howels, V.; Rasmussen, K.; Rytkonnen, P.; Westover, J.; Yuhas, J., 2002. *MOF Service Management Function Capacity Management*. Microsoft Corporation, Redmond

Barlow, Horace B., 1989. Unsupervised learning. *Neural computation* 1.3 pp. 295-311.

Broussard, Frederick W., 2008. *IT Service Management Needs and Adoption Trends: An Analysis of a Global Survey of IT Executives.* IDC White Paper. http://whitepapers.techrepublic.com.com/abstract.aspx?docid=395611 (accessed: 28.06.2009)

Chen, Y., Iyer, S., Liu, X., Milojicic, D., Sahai, A., 2007. *SLA Decomposition: Translating Service Level Objectives to System Level Thresholds*. Enterprise Systems and Software Lab, HP Labs. http://www.hpl.hp.com/techreports/2007/HPL-2007-17.pdf. (accessed: 28.12.2013)

Ehikhamenor, F.A., 2003. *Information technology in Nigerian banks: The limits of expectations. Information Technology for Development* 10, 13–24.

EMA, 2012. *IT Optimization through Predictive Capacity Management.* ENTERPRISE MANAGEMENT ASSOCIATES (EMA) White Paper. http://www.ca.com/~/media/Files/IndustryAnalystReports/it-optimization-through-predictive-capacity-management.pdf (accessed: 28.08.2012)

Fung, M.K., 2008. To What Extent Are Labor-Saving Technologies Improving Efficiency in the Use of Human Resources? Evidence from the Banking Industry. *Production and Operations Management* 17, pp. 75–92.

Heckmann, B., 2012. *Service Quality and Profit Control in Utility Computing Service Life Cycles*, Doctorate, University of Plymouth, Plymouth, http://hdl.handle.net/10026.1/1568 (accessed: 28.12.2013)

Higday-Kalmanowitz C.; Simpson E. S. (eds.), 2004. Implementing Service and Support Management Processes: A Practical Guide. Van Haren Publishing

HP, 2008. *End-to-end service management in the virtualized environment*. White Paper. https://h10078.www1.hp.com/bto/download/4AA2-3182ENW.pdf (accessed: 28.08.2012)

Kant, K.; Srinivasan, M. M., 1992. *Introduction to computer system performance evaluation.* McGraw-Hill International edition.

Kousiouris, G.; Kyriazis, D.; Gogouvitis, S.; Menychtas, A.; Konstanteli, K.; Varvarigou, T., 2011. "Translation of application-level terms to resource-level attributes across the Cloud stack layers," *Computers and Communications (ISCC), 2011 IEEE Symposium on*, vol., no., pp.153,160, June 28 2011-July 1 2011

Lee, J.W.; Asanovic, K., 2006. "METERG: Measurement-Based End-to-End Performance Estimation Technique in QoS-Capable Multiprocessors," *Real-Time and Embedded Technology and Applications Symposium, 2006. Proceedings of the 12th IEEE* , vol., no., pp.135,147, 04-07 April 2006

Lee, M.-C., 2009. Factors influencing the adoption of internet banking: An integration of TAM and TPB with perceived risk and perceived benefit. *Electronic Commerce Research and Applications* 8, pp. 130–141.

Liao, Z., Wong, W.K., 2007. The determinants of customer interactions with internet-enabled e-banking services. *Journal of the Operational Research Society* 59, pp. 1201–1210.

Metzger, A., Chi-Hung C.; Engel, Y.; Marconi, A., 2012. Research challenges on online service quality prediction for proactive adaptation. In.: *Proceedings Software Services and Systems Research - Results and Challenges (S-Cube), 2012* Workshop on European Software Services and Systems Research - Results and Challenges (S-Cube), pp. 51 - 57

Metzler, J., 2003. *The Mandate to Implement Unified Performance Management*. Accessed: 29 Sept 2009, source: http://www.comnews.com/WhitePaper_Library/Network_management/pdfs/NetScout_wp_Metzler_Mandate_to_Implement_Unified_Performance_Management.pdf

Mohri, M., Rostamizadeh, A., Talwalkar, A., 2012. *Foundations of Machine Learning*. The MIT Press ISBN 9780262018258.

OGC. 2007. *ITIL Service Design*. London: The Stationery Office Ltd.

OGC, 2011. *ITIL Service Design*. London: The Stationery Office Ltd.

Rexer, K., 2009. *Data Mining Tools Used Poll*. KDNuggets, http://www.kdnuggets.com/polls/2009/data-mining-tools-used.htm (2009.08.24)

Rountree, D., Castrillo, I. (2013) *The Basics of Cloud Computing*, Understanding the Fundamentals of Cloud Computing in Theory and Practice, Elsevier

Rumelhart, D. E., Hinton, G. E. and Williams, R. J., 1986. „Learning internal representations by error propagation". In Rumelhart, D. E. and McClelland, J. L. (eds.), „*Parallel Distributed Processing*", Vol. 1., MIT Press, Cambridge, MA.

Sifaoui, A., Abdelkrim, A., Benrejeb, M., 2008. On the Use of Neural Network as a Universal Approximator. *International Journal on Sciences and Techniques of Automatic control & computer engineering*, IJ-STA, Vol. 2, N. 1, pp. 386-399.

Sitaram, D., Manjunath, G. 2011. *Moving To The Cloud: Developing Apps in the New World of Cloud Computing*. Elsevier

Stephens, D., 2010. *The Challenges Monitoring Composite Applications*. MeasureIT 8.

Taylor, S. 2007. *ITIL: The official introduction to the ITIL Service Lifecycle*. The Stationery Office Ltd.

Turban, E., Sharda, R., Delen, D., 2011. *Decision Support and Business Intelligence System*s. 9/E, (ISBN-10: 013610729X)

Vatanasombut, B., Igbaria, M., Stylianou, A.C., Rodgers, W., 2008. Information systems continuance intention of web-based applications customers: The case of online banking. *Information & Management* 45, pp. 419–428.

Optimizing Access Control Performance for the Cloud

Slim Trabelsi[1], Adrien Ecuyer[2], Paul Cervera Y Alvarez[3] and Francesco Di Cerbo[1]
[1]*SAP Labs France, Mougins, France*
[2]*Open Systems, Zurich, Switzerland*
[3]*Amadeus, Sophia-Antipolis, France*
firstName.lastName@sap.com, aec@open.ch, pcerverayalvarez@amadeus.com

Keywords: Cloud, Access Control, Performance, Caching, Scalability, Policy, Security.

Abstract: Cloud computing is synonym for high performance computing. It offers a very scalable infrastructure for the deployment of an arbitrarily high number of systems and services and to manage them without impacts on their performance. As for traditional systems, also such a wide distributed infrastructure needs to fulfil basic security requirements, like to restrict access to its resources, thus requiring authorization and access control mechanisms. Cloud providers still rely on traditional authorization and access control systems, however in some critical cases such solutions can lead to performance issues. The more complex is the access control structure (many authorization levels, many users and resources to protect); the slower is the enforcement of access control policies. In this paper we present a performance study on these traditional access control mechanisms like XACML, which computes the overhead generated by the authorizations checking process in extreme usage conditions. Therefore, we propose a new approach to make access control systems more scalable and suitable for cloud computing high performance requirements. This approach is based on a high speed caching access control tree that accelerates the decision making process without impacting on the consistency of the rules. Finally, by comparing the performance test results obtained by our solution to a traditional XACML access control system, we demonstrate that the ACT in-memory approach is more suitable for Cloud infrastructures by offering a scalable and high speed AC solution.

1 INTRODUCTION

Cloud computing offers a high performance computing infrastructure, combined with a huge amount of resources (like storage, database, network, servers, memory, virtual machines, etc.). Resources are made available to customers with a strong guarantee on the high quality of the computing performance. Scalability, availability and accessibility are among the key pillars of a cloud service or platform; naturally, these characteristics must not be sacrificed at the altar of security. Security comprises many aspects and functionalities, and access control represents one of the most common concepts which are mandatory for any production system. Access control (AC) is a mechanism in charge of restricting and filtering the access to system resources (data objects, Personal Identifiable Information PII, services, platforms, or infrastructure). Specific authorizations (also called permissions) are assigned to system users; when a user requests access to a resource, an AC engine is in charge of checking if the requesting user's permissions are compliant with the set of AC rules associated to that resource. The AC rules can be expressed using very basic lists called ACL, or using a more sophisticated and structured representation called policy. The AC rules described via policies are largely adopted in complex systems, those that are available to many potential users and that offer different resources. The main advantage of using AC policies is the possibility to manage and maintain a huge number of complex rules. Compared to the basic ACLs, AC policies require a reasoning engine able to explore process and decide on the applicability of AC rules against access requests. Cloud computing instances rely on AC systems configured via AC policies that express AC rules targeting a complex structure of user set and resources (Popa, 2010). The more significant is the density of the user and resource model; the more complex is the structure of the policies. A complex AC policy requires an additional computing and reasoning effort to the AC engine in order to interpret its rules and enforce the decision. Some

studies (Antonios, 2011) and (Tang, 2012) pointed out the new requirements in terms of AC models introduced by cloud computing the difficulties of traditional authorization systems to handle very complex policies for cloud computing systems, and in some critical cases this can lead to performance issues in the resource access process. In this paper we conduct a performance study on a XACML (OASIS, 2013) AC engine for a cloud infrastructure. XACML, stands for eXtensible Access Control Markup Language, is an OASIS XML policy standard for AC. Although if XACML is not initially designed for the Cloud, a deployment was proposed in (Reddy, 2012). Our performance study consists of stress tests on the AC enforcement engine, requiring it to process a huge number of access requests for a wide range of resources protected by complex policies. The objective of this study is to indicate the limitation of traditional AC engines when they are deployed in cloud platforms. Subsequently, we propose two approaches to optimize the performance of the considered AC enforcement engine and make it more suitable to cloud platforms. Both solutions are based on a pre-caching authorization tree; this method consists of storing all AC rules in a tree data structure, in order to optimize the policy exploration operation. This access tree can be implemented in two ways: Using a traditional DB (loaded in memory), or using a structured hash table system (also loaded in memory). We stress the importance of the in-memory deployment, in order to take advantage of the "unlimited" resources offered by the Cloud infrastructures.

This paper is organized as follows: In section two we survey the state of the art on the different performance studies dealing with performances of AC systems, in section three we describe the deployment of the XACML engine in the cloud as a reference study basis, in section four we detail our access control tree based solutions, in the section five we compare the different approaches through performance tests and try to identify the most scalable approach.

2 RELATED WORK

Optimizing performances of AC enforcement engines is an issue that appeared and was partially addressed before the emergence of the Cloud computing paradigm. It is interesting to review these studies to learn about their approaches and try to map them to the new requirements introduced by the cloud infrastructures. An AC mechanism has its deployment and operational costs, for instance when implementing its structure in very large scale storage systems, one has to face a trade-off among performance, availability and security (Leung, 2007). Balancing this trade-off is particularly challenging in the storage cloud environment, as providers must implement an efficient access control system that scales elastically and meets the high availability requirements of the cloud. The bottleneck caused by the lack of cloud-adapted AC enforcement mechanisms can be exploited for DoS attacks to break a service or a system like in the case described in (Niu, 2009).

2.1 Performance Issues for Access Control Enforcement in Traditional Systems

One of the most interesting approaches that inspired our solution is the XEngine. In (Hwang, 2011), and (Daly, 2011) the XEngine is presented as a fast and scalable evaluation engine for XACML. This work proposes to transform a XACML policy into a different data structure; this method converts the hierarchical structure of the XACML policy into a flat structure and then transforms any set of multiple combining algorithms into a first-applicable algorithm section. This work is inspired by the decision diagram approach where the decision is made based on a tree data structure. This work addresses the problem of performing fast XACML evaluation; however their model is quite static especially with respect to the proposed tree structure that systematically uses the subject as root of the AC tree. Such model lacks of efficiency especially when many resources are requested at the same time. In our solution, we proposed several enhancements to this approach in order to make it more flexible and efficient for any type of queries.

(Squicciarini, 2011) proposed an alternative approach called reordering and clustering policy rules. Their work proposes a way to optimize policies based on statistics to reorder rules. As it consists of a reordering of rules, only the deny-overrides and permit-overrides combining algorithm are taken in consideration for the optimization. Their solution works well if the amount of requests per requesters and the type of requester are not dynamic. In the case where they are dynamic the reordering is not efficient, and a clustering solution is proposed. The clustering is based on a per subject basis. In our previous example, this leads to the creation of two views, one for each subject, student and faculty. However, Squicciardini et al consider only two

XACML combining algorithms.

2.2 Performance Issues for Access Control Enforcement in Cloud Systems

An initial study was proposed by (Punithasurya, 2012) to compare different AC models; they tried to compare different parameters including the performance aspect. This evaluation of such parameters was not really motivated by a technical analysis. Therefore, it is hard to interpret their results and compare the different AC models with respect to performance aspects.

Well known cloud providers like Amazon S3 or Microsoft Windows Azure Storage, has a simple method for granting access to third parties using ACLs, provided that they are registered as S3 users or groups (they do not implement complex AC rules). Objects can be made public by granting access to the Anonymous group but there is no way to selectively grant access to principals outside of the S3 domain. The main limitation of S3 ACLs is the restriction to list 100 principals. Instead of proposing a scalable solution, Amazon S3 proposed to split the ACLs into small independent clusters. EMC Atmos Online opted for the same approach with a more hierarchical structure between the different cluster groups.

The caching solution was also proposed for cloud computing systems (Reeja, 2012). The approach described in that paper implements two policy decision points (PDP), one for new access requests and another one for access requests that were already processed by the primary PDP.

(Harnik, 2011) conducted an analysis on the ACL based mechanism for cloud computing in order to point out their weaknesses and to propose their AC mechanism. It is based on a capability-based system that allows the integration of existing AC solutions thus leading to hybrid architectures for AC systems. Such hybrid systems combine the benefits of capability-based models with other commonly used mechanisms such as ACLs or RBAC. On the other hand this study points out the weakness of ACL used by many major cloud providers like Amazon and Microsoft.

3 DEPLOYING XACML IN THE CLOUD

In the previous section we explored the different access control system deployed in the current cloud infrastructures. We observed that most of the solutions are based on basic ACLs and not really adapted to the Cloud requirements. We explain in this section how the XACML is deployed in a Cloud platform, and how to optimise the enforcement process using access control trees. We explain how our new solution takes benefit and enhance the tree based structures proposed by other studies in the literature.

3.1 XACML Integration

XACML is a declarative AC policy language implemented in XML and a processing model describing how to evaluate authorization requests according to the rules defines in policies. It allows for the definition of AC but also usage control rules through obligations. This language is very expressive and can be used to define a lot of different kind of policies. The choice of this language was due to the completeness of its expressivity for access control rules. We can define many AC models (like RBAC, ABAC, UBAC, etc.) with this language. It has a good flexibility for defining rule conditions.

We propose the XACML engine architecture depicted in Figure 1 as an integration architecture to the Cloud infrastructure.

- Cloud User interface: UI layers provided by the cloud as a service or as a platform feature. This cloud layer offers an IDM function to manage the user identity and authentication features. Through this UI the user can access to the different services and resources offered by the cloud. The identity (or role) of the user and her access request are collected at this level in order to be exploited by the PEP (Policy Enforcement Point) of the XACML engine
- XACML Engine: it is deployed in the cloud a service or as platform functionality (in this paper we chose to deploy it as a service). It offers the traditional functionality defined by the XACML specifications: PAP (Policy Administration Point) to manage the policy repository, PDP (Policy Decision Point) that evaluates the user request, and the authorizations contained in the policies, PIP (Policy Information Point) that provides external information about the user profile, and the Context Handler that coordinates all the previous components.
- Cloud Resources: we connected the resource manager of the XACML engine to the different resources provided by the cloud infrastructure

(database, services, VMs, etc.) in order to associate a user request with the requested resources. These resources are only accessible if the PDP evaluates positively the user request.

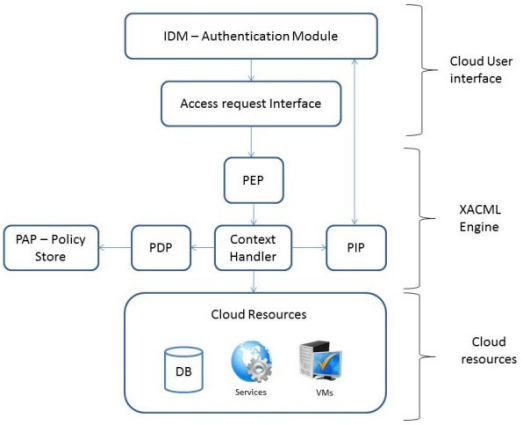


Figure 1: XACML Architecture for the Cloud.

3.2 Cloud Platform

The XACML AC engine that we used and modified in this paper is deployed as a service on a PaaS. We chose SAP Hana Cloud since it can be seen as SaaS and PaaS. SAP Hana Cloud platform also offers services like connectivity, document storing, identity, persistence and mail. But the most important selling argument offered by SAP Hana Cloud is that the platform is running over an in-memory database (Plattner, 2009), which fits perfectly with our in-memory based solution, described in details in the next sections.

4 OPTIMIZED ACCESS CONTROL ENFORCEMENT

The AC policy evaluation and enforcement can be complex and lengthy tasks to perform, impacted by the complexity of rules. In cloud computing, this issue is combined with thousands of users requesting simultaneously access to resources. This leads to an overhead during the authorization checking process. Therefore, it is important to implement a solution for speeding up the policy-based authorization checking process. Such solution should scale properly in order to handle multiple concurrent requests.

4.1 Access Control Trees

The main issue with XACML is the complexity due to its high expressivity. In order to optimize the evaluation computation process, we decided to implement a solution based on AC trees (ACT). The ACT is a concept based on aggregation. The use of pre-processing to aggregate privacy policies into the ACT will allow performance efficient AC check on mass of data. We can represent the aggregate AC information into a tree. The tree data structure has two key advantages for us: first, it provides a simple view of the AC structure; secondly, it facilitates the application of hashing techniques on a tree for efficient data search functions. This hashing technique will allow us to use an emerging database class (NoSQL) in order to improve further the already good performance of an ACT implementation in a relational database.

To map the AC rules into a tree structure we adopted the model proposed by (Bascou, 2002) and adapt it to the XACML policy schema. In this model, only accessible data objects are available. If the access to an object is denied, it will not appear. Following this model, our ACT contains only permitted data objects. If the data object cannot be accessed it will simply not appear. For this reason we call this tree the "Permit Tree" (PT). The AC tree or PT can be represented as follows:

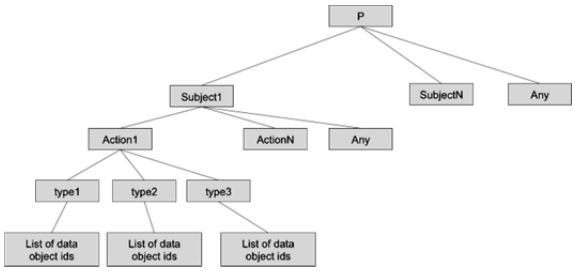


Figure 2: Access Control Tree (Permission Tree).

The PT (the same procedure can be applied for the Deny rules thus obtaining a Deny Tree) represented in Figure 2 is structured in three main levels: The first level contains the list of authorized subjects (or users, or roles, etc.) declared in the XACML policy repository. The ANY subject ID is used for objects that are accessible to all users with no restrictions. The second level represents the different actions or operations that can be executed on the data. If the list of actions is undefined, there is also an element *Any (Action)*. The third level represents the different types for data objects. Subsequently, the fourth layer contains a list of accessible data object IDs. The layer order (in the example subject, action and resource) can change according to system requirements. For example the first level can be the ID of the object. In that case the selection is made on the object to be accessed, for which one gets the list

of authorized users.

4.2 Tree Management Algorithms

In this section we describe the tree management algorithms, i.e. how the AC tree is built, maintained and explored.

4.2.1 Insertion Algorithm

This algorithm describes how one can add a new data object associated with its XACML rule. In other words, this means to insert data object references in the ACT corresponding to the XACML policy associated to the data object.

Tree Input Parameters
The input for the insertion or "insert" algorithm is a data object with a policy associated to it. The policy is seen as a rule set that can be composed by 4 rule types; they are listed in Table 1. One of them is specific, while others are not specific. The first rule type is a specific rule and can be easily mapped in the ACT tree. The others (2-4) are not specific. Not specific rules can be recognized by the "any" keyword.

Table 1: List of rule types.

Rule Type	Specific rule?
1.< *subject; action; decision* >	specific
2.< *subject;* ***any****; decision* >	non-specific
3.< ***any****; action; decision* >	non-specific
4.< ***any****;* ***any****; decision* >	non-specific

Tree Insertion Algorithm
The insertion algorithm consists of a body and two functions. The body loops over each rule and check if the rule is specific or not. If it is not specific, the algorithm creates all specific rules corresponding to it, by means of the first function *getAllSpecificRules()*. Then for all specific rules the algorithm executes the *handleSpecificRule()* function. The *getAllSpecificRules()* function receives as input a non-specific rule which is (specific type). According to the non-specific rule type, all corresponding specific rules will be created.

The *handleSpecificRule()* focuses on what to do with a specific rule. First it checks if the rule was already handled. Then it checks if the subject of the rule already exists in PT. If it is not the case, the subject is created in the tree and any sub-tree of the PT is copied. If a specific subject can access any data object ID in the "any" action sub-tree (rule type 2 in Table 1), the algorithm firstly extracts all action sub-trees then checks if the rule decision is permit or deny. If it is permit, the algorithm adds data object ID to the subject sub-tree for the corresponding action. At the end the algorithm inserts the rule in the handled rule set in order to ensure the consistency of the PT.

Insertion Algorithm

```
Input: <R1,R2,...,Rn> as P,PII, current tree as T
handeledRules={} //set of rules of type <subject,action>
for each rule R in P do
  if R is of type <subject,action,permit> OR <subject,any decision> then
    if T.subject not exist then
      Add subject in T.subjects
      Copy actions from T.any into T.subject
    end if
  endif
  if R is of type <subject ,action,decision> then
    if decision == permit then
      Add PII in T.subject.action
    endif
    Add <subject, action> in handledRules
  else if R is of type <subject, any ,decision> then
    actionList = T.decision.subject.actions
    for each action A in actionList do
      if <subject, A> not in hanledRules then
        if decision == permit then
          add PII in T.subject.A
        endif
        Add <subject, A> in handledRules
      endif
    endfor
  else if R is of type <any,action,decision> then
    Subjectlist = T.decision.subjects
    for each subject S in subjectList do
      if <S, action> not in handledRules then
        if decision == permit then
          Add PII in T.S.actions
        endif
        Add <S, Actions> in handledRules
      Endif
    Enfor
  else if R is of type <any, any, decision>  then
    subjectList = T.decision.subjects
    for each subject S in subjectList do
      actionList = T.decision.S.actions
      for each action A in actionList do
        if <S, A> not in handledRules then
          if decision == permit then
            add PII in T.S.A
          endif
          add <S, A> in handledRules
        endif
      endfor
    endfor
  endif
endfor
return T
```

4.2.2 Request Algorithm

The request algorithm is used to explore the AC tree in an optimal way and retrieve the authorizations related to a particular rule. This algorithm is executed when a user requests access to a resource.

Tree Input Parameters
The request input takes the form: < Subject; Action; Resource Type >. If one created a tree where the level 1 node is a subject, then it is possible to start the exploration using the input parameter Subject.

Tree Request Algorithm
If the subject exists in the tree then a deep exploration of the path is needed to find the data object IDs related to it. If the subject does not exist, the access to the resource is denied.

Request Algorithm

Input: *<subject, action> as R, current tree as PT*
If R.subject not exist in PT then
Return set of data objects ids resulting form PT.any.action
Else
Return set of Data object ids resulting form PT.subject.action
endif

5 IMPLEMENTING THE ACT FOR THE CLOUD

Compared to the XEngine solution, we propose implementations specially designed for the Cloud platforms and the in-memory capabilities offered by such platforms. The proposed ACT can be stored in two ways: in a relational database or in structured hash tables. Both are stored in-memory (using the in-memory Hana DB). Each of the solutions has its own advantages and drawbacks, detailed in the following performance analysis section.

5.1 Database Implementation

In this implementation, the ACTs (Permit Tree and Deny Tree) are stored in a relational database, therefore any request is performed in the database using SQL queries. The two ACTs are stored in the same table: *ACT(uniqueId, subject, name)*

This table is composed by three columns:

- UniqueId: representing the IDs of data objects.
- Subject: representing the user ID or role specified in the Policy.
- Name: the value of the data object (ex: john.doe@example.com).

The primary key of this table is composed of the three columns. The composed primary key ensures that a tuple is unique in the table and so in the ACTs as well. Additionally, the primary key also creates indexes on the three columns that will accelerate the queries performed on the table.

Each request type is translated into a SQL statement:

- Verify the access on a given data object by a given subject:

```
SELECT uniqueId FROM ACT WHERE Subject =
? AND Name = ?, If the query returns an
unique Id, then the access can be granted
```

- Retrieve the list of data objects that a given Subject can access

```
SELECT uniqueId FROM ACT WHERE subject =
?,the query returns a set of Data objects
IDs
```

- Retrieve the list of data objects from a given category (Name) that a given subject can access

```
SELECT uniqueId FROM ACT WHERE subject =
? AND name = ?, the query returns a set
of data objects ids
```

- Retrieve the list of subjects for a given data object

```
SELECT subject FROM ACT WHERE uniqueId =
?, the query returns a set of Subjects
```

5.2 Hash based Implementation

In this implementation, we proposed to store the access control trees into in-memory hash tables and persisted in a relational database. The persistence of ACTs is managed like in the previous implementation. The tree data structure fits well with the use of hash tables. This implementation consists of two different trees, *ACTbyData* and *ACTbySubject* with their own data structures.

The *ACTbyData* is implemented using the Java structure *HashTable*. It maps keys to values. In our case we mapped the Data IDs to a set of delegates. The set of delegates is represented using the java structure *HashSet*, which implements a set back by a hash table. We mapped the delegates to a nested *HashTable*. This nested *HashTable* maps the Data names to sets of Data IDs. This latter set is represented using the Java structure *HashSet*.

Therefore, we propose the following two data structures:

- `ACTbyPii: Hashtable<Long, HashSet<String>>`
- `ACTbySubject: Hashtable<String, Hashtable<String, HashSet<Long>>>`

The performance of *Hashtables* and *Hashsets* depends on two parameters: the initial capacity and the load factor. The former defines the number of buckets that are created at the creation of the hash tables. The load factor is a measure representing how full the hash table is allowed to get before its capacity is automatically increased. It is a Boolean value.

6 PERFORMANCE STUDY

In this section we evaluate the implementation of the ACT in the use case. The setup for the evaluation is the following. It is a PC with an Intel(R) Core(TM) i5-2500 CPU @ 3.30 GHz. It has 8.00 GB of RAM and is running Windows 7. The SAP HANA Cloud Platform where we deployed our application offers an elastic package with a limit of 12 GB of RAM. For the ACT we performed the tests on a traditional XACML engine HERAS-AF, the Cloud platform. We developed two different ACT implementations, one using a Derby DB loaded in-memory and the hash version using Redis.

6.1 Comparing the Three AC Implementation Versions

The basic testing scenario is a sequential query of 2000 different subject requesting 2000 different data objects (in our scenario PIIs representing e-mail addresses, and names). Every request is made by a single user that asks to access everything and only gets one object at the end. Every PII is constrained by a policy permitting one subject to access.

Both of the XACML engines are deployed as a service on the SAP Hana Cloud platform and installed on top of a database containing PIIs. Access requests are generated then sent to the the engines as a HTTP REST requests.

At 2000 data object, the Derby version is 2428x faster than the HERAS one, and the hybrid version is even 16998x faster. The difference is important since the HERAS based solution has to explore a complex structure of XML and objects representing the policy rules in order to match the applicable policy with the context of the request. This is more expensive in terms of processing time than a DB SQL query execution or a Hash function call.

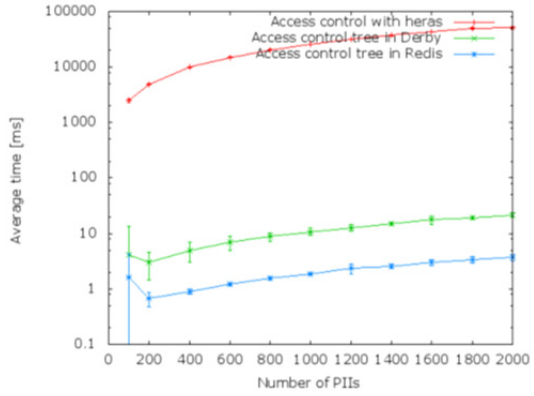


Figure 3: Comparing ACT solution with the traditional XACML HERAS engine.

6.2 Comparing the in-DB and the in-Hash ACT Solutions

The processing time of the HERAS solution gets worse with the increase of the system population (data objects + users). We decided then to focus our performance study on the comparison between the two proposed ACT implementations.

We run the same tests with a set of 6000 data objects accessible for 6000 different subjects. At 6000 data objects available the in-hash implementation of the ACT is 6x faster than the DB one. Furthermore the in-hash version scales better. Its performances are more stable as we can see with the standard deviation which is more important for the DB solution.

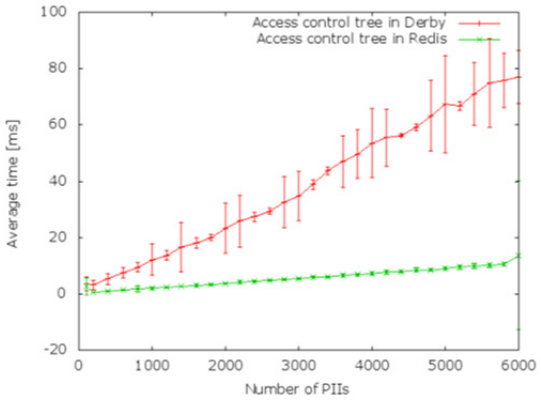


Figure 2: ACT in DB Vs. In-hash with 100% of the data objects accessible

7 CONCLUSION

The performance requirement for AC systems on the cloud is currently neglected by most of the cloud providers. Most of the cloud services are relying on single central AC systems implementing ACLs and in charge of handling all the access requests coming from all the cloud users. This issue can end up with serious performance problems especially when the cloud systems become complex. AC systems may become the main bottleneck disrupting the high speed computing capabilities of cloud servers. In our paper we evaluated a XACML engine for a cloud platform and demonstrated the limits of such policy based AC engine when access requests become huge. We proposed an AC tree caching system that can be implemented in parallel to the traditional AC systems in order to accelerate AC decisions. For this solution we proposed two implementations: one based on a relational database, and another one

based on a structured hash table system. Both solutions are stored in-memory. We tested these two solutions against a traditional XACML-based solution for the cloud. The performance discrepancy between the traditional AC system and our ACT based solutions is very important. Especially the hash based ACT seems the more scalable and the more adapted to cloud platforms.

ACKNOWLEDGEMENTS

This paper is done in the context of the PPP Fi-Ware project and the FP7 EU COCO Cloud Project.

REFERENCES

G. Antonios, "Towards new access control models for Cloud computing systems", *PhD report University of Macedonia, Department of Applied Informatics*

Z. Tang, J. Wei, A. Sallam, K. Li, R. Li, " A New RBAC Based Access Control Model for Cloud Computing", *7th International Conference, GPC 2012*, Hong Kong, China, May 11-13, 2012. Proceedings, pp 279-288

OASIS. Extensible Access Control Markup Language (xacml).https://www.oasis-open.org/committees/xacml

J. J. Bascou L. Gallon A. Gabillon, M. Munier and E. Bruno, « An access control model for tree data structures". In *ISC '02 Proceedings of the 5th International Conf. on Information Security*, 2002.

A. W. Leung, E. L. Miller, and S. Jones. Scalable security for petascale parallel file systems. In SC '07: Proceedings of the 2007 *ACM/IEEE conference on Supercomputing*, pages 1–12, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-764-3. doi: http://doi.acm.org/10.1145/1362622.1362644.

Z. Niu, H. Jiang, K. Zhou, T. Yang, and W. Yan. Identification and authentication in large-scale storage systems. *Networking, Architecture, and Storage, International Conference on,* 0:421–427, 2009.

J. Hwang A.X. Liu, F. Chen and T. Xie. Designing fast and scalable xacml policy evaluation engines. IEEE Transactions on Computers, Dec 2011.

A. Squicciarini S. Maruf, M. Shehab and S. Sundareswaran. Adaptive reordering and clustering based framework for efficient xacml policy evaluation. *IEEE Transactions on Services Computing, Oct-D*ec 2011.

J. Daly J. Brown and A. Gregory. The xengine policy decision point for xacml 3.0. Computer security *project in Department of Computer Sciences at the Michigan State University*, 26 Oct 2011.

Popa, Lucian, Minlan Yu, Steven Y. Ko, Sylvia Ratnasamy, and Ion Stoica. "CloudPolice: taking access control out of the network." In Proceedings of the 9th ACM SIGCOMM *Workshop on Hot Topics in Networks, p. 7. ACM,* 2010.

Punithasurya K and Jeba Priya S. Article: Analysis of Different Access Control Mechanism in Cloud. International Journal of *Applied Information Systems 4(2):34-39, September 2012. Published by Foundation of Computer Science*, New York, USA.

C.K. K. Reddy, P.R Anisha, K.S. Reddy, S.S. Reddy, "Third Party Data Protection Applied To Cloud and Xacml Implementation in the Hadoop Environment With Sparql", IOSR Journal of Computer Engineering (IOSRJCE) ISSN: 2278 - 0661 Volume 2, Issue 1 (July - Aug. 2012), PP 39 – 46

Reeja S L, "Role Based Access Control Mechanism in Cloud Computing using co-operative secondary authorization Recycling Method", 2012. *International Journal of Emerging Technology and Advanced Engineering* Website: www.ijetae.com (ISSN pp. 2250-2459, Volume 2, Issue 10, October 2012)

Amazon Simple Storage Service (Amazon S3). Amazon, b. http://aws.amazon.com/s3/.

Windows Azure Platform. Microsoft, a. http:// www.microsoft.com/windowsazure/windowsazure/.

Atmos Online Programmer's Guide. EMC, a. https:// community.emc.com/docs/DOC-3481, accessed Jan 12, 2010.

D. Harnik, E. K. Kolodner, S. Ronen, J. Sataran, A. Shulman-Peleg, S. Tal,"Secure Access Mechanism for Cloud Storage", Journal of Scalable Computing: Practice and Experience Volume 12, Number 3, pp. 317–336. http://www.scpe.org

HERAS-AF XACML - University of Applied Sciences Rapperswil. http://www.herasaf.org/

SAP HANA Cloud Portal http://scn.sap.com/ community/hana-cloud-portal

Apach Derby http://db.apache.org/derby/

Redis Database http://redis.io/

Plattner, H. "A common database approach for OLTP and OLAP using an in-memory column database." *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 2009.

Unified Invocation of Scripts and Services for Provisioning, Deployment, and Management of Cloud Applications Based on TOSCA

Johannes Wettinger, Tobias Binz, Uwe Breitenbücher, Oliver Kopp, Frank Leymann and Michael Zimmermann

Institute of Architecture of Application Systems, University of Stuttgart, Universitätsstraße 38, Stuttgart, Germany
{*lastname, breitenbuecher*}*@iaas.uni-stuttgart.de, zimmerml@studi.informatik.uni-stuttgart.de*

Keywords: Management Plans, Management Operations, Script Invocation, Unified Interface, TOSCA, DevOps.

Abstract: There are several script-centric approaches, APIs, and tools available to implement automated provisioning, deployment, and management of applications in the Cloud. The automation of all these aspects is key for reducing costs. However, most of these approaches are script-centric and provide proprietary solutions employing different invocation mechanisms, interfaces, and state models. Moreover, most Cloud providers offer proprietary Web services or APIs to be used for provisioning and management purposes. Consequently, it is hard to create deployment and management plans integrating several of these approaches. The goal of our work is to come up with an approach for unified invocation of scripts and services without handling each proprietary interface separately. A prototype realizes the presented approach in a standards-based manner using the Topology and Orchestration Specification for Cloud Applications (TOSCA).

1 INTRODUCTION

Automated provisioning, deployment, and management of Cloud applications are key enablers to reduce costs of their operation (Leymann, 2009). However, it is hard to implement automated processes for managing applications in production after the actual development is finished. The main reason is that key aspects for operating an application in production were not considered during development. The idea of *DevOps* (development & operations) aims to eliminate the barrier between developers and operations personnel (Humble and Molesky, 2011). Consequently, a highly automated management environment has to be established during development already to continuously deploy and manage new iterations of an application to different environments (e.g., development, test, and production).

When automating such deployment and management processes there are typically a number of building blocks involved: (i) different Cloud providers expose proprietary APIs to *provision* and *manage* resources such as storage and virtual machines. (ii) Several tools such as Chef (Nelson-Smith, 2011) originating in the DevOps community provide a means to automatically *deploy* application components on top of the Cloud resources provisioned before. The DevOps community also publicly offers reusable artifacts to be used for the deployment of particular application components. (iii) Custom scripts (e.g., Unix shell scripts) may be implemented to *manage* the application during runtime (e.g., adding new users).

However, it is hard to combine these building blocks because the invocation mechanisms of Web services/APIs, scripts, and other plans involved in these automated processes differ significantly and the majority of them is not standards-based (Breitenbücher et al., 2013). The main goal of our work is to address this deficit by creating a management bus to provide a unified invocation interface hiding all the technical details of the employed management technologies. This interface is used by management plans implementing the logic required to automate a particular management task. The main contributions of our work are outlined in the following. We present:

- Concepts for unified invocation of different kinds of scripts (e.g., used for deployment) and different APIs (e.g., used for provisioning).
- Architecture and prototype of a unified invocation bus and interface to be used by management plans based on the Topology and Orchestration Specification for Cloud Application (TOSCA) (OASIS, 2013).
- Evaluation based on an end-to-end open source toolchain to show that plans get simpler and the

number of specific "wrappers" used for invoking certain scripts and services are reduced.

The remaining of this paper is structured as follows: 2 outlines the currently existing issues and limitations we address in our work. We present TOSCA as an emerging standard for managing Cloud applications in 3 because major parts of our work is based on it. 4 discusses the notion of a management plan and the orchestration of management operations using plans. Our concepts regarding the unified invocation approach are presented in 5, followed by an evaluation of this approach in 6. Finally, we discuss some related work in 7 and conclude this paper in 8.

2 PROBLEM STATEMENT

Regarding the state of the art of provisioning and deployment of Cloud applications, several approaches emerged in the last few years. Some of the most popular approaches today are based on tools originating in the DevOps community such as Chef (Nelson-Smith, 2011), Puppet (Loope, 2011), CFEngine (Zamboni, 2012), and Juju[1]. (Delaet et al., 2010) survey and compare some of the technical aspects of these tools in more detail. The main purpose of these tools is automating the installation and configuration of application components on physical and virtual machines. This is the reason why they are often called *configuration management* tools. To implement deployment logic for application components such as a Web server or a database, scripting languages are used. As an example, Chef uses a domain-specific language (Günther et al., 2010) based on Ruby to implement deployment and management operations. In case of Chef, these operation implementations are called *recipes,* bundled in *cookbooks*[2]. Since each tooling uses different concepts for developing, managing, and invoking management operations, it is hard to combine them in order to automate the deployment of a certain application stack. Such a stack typically consists of several components, some of which are hosted on others (e.g., a PHP application *hosted on* on an Apache HTTP Server) and connected to each other (e.g., an application is *connected to* a database). We further refer to such a stack as an *application topology.*

The hard-to-realize interoperability of these approaches would make a lot of sense because the communities affiliated with these tools provide plenty of reusable management operations such as cookbooks provided by the Chef community and charms[3] offered by the Juju community. These operations can be used to implement deployment plans automating the deployment of whole application stacks consisting of different components. Especially components that are commonly used as middleware such as the Apache HTTP Server can be deployed by parameterizing these operations. As an example for the deployment of a typical stack, a publicly available Juju charm can be used to create a MySQL database cluster, an already available Chef recipe may be used to create an instance of the Apache HTTP Server, whereas the deployment of a particular PHP application is implemented by a custom Unix shell script. As of today, however, the orchestration of these operations is limited to the boundary of a single ecosystem, including its operations, its tooling, and its community.

Another limitation of the approaches described before is their limited management view on application topologies. Since an application typically is not a monolithic block, there are several application components involved that are related to each other. Some of these tools provide a means to model topologies and stacks such as the Juju GUI[4] and Amazon Web Services' OpsWorks[5], but most of them do not. This limited management view, mostly focusing on single more or less isolated parts of a topology, makes it hard to manage large and distributed applications. Relations between components are only modeled implicitly and are typically hard-wired in the operations' implementation. However, especially the relations are important for management to understand the dependencies and possible side effects of a management action. What these tools miss anyhow is a portable and standards-based approach to specify such topologies. This is another reason why the orchestration of different operation types is hard to implement because if there is a means to model an application topology it is limited to a single ecosystem. Furthermore, different approaches own different state models where properties are stored following a certain structure. A central and overarching state model is required to access these data in a unified manner.

The provisioning of resources such as storage or virtual machines in the Cloud is another key aspect in this context. This is typically the first step of a deployment process because a machine is required to be running to deploy and manage application components on it. In addition, provisioning of resources is a fundamental part of many management tasks. For example, to scale an application, additional resources must be

[1]Juju: http://juju.ubuntu.com

[2]Cookbooks: http://community.opscode.com/cookbooks

[3]Juju Charms: http://jujucharms.com

[4]Juju GUI: http://juju.ubuntu.com/resources/juju-gui

[5]AWS OpsWorks: http://aws.amazon.com/opsworks

provisioned. There are standardization and unification efforts going on such as OpenStack (Pepple, 2011) and Deltacloud[6] to provide a unified API for provisioning tasks across different Cloud providers. However, in practice there are still lots of different APIs such as the Google Compute Engine API[7], the Amazon Web Services API[8], and Baremetalcloud's API[9] to name a few. Consequently, in a multi-cloud management scenario where application components are distributed across different Cloud providers (e.g., for resilience purposes), it is hard to combine them because of their different APIs.

The goal of our work is to address the issues described before by presenting an approach for creating overarching plans orchestrating different kinds of operations and APIs. Furthermore, plans, covering both deployment and runtime management of Cloud applications, should be able to access a holistic topology model covering the whole automatically deployed application stack.

We present the most important concepts of the Topology and Orchestration Specification for Cloud Applications in the next section. The purpose of using this emerging standard in our work is to specify application topologies and package them together with corresponding management plans in a portable manner.

3 TOSCA

The Topology and Orchestration Specification for Cloud Applications (TOSCA) (OASIS, 2013) enables the definition of application topologies including their application components, the relations between them, and the resources where they are hosted on such as VMs. It is an emerging standard supported by a number of prominent industry partners such as IBM and Hewlett-Packard to enable the portable definition of topology models. 1 shows a TOSCA-based *topology model* for SugarCRM[10], an open source customer relationship management software. The SugarCRM application topology is used by the TOSCA interoperability subcommittee[11] as an example for different vendors to check whether their TOSCA runtime environment is compliant to the specification. The topology model is a graph consisting of *nodes* that represent both application components (Apache HTTP Server, SugarCRM App, etc.) and infrastructure resources (VM 1, VM 2). These nodes are linked by *relationships* to express that a particular node is hosted on another one (e.g., SugarCRM App is hosted on Apache HTTP Server) and to specify a connection between two nodes (e.g., application connects to the database).

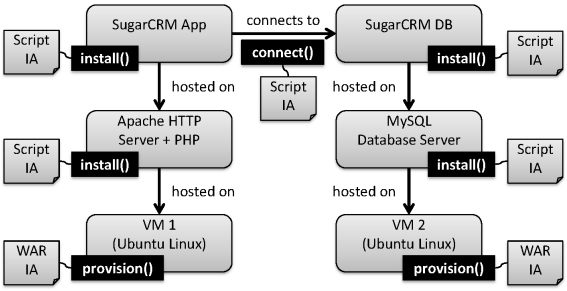


Figure 1: SugarCRM topology model.

Each node and relationship can have *management operations* attached. For instance, a VM node has a *provision* operation defined that is in charge of provisioning a new VM. Furthermore, nodes representing application components typically own an *install* operation to install and configure a particular application component. Operations can also be attached to relationships as it the case for the *connect* operation attached to the relationship wiring the application with the database. However, at this level an operation is still an abstract modeling construct. To make it executable, an *implementation artifact (IA)* needs to be attached to each operation that implements the operation. As an example, a WAR file exposing a Web service could be attached to the *provision* operation of a VM node. This WAR file is then deployed to a servlet container provided by the TOSCA runtime environment. Once the operation is invoked, a request is sent to the Web service performing the actions required to create a new VM. Another example is to attach a Unix shell script to the *install* operation of an application component node. When the operation is invoked, the script gets copied to the target VM and then is executed there. If several IAs are attached to a single operation, it is assumed that all of them implement the same logic based on different technologies. The TOSCA runtime environment is free to pick one of them to perform the operation.

In addition to defining the topology model and implementing the IAs, *plans* need to be written such as a build plan to create new instances of such a topology model. A plan basically defines the order of operation invocations to achieve a certain management goal (Binz et al., 2012). BPMN4TOSCA (Kopp et al., 2012) proposes TOSCA-specific modeling constructs to be used in plans based on BPMN (OMG, 2011). TOSCA

[6]Deltacloud: http://deltacloud.apache.org
[7]http://cloud.google.com/products/compute-engine
[8]AWS API: http://aws.amazon.com/documentation
[9]Baremetalcloud: http://www.baremetalcloud.com
[10]SugarCRM: http://www.sugarcrm.com
[11]SugarCRM Interop Topology: http://goo.gl/C6pqRd

further defines a packaging format called *Cloud Service Archive (CSAR)* to deliver all ingredients belonging to a particular topology model such as IAs, plans, and the topology model itself as a single file, a CSAR.

The following 4 outlines the notions of management plans and management operations as well as how they work together. These are the fundamentals for the concepts we describe later.

4 MANAGEMENT PLANS AND OPERATIONS

Management logic is implemented on different levels: script-centric artifacts provided by the DevOps community (Wettinger et al., 2013) typically implement the installation and configuration of application components such as an Apache HTTP Server or a PHP module on a physical or virtual machine (VM). Since the nature of these artifacts is relatively fine-grained, specifying low-level management tasks that need to be performed, we refer to these artifacts as *management operations.* They are typically designed in a way to perform a configurable "atomic" management action such as *installing an Apache HTTP Server* or *adding a new user account to an application.* These operations focus on the deployment of application components as well as their management at runtime such as changing the configuration of a particular component:

Management Operations *automate management actions that are considered "atomic", i.e., it does not make sense to split them up further to achieve a certain low-level management goal.*

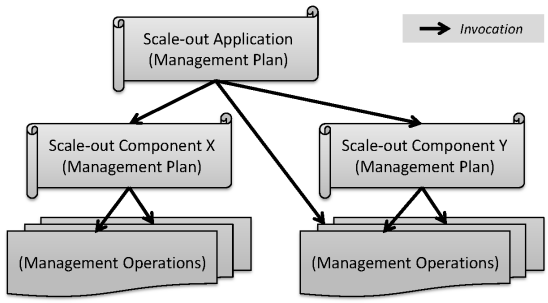


Figure 2: Plans orchestrate plans and operations.

A particular management operation may, for instance, represent a script performing a particular management action on a VM. However, it could also be an invocation of a service or an API offered by a Cloud provider, for instance, to provision new resources such as VMs. In order to enable the orchestration of several management operations, for instance, to deploy a complete application topology, more coarse-grained plans are required. We refer to these plans as *management plans:*

Management Plans *control the invocation of management operations and other management plans. They specify the proper ordering of the invocations as well as handling the input and output of each invocation.*

As mentioned in the definition before, plans may also be used to combine existing plans and operations, so plans can be positioned on different levels as shown in 2. As an example, a scalable application typically consists of several components that are deployed in a distributed manner. In order to scale such an application in or out, some of the components need to be scaled according to the current workload. There could be a management plan for each component that realizes its scale-out. A plan on this level (e.g., implemented as a Ruby script) may invoke operations (e.g., Chef recipes) to reconfigure deployed instances that belong to this particular component of the application as well as operations to create and configure new instances to enable the scale-out. Moreover, an overarching management plan (e.g., realized as a workflow) could be implemented to invoke individual management plans for each component involved in the scale-out process. Finally, management plans enable multi-cloud management scenarios: a particular plan may invoke different operations for resources distributed across different Cloud providers such as installing a database on a VM at provider A and installing an application server on a VM at provider B. Furthermore, such a management plan can use APIs and services (management operations) exposed by different providers to provision new resources such as storage or a VM.

Actually, management plans are very similar to workflows and business processes. (Binz et al., 2012) describe how to use existing workflow technology and standards such as the Business Process Model and Notation (BPMN) (OMG, 2011) and the Business Process Execution Language (BPEL) (OASIS, 2007) to implement and execute management plans. The recursive character of management plans, i.e., orchestrating existing management plans by another management plan, is similar to how BPEL deals with Web services: a BPEL workflow is not limited to invoking existing Web services. It can also be exposed as a new Web service that can then be invoked by another BPEL workflow, possibly as subprocess (Kopp et al., 2010). The distinction of management logic in plans and operations is similar to how workflows are typically distinguished: low-level workflows (similar to operations) are often referred to as *micro flows* (Manolescu and Johnson, 2001) that can be orchestrated by "real" workflows (similar to plans).

By implementing management plans as workflows based on established standards such as BPMN or BPEL, the plans' portability is improved. Existing general-purpose workflow engines can be used to execute these plans. Moreover, the plans inherit the properties of workflows such as recoverability and compensation as discussed by (Binz et al., 2012).

To enable the portability of management operations, domain-specific languages (DSLs) such as the Ruby-based DSL used by the Chef community (Günther et al., 2010) can be used. These languages provide platform-independent abstractions for typical low-level management and deployment actions such as installing a particular software package, writing a configuration file, or setting file permissions.

Again similar to workflows, there are different flavors of plans and operations: management logic can be implemented in an imperative or in a declarative manner. Whereas a declarative implementation only specifies *what* actions have to be performed, an imperative implementation additionally describes *how* these actions need to be performed. As an example, a declarative management operation may define a set of software packages that need to be installed. The imperative variant would further define how to install these packages. Imperative management plans based on workflows can be realized using BPMN or BPEL. However, there are also declarative workflow languages (Pesic and van der Aalst, 2006) available that could be used to implement declarative management plans. For management operations, the very same distinction exists: some of the DevOps tools such as Puppet provide a declarative domain-specific language, whereas Chef recipes, for instance, can be implemented using both a declarative (Ruby-based domain-specific language) and an imperative approach (plain Ruby). Furthermore, Unix shell scripts as for instance used by several Juju charms are a typical example for imperative management operations because their logic encapsulates several system commands that exactly define what action to perform and how to perform this action. Conceptually, there is no limitation to combining plans and operations of different flavors. As a result, an imperative management plan could invoke several declarative plans and operations or vice versa.

In the following section we present concepts for a standards-based approach to tackle the problems outlined in 2. Furthermore, we come up with an architecture and a prototype that realizes these concepts based on TOSCA.

5 UNIFIED INVOCATION INTERFACE FOR PLANS

As outlined in 2 it is difficult to combine different management operations such as Chef recipes, Unix shell scripts, or certain provider APIs because each of them has individual invocation mechanisms. To address this issue the goals of our concepts presented in this section are threefold:

1. Creating management plans has to be as simple as defining the flow logic (e.g., a sequence) of operations of particular nodes or relationships in a TOSCA topology model. The technical details of invoking a particular operation are completely hidden to such a management plan.
2. The number of implementation artifacts (IAs) created for a CSAR should be reduced as much as possible. As an example, the CSAR should not deliver an IA that implements the execution of scripts of a certain kind, because this logic does not immediately belong to the application's management logic and thus has to be provided by other means.
3. Different operations in a topology model can be implemented using different kinds of IAs such as Chef recipes, Unix shell scripts, or Juju charms. However, there should not be any dependency on additional IAs (e.g., an IA exposed as Web service to execute a Chef recipe) to enable the execution and orchestration of such operations.

All these goals enable *separation of concerns,* so that (i) a management plan focuses on the logic of combining operations properly, (ii) a CSAR contains artifacts and plans only related to the application it specifies, and (iii) different kinds of IAs can be used to implement a particular operation without changing higher-level management plans.

To achieve these goals a *unified invocation interface* is required that can be used by plans to invoke operations without knowing what kind of IA or technology is used in the background. The unified invocation interface is implemented based on OpenTOSCA (Binz et al., 2013), an open source TOSCA runtime environment. OpenTOSCA is one of the core components of an end-to-end toolchain to create and process TOSCA models. First, we outline how OpenTOSCA deals with plans, operations, and their invocation without such an interface. Then, we describe how we improved OpenTOSCA addressing the issues we discussed before.

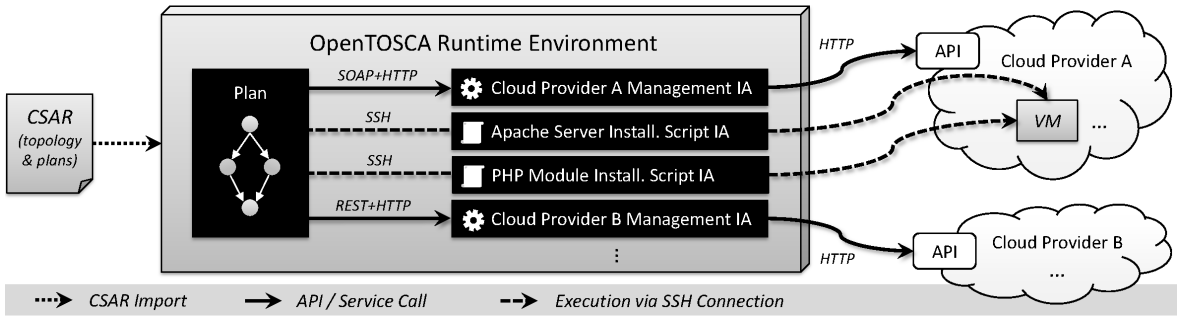


Figure 3: Original architecture without a unified invocation interface for plans.

5.1 Original Architecture of OpenTOSCA

3 shows a simplified overview of the architecture of OpenTOSCA. It further outlines the process of creating an instance of a particular application topology specified by a CSAR as it is currently implemented without the issues in mind we mentioned before. The initial step is to import a CSAR to the runtime environment, so that new instances can be created based on the model described in the CSAR. In addition to the actual topology model, the CSAR further contains all configuration and management plans as IAs and TOSCA management plans (Binz et al., 2012). We assume that each CSAR contains an overarching *build plan* specifying which operations of nodes and relationships in the topology have to be invoked in which order. This plan is executed by the OpenTOSCA runtime environment to build a new instance of the topology described in the model.

When the plan invokes a particular operation the corresponding implementation artifact (IA) is executed. Some IAs may be implemented as SOAP-based Web services in Java (WAR files), others may be implemented as Unix shell scripts or Chef recipes. When importing a CSAR, all implementation artifacts that have to run in the local OpenTOSCA management environment are deployed. For instance, the IAs providing Web services implemented in Java are hosted on a local Java servlet container that is part of the OpenTOSCA runtime environment. We refer to these IAs as *local IAs*. OpenTOSCA further supports local IAs providing RESTful Web services implemented in Java. All other IAs such as script IAs are obviously not deployed locally because they need to be executed on the target VMs remotely. Therefore, we refer to these IAs as *remote IAs*. Plans are bound to local IAs, so the corresponding plan and IAs are hard-wired. This fact decreases the portability and reusability of plans.

The sample plan outlined in 3 depicts several management operations that are performed in a multi-cloud management scenario. These represent a subset of operations required to create a new instance of the SugarCRM application topology described in 3. (i) The plan uses the local *Cloud Provider A Management IA* to interact with the API of provider A. The same applies to the *Cloud Provider B Management IA* for provider B. However, the invocation interfaces differ (SOAP+HTTP vs. REST+HTTP). Consequently, the plan needs to be aware of the interfaces' idiosyncrasies in terms of invocation, passing input data, and receiving the result of the execution. (ii) Furthermore, the plan invokes operations implemented by script IAs (Unix shell scripts) such as the *Apache Server Installation Script IA*. Since these scripts need to be executed remotely on a target VM at the provider, there are two options to deal with remote execution: the plan itself could implement the management logic for copying the scripts to the target machine using SSH, setting input and environment variables accordingly, performing the execution, and retrieving its output. Alternatively, a local wrapper IA such as an *SSH Unix Shell Script Invoker IA* could be implemented and packaged with a CSAR. This IA would expose the management logic for handling the execution of a certain type of script (Unix shell scripts) as Web service to be used by the plan. The second option is typically necessary in case the plan is implemented as workflow, e.g., based on BPEL, because such workflows cannot establish SSH connections without proprietary extensions.

Both options for executing remote IAs such as scripts pollute plans and CSARs with management logic that does not belong to the application's management logic and should therefore be provided by the management environment. In this case, such management logic can be reused by different CSARs. Furthermore, plans need to deal with different invocation mechanisms such as SOAP+HTTP, REST+HTTP, and SSH-based script execution. Thus, plans can get complex even if the application topology is relatively simple.

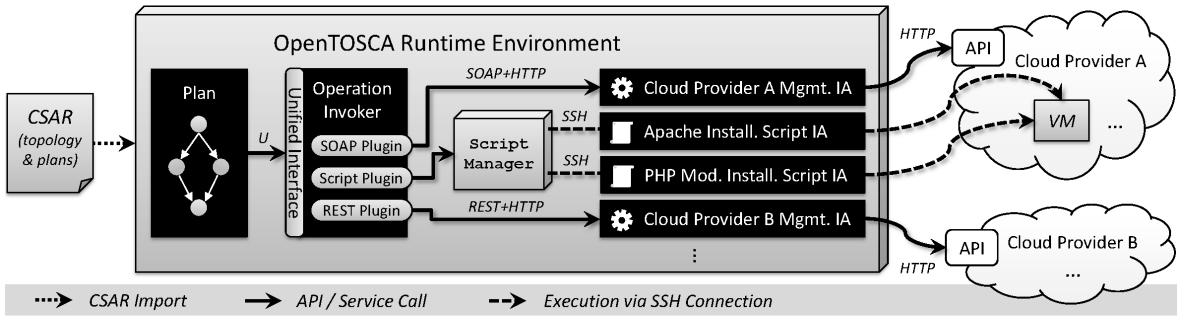


Figure 4: Improved architecture providing a unified invocation interface for plans.

5.2 Improved Architecture with Unified Invocation Interface

Based on the state of OpenTOSCA without a unified invocation interface presented before, 4 outlines how we improved OpenTOSCA's architecture to provide a unified invocation interface to plans. Thus, we implemented an additional component, namely the *Operation Invoker* acting as a bus in the sense of a service bus (Chappell, 2004). This component provides a unified interface to plans to invoke operations. In contrast to the original architecture outlined in 3, the plan does only communicate with the single unified interface of the Operation Invoker. Technically, the unified interface is implemented as SOAP-based Web service. However, it could alternatively or additionally make the interface accessible through a RESTful Web service or any other communication protocol.

The unified interface enables loose coupling between plans and IAs. When a particular operation is called through the Operation Invoker, it checks what kind of IA is available to execute the operation. Then, it checks whether there is a plugin registered that can execute IAs of the given type (e.g., an IA of type *SOAP* or of type *REST*). In case there is a corresponding plugin, this plugin gets invoked, which itself invokes the corresponding IA. The Operation Invoker is extensible by adding plugins to it enabling OpenTOSCA to process any kind of IA. These plugins are implemented using OSGi[12].

(Leitner et al., 2009) and (De Antonellis et al., 2006) present solutions similar to the Operation Invoker in order to dynamically invoke different kinds of Web services. However, these solutions are limited to service invocation and do not consider the execution of arbitrary executables such as scripts on a VM. In case of deploying and managing Cloud applications, service invocations do typically only cover provisioning tasks, e.g., creating a new VM. The actions to install and configure application components on the VM are typically implemented as scripts. We discussed this fact in 2, especially in the context of the tools such as Chef or Puppet originating in the DevOps community. Thus, it is essential for the Operation Invoker and the unified invocation interface that it provides a means to manage and monitor the execution of scripts. To be able to easily combine different kinds of scripts such as Chef recipes or Unix shell scripts the Operation Invoker has to deal with their individual invocation mechanisms and their way of handling input and output.

We decided to create the *Script Plugin* to enable the Operation Invoker to deal with IAs that are implemented as scripts. However, the actual logic for dealing with the idiosyncrasies of different types of scripts is realized by a separate reusable component, namely the *Script Manager,* which is used by the Operation Invoker's Script Plugin. When a plan invokes an operation that is implemented as a script, the Service Invoker calls the Script Manager and passes the references to the script files, some meta information such as the type of the script, and the context such as the properties of the node of which the particular operation belongs to. The following XML listing shows an example for a message that is created by the Script Plugin when a plan invokes the *install* operation of the *SugarCRM App* node:

```
<Definitions ...
 targetNamespace="http://www.opentosca.org/script"
 xmlns="http://.../tosca"
 xmlns:toscaBase="http://.../toscaBaseTypes"
 xmlns:s="http://www.opentosca.org/script">

 <ArtifactTemplate ... type="toscaBase:ScriptArtifact">
  <Properties>
   <toscaBase:ScriptArtifactProperties ...>
    <ScriptLanguage>sh</ScriptLanguage>
    <PrimaryScript>scripts/install.sh</PrimaryScript>
   </toscaBase:ScriptArtifactProperties>
  </Properties>
  <ArtifactReferences>
   <ArtifactReference reference="scripts/install.sh" />
  </ArtifactReferences>
```

[12]OSGi: http://www.osgi.org

```
17 </ArtifactTemplate>
18
19 <s:ArtifactContext>
20  <!-- OpenTOSCA's RESTful API to access CSAR content -->
21  <Files url="http://localhost/.../CSARs/SugarCRM/" />
22
23  <Operation>
24   <InputParameter name="licenseKey" type="string">
25    DUMMY_KEY
26   </InputParameter>
27   <OutputParameter name="stdout" type="string" />
28  </Operation>
29
30  <Node>
31   <!-- properties of SugarCRM App node -->
32   <NodeProperties nodeTypeName="SugarCRMApp">
33    <!-- admin password, app settings, etc. -->
34   </NodeProperties>
35
36   <!-- properties of server node on which the
37        SugarCRM App node is hosted -->
38   <HostProperties nodeTypeName="UbuntuLinuxServer">
39    <!-- public DNS address, SSH credentials, etc. -->
40   </HostProperties>
41  </Node>
42 </s:ArtifactContext>
43 </Definitions>
```

This message is then sent to the Script Manager. First, the script files (line 15) are retrieved from OpenTOSCA's RESTful API endpoint (line 21). Second, the Script Manager establishes, for instance, an SSH connection to copy the script files to the target VM, to set the corresponding environment variables, to execute the scripts, and to retrieve the result of the execution. The Operation Invoker's Script Plugin uses a RESTful API provided by the Script Manager to monitor the status of the execution and to finally retrieve the output to pass it back to the plan through the unified invocation interface. Such a message is structured as shown in the following XML listing:

```
1 <body>
2  <status>completed</status>
3  <output>
4   <param>
5    <name>stdout</name>
6    <type>string</type>
7    <value>SugarCRM installed successfully</value>
8   </param>
9  </output>
10 </body>
```

We evaluated our concepts based on the improved architecture of OpenTOSCA to show that our solution proposal can be implemented in practice to address the issues outlined in 2. The following section discusses the evaluation in more detail.

6 EVALUATION

Based on the topology of SugarCRM described in 3 we show that the concepts and the improved architecture presented in the previous 5 can be implemented in practice to address the issues discussed in 2. The overarching build plan to create new instances of SugarCRM is implemented using BPEL (OASIS, 2007). The main reason for using standards-based workflow languages as proposed in TOSCA is to benefit from the workflows' advantages such as recoverability and to reuse existing workflow engines for plan execution. However, the plan's implementation does not rely on any unique feature of BPEL. If certain workflow properties such as recoverability or compensation are not required for a particular plan it could also be implemented, for instance, using a scripting language such as Ruby or Python.

For evaluation purposes we created two CSARs including a build plan for each of them: (i) the original one does not use the Operation Invoker at all, so the build plan directly invokes Web services provided by local IAs bound to certain operations. The IAs' implementation is part of the CSAR. Consequently, a local wrapper IA was created to copy scripts on VMs and execute these scripts because a BPEL plan cannot do this directly. (ii) The improved CSAR's build plan invokes all operations through the Operation Invoker, meaning there is no direct dependency to any IA. Furthermore, there is no IA needed to invoke scripts because this task is delegated to the Script Manager. By using the Operation Invoker for each operation invocation centralized monitoring can be implemented because all input and output is going through a central component. 1 summarizes the evaluation results showing that (i) the complexity (measured as the number of XML elements in the BPEL plan) of the improved CSAR's build plan is lower, so plan creation gets simplified by using the Operation Invoker. This is mainly because many low-level actions such as copying scripts to the target machine do not appear in the plan. (ii) In contrast to the original CSAR the improved one does not require a special IA for invoking scripts because the

Table 1: Evaluation Results.

CSARs:	**Original**	**Improved**
Complexity of build plan (number of XML elements)	563	161
Wrapper IA for script invocation	yes	no
IAs exchangeable without changing plans	no	yes

Operation Invoker uses the Script Manager for this purpose. (iii) IAs can be arbitrarily exchanged in case of the improved CSAR because the plan only points to the operations that need to be invoked. Then, the Operation Invoker deals with the IAs and invokes them accordingly.

7 RELATED WORK

Invoking Web services dynamically by using some kind of unified invocation interface was discussed before (Leitner et al., 2009; De Antonellis et al., 2006). These approaches do not consider the invocation of arbitrary management operations such as Chef recipes or Unix shell scripts. However, their invocation is of utmost importance in case of creating management plans because many operations such as installing and configuring application components are implemented as scripts. In the field of multi-cloud research there is related work (Petcu et al., 2011; Moscato et al., 2011; Sampaio and Mendonça, 2011) proposing concepts and architectures to talk to different Cloud providers using a unified API. Only some of them (Liu et al., 2011) marginally consider the invocation of scripts on VMs in the target Cloud. Most of the time it is assumed that application components are bundled and deployed as virtual image files, e.g., using OVF.

In terms of provisioning resources such as VMs, libraries (e.g., jclouds[13] and fog[14]) enable the abstraction of specific Cloud provider APIs. However, their functionality is limited to provisioning tasks and they can only be used in a programmatic manner. Consequently, a standards-based plan (e.g., based on BPEL or BPMN) cannot use such libraries directly without a wrapper exposing the libraries' functionality as Web services. Beside these libraries, approaches such as cloud-init[15] can be used to run scripts on a VM when it gets started. This may work for the initial deployment of application components. However, arbitrary management operations such as a database backup that need to be performed later cannot be covered by this approach.

Some DevOps approaches can also perform provisioning actions. However, they are either bound to specific Cloud providers such as AWS OpsWorks[16], or they are mainly used as a command-line tool such as Chef's knife[17] and Juju[18]. Beside the DevOps tools such as Chef or Puppet, there is another category of tools emerging. These are in particular platform-as-a-service (PaaS) frameworks such as Cloud Foundry[19]. Their goal is to provide an integrated environment, namely a PaaS environment (Mell and Grance, 2011), for deploying and running applications and all their management processes. However, standards such as TOSCA are not considered there.

[13]jclouds: http://jclouds.incubator.apache.org

[14]fog: http://fog.io

[15]cloud-init: http://launchpad.net/cloud-init

[16]AWS OpsWorks:http://aws.amazon.com/opsworks

[17]Knife: http://docs.opscode.com/knife.html

[18]Juju: http://juju.ubuntu.com

8 CONCLUSIONS AND FUTURE WORK

The unified invocation interface presented in 5 cleanly decouples overarching management plans (e.g., the application's build plan) from their underlying management operations (e.g., IAs implemented as Chef recipes and Unix shell scripts). The Script Manager used to invoke management operations implemented as scripts can be arbitrarily extended to support the execution of any kind of scripts. This enables reusing existing management operations published by the DevOps community that can be embedded into CSARs to implement certain operations. Finally, IAs exposing SOAP-based or RESTful Web services (e.g., to provision a new VM) can also be invoked by a plan through the Operation Invoker without binding the plan to a certain Web service. As a result, management plans focus on the actual orchestration of different operations and thus do not have any dependency on particular IAs or scripts. Our evaluation presented in 6 confirmed that this approach can be implemented in practice. Furthermore, the complexity of plans can be notably decreased.

In terms of future work we aim to extend our prototype in two dimensions: (i) extending the Script Manager to process a wide range of existing management operations (Puppet, Juju, etc.) and (ii) exposing the Operation Invoker's unified interface as RESTful Web service to make it directly usable for scripts implemented in Ruby, Python, etc. This enables the implementation of plans based on scripting languages as an alternative to workflows. Provisioning libraries such as jclouds mentioned in 7 can be wrapped as IAs to be reused in different CSARs for provisioning tasks.

ACKNOWLEDGEMENTS

This work was partially funded by the BMWi project CloudCycle (01MD11023).

[19]Cloud Foundry: http://cloudfoundry.org

REFERENCES

Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., and Wagner, S. (2013). OpenTOSCA - A Runtime for TOSCA-based Cloud Applications. In *Proceedings of 11th International Conference on Service-Oriented Computing (ICSOC'13)*, volume 8274 of *LNCS*, pages 694–697. Springer Berlin Heidelberg.

Binz, T., Breiter, G., Leymann, F., and Spatzier, T. (2012). Portable Cloud Services Using TOSCA. *Internet Computing, IEEE*, 16(3):80–85.

Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., and Wettinger, J. (2013). Integrated Cloud Application Provisioning: Interconnecting Service-centric and Script-centric Management Technologies. In *Proceedings of the 21st International Conference on Cooperative Information Systems (CoopIS 2013)*.

Chappell, D. A. (2004). *Enterprise Service Bus*. O'Reilly.

De Antonellis, V., Melchiori, M., De Santis, L., Mecella, M., Mussi, E., Pernici, B., and Plebani, P. (2006). A layered architecture for flexible Web service invocation. *Software: Practice and Experience*, 36(2):191–223.

Delaet, T., Joosen, W., and Vanbrabant, B. (2010). A Survey of System Configuration Tools. In *Proceedings of the 24th Large Installations Systems Administration (LISA) conference*.

Günther, S., Haupt, M., and Splieth, M. (2010). Utilizing Internal Domain-Specific Languages for Deployment and Maintenance of IT Infrastructures. Technical report, Very Large Business Applications Lab Magdeburg, Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg.

Humble, J. and Molesky, J. (2011). Why Enterprises Must Adopt Devops to Enable Continuous Delivery. *Cutter IT Journal*, 24(8):6.

Kopp, O., Binz, T., Breitenbücher, U., and Leymann, F. (2012). BPMN4TOSCA: A Domain-Specific Language to Model Management Plans for Composite Applications. In *Business Process Model and Notation*, volume 125 of *Lecture Notes in Business Information Processing*, pages 38–52.

Kopp, O., Eberle, H., Leymann, F., and Unger, T. (2010). The Subprocess Spectrum. In *Proceedings of the Business Process and Services Computing Conference: BPSC 2010*, volume P-177 of *Lecture Notes in Informatics*, pages 267–279. Gesellschaft für Informatik e.V. (GI).

Leitner, P., Rosenberg, F., and Dustdar, S. (2009). Daios: Efficient Dynamic Web Service Invocation. *Internet Computing, IEEE*, 13(3):72–80.

Leymann, F. (2009). Cloud Computing: The Next Revolution in IT. In *Photogrammetric Week '09*. Wichmann Verlag.

Liu, T., Katsuno, Y., Sun, K., Li, Y., Kushida, T., Chen, Y., and Itakura, M. (2011). Multi cloud management for unified cloud services across cloud sites. In *Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on*, pages 164–169.

Loope, J. (2011). *Managing Infrastructure with Puppet*. O'Reilly Media, Inc.

Manolescu, D.-A. and Johnson, R. E. (2001). *Micro-Workflow: A Workflow Architecture Supporting Compositional Object-oriented Software Development*. University of Illinois at Urbana-Champaign.

Mell, P. and Grance, T. (2011). The NIST Definition of Cloud Computing. *National Institute of Standards and Technology*.

Moscato, F., Aversa, R., Di Martino, B., Fortis, T., and Munteanu, V. (2011). An analysis of mosaic ontology for cloud resources annotation. In *Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on*, pages 973–980.

Nelson-Smith, S. (2011). *Test-Driven Infrastructure with Chef*. O'Reilly Media, Inc.

OASIS (2007). Web Services Business Process Execution Language (BPEL) Version 2.0.

OASIS (2013). Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0, Committee Specification 01.

OMG (2011). Business Process Model and Notation (BPMN) Version 2.0.

Pepple, K. (2011). *Deploying OpenStack*. O'Reilly Media.

Pesic, M. and van der Aalst, W. M. (2006). A Declarative Approach for Flexible Business Processes Management. In *Business Process Management Workshops*, pages 169–180. Springer.

Petcu, D., Craciun, C., Neagul, M., Lazcanotegui, I., and Rak, M. (2011). Building an interoperability api for sky computing. In *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, pages 405–411.

Sampaio, A. and Mendonça, N. (2011). Uni4cloud: An approach based on open standards for deployment and management of multi-cloud applications. In *Proceedings of the 2Nd International Workshop on Software Engineering for Cloud Computing*, SECLOUD '11, pages 15–21. ACM.

Wettinger, J., Behrendt, M., Binz, T., Breitenbücher, U., Breiter, G., Leymann, F., Moser, S., Schwertle, I., and Spatzier, T. (2013). Integrating Configuration Management with Model-Driven Cloud Management Based on TOSCA. In *Proceedings of the 3rd International Conference on Cloud Computing and Services Science (CLOSER)*. SciTePress.

Zamboni, D. (2012). *Learning CFEngine 3: Automated System Administration for Sites of Any Size*. O'Reilly Media, Inc.

Analysing the Lifecycle of Future Autonomous Cloud Applications

Geir Horn[1], Keith Jeffery[2], Jörg Domaschka[3] and Lutz Schubert[3]
[1]*University of Oslo, Olav Dahls hus, P.O. Box 1080, 0316 Oslo, Norway*
[2]*Keith G Jeffery Consultants, 10 Claypits Lane, Shrivenham, U.K.*
[3]*University of Ulm, 89069 Ulm, Germany*
geir.horn@mn.uio.no, {joerg.domaschka, lutz.schubert}@uni-ulm.de

Keywords: Cloud Computing, Software Design, Autonomy, Adaptability.

Abstract: Though Cloud Computing has found considerable uptake and usage, the amount of expertise, methodologies and tools for efficient development of in particular distributed Cloud applications is still comparatively little. This is mostly due to the fact that all our methodologies and approaches focus on single users, even single processors, let alone active sharing of information. Within this paper we elaborate which kind of information is missing from the current methodologies and how such information could principally be exploited to improve resource utilisation, quality of service and reduce development time and effort.

1 INTRODUCTION

Clouds are on the rise. A major problem is heterogeneity of offered platforms and their interfaces with consequent inability to port applications. Traditional methodologies and software engineering principles still dominate the commercial software industry. These principles were generated for single-user and typically single-processor use, i.e. not for cloud concepts.

This paper argues the necessity to involve additional skills beyond traditional software development in order to master successfully the Cloud deployment of an application. We describe the application aspects that need to be provided as a profile by the application owner; the characteristics of Cloud platforms that are supplied by the platform operator; the characteristics of the data used by the application and the characteristics of the user input must be provided by the user as a profile partly modifiable for each individual execution of the application. This information is metadata to be used by the systems development process.

The main contribution of this paper is to identify the data that must be considered for an effective deployment, and show that the data requires continuous monitoring to unlock the full benefits offered by the elasticity and scalability of the Cloud. This implies that the profiles may need continuous monitoring and essentially that the application owner needs to be aware of this application lifecycle. Section 2 highlights the core aspects of application deployment, and Section 3 identifies the data needed for the deployment and the data profiles.

The results presented here base on the initial work in the PaaSage project, and the architecture for a system to support model driven autonomic deployment and application adaptation is described in Section 4. Although the PaaSage project is working on implementing the necessary building blocks, it is too early to report on the implementation of the application lifecycle in this paper.

2 ASPECTS OF APPLICATION DEPLOYMENT

Current cloud applications are designed and developed in much the same way as traditional applications. The developer / provider only starts thinking about the cloud specific characteristics once he starts to deploy, respectively host the application.

In order to understand why traditional engineering principles are insufficient and how they need to change, it is necessary to analyse which kind of factors play a role at executing the main Cloud characteristics, including specifically:

1. sharing data (and computation)
2. replication and elasticity
3. (re)location and distribution

With traditional software engineering, the

application would be designed to essentially host one complete instance for a single user. Data sharing has to be explicitly encoded and is not represented on the level of actual execution state, let alone that it automatically caters for consistency problems. To make best use of resources (and thus to optimise cost and quality of service), it is furthermore necessary to assess the scaling behaviour of the individual functionalities that compose the full application logic.

Figure 1 depicts such varying scaling behaviour of a simple online book store: following traditional principles would result in a single workflow per each user, including the services and data base. However, information such as the database needs to be available to all users, but the store front and the publisher access are not necessary per user. However, no current software engineering principle allows for description of a behaviour as depicted.

The implication of this scaling behaviour, however, has to be assessed on the overarching level, as for example two individual services in the application workflow may scale out beyond the total cost agreements, simply because the behaviour of the individual services did not cater for the total impact.

We will elaborate the impacting characteristics in more detail in the following sections:

2.1 Usage Aspects

There is a general tendency to treat a Cloud application as a self-sustained application in a full-blown virtual image, though an increasing number of modern cloud applications are actually complex business processes distributed over multiple resources and multiple virtual images and potentially integrating the capabilities of different Cloud infrastructures at the same time, cf. (Schubert and Jeffery, 2012). As has been shown, the intrinsic characteristics of such applications determine the scaling behaviour, and in particular its constraints, and thus the optimal resource utilisation, see e.g. (Becker, Koziolek and Reussner, 2009), (Rubio Bonilla, Schubert and Wesner, 2012) (cf. Figure 1).

Not only the number of users (which may vary significantly over time) impacts on application scale, but also the usage behaviour of each individual user. Already a comparatively simple application, such as book selling exposes multiple functionalities: searching for books, adding / removing from lists, ordering from publisher etc. – all triggering different processes and necessitating different data.

It may be safely assumed that the application structure, namely its processes and their potential relationship, is already known. This – structural – information serves also one further purpose: the relationship between processes (usually as services) and their potential impact on execution criteria such as performance, data consumption etc.:

Each service contributes differently to the overall

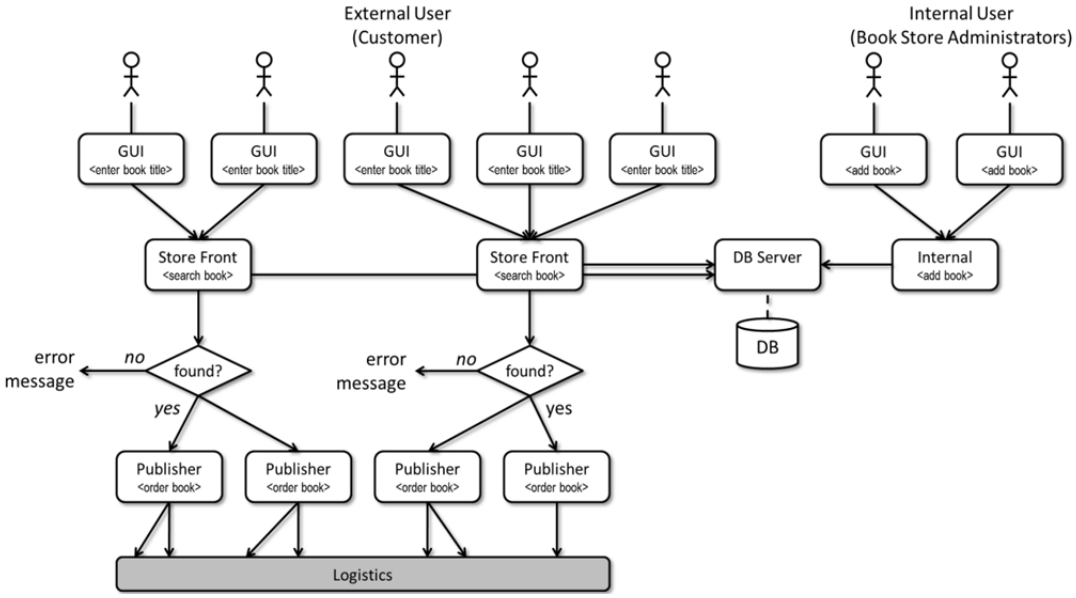


Figure 1: Potential distributed workflow with different overlapping processes and different scaling behaviour per service instance.

application workflow and hence exposes different (a) resource requirements and (b) connections during scaling. For example an individual instance of service A may serve up to 9(!) instances of service B provided that B only contributes 1/10 to the total load. The multiple instance of B may be the consequence of scaling out due to increased user demand.

2.2 Performance Aspects

The application workflow information can indicate problem areas, such as bottlenecks. In order to assess these, standard relationship information needs to be extended with data about the communication frequency and width (average size and range size of messages), as well as performance impact per delay.

It will be noted that the communication impact is not only defined by the data exchange, but logically by the data in use itself. As with any application, data size and structure can have strong impact on overall performance. Non- or semi-structured data has particular characteristics when used for searching, pattern detection etc. and streamed data has even more difficult characteristics requiring windowing of the stream and multiple complex parameters being maintained to assure effective and efficient management of the data. Restructuring for efficiency improvement may prove difficult to achieve in a Cloud environment especially if the data is shared.

Even if performance plays a secondary role in the context of cloud computing so far, *quality* of the provided services is a major concern. In this context, availability (uptime and response time) is crucial for customer satisfaction: if a specific service is not available at request time, it will stall the execution of the total application workflow. Scaling out to reach the desired quality of service would mean: (1) higher resource usage and thus higher cost, as well as (2) higher risk of bottlenecks and potential problems that cannot be compensated by scale out.

Also the technical characteristics of the hosting environment plays a major role in meeting the quality of service: for example, the database system may not be suited for the type of access requirements by the application, or storage may be insufficient etc.

2.3 Deployment Criteria Overview

We can thus isolate the following main "external" criteria that (should) influence the behaviour of a cloud application, but are not accessible to the execution environment in any form, so that it must be explicitly provided by the application owner (or the Cloud provider):

(1) *Number of service consumers (users)*: increased number of users will increase traffic and potential resource conflicts associated with access to threads of code, data etc.

(2) *Communication behaviour:* the time and size of data exchange is a critical performance factor and may lead to conflicts if it coincides with a large number of users or background processes

(3) *Usage behaviour*: process invocation frequency and sequence may lead to different resource usage and potentially conflicts

(4) *Workflow:* specifies the application's structure and thus potential bottlenecks, scaling conditions etc.

(5) *Data structure*: classify the impact arising from typical data types and structures related to the execution of the application

(6) *Quality constraints* (provider, host, consumer) specify the actual business constraints and goals of the actors and must therefore be respected throughout the whole application lifecycle.

(7) *Technical characteristics*: define the performance of a given application with respect to its explicit and implicit resource need. This is especially true for specialised platforms.

Up until now, the developer and potentially the application provider had to cater for these aspects implicitly by generating the necessary configuration information manually, respectively by incorporating the according behaviour directly into the application code or workflow. Not only do most developers lack such expertise, but it also makes the code difficult to adjust to new environments and usage conditions.

3 CONCRETE DEPLOYMENT INFORMATION

The information identified in the previous section must not only be captured in a form that is intuitive and meaningful – more importantly, it must be specified in a form that can be used for deployment and execution control purposes. Therefore this information must be provided in a common format that allows combination, selection and reasoning over it, so as to select the right rules that apply under the given circumstances.

In general, this means that the specifications must consider the following aspects:

- *Conditions:* under which circumstances does the information apply. For example, given a generic rule to scale out when the number of users increase, when does this behaviour really apply? When would it create a bottleneck? The main problem with these conditions is that they are generally not well-defined and cannot be represented as simple scalar tests. Hence they must be able to cater for probabilistic values.
- *Actions* to perform when the conditions are met, e.g., to scale out by a certain number, to relocate the service etc. The composed actions of all services must thereby meet the overall quality constraints. This means that actions may vary between use cases.
- *Events:* conditions are only evaluated if certain events occur. In most cases these events will be triggered by environmental circumstances. Like conditions, they will have to deal with uncertainty in order to allow for delays and predict likely occurrences.

Even though these three aspects are clearly related to the Event-Condition-Action definition commonly used in logic reasoning (Helmer, Poulovassilis and Xhafa, 2011), there are some substantial differences, as will be elaborated in section 4.

3.1 User Profile

The *user profile* describes the way in which the end-user (application end user, application developer or administrator) interacts with the Cloud environment. The expected parameters are required for access control and security – however, other user characteristics are also of interest for better interaction, such as preferred interaction mode, preferred language, screen set-up etc. It should be noted that the user profile information can change dynamically, e.g. access control as the application accesses different (parts of) datasets or databases.

3.2 Application Profile

An application can be generalised to a set of modules exchanging data. The modules can be functions, objects, actors, processes, services etc. The *application profile* captures the characteristics of these modules and their relationships.

The application developer typically has limited knowledge about the temporal aspects of the application, e.g. how much data will be exchanged among the modules at what time, CPU and memory consumption of each module etc. These factors may depend on the data processed (see data profile) and the user behaviour (user profile). Such parameters must therefore be deduced from past executions and from the software structure and its deployment.

Due to the nature of these properties, they are mostly stochastic. In order to evaluate conditions on such parameters, one will have to resort to hypothesis testing, or observation of summarising quantities. For properties with high variability, such as CPU load, a high sampling frequency will be necessary to capture the variations and the large data volumes resulting might not be possible to store. Hence, one will have to resort to generalisations and fitting of standard parameterised density functions to capture the essential variability.

Finally, the application profile must encompass the operational goals, policies, and preferences of the application owner. These are not constraints by themselves, though they may be linked to constraints. Policies can, for example, put restrictions on where data is stored, or which Cloud providers can (not) be used for deployment – these could also indicate preferences of one Cloud provider over others.

The specification of constraints and goals is difficult – there is a clear need for a rich language to capture all aspects, that is tractable using knowledge engineering techniques but which in addition is able to capture the intent of the user.

3.3 Data Profile

The *data profile* characterises the data to be used by the application – this includes size (number and size of records), the kind of dataset (alphanumeric, file etc.), degree of dynamicity, privacy and security considerations. Even more important for Cloud performance are the access characteristics: do applications typically read the dataset sequentially or randomly based on some key attribute value? Is the data used concurrently? Does the data need to be kept consistent? Finally backup mechanisms need to be in place for system recovery.

3.4 Host Profile (Infrastructure Characteristics)

The technical characteristics of the infrastructure play a major role with two respects: (1) whether the host is suitable for the technical requirements and (2) what performance can be expected by using the respective host. The relationship between performance and executing platform is however generally undefinable, see e.g. (Skinner and Kramer,

2005). Similar to the technical requirements (see below), there are however indicators that help in performing an "educated guess" about the right matching, such as bandwidth between nodes, size of storage, database type etc.

Some of these factors can be influenced by the host, such as the number of virtual machines, other factors depend completely on the application and the data processed by it.

The developer should therefore be able to specify all the characteristics that are of essence to executing the application. Typically, this will be constrained to low-level aspects, such as the type of operating system, necessary libraries etc., as the average developer will not be able to specify all performance impacting factors. On the other hand, the developer is the only person able to assess what kind of communication density to expect etc.

Therefore the relevant information must be conveyed in a way that is intuitive and yet meaningful for deployment and configuration. This means that the developer will have to provide (a) the concrete technological deployment requirements (*deployment model*) and (b) any performance related information that compensate the *application profile* – this means in particular information regarding likelihood and category of occurrence. For example, the developer indicates how likely specific data sets occur, such as frequency of address changes versus person queries in a personnel database.

3.5 Expertise

As noted earlier (see Section 2.3), the translation of all the expectations towards the application is currently achieved through manual labour, i.e. by the expertise of the developers involved in the process. However, the scope of information to be considered may easily exceed any current developer's expertise.

The problem is thereby not so much the complexity of the individual knowledge "items", but the sheer scope of such expertise and the according fine-grained application context. For example

```
Scale out with numbers of users
```

is fairly easy to grasp and apply in general. However, correct application depends on a number of factors, starting with the networking capacity of the host, over its adaptation speed, up to the overall quality expectations. But there are also commercial aspects to be considered, such as maximum costs and cost to quality ratio.

What is more, however, is that the combination of rules for different services within an application can lead to undesired side effects, if not properly respected when applying these rules. As such, two connected components that communicate with each other may e.g. lead to mutual scale up beyond given boundaries.

As already noted in the beginning of this section, all rules must generally take an event-condition-action (ECA) like structure in order to be applicable under the right circumstances. As opposed to the other information types, which generally are of the form of conveying only parts of the ECA rules (such as the conditions), the "expertise rules" directly incorporate all information and should therefore be formalised as ECAs. In principle, "expertise rules" are the results of gathering all the historical information, generating the rules and improving them over time – with the difference, that such knowledge is already available for exploitation, though not properly formalised for automated usage and application across domains and use cases.

4 AN ARCHITECTURE FOR APPLICATION DEPLOYMENT

As noted, the rules and information to be applied to the actual execution of an application take an event-condition-action (ECA) like form – however:

(1) many of these rules have to be combined under *uncertain and partial unspecified* conditions, and

(2) the conditions will take the form of *mathematical equations* with domains of applicable values, rather than concrete Boolean formulas

4.1 Contextualising Rules

Normal ECA rules apply to strictly logical expressions that evaluate to true or false, similar to

```
if (10 < #users < 100) then
```

However, in the case here, the actual conditions (and also the events) are of a form that is closer to

$$\texttt{if (p(invocation}_{\texttt{processA}}\texttt{)>0.7) then}$$

where *p* means the probability for invocation. In other words, the rules apply when uncertain boundary conditions are met. This can be for example

Rule#1: scale out whenever the number of users (per instance) is bigger than 10

Rule#2: only scale out, when the service is not a bottleneck

Rule#3: service B has high communication needs when process A is invoked (bottleneck)

Rule#4: probability for invoking process A is 0.25

In other words, rule#1 applies successfully for service B in 75% of all cases.

To derive such rules and reason over them, the information described (section 3) must be formalised and provided to the executing environment.

4.1.1 Exploiting Specialisation

It will be noted that the rules in this form are per se infrastructure agnostic – meaning in particular, that they generalise over the potential cloud behaviour. At this level, the rules can be executed on any platform that supports the necessary actions.

Some providers may however not offer full support for all of this information, e.g. not all platform providers allow fully detailed monitoring information and not all allow full control over scaling behaviour. The implication of this is, that not all of the rules are equally applicable to all environments.

By exploiting the *host profiles*, however, it can be easily identified which provider supports which rules. Implicitly, the combination of selected rules constraints the selection of providers.

Vice versa, and more interestingly, the same principle holds true to exploit specialised capabilities: these form nothing but action rules that only apply to a dedicated platform or host. Thus, these rules restrict the choice of providers, if the constraints demand for the according rule set. Hence, the probability of fulfilment may alter by selecting an alternative host.

This principle allows full exploitation of specialisation according to the abstract requirements of the application (see above).

4.2 Conceptual Architecture

Self-adaptive software systems have been studied extensively ever since the vision of autonomic computing arose (Jeffrey O. Kephart and David M. Chess 2003). This vision basically specified that the management of any adaptive system would have to carry out four essential processes: (1) the system must be monitored, (2) the gathered information must be analysed to decide what the current system state is and to decide if an adaptation is necessary, after which the adaptation is (3) planned before it is (4) executed. This has become known as the MAPE-K loop of autonomic systems, where the K stands for the knowledge available and exploited executing this loop. The MAPE-K loop must also be enacted for autonomous cloud applications.

One could build the adaptation plan reactively taking only the current application context into consideration when defining the next model and then

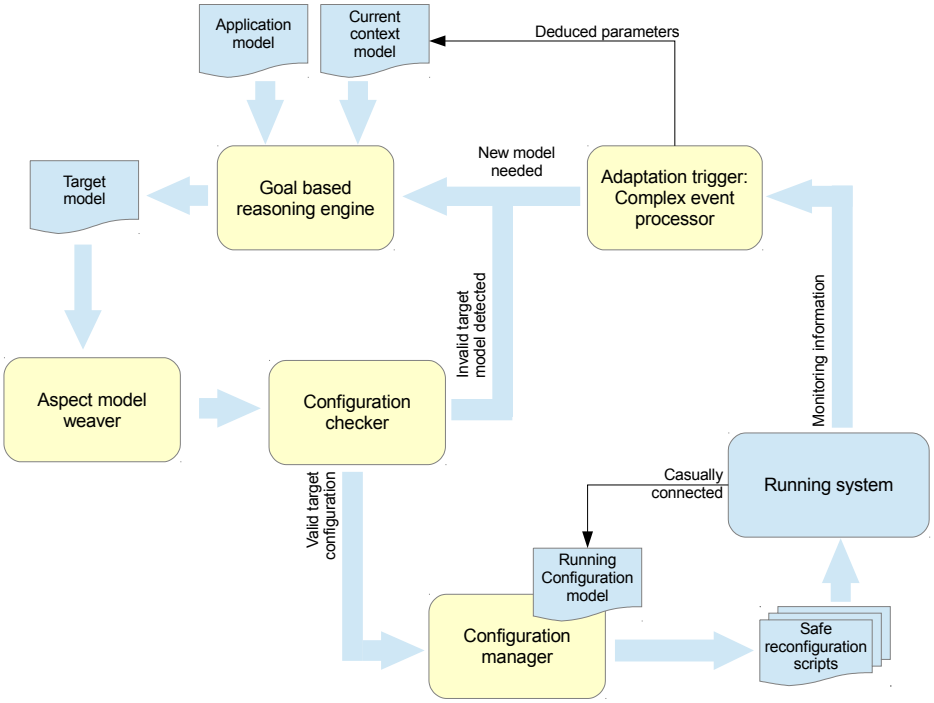


Figure 2: The fundamental modules for adapting a model-based application at runtime. Platform specific scripts ensure a safe transition from the current to the new application configuration. Taken from (Geir Horn 2013).

plan the adaptation as transformation from the running application model to the new one. This approach, known as models@run.time (Gordon Blair, Nelly Bencomo, and Robert B. France 2009) was used in (Brice Morin et al. 2009), which identified the following five modules necessary in order to carry out the MAPE-K loop:

(1) A *complex event processor* to identify when the running context of the application has changed to the extent that an adaptation is necessary.

(2) A *goal based reasoning engine* to select the model configuration that would be "best" for the current context, i.e. the target model.

(3) An *aspect model weaver* to realise the target model and its bindings.

(4) A *configuration checker* to verify that the target model is consistent and deployable.

(5) A *configuration manager* to derive a safe sequence of application reconfiguration commands.

The relations of these modules are depicted in Figure 2, which also advocates the need for a stochastic goal based reasoning engine.

4.2.1 Monitoring

Monitoring is done by the execution platform (inside the virtual machine or on system level). The monitor provides information about the current execution status, insofar as supported by the host (cf. section 4.1.1). Therefore the monitor provides relevant information whether the expected constraints are met.

Each monitoring instance fulfils a two-fold role (cf. next section): (1) checking the performance of the individual service (instance), and (2) contributing to the analysis of the overall behaviour.

A definition of a monitor incorporates (1) what type of information needs to be gathered (2) at what frequency and (3) potentially how it needs to be generated (metric), as well as (4) where it needs to be sent to (such as the event filter).

Obviously, the information is constrained by the hosting platform's capabilities - in other words, no metrics can be applied to data that is not provided by the host. This is hence a selection criteria for the right host (see section 4.1.1).

4.2.2 Analysis

The analysis of the monitored information can be done locally or centrally: for instance local monitoring can decide to scale out a local service, but application level adaptation must be analysed centrally to weigh the effect of all individual actions.

The simplest form of analysis is to compare the monitored data against a threshold. However, this will be vulnerable to variations. For instance, the response time of a database query depends on its complexity - hence, a complex query may exceed the response threshold. Adapting on a single event may therefore lead to erratic behaviour.

Statistical hypothesis testing is one alternative. However, one should bear in mind that most methods are developed for the linear model (Franklin A. Graybill, 1976) and therefore sensitive to deviations from the normal distribution of large samples. The alternative is to use non-parametric tests (Conover, 1999) or, if one is just interested in detecting out of bounds events, one could use the sample version of the Chebyshev inequality (John G. Saw, 1984).

Finally, there are numerous techniques from the field of information flow processing that can be applied to correlate and understand the true system state based on monitored events (Gianpaolo Cugola and Alessandro Margara, 2012).

4.2.3 Planning

Adaptation planning takes place at three levels:

(1) The developer plans adaptation steps that make sense for the application logic – for example related to the workload assignment in distributed applications.

(2) Plans for each Cloud provider adaptation using the elasticity mechanisms offered by that infrastructure. Such plans are provided as part of the deployment of the application, and they are therefore a part of the currently deployed configuration.

(3) When the system requires adaptation that is not catered for, a new configuration is needed. Such a global adaptation must be planned in response to the detected changes in the application's running context using models@run.time as outlined above, trying to meet all operational constraints (Geir Horn 2013).

4.2.4 Execution

The adaptation plan should be executed hierarchically with the central authority maintaining overall application consistency, whereas platform level mechanisms are used to the largest possible extent to allow a smooth transition.

The configuration manager will need to identify the difference between the running and the new configuration, to generate reconfiguration scripts for each Cloud platform. This is also highly use case dependent, as some applications allow checkpointing and state recovery, whereas other

applications must simply be stopped with the new configuration started from scratch.

The main point thereby is that not the services themselves are adapted (in the sense of re-programmed or re-configured with new parameters), but the execution context instead. Using above building blocks, the profiles and service descriptions, the tasks (i.e. services) can be treated individually as black boxes that are scaled out, relocated, reconfigured much the same way a virtual image would. In other words, the services contributing to the overarching application logic are basically subject to common cloud strategies that together meet the overarching goals and constraints.

To realise such behaviour, the configuration manager is closely linked to local execution engines that perform the actual adaptation on a service level, ensuring that the individual steps in the adaptation script are executed correctly.

5 CONCLUSIONS

Our proposed model aims at integration across multiple Cloud platforms of any kind. It will also allow the application to be deployed optimally taking account of the specialised characteristics of different platforms matched to the requirements of the application and its usage constraints.

Not all additional information necessary can always be provided, or properly matched: so far, no proper programming and modelling mechanism exists that allows easy and intuitive definition of the right type of information. Furthermore, reactive adaptation planning using models@run.time is an active research topic (Svein Hallsteinsen et al. 2012), and using these concepts for Cloud deployment is currently under investigation.

A major open challenge thereby remains in maintaining the compositional correctness of the decomposed rules and actions: during deployment and adaptation, the overarching constraints have to be broken down to low-level rules that can be enacted individually. To this end, the information does have to be provided in a fashion that incorporates both high- and low-level descriptions.

This paper described the approach pursued in the PaaSage project, which develops the necessary language, modelling and reasoning tools to allow provisioning and exploitation of the type of information described here. The tools will allow the individual stakeholders to provider their respective view on the goals and constraints by building up from proven patterns that can be decomposed through according reasoning mechanisms. The goal is to make it easier to create and host applications that can run effectively and efficiently on various Cloud, thereby addressing a major barrier to take-up of Clouds.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 317715. The views expressed in this paper are those of the authors and do not necessarily represent those of the consortium.

REFERENCES

Becker, S., Koziolek, H. & Reussner, R., 2009. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software, 82:3-22*.

Brice Morin, Olivier Barais, Jean-Marc Jézéquel, Franck Fleurey, and Arnor Solberg. 2009. 'Models@run.time to Support Dynamic Adaptation'. *Computer* 42 (10): 44–51. doi:10.1109/MC.2009.327.

Conover, W. J.., 1999. *Practical Nonparametric Statistics*. 3rd ed. Wiley Series in Probability and Statistics: Applied Probability and Statistics Section. John Wiley & Sons.

Franklin A. Graybill. 1976. *Theory and Application of the Linear Model*. North Scituate, MA, USA: Duxbury Press.

Geir Horn. 2013. 'A Vision for a Stochastic Reasoner for Autonomic Cloud Deployment'. In *Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies*, edited by Muhammad Ali Babar, Marlon Dumas, and Arnor Solberg, 46–53. NordiCloud '13. New York, NY, USA: ACM. doi:10.1145/2513534.2513543. http://doi.acm.org/10.1145/2513534.2513543.

Gianpaolo Cugola, and Alessandro Margara. 2012. 'Processing Flows of Information: From Data Stream to Complex Event Processing'. *ACM Comput. Surv.* 44 (3) (June): 15:1–15:62. doi:10.1145/2187671.2187677.

Gordon Blair, Nelly Bencomo, and Robert B. France. 2009. 'Models@run.time'. *Computer* 42 (10): 22–27. doi:10.1109/MC.2009.326.

Hallsteinsen, S., Geihs, K., Paspallis, N., Eliassen, F., Horn, G., Lorenzo, J., Mamelli, A., and Papadopoulos, G. A., 2012. A Development Framework and Methodology for Self-adapting Applications in Ubiquitous Computing Environments. *Journal of Systems and Software* 85 (12) (December): 2840–

2859. doi:10.1016/j.jss.2012.07.052.

Helmer, S., Poulovassilis, A. & Xhafa, F., 2011. Reasoning in Event-Based Distributed Systems. *Studies in Computational Intelligence 347*, Springer-Verlag Berlin Heidelburg.

Jeffrey O. Kephart, and David M. Chess. 2003. 'The Vision of Autonomic Computing'. *Computer* 36 (1): 41–50. doi:10.1109/MC.2003.1160055.

John G. Saw, Mark C. K. Yang, and Tse Chin Mo. 1984. 'Chebyshev Inequality with Estimated Mean and Variance'. *The American Statistician* 38 (2) (May): 130. doi:10.2307/2683249.

Rubio Bonilla, D., Schubert, L. and Wesner, S., 2012, The Need to Comprehend Clouds: Why We Still Can't Use Clouds Properly. *CloudComp Conference Proceedings*.

Schubert, L., & Jeffery, K., 2012. Advances in Clouds - Research in Future Cloud Computing. Cordis (Online). Brussels, BE: European Commission. Retrieved from http://cordis.europa.eu/fp7/ict/ssai/docs/future-cc-2may-finalreport-experts.pdf.

Skinner, D. & Kramer, W., 2005. Understanding the causes of performance variability in HPC workloads. *International Symposium on Workload Characterization.*

QoS- and Security-aware Composition of Cloud Collaborations

Olga Wenge, Ulrich Lampe and Ralf Steinmetz
Multimedia Communications Lab (KOM), TU Darmstadt, Rundeturmstr. 10, 64283 Darmstadt, Germany
{firstname.lastname}@kom.tu-darmstadt.de

Keywords: Cloud Computing, Collaboration, Quality of Service, Security.

Abstract: While cloud computing promises virtually unlimited resource supplies, smaller providers may not be able to offer sufficient physical IT capacity to serve large customers. A solution is cloud collaborations, in which multiple providers unite forces in order to conjointly offer capacities in the market. Unfortunately, both the QoS and security properties of such collaborations will be determined by the "weakest link in the chain", hence resulting in a trade-off between the cumulative capacity and the non-functional characteristics of a cloud collaboration. In this position paper, we examine how cloud collaborations can be optimally composed in a QoS- and security-aware fashion within a market scenario involving multiple cloud providers and users. We propose a Mixed Integer Programming-based exact optimization approach named CCCP-EXA.KOM. Based on a quantitative evaluation, we find that the practical applicability of CCCP-EXA.KOM is limited to small-scale problem instances and conclude that the development of tailored heuristic approaches is required.

1 INTRODUCTION

Cloud computing promises to supply virtually unlimited IT capacities in a scalable, pay-as-you-go fashion (Buyya et al., 2009). Yet, specifically smaller providers may not be able to satisfy the resource demands of large customers on their own due to insufficient data center capacity. A solution lies in cloud collaborations, i. e., the cooperation of multiple providers to conjointly satisfy user demands. Unfortunately, such cloud collaborations have both Quality of Service (QoS) and security implications: since a user may potentially be served by any provider within a collaboration, the non-functional service attributes – e. g., availability, latency, or data center location – will be determined by the "weakest link in the chain", i. e., the provider with the lowest guarantees.

Take the example of two providers, one of which uses encryption for data storage and one which does not. Once these providers join forces within a collaboration and act as one common provider, data may be stored at either one of them. Hence, the collaboration *cannot* be assumed to offer data encryption, even though the data may in fact physically reside with the first provider, i. e., the one which applies encryption. A similar problem occurs if two providers reside in different jurisdictions, such as the European Union (EU) and the United States, where data privacy laws substantially differ (Wenge et al., 2012).

Yet, given the wide range of legal and regulatory requirements that apply in many industries, a cloud user faces certain requirements in choosing his/her cloud provider, and these requirements may not be fulfilled once two or more providers join forces within a collaboration. Based on this scenario, we examine the *Cloud Collaboration Composition Problem* (CCCP) in the work at hand. Our focus is on a broker within the cloud market, who aims to maximize his/her profit through the composition of cloud collaborations from a set of providers and assignment of users to these collaborations. In that process, QoS and security requirements should also be satisfied.

This work introduces the CCCP as a new research problem in the context of cloud computing. The paper also presents a formal optimization model, which permits the computation of exact, i. e., profit-optimal, solutions for specific problem instances.

The remainder of this paper is structured as follows: In Section 2, we describe the problem in detail and introduce formal notations. Based on this, the subsequent Section 3 introduces an exact optimization approach, called CCCP-EXA.KOM, which is quantitatively evaluated in Section 4. Section 5 gives an overview of related work, and Section 6 concludes the paper with a summary and outlook.

2 FORMAL PROBLEM STATEMENT

In our work, we take the perspective of a cloud broker, who is acting within a cloud market. This cloud market consists of a set of cloud providers and a set of users, formally denoted as $P = \{1, 2, \ldots, P^{\#}\}$ and $U = \{1, 2, \ldots, U^{\#}\}$, respectively.

Each user $u \in U$ exhibits a certain resource demand of $RD_u \in \mathbb{R}^+$ units, for which he/she is willing to pay a total of $M_u^+ \in \mathbb{R}^+$ monetary units. Furthermore, each cloud provider $p \in P$ is able to provide a resource supply of $RS_p \in \mathbb{R}^+$ units at a total cost of $M_p^- \in \mathbb{R}^+$. Please note that resource demands and supplies could also be expressed in a multi-dimensional fashion, i. e., with respect to different resource types. However, for the sake of simplicity, we assume one-dimensional resource constraints at this point, a notion that is also followed in related publications (Hans et al., 2013).

Consumption and provision of services is subject to certain QoS and security constraints, which we refer to by the common term of *non-functional* constraints. Specifically, we assume two sets, $A = \{1, 2, \ldots, A^{\#}\}$ and $\hat{A} = \{1, 2, \ldots, \hat{A}^{\#}\}$, of quantitative and qualitative non-functional attributes. Quantitative attributes represent numerical properties, e. g., availability or latency. In contrast, qualitative attributes correspond to nominal properties, e. g., data center location in the European Union or adherence to a certain security policy.

The cloud providers make certain guarantees with respect to the non-functional attributes. For each quantitative attribute $a \in A$, the value guaranteed by provider $p \in P$ is denoted as $AG_{a,p} \in \mathbb{R}$. For each qualitative attribute $\hat{a} \in \hat{A}$, the corresponding information is given by $\hat{AG}_{\hat{a},p} \in \{0, 1\}$.

Inversely, the cloud users specify certain requirements concerning the non-functional attributes. With respect to each quantitative attribute $a \in A$, the value required by user $u \in U$ is denoted as $AR_{a,u} \in \mathbb{R}$. Likewise, $\hat{AR}_{\hat{a},u} \in \{0, 1\}$ denotes the requirement for each qualitative attribute $\hat{a} \in \hat{A}$, i. e., indicates whether this attribute is mandatory or not. Without loss of generality, we assume that the users specify lower bounds (e. g., minimum availability) on their quantitative non-functional requirements. Upper bounds (e. g., maximum latency) can be easily incorporated into the model by negation of the respective values.

As it has been briefly explained in the previous section, the challenge for the cloud broker consists in composing cloud collaborations, consisting of multiple cloud providers, and subsequently assigning users to them. In that process, the objective for the broker is to maximize his/her profit, i. e., the difference between the revenue from the served cloud users and the spending on the incorporated cloud providers. As most important constraint, each collaboration should obviously offer sufficient resource supplies to serve the assigned users. The process is further subject to the constraint that the QoS and security requirements of each user are fulfilled by the cloud collaboration to which he/she has been assigned.

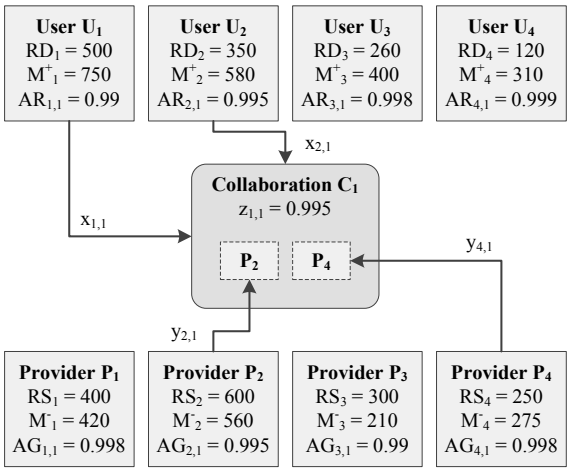


Figure 1: Tangible example of a CCCP instance with four users and providers.

A tangible, simplified example for a CCCP instance is provided in Figure 1. The instance exhibits four users and providers with different resource demands/supplies and non-functional requirements/guarantees, respectively. In the example, providers P_2 and P_4 form a collaboration, which enables them to conjointly serve users U_1 and U_2 under the given constraints. Both providers substantially profit from the collaboration, since their combined resource supply permits to serve larger customers and allows to achieve a higher degree of resource utilization.

3 OPTIMIZATION APPROACH CCCP-EXA.KOM

Based on the notations that were introduced in the previous section, the CCCP can be transformed into an optimization model. The result is given in Model 1 and will be explained in the following.

To start with, $x_{u,c}$ and $y_{p,c}$ are the *main* decision variables in the model (cf. Equation 11). They are defined as binary and indicate whether user u or provider p, respectively, has been assigned to collaboration c or not. As additional axillary decision vari-

ables, we introduce $y'_{p,c}$, which are also binary and serve as complement to $y_{p,c}$, hence indicating the non-assignment of a provider p to a collaboration c. Furthermore, $z_{a,c}$ and $\hat{z}_{\hat{a},c}$ are specified (cf. Equation 12). They are defined as real and binary, respectively, and represent the cumulative value of the non-functional property a or $\hat{a}$, respectively, for collaboration c. The variables x and y are referred to as main decision variables, since they have a direct impact on the objective function. In contrast, y', z, and $\hat{z}$ only have an indirect influence.

As outlined before, the objective consists in profit maximization (cf. Equation 1). That is, the difference between the revenue from the served cloud users and the spending on the used cloud providers should be maximized, depending on the values of the decision variables. Please note that other objectives, such as maximizing the number of served users or the overall resource utilization, may easily be incorporated into the optimization model as well. However, given the general idea of a competitive cloud market, our initial focus is placed on monetary objectives.

Equations 2 and 3 make sure that each user and provider is assigned to not more than one collaboration. Thus, the broker may opt to *not* satisfy certain users' demands, but also to not exploit cloud providers as part of a collaboration. Equation 4 determines the inverse variable $y'_{p,c}$ for each decision variable $y_{p,c}$. This definition is used in the following two Equations 6 and 7. They determine the cumulative non-functional values for quantitative and qualitative attributes, respectively. Both equations are formulated such that quantitative properties are given by the "worst" value among all providers in a certain collaboration, i. e., the "weakest link in the chain". Equations 8 and 9 make sure that users can only be assigned to such collaborations that make sufficient non-functional guarantees, given the users' specific non-functional requirements. Lastly, Equation 10 defines a set of potential cloud collaborations. The underlying notion for the given definition is that no user or provider will be assigned to more than one collaboration (recall Equations 2 and 3). Hence, the maximum number of collaborations is given by the number of users or providers, whichever is lower.

Based on the given model, it can easily be seen that the CCCP constitutes a Mixed Integer Program (MIP), i. e., a special form of Linear Program (LP) that features both integer (in this case, binary) and natural decision variables. Thus, the problem can be solved using off-the-shelf optimization algorithms, such as branch-and-bound (Hillier and Lieberman, 2005), in order to obtain an exact (i. e., profit maximal) solution.

While branch-and-bound is known to perform very well on many Integer Program (IP) and MIP problems, it is ultimately still based on the principle of enumeration (Hillier and Lieberman, 2005). Thus, in the worst case, the computational complexity of obtaining an exact solution grows with the size of the solution space. In the specific case of the CCCP, this translates into an exponential growth with the problem size, i. e., the number of considered providers and users.

4 EVALUATION

To assess the practical applicability of our proposed approach CCCP-EXA.KOM, we have prototypically implemented it in Java 7. In order to transfer Model 1 into a programmatic representation, we use the free JavaILP framework[1]. While this potentially permits for the application of different backend solver framework, we have selected the commercial IBM ILOG CPLEX framework[2] as default due to its favorable performance (Meindl and Templ, 2012) and its popularity in related research, e. g., (Hans et al., 2013; Mashayekhy and Grosu, 2012).

4.1 Evaluation Setup and Procedure

The main objective of our evaluation is to assess the required computation time of CCCP-EXA.KOM for different problem sizes. This allows us to judge the applicability of the proposed approach under practical conditions, where time constraints in the decision process play an important role. Thus, formally, we regard *computation time* as the *dependent* variable of our evaluation.

As *independent* variables, we include the number of considered users and providers, i. e., $U^{\#}$ and $P^{\#}$. In contrast, the number of quantitative and qualitative non-functional attributes were fixed ($A^{\#} = 1$ and $\hat{A}^{\#} = 1$); hence, they constitute *controlled* variables. This is justified by two aspects: First, these variables are likely also predefined in practice. Second, they do not have an impact on the number of decision variables and hence, the size of the solution space. Each specific combination of $U^{\#}$ and $P^{\#}$ results in a *test case*. For each test case, we created 100 specific CCCP instances with the according dimensions.

The parameter values or distributions that were used in the problem generation process are summa-

[1] http://javailp.sourceforge.net/.

[2] http://www.ibm.com/software/integration/optimization/cplex-optimizer/.

Model 1: Cloud Collaboration Composition Problem

$$\text{Max. } Pr(x,y,y',z,\hat{z}) = \sum_{u\in U, c\in C} x_{u,c} \times M_u^+ - \sum_{p\in P, c\in C} y_{p,c} \times M_p^- \quad (1)$$

such that

$$\sum_{c\in C} x_{u,c} \leq 1 \quad \forall u \in U \quad (2)$$

$$\sum_{c\in C} y_{p,c} \leq 1 \quad \forall p \in P \quad (3)$$

$$y_{p,c} + y'_{p,c} = 1 \quad \forall p \in P, \forall c \in C \quad (4)$$

$$\sum_{u\in U} x_{u,c} \times RD_u \leq \sum_{p\in P} y_{p,c} \times RS_p \quad \forall c \in C \quad (5)$$

$$z_{a,c} \leq y_{p,c} \times AG_{p,a} + y'_{p,c} \times \max_{p\in P}(AG_{p,a}) \quad \forall p \in P, \forall c \in C, \forall a \in A \quad (6)$$

$$\hat{z}_{\hat{a},c} \leq y_{p,c} \times \hat{AG}_{p,\hat{a}} + y'_{p,c} \quad \forall p \in P, \forall c \in C, \forall \hat{a} \in \hat{A} \quad (7)$$

$$z_{a,c} \geq x_{u,c} \times AR_{u,a} \quad \forall u \in U, \forall c \in C, \forall a \in A \quad (8)$$

$$\hat{z}_{\hat{a},c} \geq x_{u,c} \times \hat{AR}_{u,\hat{a}} \quad \forall u \in U, \forall c \in C, \forall \hat{a} \in \hat{A} \quad (9)$$

$$C = \{1, 2, \ldots, \min(P^{\#}, U^{\#})\} \quad (10)$$

$$x_{u,c} \in \{0,1\} \quad \forall u \in U, \forall c \in C \quad (11)$$
$$y_{p,c} \in \{0,1\} \quad \forall p \in P, \forall c \in C$$

$$y'_{p,c} \in \{0,1\} \quad \forall p \in P, \forall c \in C \quad (12)$$
$$z_{a,c} \in \mathbb{R} \quad \forall a \in A, \forall c \in C$$
$$\hat{z}_{\hat{a},c} \in \{0,1\} \quad \forall \hat{a} \in \hat{A}, \forall c \in C$$

rized in Table 1. The specifications of the non-functional parameters are based on the notion that the sole quantitative and qualitative attribute represent availability (a QoS aspect) and data center location in the European Union (a security aspect), respectively. Furthermore, monetary parameters were set such that higher availability results in quickly increasing values, based on the observation that each additional "nine" in the availability figure results in doubled cost (Durkee, 2010). In contrast, a EU data center location only leads to a moderate increase of 10%, which closely corresponds to the price difference observed for Eastern U.S. and Ireland-located Amazon EC2 VM instances (Amazon Web Services, Inc., 2013).

Table 1: Parameter values and distributions used in the problem instance generation. Abbreviations: Uni – Uniform distribution; Ber – Bernoulli distribution.

Param.	Value/Distribution
$AR_{1,u}$	Uni$(0.99, 0.9995)$
$\hat{AR}_{1,u}$	Ber(0.5)
$AG_{1,p}$	Uni$(0.995, 0.9995)$
$\hat{AG}_{1,p}$	Ber(0.5)
RD_u	Uni$(1000, 5000)$
RS_p	Uni$(1000, 5000)$
M_u^+	$\alpha_u \times RD_u \times log_{10}(1 - AR_{1,u})^2 \times (1.1^{\hat{AR}_{1,u}})$
M_p^-	$\beta_p \times RS_p \times log_{10}(1 - AG_{1,p})^2 \times (1.1^{\hat{AG}_{1,p}})$
α_u	Uni$(1.5, 1.75)$
β_p	Uni$(1.0, 1.25)$

Following the generation, we computed a solution to each problem instance using our prototypical implementation of CCCP-EXA.KOM. In that process, we imposed a timeout of 300 seconds (i. e., five minutes) per problem instance. Based on the resulting sample of computation times for the successfully solved problems, we computed the mean computation time, as well as the 95% confidence interval. The evaluation was conducted on a desktop computer, equipped with an Intel Core 2 Duo E7500 processor and 4 GB of memory, operating under the 64-bit edition of Microsoft Windows 7.

4.2 Evaluation Results and Discussion

The results of our evaluation, i. e., the observed mean computation times per test case, are graphically illustrated in Figure 2. As can be clearly seen, the computation times quickly increase with the problem size, i. e., the considered number of users and providers. The effect is less pronounced for the smallest two problem classes (with $U^{\#} \leq 6$ and $P^{\#} \leq 9$); in fact, for these two test cases, there is no statistically significant difference in mean computation time observable at the 95% confidence level. In absolute terms, we already find absolute computation times in the order of magnitude of one-hundred seconds and one second respectively for the medium-sized test cases with $U^{\#} \leq 8$. For these test cases, increasing the number of

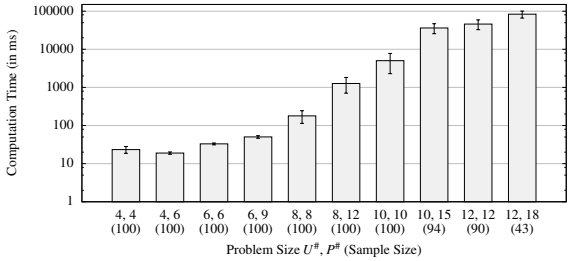


Figure 2: Evaluation results, i. e., observed mean computation times (with 95% confidence intervals) for CCCP-EXA.KOM by test case. Please note the logarithmic scaling of the ordinate.

providers increases the computation time by a factor of approximately ten already.

For the four largest test cases (with $U^{\#} \geq 10$ and $P^{\#} \geq 10$), the absolute computation times reach the order of magnitude of seconds and ten seconds. All observed increases are statistically significant at the 95% confidence level. In addition, the ratio of solved problem instances sharply drops with growing problem size. This effect is most notable for the largest problem class that involves 12 users and 18 providers, where only 43% of the 100 problem instances could be solved within the timeout period of five minutes.

Given that the considered problem dimensions are still relatively small in the context of a large cloud market, it can be concluded that the practical applicability of the proposed optimization approach CCCP-EXA.KOM is rather limited. As it has already been explained before, a broker will likely have to decide on the composition of collaborations under rigid time constraints, since users likely require resources at short notice.

Hence, an important future challenge consists in the development of appropriate heuristics, which permit to trade reductions in computation time against small degradations in broker profit, and are consequently applicable to practically relevant, large-scale problem instances. In that context – apart from its potential application to small-scale problem instances – CCCP-EXA.KOM can serve as a valuable performance benchmark.

5 RELATED WORK

Niyato et al. (Niyato et al., 2011) study the cooperative behavior of multiple cloud providers in order to cooperate and support the establishment of resource pools to offer services to public cloud users. The authors present a stochastic LP game model which takes the random internal demand of cloud providers and a transferable utility into account to define and commit the optimal offer of cooperated cloud providers. In contrast to our work, Niyato et al. do not consider non-functional constraints, i. e., QoS and security requirements.

In a more recent work, Niyato et al. (Niyato et al., 2012) examine building coalitions between cloud providers as a novel approach to optimize the capacity expansion and maximize the mobile cloud providers' monetary benefits. The authors consider cooperative game theory and the Nash equilibrium principles in their approach and propose admission control and revenue sharing strategies for building cloud provider coalitions and a resource pool for mobile applications. The provided results illustrate improvements in cloud providers' capacity and profit maximization by entering such cloud coalitions. Similar to their previous work, the authors do not consider non-functional constraints, which are an important aspect of our work.

Gohad et al. (Gohad et al., 2013) propose a dynamic algorithm for forming self-adaptive cloud collaborations based on the identifying most appropriate healthy set of cloud provider resources (cloud provider capabilities and functional abilities at the SaaS layer), cost modeling and tenancy requirements. The approach is highlighted with a realistic example. In contrast to us, Gohad et al. focus on ad-hoc resource provisioning, rather than the long-term formation of cloud collaborations, and do not consider security aspects. This specifically includes the cumulative security properties of cloud collaborations that were a focal point of our work.

Song et al. (Song et al., 2010) examine the problem of task selection and allocation to physical machines in the context of dynamic cloud collaborations. Their objective consists in the balancing of resource demands under consideration of different resource types, such as CPU and memory. For that purpose, the authors propose three heuristic optimization approaches, and demonstrate that a cooperative heuristic has benefits with respect to the objective of balanced resource utilization. In contrast to us, Song et al. focus on individual cloud providers and do not regard security requirements.

Mashayekhy and Grosu (Mashayekhy and Grosu, 2012) model a cloud federation formation problem based on the game theory and formulate a corresponding IP-based optimization approach. In their model, the authors consider the cooperative provisioning of VM instances and storage by federated cloud providers. Their objective consists of profit maximization combined with the formation of stable coalitions, i. e., coalitions in which cloud providers do not have a monetary incentive to switch to differ-

ent coalitions. In contrast to our work, the authors only consider resource constraints, but do not regard non-functional requirements. Their work also aims at low-level VM provisioning, rather than strategic composition of collaboration.

Lastly, Hans et al. (Hans et al., 2013) have examined the cost-efficient selection of cloud data centers for the delivery of multimedia services. In that context, the authors propose an exact optimization approach based on IP. While their work is similar with respect to the consideration of resource and QoS constraints, it focuses on a single cloud provider and does neither regard the composition of collaborations nor qualitative non-functional aspects.

In conclusion, to the best of our knowledge, we are the first to examine the profit-maximal, strategic composition of cloud collaborations under consideration of cumulative non-functional properties that result from the very formation of these collaborations, i. e., are determined by the "weakest link in the chain". Apart from the identification of that specific problem, our main contribution consists in the proposal of an exact optimization approach, which can serve as benchmark for future heuristic approaches.

6 SUMMARY AND OUTLOOK

While cloud computing promises access to virtually unlimited IT resources, the physical infrastructure of cloud providers is actually limited. Hence, smaller providers may not be able to serve the demands of larger customers. A possible solution is cloud collaborations, where multiple providers join forces to conjointly serve customers. Unfortunately, in such scenario, non-functional QoS and security properties are determined by the "weakest link in the chain", rendering the process of composing collaborations cumbersome.

In this work, we introduced the corresponding *Cloud Collaboration Composition Problem*. We proposed an initial solution approach named CCCP-EXA.KOM based on Mixed Integer Programming, and evaluated it with respect to its computation time requirements. Our results indicate that exact optimization approaches are only applicable to small-scale problem instances, thus indicating the need for the development of custom-tailored heuristic approaches.

Accordingly, the development of such heuristics will constitute the primary focus of our future work. In addition, we plan to extend the proposed model to cater for more complex non-functional constraints, such as conditional requirements (e. g., strong data encryption is only required if data is placed outside the European Union).

ACKNOWLEDGEMENTS

This work has partly been sponsored by the E-Finance Lab e. V., Frankfurt a.M., Germany (www.efinancelab.de.).

REFERENCES

Amazon Web Services, Inc. (2013). Amazon EC2 Pricing, Pay as you go for Cloud Computing Service. http://aws.amazon.com/en/ec/pricing/.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. (2009). Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility. *Future Generation Computer Systems*, 25(6):599–616.

Durkee, D. (2010). Why Cloud Computing Will Never Be Free. *Queue*, 8(4):20–29.

Gohad, A., Ponnalagu, K., Narendra, N. C., and Rao, P. S. (2013). Towards Self-Adaptive Cloud Collaborations. In *2013 Int. Conf. on Cloud Engineering*.

Hans, R., Lampe, U., and Steinmetz, R. (2013). QoS-Aware, Cost-Efficient Selection of Cloud Data Centers. In *6th Int. Conf. on Cloud Computing*.

Hillier, F. and Lieberman, G. (2005). *Introduction to Operations Research*. McGraw-Hill, 8th edition.

Mashayekhy, L. and Grosu, D. (2012). A Coalitional Game-Based Mechanism for Forming Cloud Federations. In *5th Int. Conf. on Utility and Cloud Computing*.

Meindl, B. and Templ, M. (2012). Analysis of Commercial and Free and Open Source Solvers for Linear Optimization Problems. Technical report, Technische Universität Wien.

Niyato, D., Vasilakos, A. V., and Kun, Z. (2011). Resource and Revenue Sharing with Coalition Formation of Cloud Providers: Game Theoretic Approach. In *11th Int. Symp. on Cluster, Cloud and Grid Computing*.

Niyato, D., Wang, P., Hossain, E., Saad, W., and Han, Z. (2012). Game Theoretic Modeling of Cooperation Among Service Providers in Mobile Cloud Computing Environments. In *2012 Wireless Communications and Networking Conf.*

Song, B., Hassan, M. M., and Huh, E.-N. (2010). A Novel Heuristic-Based Task Selection and Allocation Framework in Dynamic Collaborative Cloud Service Platform. In *2nd Int. Conf. on Cloud Computing Technology and Science*.

Wenge, O., Siebenhaar, M., Lampe, U., Schuller, D., and Steinmetz, R. (2012). Much Ado About Security Appeal: Cloud Provider Collaborations and Their Risks. In *1st Europ. Conf. on Service-Oriented and Cloud Computing*.

A Requirements Analysis for IaaS Cloud Federation

Alfonso Panarello, Antonio Celesti, Maria Fazio, Massimo Villari and Antonio Puliafito
DICIEAMA, Università degli Studi di Messina, Contrada Di Dio, S. Agata 98166, Messina, Italia
alfonso.panarello@gmail.com, (acelesti, mfazio, mvillari, apuliafito)@unime.it

Keywords: Cloud Computing, Federation, Requirements Analysis, XMPP, SAML, XACML.

Abstract: The advent of Cloud computing offers different ways both to sell and buy resources and services according to a pay-per-use model. Thanks to virtualization technology, different Cloud providers supplying cost-effective services provided in form of Infrastructure as a Service (IaaS) have been rising. Currently, there is another perspective which represents a further business opportunity for small/medium providers known as Cloud Federation. In fact, the Cloud ecosystem includes hundreds of independent and heterogeneous cloud providers, and a possible future alternative scenario is represented by the promotion of cooperation among them, thus enabling the sharing of computational and storage resources. In this paper, we specifically discuss an analysis of the requirements for the establishment of an IaaS Cloud Federation.

1 INTRODUCTION

Cloud computing is a new paradigm able to provide on-demand servicesin a transparent way to usersaccording to given pay-per-use constrains.These services are arranged by providers using distributed and virtualized computing resources. This approach avoids to small and medium enterprises to make large investments of capital for purchasing their hardware/software equipments. The flexible and dynamic use of services takes place by means of thevirtualization technology that allows to decouple applications from the physical machine on which they run by means of Virtual Machines (VMs). Furthermore, VM migration gives the opportunity to guarantee a particular degree of Quality of Service (QoS).Thus, aggregating and mapping VMs, a Cloud provider is able to supply different levels of service: Infrastructure as a Service (IaaS), Platform as a service (PaaS) and Software as a Service (SaaS). Considering the cloud computing ecosystem, besides large cloud providers, smaller ones are also becoming popular even though their own virtualization infrastructures (i.e., deployed in their datacenters) cannot directly compete with the bigger market leaders. The result is that often small/medium cloud providers have to exploit the services of mega-providers in order to develop services. Thus, a possible future alternative scenario is represented by the promotion of a cooperation among small/medium providers, enabling the sharing of computational and storage resources. IaaS Cloud Federation is a partnership between providers for the borrowing/lending of virtual equipments including VMs, virtual clusters, virtual networks, and so on.However, guidelines regarding the design and implementation of the functionalities enabling IaaS Cloud Federation are not precisely defined.

In this paper, we discuss a requirement analysis for the establishment of an IaaS Cloud Federation. In particular, considering the "three-phase model" (A. Celesti, et al, 2010a), we identify the involved actors, also discussing automatism, scalability, versatility, and security features.Furthermore, we will analyze the major standards and protocols useful for the establishment of an IaaS Cloud Federation.

The paper is organized as follows. In Section II, we summarize several initiatives regarding Cloud Federation. An overview on the major IaaS pieces of middleware is discussed in Section III. In Section IV, we provide a detailed analysis of the requirements regarding the IaaS Cloud Federation. In Section V, we discuss a case of study considering the three-phase model and several emerging technologies. Section VI concludes the paper.

2 RELATED WORKS

Several architecture and models for Cloud Federation have been proposed recently with

different motivations. A three-phase cross-cloud model was presented in (Celesti et al., 2010b) In this work, the federation process takes place according to three subsequent phases, i.e., discovery, match-making, and authentication. The Reservoir FP7 European project (Rochwerger et al., 2010) introduced a modular and extensible Cloud supporting the management of business services and cloud federation. Claudia (Rodero-Merino et al,. 2010) provides an abstraction layer that allows the execution of services on top of a transparent federation of Cloud providers. Open Cirrus (Avetisyan et al., 2010) is a federation initiative between universities and research centers promoting the research in design, provisioning,and management of services in scale of multi data centers. Sky Computing (Keahey et al., 2009) introduces a virtual site layer on dynamically provisioned distributed resources provided by several data centers and a closed federation model, where the sharing of resources is based on cooperation basis like in a grid. OPTIMIS (Ferrera et al., 2012) is a platform for Cloud service provisioning that manages the whole lifecycle of the service and that also addresses issues such as risk, trust management, energy efficiency, and legislation. However, OPTIMIS does not address negotiation and a marketplace for discovery of resources. Other work which focus on low-level aspects of federation, such as security, networking, and disk image management is (Bernstein et al., 2009). Regarding scalable applications across multiple independent Cloud data centers, a market-based trading mechanism is required. The *InterCloud project* (Buyya et al., 2010) focuses on Cloud data centers and brokers that dynamically negotiate resources. Key component of *InterCloud* is the *Cloud coordinator*, whose architecture is described in (Calheiros et al., 2012). Other recent works that focus on other aspects of CoudFederation brokering are discussed in (Garge et al., 2011). A Cloud federation architecture based on satellite communications is discussed in (Celesti et al., 2012c). In (Celesti et al., 2012), the authors discuss how the data web can support Cloud federation. Furthermore, a mathematical programming technique to minimize the cost of leased VMs in investigated in (Chaisiri et al., 2009). In (Celesti et al., 2013), an architecture to build Platform as a Services (PaaS) exploiting different IaaS Cloud providers is proposed. These works focus on howto minimize the cost of external provisioning in hybrid Cloud providers. In (Breitgand et al., 2011), it is proposed a integer programming formulations for placement of VM workloads within as well as across multiple Cloud providers collaborating in a federation.

3 A SURVEY ON IaaS MIDDLEWARE

In this Section, we providean overview of the major existing open source IaaS Cloud pieces of middleware, evaluating their main features.

Nimbus (Nimbus, 2013) is an open source toolkit that allows turning a set of computing resources into an IaaS cloud. Nimbus comes with a component called workspace control, installed on each node, used to start, stop,and suspend VMs. It implements disk-image reconstruction and management, and securely connects the VMs to the network, and delivers contextualization. Nimbus's workspace control tools work with Xen and KVM but only the Xen version is distributed. Nimbus provides interfaces to VM management functions based on the WSRF set of protocols.

Eucalyptus (Eucalyptus, 2013) is an open-source framework that uses the computational and storage infrastructures commonly available at academic research groups to provide a platform that is modular and open to experimental instrumentation and study. Eucalyptus addresses several crucial Cloud computing questions, including VM instance scheduling, administrative interfaces, construction of virtual networks, definition and execution of service level agreements (cloud-to-user and cloud-to-cloud), and cloud computing user interfaces.

OpenQRM (OpenQRM, 2013) is an open-source platform for enabling flexible management of computing infrastructures. Thanks to its pluggable architecture, OpenQRM is able to implement a cloud with several features that allows the automatic deployment of services. It supports different virtualization technologies managing Xen, KVM and Linux-VServer. It also supports P2V (physical to virtual), V2P (virtual to physical) and V2V (virtual to virtual) migration. This means Virtual Environments (VEs) (appliances in the OpenQRM terminology) cannot only easily move from physical to virtual (and back), but that they can also be migrated from different virtualization technologies.

OpenNebula (OpenNebula, 2013) is an open and flexible tool that fits into existing data center environments to build a Cloud computing environment. OpenNebula can be primarily used as a virtualization tool to manage virtual infrastructures

in the data center or cluster, which is usually referred as private Cloud. OpenNebula also supports public Clouds by providing Cloud interfaces to expose its functionalities for VM, storage, and network management.

OpenStack (OpenStack, 2013) is IaaS cloud computing project that is a free open source software released under the terms of the Apache License. The project is managed by the OpenStack Foundation, a non-profit corporate organization. The technology consists of a series of interrelated projects that controls large pools of processing, storage, and networking resources throughout a datacenter, all managed through a dashboard.

CLEVER (CLoud-Enabled Virtual EnviRonment) (Celesti et al., 2012) is modular and pluggable middleware that specifically aims at the administration of private Cloud infrastructures. CLEVER is able to manage cluster of nodes each containing a host level management module (Host Manager). A single node may also include a cluster level management module (Cluster Manager). All these entities interact exchanging information by means of the Communication System based on the *Extensible Messaging and Presence Protocol (XMPP)*. The set of data required to enable the middleware functionalities is stored within a specific database deployed in a distributed fashion. CLEVER offers security and fault-tolerance.

4 FEDERATION REQUIREMENTS ANALYSIS

Despite of the obvious advantages that Cloud Federation offers, its implementation is not at all trivial. The main reason is that Cloud providers are more complicated than traditional systems and the existing federation models are not applicable. In fact, while Cloud providers are typically heterogeneous and dynamic, the existing federation models are designed for static environments where it is needed an a priori agreement among the parties. With regard to IaaS Cloud providers, we think federation needs to meet the following requirements:

a) *Identification of the Actors in a Federated System*: it is necessary to define which are the entities that cooperate each other in a federated environment. In particular, it is necessary to exactly know who plays the role of the cloud system provider and what is the role of Cloud system consumer.

b) *Automatism and Scalability*: In a federatedenvironment, a Cloud provider that accomplishautomated decisions has to determine which external provider has to be used for a particular workload, because not all Clouds are equal in terms of warranty, cost, reliability, QoS and resource availability. For example, a particular Cloud provider may be cheaper, but it could not provide guarantees of availability, making it unsuitable for mission-critical workloads. Another Cloud provider, however, could provide "five nines" availabilitybut be more expensive. For this reason, a Cloud provider requiring additional resources, should be able to pick out the right Clouds provider which satisfies its requirements reacting also to sudden environmental changes;

c) *Versatility and Security*: In a heterogeneous Cloud federation scenario, interoperability is a key concept. A Cloud provider requiring additional resources must be able to work with multiple Cloud providers based on different pieces of middleware. The ability of a Cloud consumer to be able to integrate different external Cloud providers in a transparent manner allows to integrate computing, storage, and network services across multiple operators simultaneously.

In order to achieve federation, it is fundamental the integration of different security technologies, for example, permitting a Cloud provider to be able to join the federation without changing its security policies. The Cloud systems require, generally, a simple, secure access to resources that they make available. The access to Cloud services is often achieved through web interfaces. Regardless of the model that will be implemented, an authentication system that allows a Cloud provider to access the resources offered by other operators maintaining its own identity, it is a necessary element. Hence, a Single Sign-On (SSO) authentication system is required.

d) *Authorization:* In a federated environment it is necessary that the internal management policies of the Cloud operators coexist without interfering with each other. Nowadays, it is necessary to apply access policies to user profiles and resources, because in a distributed system the access to a particular resource must be controlled. Typically, these policies establish which are the entities that may access a specific resource in a distributed system according to a matching criterion that allows to check whether the requirements of the subject correspond to the requirements of the system.

5 CASE OF STUDY

Currently, there are not standards that determine how the process of federation should be accomplished. In this Section, in order to describe how the *IaaS Cloud Federation* can take place, we will consider the three-phase model and several emerging technologies for its implementation.

A – The Three-phase Model. The three-phase model implies three subsequent phases for federation establishment: discovery, match-making, and authentication. During the discovery phase, all the available Cloud providers have to be discovered. As this environment cannot be a priori known, but it is pretty flexible and dynamic, the discovery process should be implemented in a totally distributed fashion with a peer-to-peer (p2p) approach. After that, the match-making phase has to choose the more convenient cloud providers with which to establish federation. All the available (discovered) providers have to be associated to several policies, describing the offered resources according to particular conditions. In addition, these policies have to match the policies of the provider requiring external resources. During the authentication phase, after that the Cloud providers have been selected for federation, a mechanism for creating a security context among the involved Cloud providers should be accomplished by means of Single Sign-On mechanisms.

B - XMPP or Web Services for Discovery Phase. As depicted in Figure 1, possible solutions to the discovery problem consist in using XMPP or Web Services. The architecture of the XMPP (Ejabberd, 2013) network is quite similar to the approach used by the e-mail service. Every user on the network is associated to a Jabber ID (JID) and communicate each other using a chat room. Scenario A shows a Cloud consumer that in order to gain knowledge about all the available Cloud providers, retrieves other providers' availability information merely querying the "shared location" on which it is published. More specifically, in a XMPP based scenario, the "location" is identified with a specific chat room, where the operators aiming to take part the federation hold a subscription.

Scenario B shows an alternative solution based on Web Services. This scenario includes a UDDI web server that is a xml-based registry (a sorted , indexed database) which allows Cloud providers to publish their WSDL document. A WSDL document give e formal description of the public interface of a web service. Therefore, a Cloud consumer can "read" the WSDL document related to a Web Service to determinate how the offered service can be used.

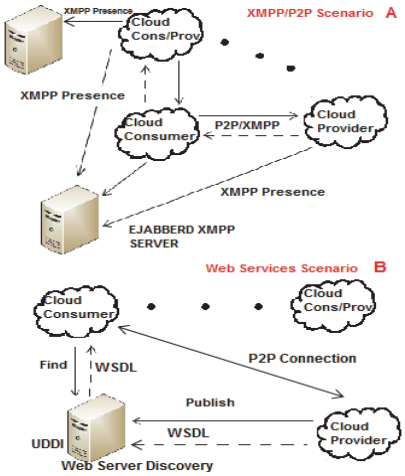


Figure 1: XMPP/P2P/Web Services for Discovery.

C - XACML for Match-Making Phase. During the Match-Making phase, it is needed to pick out which of the discovered operators. More specifically, the policies of the involved Cloud providers must coexist without affecting each other. Nowadays, in large-scale distributed systems there is the need to enforce policies in many different enforcement points. Typically such policies govern which subjects can access to a target resource of a distributed system according to a policy matching task performed in order to check if the requirements of the subject match the requirements of the system. To this end, a Cloud provider should be able to choose whether to accept or not a given Cloud consumer and it must also be able to restrict such access as appropriate. One of the best solutions which is able to address the aforementioned scenarios is the *eXtensible Access Control Markup Language* (XACML) technology (XACML, 2013). XACML allows to express policies by means of four major components: attributes, rules, policies, and policy set.

Attributes are characteristics of subjects, resources, actions, or environments that can be used to define a restriction. A *Rule* is the basic element of a policy. It identifies a complete and atomic authorization constraint which exists in isolation respecting to policy in which it has been created. A rule is composed by a Target, to identify the set of requests the rule is intended to restrict, an Effect, which is either "Permitted" or "Denied. *Policies*. A *Policy* is a combination of one or more rules. A policy contains a Target (specified using the same

components as the rule Target), a set of rules, and a rule combination algorithm.

Due to its nature, the XACML technology can be successfully used toachieve the match-making phase. Figure 2 shows how XACML allows to accomplish the match-making phase considering two Cloud operators. The actors of the above scenario are: a Cloud Consumer, a Cloud Provider, a server PEP (Policy Enforcement Point), a server PDP (Policy Decision Point) and a Policy Store. In the first step of the authorization process, the Cloud Consumer sends a Resources Request to PEP.

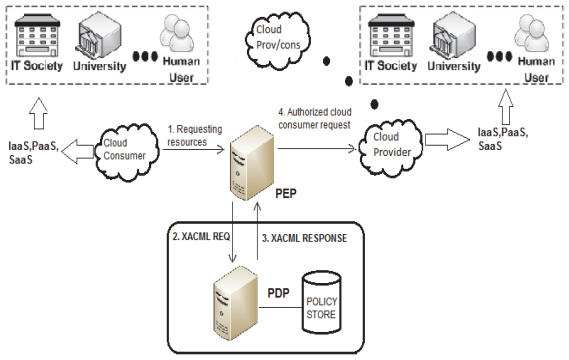


Figure 2: XACML Authorization Scenario.

The second step shows that the PEP translates the resource of the Cloud consumer in a XACML request. PEP forwards XACML request to PDP and asks whether the cloud consumers is authorized to access the requested resources. PEP evaluates the policies in the Policy Store and takes the decision according to the defined policies. At the third step PDP returns to PEP a XACML response message which contains the taken authorization decision. At the fourth step, the PEP depending on the XACML message received grants or denies access to resources.

D – SAML, OpenID and Shibboleth for Authentication Phase. During the Authentication phase it is needed to establish a trust context between different operators by means of SSO authentication mechanisms. In order to achieve such a goal, one of the major technologies using the Identity Provider/Service Provider (IdP/SP) model is *the Security Assertion Markup Language* (SAML) (SAML, 2013). It is an open standard based on XML for the exchange of authentication assertions between different parties. Figure 3 shows how the IdP/SP model can be used to establish a secure context between two operators. The Cloud consumer tries to access a Cloud provider (step 1). The Cloud provider requires a SAML assertion type (step 2). Therefore, the Cloud provider redirects the user to the IdP (step 3). The IdP authenticates the Cloud consumer that responses with authentication information provided by the IdP (step 4). The Cloud consumer forwards to Cloud provider a resources request attaching to it the authentication token obtained by IdP (step 5).

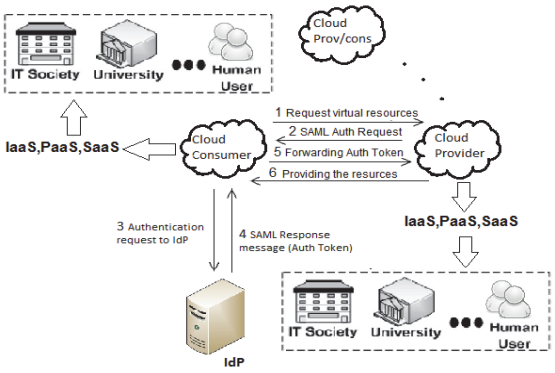


Figure 3: SAML Authentication Scenario.

Cloud provider unpacks the request obtaining the token and verifies its correctness. Finally, the Cloud provider contacts the cloud consumer notifying where and how to access the requested resources (step 6).

The Cloud consumer, subsequently, can request access the resources of other Cloud providers relying on IdP without further authentication tasks.

Other valuable technologies used to accomplish the SSO authentication are OpenID (OpenID, 2013) and Shibboleth (Shibboleth, 2013) .

6 CONCLUSIONS

Cloud federation is an emerging topic. Currently there are not so many standards and guidelines to accomplish a federation. In this paper, firstly, we performed a requirement analysis for the establishment of the IaaS Cloud federation, analyzing different architecture and models. Secondly, we discuss a case of study adopting the three-phase model and considering several technologies. We hope, our analysis will be useful for Cloud architects facing federation issues.

REFERENCES

A. Celesti, F. Tusa, M. Villari and A. Puliafito, "How to Enhance Cloud Architectures to Enable Cross-

Federation", *IEEE 3rd International Conference on Cloud Computing (CLOUD)*, pp.337-345, 2010.

B. Rochwerger, D. Breitgand, A. Epstein, D. Hadas,I.Loy, K. Nagin, J. Tordsson, C. Ragusa, M. Villari, S. Clayman, E. Levy, A. Maraschini, P. Massonet, H. Munoz and G. Toffetti, "Reservoir - when one cloud is not enough", in *Computer*, pp. 44-11, 2010.

L. Rodero-Merino, L. M. Vaquero, V. Gil, F. Galán, J. Fontán, R.S. Montero and I. Llorente, "From infrastructure delivery to service management in clouds" in *Future Generation Computer Systems*, 1226-1240, 2010.

A. Avetisyan , R. H. Campbell et al, "Open Cirrus: A Global Cloud Computing Testbed", in *Computer*, vol. 43, no. 4, pp. 35-43, 2010.

K. Keahey and M. Tsugawa and A. Matsunaga and J. A. B. Fortes, "Sky Computing", in *IEEE Internet Computing*, pp. 43-51, 2009.

A. J. Ferrera, F. Hernándezb et al, "OPTIMIS: A holistic approach to cloud service provisioning", in *Future Generation Computer Systems*, 66-77, 2012.

D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond and M. Morrow, "Blueprint for the InterCloud - protocols and formats for cloud computing interoperability", in *Fourth International Conference on Internet and Web Applications and Services (ICIW '09)*, pp. 328-336, 2009.

R. Buyya , R. Ranjan, R. N. Calheiros, InterCloud: utility-oriented federation of cloud computing environments for scaling of application services, *Algorithms and Architectures for Parallel Processing, ICA3PP'10*, pp. 13-31, 2010.

R. N. Calheiros, A. N. Toosi, C. Vecchiola, R. Buyya, A coordinator for scaling elastic applications across multiple clouds, *Future Generation Computer Systems,* 1350-1362, 2012.

S. K. Garg, C. Vecchiola, R. Buyya and Mandi, June 2011, A market exchange for trading utility and cloud computing services, *The Journal of Supercomputing*, pp. 1153-1174.

A. Celesti, M. Fazio, M. Villari, A. Puliafito, Virtual machine provisioning through satellite communications in federated Cloud environments, *Future Generation Computer Systems*, Volume 28, Issue 1, pp. 85-93, 2012.

A. Celesti, F. Tusa, M. Villari and A. Puliafito, "How the Dataweb Can Support Cloud Federation: Service Representation and Secure Data Exchange", *Second Symposium on Network Cloud Computing and Applications (NCCA)*, pp.73-79, 2012.

S. Chaisiri, B. Lee and D. Niyato, "Optimal virtual machine placement across multiple cloud providers", in *IEEE Asia-Pacific Services Computing Conference (APSCC 2009)*, pp. 103-110, 2009.

A. Celesti, N. Peditto, F. Verboso, M. Villari, A. Puliafito, "DRACO PaaS: A Distributed Resilient Adaptable Cloud Oriented Platform", in *IEEE 27th Internationa Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*, pp.1490,1497, 2013.

D. Breitgand, A. Marashini and J. Tordsson, "Policy-driven service placement optimization in federated clouds", in *Computer Science Technical Report*, IBM Haifa Labs, 2011.

Nimbus, "Cloud Computing for Science". http://www.nimbusproject.org, 2013

Eucalyptus, "Open Source Aws Compatible Private Clouds", http://eucalyptus.cs.ucsb.edu, 2013.

OpenQRM Enterprise, http://www.openqrm-enterprise.com/, 2013.

OpenNebula, "Flexible Enterprise Cloud Made Simple, opennebula.org", 2013

OpenStack, "Open source software for building", 2013

private and public clouds", http://www.openstack.org.

A. Celesti, F. Tusa, M. Villari, and A. Puliafito, "Integration of CLEVER clouds with third party software systems through a REST web service interface", in *IEEE Symposium on Computers and Communications (ISCC '12),* pp. 827-832, 2012.

Ejabberd, "The Erlang Jabber/XMPP Deamon", http://www.ejabberd.im, 2013.

XACML, https://www.oasis-open.org/, 2013

SAML,https://www.oasis-open.org/committees/, 2013

OpenID, http://openid.net/developers/specs/ 2013,

Shibboleth, https://shibboleth.net, 2013.

Analysing the Migration Time of Live Migration of Multiple Virtual Machines

Kateryna Rybina, Abhinandan Patni and Alexander Schill
Chair of Computer Networks, Faculty of Computer Science, Technical University Dresden, 01062 Dresden, Germany
{*kateryna.rybina, alexander.schill*}*@tu-dresden.de, abhinandan.patni2010@vit.ac.in*

Keywords: Virtual Machine, Live Migration of Multiple VMs, Interference Effects, KVM, SPEC CPU2006, VM Migration Costs, VM Migration Time, Energy-efficient Computing.

Abstract: Workload consolidation is a technique applied to achieve energy efficiency in data centres. It can be realized via live migration of virtual machines (VMs) between physical servers with the aim to power off idle servers and thus, save energy. In spite of innumerable benefits, the VM migration process introduces additional costs in terms of migration time and the energy overhead. This paper investigates the influence of workload as well as interference effects on the migration time of multiple VMs. We experimentally show that the migration time is proportional to the volume of memory copied between the source and the destination machines. Our experiment proves that the VMs, which run simultaneously on the physical machine compete for the available resources, and thus, the interference effects that occur, influence the migration time. We migrate multiple VMs in all possible permutations and investigate into the migration times. When the goal is to power off the source machine it is better to migrate memory intensive VMs first. Kernel-based Virtual Machine (KVM) is used as a hypervisor and the benchmarks from the SPEC CPU2006 benchmark suite are utilized as the workload.

1 INTRODUCTION

Virtualization of the physical machine allows multiple operating systems to be run on it. These operating systems are running in VMs, which are logically isolated from each other. Each VM gets its share of the server's resources assigned to it by a midleware called hypervisor, and may run its own application(s). This allows the physical resources of one server to be efficiently shared between multiple VMs, thus reducing the amount of needed physical hardware (Kofler and Spenneberg, 2012). It leads to energy saving in the IT infrastructure such as big data centres and clouds, as well as reducing the maintenance costs (Koomey, 2011).

KVM [1] is one of the examples of hypervisors that in combination with QEMU [2] hardware emulation program and application programming interface libvirt[3] constitute a complete virtualization system.

Another merit of virtualization is that it enables the live migration of the virtual machines. Live migration of the VMs is a transparent process of physically moving a VM from one server to another. During this, the applications continuously run on the VMs. It allows to consolidate the workload on the smaller number of physical machines, power off idle ones, and thus, save energy. Migration can be also applied for the load balancing policy (Gerofi et al., 2010) or for the reason of transparent infrastructure maintenance.

But the live VM migration itself introduces non negligible overhead to the system, in the literature referred to as migration costs. Migration time, during which service execution latency is observed and the energy overhead are the examples of the migration costs (Strunk, 2012). The latter occurs because the migration process requires additional resources such as CPU cycles and network bandwidth (Wu and Zhao, 2011). And the CPU is the main energy consumer of the system (Beloglazov et al., 2012).

Moreover, the applications on the co-located VMs may compete for the resources as they share cashes, memory channels, network and storage devices (Delimitrou and Kozyrakis, 2013). This can be understood as *interference effects*. The hypervisor first fairly allocates the resources to the running VMs on the server and then to the migration process. We believe, if there are not enough free resources (e.g. CPU

[1] http://www.linux-kvm.org/page/Main_Page.

[2] QUEMU: source machine emulation program. http://wiki.qemu.org/Main_Page.

[3] Libvirt: The virtualization API. http://libvirt.org/.

cycles, available bandwidth) on the server due to interference effects the migration time may expand.

In this paper we investigate the migration time of the VMs and how it is influenced by the workload running on the VM and the interference effects, caused by running co-located virtual machines on the server during migration. This paper makes the following contribution:

- We show experimentally that when running co-located VMs on the physical server the interference effects occur and investigate how they influence the migration time of multiple VMs.

Our paper is organized as follows. Section 2 explains how the live migration of virtual machines is realized. In Section 3, we present the utilized benchmarks and the experimental settings for carrying our live migration. Section 4 goes into the details of the experiments. Observations and results of the experiments are presented in Section 5. Then, the related works are discussed in Section 6 and in Section 7 we make the conclusion and outline the future research work.

2 PRE-COPY LIVE MIGRATION TECHNIQUE

Live migration of the VMs is a technique that allows VMs to be physically moved between servers when VMs are operating, namely running their workload without interrupts. Modern virtualization technology stores the images of the VMs in the network attached storage (NAS). In this case the migration process is reduced to copying the in-memory state and the content of CPU registers between source and destination physical machines. Thus, it is realized faster and does not create so much network traffic. KVM applies a technique for the live migration called pre-copy (Liu et al., 2011) which is explained in the following (see Figure 1). The migration algorithm is realized in several iterative rounds. At the first round (see Round 0 Figure 1) all memory pages used by the VM are pagewise copied from the source machine to the destination host. This is realized while the VM is running (executing its applications). As this process takes some time some of the memory pages on the source machine may be modified (dirtied), and thus no longer be consistent with the copy version on the destination host. Only these pages have to be re-sent in the consecutive round(s) (see Round 1...n Figure 1) in order to ensure memory consistency.

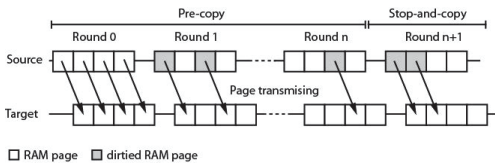


Figure 1: Pre-copy live migration algorithm (Liu et al., 2011), (Strunk and Dargie, 2013), (Rybina et al., 2013).

In order to prevent the first Pre-copy phase from executing indefinitely, it is important to introduce some threshold, so called stop condition. Usually, one of the following three thresholds (limits) are considered in the modern hypervisors as well as KVM (Liu et al., 2011), (Strunk and Dargie, 2013): (1) the number of iterations exceeds a pre-defined limit value ($n > n_{lim}$), (2) the total amount of memory that has already been transmitted exceeds a pre-defined limit ($memory_{mig} > memory_{lim}$), or (3) the number of dirtied pages in the previous round falls below a set limit value ($pages < pages_{lim}$). After one of the thresholds is reached, the hypervisor shortly suspends the VM to prevent further dirtying of the memory pages and copies the remaining dirtied memory pages as well as the state of the CPU registers to the destination host (see Stop-and-copy Figure 1). At this point the migration process is finished and the hypervisor on the destination host resumes the VM.

3 CONCEPT

Use case 1: In a scenario of having two underutilized servers, the idea is to migrate the workload (all VMs) from server one to server two and to power off the first server in order to save energy. This scenario would also suffice for the situation when server one has to be switched off for maintenance reasons. If on the first server k VMs are running, then the interference effects occur and the migration time might be influenced by it. It is of great interest to investigate whether the order at which we migrate all VMs influence the total migration time as well as the power consumption during migration.

Knowing these two parameters, we will be able later to define another migration cost factor called energy overhead. It is the power consumed during migration duration:

$$E_{overhead} = \left(P_{mig} - P_{bmig}\right) \times t_{mig} \qquad (1)$$

where , the P_{mig} is the power consumption of the system under consideration during migration, P_{bmig} stands for the average power consumption of the system before a migration process took place, and t_{mig} is the migration duration.

Use case 2: In *M* heterogeneous server scenario, when we have to migrate *N* VMs with the aim to make load balancing in the system, the problem of optimal resource consolidation becomes more complex and in the literature often referred to as the multiple "bin packing" problem (Verma et al., 2008), (Li et al., 2009). While in this case the bins having different volume refer to heterogeneous physical machines and the balls of different size with which we have to fill these bins refer to VMs.

In both scenarios it will be beneficial to leverage the knowledge about interference effects of different VMs running together in order to find later the best candidates for migration. For the Use case 1, that permutation of VMs which will result in least migration time, will be considered as the best permutation. As from the application perspective the migration time can be also considered as the application performance degradation time (Wu and Zhao, 2011).

The **three goals** of our experiment are to investigate:

1. how the migration time of a single as well as multiple VMs depends on the benchmarks (workload) running on the VMs;
2. whether interference effects occur in multiple VMs migration scenario and how they influence the migration time; and,
3. possible patterns which could indicate the favourable VMs migration order when we want to free the server by migrating all VMs from it.

We realize multiple VMs migration in all possible permutations and investigate into the migration times.

3.1 Benchmarks

As the workload for our experiments four benchmarks from the SPEC CPU2006 [4] benchmark suite were selected. These benchmarks stress the CPU and memory of the VMs to different extent, thus allowing us to observe dependence of the migration time on the applied workload. Two of the benchmarks are CPU intensive (and have slight memory usage) and another two are memory as well as CPU intensive. This creates an environment where the interference effects occur. For example, the memory intensive processes on different VMs may use the common data bus, thus slowing down the access to memory.

These benchmarks were chosen on the basis of the results from the hardware based profiler called Oprofile[5] which was used to fetch values recorded in the performance counters of the host CPU. The CPU intensive benchmarks used the maximum CPU time, implying that they kept the CPU busy most of the time. The memory intensive benchmarks had the maximum number of read/writes from/to the memory subsystem. After determining the highest values of the aforementioned performance counters the following four SPEC CPU2006 benchmarks were chosen (see Table 1).

For example 464.h264ref and 444.namd are two CPU intensive benchmarks that cause about 100% CPU utilization and the former creates a slightly more memory utilization than the latter. Though, the memory utilization of both these benchmarks is slight. 429.mcf and 401.bzip are highly memory intensive. For example it is required to have 1700 MB of memory free in order to run 429.mcf and it causes 100% CPU utilization. An extensive analysis of memory behaviour of SPEC CPU2006 was done by (Jaleel, 2010).

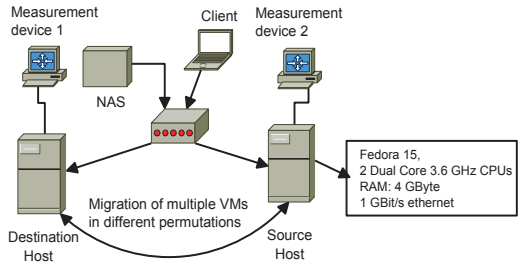


Figure 2: Experimental setup for live migration of multiple virtual machines in different permutations.

3.2 Settings of the Experiment

The experiment was based on the live migration of four virtual machines from the source physical machine (Source host) to the destination physical machine (Destination host) using network attached storage (NAS), and the client machine to trigger migration as depicted in Figure 2 .

Two physical machines under test are homogeneous with the following parameters: two Intel 15-680 Dual Core 3.6 GHz processors, 4 GB DDR3-1333 SDRAM, and with 1 Gbit/s Ethernet NIC. They are interconnected via a 1 Gbit/s switch. The NAS on which the VM images are located has the following characteristics: Intel Xeon E5620 Quad-Core 2.4 GHz processor, 10 GB DDR3-1333 SDRAM memory, and 1 Gbit/s Ethernet NIC. NAS is always accessed by the source and the destination servers.

Both physical machines run under Fedora 15[6] (Linux kernel v. 2.6.38, x86_64) operating system.

[4]SPEC CPU2006: http://www.spec.org/cpu2006

[5]OProfile: Profiler for Linux systems. http://oprofile.sourceforge.net

[6]Fedora 15. http://fedoraproject.org/.

Table 1: Description of the benchmarks used in the experiments.

Benchmark name	Description	Runs on VM
464.h264ref	CPU intensive; Integer benchmark	VM1 - $C1$
444.namd	CPU intensive; Floating point benchmark	VM2 - $C2$
429.mcf	Memory and CPU intensive; Integer benchmark	VM3 - $M1$
401.bzip2	Memory and CPU intensive; Integer benchmark	VM4 - $M2$

As a hypervisor we use KVM and apply a toolkit libvirt, to manage the interaction between the hypervisor and the operating system during migration. Furthermore, we installed a Virtual Machine Manager[7] (VMM). It is a desktop user interface which can be used to create, configure, and manage VMs through libvirt. Open source operating system FreeNAS [8], is used as the NAS.

All four virtual machines we migrated were allocated 1 virtual CPU and 30 GB disc space on the NAS. They run Fedora 15 as their operating system as well. Two identical measurement devices (Yokogawa WT210 digital power analyzers) are used to measure the overall AC power consumption from two physical machines between which the migration takes place. These devices are able to measure DC and AC power consumption at the rate of 10 Hz and a DC current between 15 μA and 26 A. The four virtual machines were executing as a workload benchmarks from the SPEC CPU2006 benchmark suite.

4 EXPERIMENT

We run Spec CPU2006 benchmarks on four virtual machines and perform the live migration of the VMs from the source to destination server at the network bandwidth of 100 MBps. Each VM runs its own benchmark, is assigned one core, and is executed on the separate core of the physical server. The benchmark 464.264ref is run on VM1. We depict this VM on figures as C1 (see Table 1). The benchmark 444.namd is run on VM2, is depicted as C2; 429.mcf is run on VM3 and is depicted as M1; and 401.bzip2 is run on VM4 and is depicted as M2.

In order to be able to reason about interference effects and their influence on the migration time, we were migrating all four VMs from the source server to destination server sequentially one after another till the source server was running idle (see Use case 1). We did the experiment for all possible permutations of four virtual machines, which resulted in (4!) 24 migration permutations (see Table 2).

The benchmarks are continuously running on the virtual machines during live migration. This means that during measurements, different functions of the benchmarks were executing. Hence, there are some discrepancies in terms of migration times. These have been minimized by having a higher number of migration iterations, namely ten repetitions for every permutation of VMs, so that statistical consistency is maintained. Every experimental run for each permutation was done ten times, in all, there were 240 migrations done for 24 test cases.

During the measurements, we ran the dstat[9] program on the source and destination servers, as well as VMs to observe and record resource utilization (CPU and memory). The migration data, namely the beginning of migration, migration duration of each VM in the permutation, and the end time of migration were saved as .csv files in the client machine to enable future analysis. Before the experiments took place all the servers were time synchronized in order to accurately determine the beginning and the end of a VM migration.

Besides the resource utilization and migration times we also recorded the power consumption of the two servers to enable our future work on the energy overhead of migration of multiple VMs (see Equation 1).

5 OBSERVATIONS AND RESULTS

We will step by step address the three goals (1), (2), (3) set in this paper.

(1) In agreement with other research works (Clark et al., 2005) we experimentally proved that the migration time depends on the type of workload running on the VM. Figure 3 displays the average migration time of each of the four VMs which were executing their corresponding SPEC CPU2006 benchmarks during migration. We can see that virtual machine C2 needed on average about 50 seconds less for migration than virtual machine M1. It can be explained

[7] Virtual machine manager. http://virt-manager.org

[8] Freenas: Storage for open source. http://www.freenas.org/.

[9] dstat: Resource statistics tool. http://linux.die.net/man/1/dstat

Table 2: Migration settings.

Benchmark name	464.h264ref; 444.namd; 429.mcf; 401.bzip2
Bandwidth	100 MBps
VMs permutations (4!)	C1C2M1M2; C1C2M2M1; ... M2M1C2C1

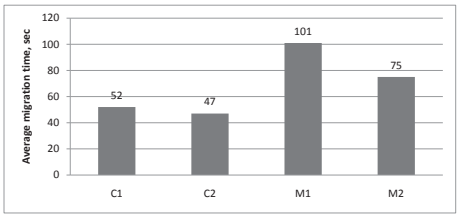


Figure 3: The average migration times of each VM over all the experiments. Each VM is running one of prescribed SPEC CPU2006 benchmarks. Network bandwidth is 100 MBps.

by the nature of workload running on these machines and the pre-copy migration approach. M1 is the VM, which is running in this test-case the most memory intensive benchmark (429.mcf). Thus, more memory pages have to be copied between two servers and during the migration process more pages might be dirtied and need to be resent. This results in high migration times. In agreement with the works of (Clark et al., 2005), (Strunk and Dargie, 2013) we are in a position to say that the migration time is proportional to the VM's memory volume (RAM) which has to be copied and sent between two servers. On the contrary VM C2 is executing predominantly CPU intensive tasks that resulted in smaller migration times (47 seconds). VM C1 is running CPU intensive tasks with slightly higher memory utilization than C2, that is why C1 needed in average more time to be migrated.

Thinking about the bin-packing problem (see Use case 2), where e.g. only one or some of the VMs have to be migrated from the overloaded to underutilized servers for load balancing reason, we are in a position to say that the best VM candidates would be the

- VMs running CPU intensive tasks rather than memory intensive tasks. As the migration time for the former is smaller and the services running on such VMs are less likely to be degraded.

This assumption was confirmed by our experiment, hence VM C2 would be a better candidate to migrate than VM M1 (see Figure 3).

(2) We migrated four VMs from the source to the destination server in all 24 possible permutations. Afterwards we analysed the total migration time of each single VM in all the permutations and made the following observations:

- when running multiple VMs on the source server during migration the interference effects occur and they significantly influence the migration time.

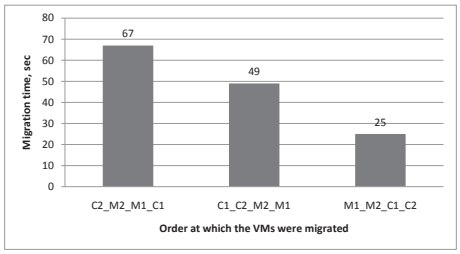


Figure 4: Migration time of the virtual machine C2, executing 444.namd benchmark, when migrated in the three displayed permutations at the network bandwidth of 100 MBps.

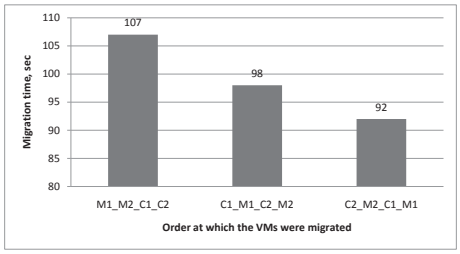


Figure 5: Migration time of the virtual machine M1, executing 429.mcf benchmark, when migrated in the three displayed permutations at the network bandwidth of 100 MBps.

For better visibility and presentation of results we selected each time three out of the 24 VM's migration permutations in the following-up figures.

At first all four VMs are running on the source machine and then they are one by one migrated from it to the destination machine. Figure 4 displays the average migration time of the single virtual machine C2 in the three following permutations: C2-M2-M1-C1, C1-C2-M2-M1, and M1-M2-C1-C2. We can clearly

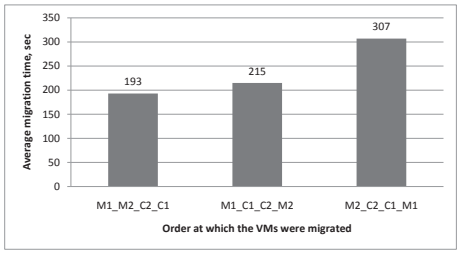


Figure 6: The summed migration time of four VMs in the described permutations.

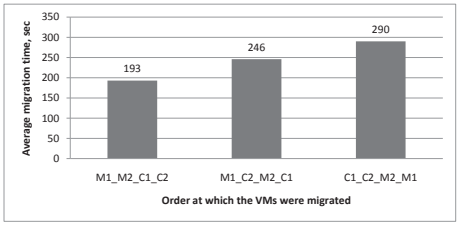


Figure 7: The summed migration time of four VMs in the depicted permutations.

see the influence of the interference effects on the migration time of C2. In the first permutation the VM C2 is migrated first from the source machine, namely when VMs M2, M1 and C1 are normally executing on the source machine. We can see that the migration time in this case is the biggest, namely 67 seconds. In this situation the hypervisor fairly allocates the CPU cycles first to the running VMs and then to migration process. And as there are not enough free resources on the source host to cover the migration overhead, the migration time is high. In the second displayed permutation virtual machine C2 is migrated after VM C1 has already been migrated to the destination server, thus migration process gets more CPU cycles and the migration now is realized faster. In the last third depicted permutation C2 is migrated in the last case from the source machine, thus having more than enough free resources to cover the migration overhead, thus the smallest 25 seconds migration time. This applies also for other VMs (see for example Figure 5 for the virtual machine M1). When M1 is migrated at the first, second and the last place in permutations its migration time is 107, 98 and 92 seconds correspondingly.

(3) Then, we investigated the time needed to free the source server e.g. for maintenance reason or with the aim to power it off and save energy. It is the summed total migration time needed to migrate four VMs from the source to the destination server. We did the experiment for all 24 possible permutations of four VMs and made the following observations regarding migration time patterns:

- It is better to migrate virtual machines running more intensive benchmarks first for the case when all VMs have to be migrated from the server (Use case 1).

Having the knowledge about utilized benchmarks from Oprofile and resource utilization statistics (dstat), we can conclude that between memory benchmarks, M1 is running a more intensive benchmark compared to M2. The same conclusion can be made for C1 being more intensive than C2, as besides being CPU intensive C1 also causes slight utilization of the memory subsystem.

Figure 6 presents the total migration time of all four VMs in three depicted permutations. The permutations M1-M2-C2-C1 or M1-C1-C2-M2 need less time to be migrated than M2-C2-C1-M1. It can be explained by the fact, that the migration (pre-copy and stop-and-copy phases) requires additional CPU cycles.The hypervisor first allocates resources to the running VMs, and then to the migration process. And thus, migrating first intensive tasks releases more resource for allocating them to the migration process of other VMs.

- When the goal is to free the server (migrate all VMs), then it is better to migrate virtual machines running the memory intensive benchmarks first rather than machines running CPU benchmarks (for Use case 1).

In Figure 7, the permutation M1-M2-C1-C2 has the least migration duration. And the total migration time is 97 seconds less than for permutation C1-C2-M2-M1. Which is quite a considerable value, that would allow to power off the source machine 1,5 minutes faster and reduce the total migration time and at the same time the service degradation time. The VMs during migration are executing only on the source machine. The hypervisor labels all memory pages occupied by the VM as read only. When some of the memory pages were overwritten during the migration (pre-copy iterations), the exception will be raised that the memory pages are faulted and have to be resent. Thus, the more memory intensive benchmarks are running, the more they modify the memory and use CPU, so migrating them first reduces the total migration time.

Thus, if one follows the goal to migrate exactly all VMs from the source host in order to switch it off, then considering the migration patterns might be beneficial.

6 RELATED WORK

Workload consolidation realized via live VM migration has been investigated in many research works (Akoush et al., 2010), (Wu and Zhao, 2011), (Kuno et al., 2011), (Andreolini et al., 2010), (Mi et al., 2010), (Orgerie et al., 2010), (Imada et al., 2009). The costs of migration process considered in the literature so far, were summarized in a survey paper (Strunk, 2012). The main migration costs addressed in research works are the total migration time and the service downtime. For the services which are running in VMs the migration time is at the same time their performance degradation time (Wu and Zhao, 2011). Clark et al. (Clark et al., 2005) reveals that the migration process slows down the transmission rate of the

Apache Web Server by up to 20%. Kuno et al. (Kuno et al., 2011) analysed the processing speed of CPU-intensive and the reading speed of IO-intensive applications during migration process. The performance of CPU intensive processes reduced by 15%. When additionally starting memory writing process on the same VM the performance declined by 40%. So it is important to minimize migration time and to understand the main parameters that influence it.

Wu et al. (Wu and Zhao, 2011) established a performance model which enables to derive dependency between the resource allocation (CPU) to the migration process and VM migration time. They set up four models for each of the application types (CPU, memory, disk I/O, and network I/O intensive) running in isolation on the migrated VM. They proved that the migration process requires additional CPU cycles and increasing the CPU share for migration process from 10% to 50% resulted in shorter migration times. But the VMs were running in isolation, and this scenario does not account for the interference effects which occur when several VMs with different workloads are running on the server when the migration takes place.

In (Rybina et al., 2013) the authors investigated the migration time and energy overhead of single VM migration under varying parameters such as available network bandwidth, size of the VM, and different CPU intensive workloads. It was revealed that migration time depends on the size of VM and the network bandwidth. Migration time decreases with higher network bandwidth and smaller VM size (RAM), which was also proved by Strunk et al. (Strunk and Dargie, 2013). But the other workloads rather than CPU intensive were not considered and the VMs were running in isolation which is not usually the case in real world scenario.

The negative interference effects of co-locating different workloads on the server have been investigated (Govindan et al., 2011). The interference happens even when running workloads on the separate processor cores, because the applications share the same resources such as cashes, memory channels, networking devices and storage (Delimitrou and Kozyrakis, 2013), (Govindan et al., 2011). But to the best of our knowledge the influence of interference effects on the live migration of multiple VMs was not addressed for far.

In our work we are going to migrate multiple virtual machines and to run on these VMs different types of workloads, thus to enable us to investigate into the interference effects and discover how the total migration time is influenced by them.

7 CONCLUSION

In this paper, we investigate the VM migration costs, namely VM migration time. We showed that the migration time depends on the interference effects caused by simultaneously running multiple virtual machines with different workloads on the source server when the migration was taking place. We migrated four VMs one after another from the source server to destination server in 24 possible permutations. Each of these VMs was running memory and/or CPU intensive benchmark from the SPEC CPU2006 benchmark suite. We migrate VMs at a bandwidth of 100 MBps. During migration we were recording the migration time, the resource utilization (CPU, memory) of both servers as well as the the power consumption.

We addressed three main goals with our experiment, namely how the migration time is influenced by (1) workload running on the VM, (2) interference effects which occur; and (3) we discovered the migration patterns which might be applicable when all VMs have to be migrated from the source server. Our experiment observations are as follows:

1. The migration time depends on the type of workload running on the VM and it is proportional to the volume of memory which has to be copied and sent between source and destination server. This observation is in agreement with other research work results (Clark et al., 2005), (Strunk and Dargie, 2013);
2. We showed that interference effects occur when running multiple VMs with different workloads on the source machine during migration and they significantly influence the migration time.
3. We discovered the migration patterns, that might be used in case all VMs have to be migrated from the source server, in order to repair or to switch it off.
   - It is better to migrate virtual machines running more intensive benchmarks first. Hence, permutations of VMs M1-M2-C2-C1 or M1-C1-C2-M2 need less time to be migrated than M2-C2-C1-M1.
   - It is better to migrate virtual machines running the memory intensive benchmarks first rather than machines running CPU benchmarks. Hence, permutation M1-M2-C1-C2 is better than C1-C2-M2-M1.

In our follow-up experiments, we will go into details of quantifying the interference effects of co-located VMs and their influence on the migration times. We will also continue our work on deriving the

energy overhead of migration of multiple VMs and modelling it.

ACKNOWLEDGEMENTS

This work has been partially funded by the German Research Foundation (DFG) under project agreement: SFB 912/1 2011.

REFERENCES

Akoush, S., Sohan, R., Rice, A., Moore, A., and Hopper, A. (2010). Predicting the performance of virtual machine migration. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE Int. Symposium on*, pages 37 –46.

Andreolini, M., Casolari, S., Colajanni, M., and Messori, M. (2010). Dynamic load management of virtual machines in cloud architectures. In Avresky, D., Diaz, M., Bode, A., Ciciani, B., and Dekel, E., editors, *Cloud Computing*, volume 34 of *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*, pages 201–214. Springer Berlin Heidelberg.

Beloglazov, A., Abawajy, J., and Buyya, R. (2012). Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, 28(5):755 – 768.

Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A. (2005). Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 273–286, Berkeley, CA, USA. USENIX Association.

Delimitrou, C. and Kozyrakis, C. (2013). Paragon: Qos-aware scheduling for heterogeneous datacenters. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, pages 77–88.

Gerofi, B., Fujita, H., and Ishikawa, Y. (2010). An efficient process live migration mechanism for load balanced distributed virtual environments. In *Cluster Computing (CLUSTER), IEEE Int. Conference on*, pages 197–206.

Govindan, S., Liu, J., Kansal, A., and Sivasubramaniam, A. (2011). Cuanta: Quantifying effects of shared on-chip resource interference for consolidated virtual machines. In *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, SOCC '11, pages 22:1–22:14.

Imada, T., Sato, M., and Kimura, H. (2009). Power and qos performance characteristics of virtualized servers. In *Grid Computing, 2009 10th IEEE/ACM International Conference on*, pages 232–240.

Jaleel, A. (2010). Memory characterization of workloads using instrumentation-driven simulation. *Web Copy: http://www. glue. umd. edu/ajaleel/workload*.

Kofler, M. and Spenneberg, R. (2012). Kvm fuer die server-virtualisierung - von konfiguration und administration bis clustering und cloud. In *ADDISON-WESLEY, ISBN 978-3-8273-3149-6*.

Koomey, J. (2011). Growth in data center electricity use 2005 to 2010. Technical report.

Kuno, Y., Nii, K., and Yamaguchi, S. (2011). A study on performance of processes in migrating virtual machines. In *Autonomous Decentralized Systems (ISADS), 10th Int. Symposium on*, pages 567 –572.

Li, B., Li, J., Huai, J., Wo, T., Li, Q., and Zhong, L. (2009). Enacloud: An energy-saving application live placement approach for cloud computing environments. In *IEEE CLOUD'09*, pages 17–24.

Liu, H., Xu, C.-Z., Jin, H., Gong, J., and Liao, X. (2011). Performance and energy modeling for live migration of virtual machines. In *Proceedings of the 20th int. symposium on High performance distributed computing*, HPDC '11, pages 171–182, NY, USA. ACM.

Mi, H., Wang, H., Yin, G., Zhou, Y., Shi, D., and Yuan, L. (2010). Online self-reconfiguration with performance guarantee for energy-efficient large-scale cloud computing data centers. In *Services Computing (SCC), 2010 IEEE International Conference on*, pages 514–521.

Orgerie, A.-C., Lefevre, L., and Gelas, J.-P. (2010). Demystifying energy consumption in grids and clouds. In *Green Computing Conference, 2010 International*, pages 335–342.

Rybina, K., Dargie, W., Strunk, A., and Schill, A. (2013). Investigation into the energy cost of live migration of virtual machines. In *Sustainable Internet and ICT for Sustainability (SustainIT)*, pages 1–8.

Strunk, A. (2012). Costs of virtual machine live migration: A survey. In *Services (SERVICES), 2012 IEEE Eighth World Congress on*, pages 323 –329.

Strunk, A. and Dargie, W. (2013). Does live migration of virtual machines cost energy? In *The 27th IEEE Int. Conference on Advanced Information Networking and Applications (AINA-2013)*.

Verma, A., Ahuja, P., and Neogi, A. (2008). pmapper: Power and migration cost aware application placement in virtualized systems. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '08, pages 243–264, New York, NY, USA. Springer-Verlag New York, Inc.

Wu, Y. and Zhao, M. (2011). Performance modeling of virtual machine live migration. In *Cloud Computing (CLOUD), 2011 IEEE Int. Conference on*, pages 492–499.

Towards a Security SLA-based Cloud Monitoring Service

Dana Petcu and Ciprian Crăciun
Computer Science Department, West University of Timişoara, Timişoara, Romania
{*petcu, ccraciun*}*@info.uvt.ro*

Keywords: Cloud Monitoring, Security Monitoring, SLA Management.

Abstract: Following the community concerns related to security and trust in cloud services, services level agreements (SLAs) are nowadays revised to include security requirements. In order to speedup their take-up by the service providers and consumers, security SLAs monitoring at run-time should be ensured. Several tools for SLA management are available, but most of them are dealing with performance parameters, and not referring to security. Other tools are available for cloud security monitoring, but not currently related or mapped to security SLAs. Aiming to design and develop a security SLA-based cloud monitoring service, which can be deployed or hosted, we identify in this paper the concepts, mechanism and available tools which can lead to a proper design of such a service, as well as the main barriers to overcome.

1 INTRODUCTION

The monitoring process is a key element to be enhanced for upgrading the quality level of the current cloud services. The monitoring of the cloud and of its service level agreements (SLAs) in simple terms like workload, performance or availability, offers both to the providers and the consumers the information necessary to implement mechanisms to prevent or recover from agreement violations. However there are only a few SLA management systems which are hosted or deployable in clouds.

Security monitoring is less developed than operational performance monitoring in cloud environments. Moreover, it is laging behind other security features at cloud providers. Security obligations associated with a service should be specified at request in a SLA (in security SLA, or, shortly, Sec-SLA). The absence of security aspects in the currently used SLAs, combined with the lack of methods for making objective comparisons between different service offerings, makes it impossible for providers to offer trustworthy services to their customers (Bernsmed et al., 2011). Moreover, according (Wagner et al., 2012), cloud services provider contracts will not provide detailed and substantive security SLAs before 2016.

In order to address these problems, we are interested in developing an open-source SLA-based Cloud security monitoring system that can act as monitoring-as-a-service or can be deployed in conjunction with the open-source mOSAIC PaaS (Petcu et al.,) which offers a certain degree of portability of applications consuming IaaSs. An incipient form of SLA-based monitoring system with no security parameters was designed, prototyped and reported in (Rak et al., 2011). The role of a new SLA-based cloud security monitoring service, in the SPECS framework based on mOSAIC PaaS, under design and development, was exposed in (Rak et al., 2013): a Sec-SLA that is negotiated will be monitored for compliance, and alerts will be generated in case of security changes or in case of Sec-SLA violations, leading to its enforcement. In this paper we discuss the availability or lack of adequate concepts, mechanisms and tools to be reused when such a service is designed. The next section identifies the state-of-art, while its following section evaluates the appropriateness of the existing solutions for the new service.

2 RELATED WORK

We have recently collected data and reports about the current tools and prototypes that are available for SLA monitoring or security monitoring in clouds. Tables 1-2 are pointing towards the most significant ones (not exhaustive lists). Here the tools and frameworks are organized in three categories: open-source, commercial or research prototypes.

Surprisingly, there is no report until now of any effort for developing a Sec-SLA based Cloud monitoring service. In what follows we point to some of

Table 1: Open-source and Commercial Cloud monitoring tools which are SLA-based or security-oriented.

Acronym	Short description
	Open-source SLA-oriented Cloud monitor tools
CloudCompass[1]	SLA-aware PaaS featuring an extension of the SLA specification WS-Agreement for cloud. The monitor module performs dynamic assessment of the QoS rules from active SLAs.
Sandpiper[2]	Automates the process of monitoring, detecting hotspots and reconfiguring VMs whenever necessary. The monitoring system uses thresholds to check whether SLAs can be violated.
SLA@SOI[3]	SLA management platform that includes at monitoring layer EVEREST, a general-purpose engine for monitoring the behavioural and quality properties of distributed systems based on events
	Open-source Cloud security monitor tools
FBCrypt[4]	Prevents information leakage; it encrypts the I/O between a client and a user VM using a virtual machine monitor -VMM: intercepts reads of virtual devices by a user VM and decrypts inputs.
Snorby[5]	Application for network/host security monitoring; integrates with intrusion detection systs (IDS)
	Commercial SLA-oriented Cloud monitor tools
NMS[6]	'NimSoft Monitoring Solution' is a unified monitoring dashboard able to monitor data centers of both private and public Clouds; it can be used for monitoring SLAs.
Site24x7[7]	Online website monitoring service continuously checking the availability of web-sites/apps; it includes application performance monitoring, SLA management, uptime reporting.
	Commercial Cloud security monitor tools
CipherCloud[8]	Service and virtual appliance delivering a set of protection controls including encryption, tokenization, activity monitoring, data loss prevention, malware detection.
CloudFlare[9]	Protects websites (optimize delivery, blocks threats, limit abusive bots and crawlers): after the enrolement of the website in the community, its web traffic is re-routed.
CloudPassage[10]	Automate security and compliance: integrated into platforms like OpenStack or by providers like AWS, it features a cloud-agnostic architecture, continuous security monitoring/control, and REST APIs for integration with automation tools like Puppet and security systs like Splunk
MARS[11]	Cisco's 'Monitoring, Analysis & Response System' designed to monitor logs and threats
SPAE[12]	'Security Performance Availability Engine' is a network monitoring tool supporting a various protocols and using SNMP; enables consumers to monitor networked Cloud resources.
Splunk' Storm[13]	Cloud-based service for analyzing machine data generated by web sites, applications, servers, networks, mobile device.
ThreatStack[14]	Deployable, profiling for normal behaviour, real-time monitoring: user loggings, network connections, data send to ThreatStack analyzer servers, firewall policies, forensics, audit alerts

the issues that have led to this current status, with accent on complexity and novelty.

Cloud monitoring is done at high or low level (Aceto et al., 2013). A high-level monitoring is related to information on the status of the virtual platform, collected at the middleware, application and user layers by service providers or consumers through platforms and services operated by themselves or by third parties. A low-level monitoring is related to information collected by the service provider and usually not exposed to the consumer, and is concerned with the status of the physical infrastructure.

The security monitoring is falling mostly in the category of high-level monitoring. For low-level monitoring specific utilities for collecting information about security are referring to software vulnerabilities or bugs (at OS and middleware layer), IDS or firewalls (at network layer), authentication systems or surveillance (at facility layer), workload, voltage or temperature, memory or CPU (at hardware level).

The definition of a security that can be quantifiable and can be expressed in a service level is a very complex task (de Chaves et al., 2010). In particular the definition of the security metric is challenging and

[1] github.com/angarg12/cloudcompaas-common
[2] lass.cs.umass.edu/projects/virtualization/sandpiper
[3] sourceforge.net/apps/trac/sla-at-soi/
[4] ksl.ci.kyutech.ac.jp/oss/fbcrypt
[5] github.com/snorby/snorby
[6] www.nimsoft.com/solutions/cloud-monitoring
[7] www.site24x7.com
[8] www.ciphercloud.com
[9] www.cloudflare.com/overview
[10] cloudpassage.com
[11] www.cisco.com/en/US/products/ps6241/
[12] shalb.com/en/spae/spae_features/
[13] www.splunkstorm.com
[14] www.threatstack.com

Table 2: Research prototypes of Cloud monitoring tools which are SLA-based or security-oriented.

Acronym	Short description
	SLA-oriented Cloud monitoring tools
CASViD	'Cloud application SLA violation detection' aims at SNMP-based monitoring and detecting SLA violations at application layer; it includes tools for resource allocation (Emeakaroha et al., 2012).
LoM2HiS	'Low-level Metrics to High-level SLA monitoring and mapping' monitors resource metrics and maps the metric values to high-level SLA parameter objectives (Emeakaroha et al., 2010).
QoS-MONaaS	'Quality of Service MONitoring as a Service' allows to describe in a formal SLA the key performance indicators of interest and the alerts in case of SLA violation (Adinolfi et al., 2012).
	Cloud security monitoring tools
Aftersight	Can be used to analyze the behavior of a VM; it decouples execution of the VMs from this analysis of the execution; it records non-deterministic events, inputs to a VM (Chow et al., 2008).
CloudSec	Provides active, transparent and real-time security monitoring for multiple concurrent VMs hosted on a cloud platform in an IaaS setting (Ibrahim et al., 2011).
CloudWatcher	Security monitoring as a service automatically detouring network packets to be inspected by pre-installed network security devices (Shin and Gu, 2012).
HyperWall	A hypervisor that is not be able to snoop on, or modify, the data (or code) that is exchanged between the VM and the resource, or on computation done in the VM (Szefer, 2013).
K-Tracer	Dynamically analyzes Windows kernel-level code and extracts malicious behavior from rootkits, based on QEMU virtualization technology (Lanzi et al., 2009).
KVMSec	Extension to the KVM to check the integrity of guest VMs by adding modules in host & guest side: guest OS sends information to the host about the VM status (Lombardi and Di Pietro, 2009).
Lares	Based on two VMs: an untrusted monitored VM and a security VM. The last monitors the first and can see into the state of the monitored VM using an introspection API (Payne et al., 2008).
Livewire	An IDS which uses the VMM to pull the intrusion detection logic out of a monitored VM; the IDS VM runs on the same server as the VM being monitored (Garfinkel and Rosenblum, 2003).
Lycosid	Detects hidden process in VMs comparing guest view with a VMM image (Jones et al., 2008).
MAVMM	A VMM for malware analysis extracts features of the applications running inside a guest OS: execution trace, memory pages, system calls, disk accesses, network (Nguyen et al., 2009).
MISURE	'Monitoring Infrastructure using Streams on an Ultra-scalable, near-Realtime Engine' is a monitoring-as-a-service for data analysis; uses stream processors like S4, Storm (Smit et al., 2013)
NICKLE	A VMM based on memory shadowing: the trusted VMM maintains a shadow physical memory for a running guest VM & performs real-time authentication of the kernel code (Riley et al., 2008).
Overshadow	Protects the privacy and integrity of application data in a guest VM even if the guest OS is compromised (the VMM provides guest physical memory pages accordingly) (Chen et al., 2008).
PoKeR	'Profiler of Kernel Rootkits' profiles four rootkit characteristics: hooking behavior, kernel object modifications, impact on user applications, code injection (Riley et al., 2009).
Revirt	Ensure secure logging and logs information: real-time clock, keyboard, mouse events, user inputs, system calls, enabling admin to replay the execution of VM/ analyze attacks (Dunlap et al., 2002).
Rkprofiler	Sandbox-based malware tracking using QEMU virtualization for Windows (Xuan et al., 2009).
SecMon	Secure introspection framework using a VMM for Windows OS (Wu et al., 2013).
SecVisor	Hypervisor supporting one guest VM to protect it from rootkits (Seshadri et al., 2007).
SIM	'Secure-in-vm monitoring': monitoring code in the VM with monitored code (Sharif et al., 2009).
VMWatcher	An out-of-the-box malware detection mechanism addressing the gap between observed events at the VMM level and guest OS context; ensures strong tamper-resistance (Jiang et al., 2007).
TrustVisor	Hypervisor which protects pieces of application logic to be execute in isolation: the programmer specifies these pieces as well as valid entry and exit points (McCune et al., 2010).

numbers are not appropriate as the security is related to a variety of properties, varying from a service performance to process maturity.

Fortunately, security parameters for a security monitoring framework were defined and classified recently in (Hogben and Dekker, 2012). Beyond the parameter definition, methods and techniques for measuring parameters in practice were defined. Moreover, thresholds were established to indicate when to trigger an event (how to determine the ranges of parameters that would trigger an incident report, or response and remediation based on real-time or regular service

level reports). However security indicators were not provided (a security indicator is an observable characteristic that correlates with a desired security property; the set of feasible indicator values is expected to form a nominal scale).

To overcome this problem, a step forward was made in (Pannetrat et al., 2013) by providing an attribute-based security property vocabulary and by developing security properties in abstract terms and as a properties with a set of defined attributes.

Cloud security monitoring is currently done on-premises, on the monitored infrastructure, or via a SaaS. In the case of monitoring on-premises, a security tool is able to make use of specific APIs as well to collect logs from cloud services. In the second case, of monitoring on monitored IaaS, a security tool is loaded directly into an IaaS (no high bandwidth requirement, possible some high storage costs, but currently there is a lack of a unified view on the approach). In the third case, monitoring data is obtained from the cloud service (if available), and hand it to a managed security service provider.

Looking at the available tools for Cloud security monitoring systems displayed in Tables 1-2, we see that most of them are low level. Their approach is either to take a complete VM as the monitoring granularity, such that they cannot capture the security incidents within individual VMs, or to focus on specific monitoring function that cannot be used for heterogeneous VMs concurrently running on one cloud node (Zou et al., 2013).

The few SLA management systems that are including monitoring features are comprehensive in terms of covering various cloud services. In particular several domain specific languages were developed to describe the monitored properties present in the SLA as well as the alert or SLA violation thresholds (e.g. in SLA@SOI and QoS-MONaaS, the first being re-used in various follow-up research projects). LoM2HiS is the first try to map high-level parameters objectives to low-level metrics. Despite the degree of granularity of those tools, most of their reported use-cases are concerned with performance monitoring. The security parameters have not been taken into consideration by them.

3 POTENTIAL COVERAGE FOR SEC-SLA MONITORING

We assume in what follows that the Sec-SLA to be adopted by the cloud service providers and consumers follows the vocabulary reported in (Pannetrat et al., 2013), including also its security indicators. We are interested in mapping the available mechanisms and tools to the various security properties from this vocabulary and in filling the gaps where this mapping is not possible. As targeting an open-source service, we are referring here only to the open-source (extensible) tools described in the previous section, as well as to other open-source general monitoring tools, like collectl[15], CloudCmp[16], Cloudstone[17], Ganglia[18], Groundwork[19], Hyperic-HQ[20], JasMINe[21], MonALISA[22], Nagios[23], PCMons[24], SIGAR[25], Zabbix[26].

We have build a matrix of coverage visible in Tables 3 and 4. We considered five levels for indicate if a monitor feature is provided by a service provider or a consumer software (0 – not possible; 1 – there are some serious issues with the proposed property; 2 – although technically conceivable it is less likely to be implemented due to cost & effort; 3 – not usually provided, or it is complicated to provide; 4 – usually happens) according to three categories, IaaS (I), P/SaaS (P: provider oriented) and App (A: consumer oriented, i.e. a VM in case of IaaS, or a software service in case of a PaaS/SaaS). S stands for a deployable service, while L refers to a programming library. We then considered in the last two columns the case of a deployable service and a programming library; we refined further the expectation levels in eleven levels (a – possible with the current tools to monitor and enforce; b – possible with the current tools to monitor; c – if tools are build to monitor/enforce; d – if tools are build to monitor; e – if tools are build to monitor but not straightforward; f – if tools are build to monitor/enforce but it is complex issue; g – if tools are build to monitor but it is complex issue; h – if tools are build to enforce but it is complex issue; i – although technically conceivable it is less likely to be implemented due to cost & effort; j – less likely to be implemented; k – not possible).

Such a matrix indicates not only the hot-spots in developing a Sec-SLA monitoring system (the ones near to zero, or near k), but also the probability that a service provider will comply with the Sec-SLA. For example, as '4' indicates 'easy to comply with', the

[15] collectl.sourceforge.net
[16] github.com/angl/cloudcmp
[17] radlab.cs.berkeley.edu/wiki/Projects/Cloudstone
[18] ganglia.sourceforge.net
[19] sourceforge.net/projects/gwmos
[20] sourceforge.net/projects/hyperic-hq
[21] maven.ow2.org/maven2/org/ow2/jasmine
[22] monalisa.caltech.edu/monalisa__Download.htm
[23] nagios.sourceforge.net
[24] code.google.com/p/pcmons
[25] sourceforge.net/projects/sigar
[26] www.zabbix.com/download.php

Table 3: Matrix of coverage for Sec-SLA monitoring by current mechanisms and tools.

Security property	I	P	A	S	L
Software integrity protection	3	3	2	g	i
Software integrity detection	0	3	2	i	h
Malware protection	0	3	3	k	i
Data alteration prevention	2	0	0	k	k
Data alteration detection	4	4	3	d	d
Data access level	4	4	4	c	c
External data exchange confidentiality	3	4	4	c	c
Authentication of data origin	3	3	3	c	c
Network authenticated server access	4	4	4	c	c
Network mutually authenticated channel	3	3	3	c	c
Non repudiation of origin	3	3	3	e	e
Non repudiation of receipt	2	2	2	i	i
Information flow control: blacklist	4	3	4	a	a
Information flow control: whitelist	4	3	4	a	a
% of systems with time synchronization	2	2	2	g	g
User traceability	2	3	4	i	d
Security event storage integrity level	3	3	3	h	h
Tenant isolation level	4	4	4	h	k
Collocation indistinguishability	3	3	2	k	k
Data portability	3	3	3	c	c
Mean time between incidents	4	4	4	b	b
% of timely incident reports	4	4	4	b	b
% of timely incident resolutions	4	4	4	b	b
User authentication & identity assurance level	4	4	4	a	a
Password storage protection level	4	4	4	a	a
% of timely suspension of unused accounts	4	4	4	b	b
Limitation of failed user authentications	4	4	4	a	a
Inactive session blocking	4	4	4	a	a
Limitation parallel active sessions	4	4	4	a	a
Cryptographic brute force resistance	4	4	4	i	i
Key generation quality	4	1	1	k	k
Key access control level	4	4	4	i	i
Cryptographic module protection level	3	3	2	f	f
% of systems with formal risk assessment	3	3	3	j	j
% of systems with tested controls	3	3	3	j	j
Country level anchoring	3	3	3	a	a
Personal data: consultation ability	2	2	2	k	k
Personal data: modification ability	2	2	2	k	k
Personal data: deletion ability	2	2	2	k	k
Personal data: timely access	2	2	2	k	k
Vulnerability exposure level	3	3	3	h	h
% of timely vulnerability corrections	4	4	4	b	b
% of timely vulnerability reports	4	4	4	b	b
Data deletion quality level	4	1	1	k	k
% of timely effective deletions	4	4	4	k	k
Data leakage detection	2	2	2	f	f
Data leakage prevention	2	2	2	f	f
Storage freshness	4	4	4	g	g
Storage retrievability	2	2	2	g	g
% durability	4	4	4	g	g
Authentication feature count	2	3	3	k	k
Tamper evidence	2	2	2	k	k
Tamper resistance	2	2	2	k	k
% of uptime	4	4	4	b	b
% of processed requests	4	4	4	b	b
% of timely recoveries	4	4	4	b	b
Mean time between failure	4	4	4	b	b
Recovery point objective	4	4	4	j	j
Recovery time actual	4	4	4	j	j
Recovery success ratio	4	4	4	j	j

Table 4: Continuation of Table 3.

Security property	I	P	A	S	L
Elasticity reserved capacity	3	3	3	k	k
% of timely provisioning requests	4	4	4	b	b
Allocation limitation	4	4	4	b	b
Denial of service attack resistance	4	4	4	k	k
% of compliant devices	4	4	4	b	b
% of compliant software	4	4	4	b	b
% of timely configuration change notifications	4	4	4	b	b
Configuration change reporting capability	4	4	4	b	b

38 appearances of '4's in 68 security properties means an expectation of 56% for an IaaS provider to easily comply. This percentage can reflect partially a lack of knowledge about some tools availability or a certain subjectivity in defining implementation difficulty. However, such uncertainty can generate a certain deviation from the above percentage, but not leading it to 100%. With this percentange we can explain why Sec-SLA monitoring systems are not yet in place.

4 CONCLUSIONS

Multiple conceptual and technical barriers must be overcome in order to implement a Sec-SLA monitoring service. Some of them were underlined in this paper: lack of acceptance and maturity of the SLA management systems, difficulty of mapping high level security properties to low level monitoring parameters, lack of deployment-layer agnosticism, extra complexity introduced by the virtualization, and so on. The identification of the available concepts, methods and available tools is only the first step for the implementation of the SPECS's Sec-SLA monitoring system in its two intended versions, deployable or hosted service. The first stable version of the open-source code is expected to be available in one year.

ACKNOWLEDGEMENTS

This research is partially supported by the Romanian grant PN-II-ID-PCE-2011-3-0260 (AMICAS), as preliminary study for the grant FP7-ICT-2013-10-610795 (SPECS).

REFERENCES

Aceto, G., Botta, A., De Donato, W., and Pescapè, A. (2013). Survey cloud monitoring: A survey. *Computer Networks*, 57(9):2093–2115.

Adinolfi, O., Cristaldi, R., Coppolino, L., and Romano, L.

(2012). Qos-monaas: A portable architecture for qos monitoring in the cloud. SITIS '12, pages 527–532.

Bernsmed, K., Jaatun, M. G., Meland, P. H., and Undheim, A. (2011). Security slas for federated cloud services. ARES '11, pages 202–209.

Chen, X., Garfinkel, T., Lewis, E. C., Subrahmanyam, P., Waldspurger, C. A., Boneh, D., Dwoskin, J., and Ports, D. R. (2008). Overshadow: A virtualization-based approach to retrofitting protection in commodity operating systems. *SIGOPS Oper. Syst. Rev.*, 42(2):2–13.

Chow, J., Garfinkel, T., and Chen, P. M. (2008). Decoupling dynamic program analysis from execution in virtual environments. ATC '08, pages 1–14.

de Chaves, S., Westphall, C., and Lamin, F. (2010). Sla perspective in security management for cloud computing. ICNS '10, pages 212–217.

Dunlap, G. W., King, S. T., Cinar, S., Basrai, M. A., and Chen, P. M. (2002). Revirt: Enabling intrusion analysis through virtual-machine logging and replay. *SIGOPS Oper. Syst. Rev.*, 36(SI):211–224.

Emeakaroha, V., Brandic, I., Maurer, M., and Dustdar, S. (2010). Low level metrics to high level slas - lom2his framework: Bridging the gap between monitored metrics and sla parameters in cloud environments. HPCS '10, pages 48–54.

Emeakaroha, V., Ferreto, T., Netto, M., Brandic, I., and De Rose, C. (2012). Casvid: Application level monitoring for sla violation detection in clouds. COMPSAC '12, pages 499–508.

Garfinkel, T. and Rosenblum, M. (2003). A virtual machine introspection based architecture for intrusion detection. NDSS'03, pages 191–206.

Hogben, G. and Dekker, M. (2012). Procure secure. a guide to monitoring of security service levels in cloud contracts. Technical report, European Network and Information Security Agency (ENISA).

Ibrahim, A., Hamlyn-Harris, J., Grundy, J., and Almorsy, M. (2011). Cloudsec: A security monitoring appliance for virtual machines in the iaas cloud model. NSS '11, pages 113–120.

Jiang, X., Wang, X., and Xu, D. (2007). Stealthy malware detection through vmm-based "out-of-the-box" semantic view reconstruction. CCS '07, pages 128–138.

Jones, S. T., Arpaci-Dusseau, A. C., and Arpaci-Dusseau, R. H. (2008). Vmm-based hidden process detection and identification using lycosid. VEE '08, pages 91–100.

Lanzi, A., Sharif, M. I., and Lee, W. (2009). K-tracer: A system for extracting kernel malware behavior. NDSS'09.

Lombardi, F. and Di Pietro, R. (2009). Kvmsec: A security extension for linux kernel virtual machines. SAC '09, pages 2029–2034.

McCune, J., Li, Y., Qu, N., Zhou, Z., Datta, A., Gligor, V., and Perrig, A. (2010). Trustvisor: Efficient tcb reduction and attestation. SP '10, pages 143–158.

Nguyen, A., Schear, N., Jung, H., Godiyal, A., King, S., and Nguyen, H. (2009). Mavmm: Lightweight and purpose built vmm for malware analysis. ACSAC '09, pages 441–450.

Pannetrat, A., Hogben, G., Katopodis, S., Spanoudakis, G., and Cazorla, C. S. (2013). D2.1: Security-aware sla specification language and cloud security dependency model. Technical report, Certification infrastrUcture for MUlti-Layer cloUd Services (CUMULUS).

Payne, B., Carbone, M., Sharif, M., and Lee, W. (2008). Lares: An architecture for secure active monitoring using virtualization. SP '08, pages 233–247.

Petcu, D., Di Martino, B., Venticinque, S., Rak, M., Máhr, T., Esnal Lopez, G., Brito, F., Cossu, R., Stopar, M., Šperka, S., and Stankovski, V. Experiences in building a mosaic of clouds. *Journal of Cloud Computing: Advances, Systems and Applications*, 2:12.

Rak, M., Luna, J., Petcu, D., Casola, V., Suri, N., and Villano, U. (2013). Security as a service using an sla-based approach via specs. CloudCom ' 13.

Rak, M., Venticinque, S., Máhr, T., Echevarria, G., and Esnal, G. (2011). Cloud application monitoring: The mosaic approach. CloudCom '11, pages 758–763.

Riley, R., Jiang, X., and Xu, D. (2008). Guest-transparent prevention of kernel rootkits with vmm-based memory shadowing. RAID '08, pages 1–20.

Riley, R., Jiang, X., and Xu, D. (2009). Multi-aspect profiling of kernel rootkit behavior. EuroSys '09, pages 47–60.

Seshadri, A., Luk, M., Qu, N., and Perrig, A. (2007). Secvisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity oses. SOSP '07, pages 335–350.

Sharif, M. I., Lee, W., Cui, W., and Lanzi, A. (2009). Secure in-vm monitoring using hardware virtualization. CCS '09, pages 477–487.

Shin, S. and Gu, G. (2012). Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?). ICNP '12, pages 1–6.

Smit, M., Simmons, B., and Litoiu, M. (2013). Distributed, application-level monitoring for heterogeneous clouds using stream processing. *Future Generation Computer Systems*, 29(8):2103–2114.

Szefer, J. M. (2013). *Architectures for Secure Cloud Computing Servers*. PhD thesis, University of Princeton.

Wagner, R., Heiser, J., Perkins, E., Nicolett, M., Kavanagh, K. M., Chuvakin, A., and Young, G. (2012). Predicts 2013: Cloud and services security. Technical report, Gartner ID:G00245775.

Wu, X., Gao, Y., Tian, X., Song, Y., Guo, B., Feng, B., and Sun, Y. (2013). Secmon: A secure introspection framework for hardware virtualization. PDP '13, pages 282–286.

Xuan, C., Copeland, J., and Beyah, R. (2009). Toward revealing kernel malware behavior in virtual execution environments. RAID '09, pages 304–325.

Zou, D., Zhang, W., Qiang, W., Xiang, G., Yang, L. T., Jin, H., and Hu, K. (2013). Design and implementation of a trusted monitoring framework for cloud platforms. *Future Generation Computer Systems*, 29(8):2092 – 2102.

Extending Hypervisor Architecture to Allow One Way Data Transfers from VMs to Hypervisors

Mustafa Aydin and Jeremy Jacob
Department of Computer Science, University of York, York, U.K.
{*mma510, jeremy.jacob*}*@york.ac.uk*

Keywords: Cloud, Hypervisor, Forensics, Security, Energy.

Abstract: We propose an alternative architecture to existing hypervisors, which allows for more data to be moved whilst requiring less work for hardware and networks. Our suggestion is to develop an extension to hypervisors for an interface which can allow data transfer one way from virtual machines to hypervisors. We argue that the ability to transfer data directly in this way can provide a number of benefits to cloud users and providers, namely in the areas of security (confidentiality, integrity, and through decreased overhead), reduced energy consumption, and better use of hardware resources.

1 INTRODUCTION

Cloud computing has started to enjoy widespread use, with the convenience of on-demand IT resources proving to be increasingly popular among its users. Although cloud customers are able to employ their machines in much the same way as they can local physical machines, the fact that cloud services are delivered remotely has forced many users to rethink the way in which certain tasks are performed. Among these tasks for example, is the need to back up important data.

The backing up of data is a common operation in any system, and in cloud systems backing up has generally meant redirecting data to secure locations elsewhere. Data backup and redirection may be done for a variety of reasons. Some of the more common reasons include the backing up of log data for security purposes, and the need to backup valuable information which is too precious to lose. The fact that cloud providers immediately delete VM images upon VM termination is one example of why cloud users may wish to redirect data before they no longer have access to it. For this reason redirection of important data is used as an important component of research looking into providing ways to aid cloud forensics investigations (Marty, 2011) (Zawoad et al., 2013).

The fact that cloud computing is becoming more popular means that the need to redirect data will also need to increase. Along with an increase in the amount of data being produced by applications generally, this means the amount of information needing to be backed up and/or redirected is getting larger all of the time. Sending this data will have an effect on performance and network overhead, and may even result in associated costs from the cloud provider, depending on the service used. In a world in which more and more data and metadata is being produced, we need a way in which data can be sent around or redirected in more efficient ways than we currently have at the moment.

Our aim in this work is to provide more details of our suggested design for an extended hypervisor architecture, outlining the advantages of using our approach and responding to possible concerns. Our motivation is provided by the belief that the implementation of our approach would help to make better use of available resources, and also make it more feasible for larger amounts of information to be redirected for use whenever required.

Our paper is structured in the following way: we describe the area of virtualisation in Section 2, we cover the design principles of our suggested architecture in Section 3, and the advantages provided by our approach in Section 4. We look at possible problems in Section 5, discuss other related work in Section 6, and provide a conclusion in Section 7.

2 VIRTUALISATION

Virtualisation is one of the key enablers of cloud computing, allowing a way of operating multiple in-

stances of operating systems (known as virtual machines, or VMs) on a single physical machine. VMs are run by a more privileged software layer known as a hypervisor, which controls and allocates access to the hardware resources between VMs. Virtualisation has many benefits, providing secure isolation between VMs, and allowing much more efficient use of physical machines by utilising spare hardware capacity. The need to make hypervisors use hardware ever more efficiently has meant the development of techniques which allow them to use available resources better.

One of the biggest developments in this area was the introduction of paravirtualisation. This is a technique in which the hypervisor presents guest machines with an interface through which certain executable commands are directly passed to the hypervisor, commands able to be performed much more quickly than they would if passed through the layers included within the virtual machine. In order to do this, both the hypervisor and the guest operating system must be configured to take advantage of this ability to paravirtualise. By cutting out unnecessary movement of calls through different layers of the virtualised environment, paravirtualisation allows for better utilisation of the hardware.

The successive adoption of paravirtualisation suggests that using effective cooperation between hypervisors and operating systems in order to speed up useful operations can see widespread use. Although paravirtualisation has so far been concerned with allowing operating system executions to be accelerated, there is no reason why the principle of providing faster services cannot be adapted to be useful in other contexts. If the production and subsequent transfer of data is an operation using more resources than necessary, it makes sense to be able to reduce the impact of this where possible.

We believe that taking advantage of similar principles to paravirtualisation may be one way to create a more efficient environment for the cloud. The ability to pass data directly from a VM to a hypervisor could reduce the amount of overall work required compared to sending the same data to a destination over a network. This can be seen in Figure 1, which shows data going through multiple virtual and hardware layers, before reaching its final destination. The forwarded data is first sent to the VM from the application, before being sent off to the outside world for storage. In contrast, in our proposed architecture, the data is sent directly to the hypervisor and VM in parallel, and reaches its destination having gone through as few layers as possible. This would help cloud users and cloud providers to make better use of the available resources, and reduce both performance and network overhead in particular instances.

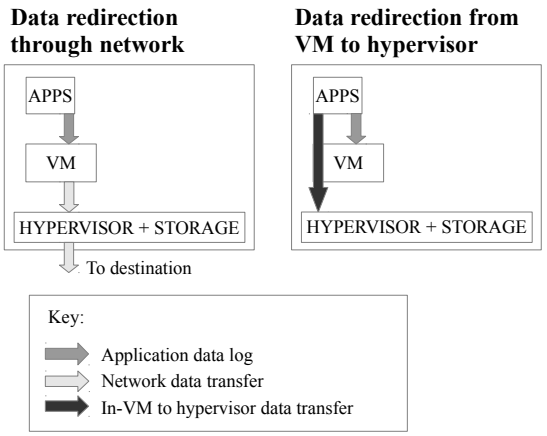


Figure 1: The diagram on the left shows the current architecture, while that on the right shows our proposal.

3 DESIGN

In this section we consider some of the important design principles, in order to provide a better understanding of the way in which the suggested architecture would work.

3.1 Storage

Redirecting data over the internet requires the data to go through many virtual and physical layers. In our proposed hypervisor extension, the data can be redirected directly to the hypervisor layer on the same physical machine (see Figure 1). Instead of passing through all of those layers and causing the related performance and network overheads, the data can be redirected with minimal effort to an environment which is already trusted by the user.

The location of storage of the transferred data is intended to be on the hypervisor layer, in a partition which is linked to the running VM (and associated user). This will allow subsequent retention of the information even when the VM has been terminated, and can be moved to a more convenient place after the VM has been shutdown. The ability to access the data will need to be offered to the user, but this may be of more concern to the cloud provider than to the design of the extended hypervisor.

3.2 Privacy

Privacy is an important concept to cloud users, who want assurances that their running VMs and any as-

sociated data is accessible only to themselves. Users want to have the option of privacy for all data they intend to send to alternative locations in the cloud. The use of techniques such as encryption is one way to provide this assurance.

Encryption of data at the moment of creation (*i.e.* before being passed to the hypervisor) would be a useful way of doing this. Users would have a level of assurance that even an administrator would be unable to access the transferred data (unless they somehow learned the encryption key). Techniques which allow the hypervisor to monitor some of the high level activity within a VM do actually exist, such as virtual machine introspection (Garfinkel and Rosenblum, 2003). However without the ability to monitor low level OS calls to the processor, the only other way someone would be able to break the encryption would be poor security on the part of the user.

Using established encryption techniques within the interface for data transfer should be enough to satisfy users that they have strong privacy in place for their data.

3.3 One Way

The interface for data transfer is intended to be only one way, allowing for the transfer of data from the virtual machine to the hypervisor. The reason for this limitation is that there is no conceivable reason for the hypervisor to send any data to the virtual machine, other than passing the communications which are required to be managed between the OS and the hardware.

Preventing the ability to send information from the hypervisor to the VM would be a necessary safety measure. Communication from hypervisors to VMs would provide significant security worries for users, and make hypervisors targets for hackers. Attackers would try to use any such abilities to attack VMs from the hypervisor level. Threats could even try to make use of such opportunities from within VMs, by communicating with other VMs on the same hardware to coordinate attacks. Maintaining the isolation of VMs is an important security principle which could be violated by the ability to have two way messaging.

A problem created by the need to have one way messaging would be the lack of acknowledgement from the hypervisor for transferred data, which may result in some incorrect data transfers.

3.4 Integrity

Integrity frequently comes into question after data moves location. Redirecting data across the cloud is an obvious case of when data integrity needs to be verified. It may be less obvious that it would need to be done when moving data from a VM to a hypervisor, since it does not move off the physical machine. However the data could always move later on, in which case its integrity would still need to be verified. In addition, in the cloud the hypervisor layer does not belong to the cloud user, making integrity an important feature of the data.

The interface used for sending data to the hypervisor needs to also have the ability to sign information with keys, in order to satisfy this need. Quicker signing methods using single keys would be useful to keep the performance overhead down, but more complicated ones using changing keys could also be considered depending on a user's preferences. Doing this at the hypervisor level could be problematic, since there may need to be a requirement to manage different keys for VMs, since a single physical machine in a cloud environment is likely to be providing services to more than one customer. It would make more sense for each operating system to be able to manage its own keys, reflecting the need for both hypervisors and operating systems to work well together in order to make better use of their environment.

3.5 Trust

The design presented here is based on the assumption that users trust the hardware owner of the hypervisor they are using. There is no more trust for the presented hypervisor architecture than is needed in any other proprietary hypervisor. The point here is only to reinforce the idea that whenever hypervisors are being used, there is a level of trust placed in the hardware owner the moment that a user chooses to use a VM on that platform.

Future possibilities for those who do not trust the hardware owner may lie in the use of homomorphic encryption technology, in which fully encrypted data is still able to be processed without the need for decryption (Gentry and Halevi, 2011). Use of such methods could be useful in order to keep valuable information safe on untrusted environments like the cloud, although the problem of its slow running speed will need to be solved.

4 ADVANTAGES

In this section we consider possible advantages from using the proposed architecture.

4.1 Speed

One of the most obvious benefits of the proposed architecture is the fact that the delivery of data to the intended destination would be performed as quickly as the hardware would allow. Delivery of data to any other location would be dependant on networks and network layers, and hardware and hardware layers. Even though data may always succeed in getting to the intended destination, services which would require fast delivery would still have to wait for data to be sent. For security monitoring processes, the difference in the time taken to do this could be important.

4.2 Overhead

Overhead can be broken down into two main areas, performance overhead and network overhead. The redirection of data to alternative locations affects both of these. Even copying a file onto another location on a local hard disk can noticeably slow down a running machine, so any opportunity with which this kind of costly data transfer can be avoided would be preferable to the average user.

An interface able to move data from one layer to another could also be used to specify a duplicate location in the hypervisor layer to place data at the moment it is created in the hardware. The simultaneous writing of the data to both locations avoids the need to copy any data later on, so the job does not need to be performed a second time (unless required for another reason).

The fact that data would no longer necessarily need to be sent over a network would relieve the pressure on networks too. This would allow for virtual machines which have a need to preserve network capacity to use their network speed for purposes other than the back up of data.

4.3 Security

Although sending information over networks is not inherently insecure if done correctly, keeping data within a trusted boundary does have its advantages. The ability to send data directly to another level on the same hardware prevents attacks such as a man in the middle attack from taking place, a worry for those with particularly sensitive information.

The most obvious point of attack for the VM owner's data is by a malicious administrator, who would be able to access the VM owner's data and their VM image, but would not be able to access them without also gaining the passwords with which they were encrypted. However, this danger would exist anywhere on the cloud on which a user stores data. By keeping data within trusted confines, it would at least reduce the need to look for additional services which can offer third party storage.

4.4 Energy

Energy use has become a much more important concern for organisations, as they have started to become aware of the costs of its use. Energy costs will be of most concern to cloud providers, due to the huge amounts of energy required to power their data centres. There are ways in which this concern could be passed to users as well. Firstly, if cloud providers ever decide to start using energy units as part of the cost calculations for their services. Secondly, due to wider concern about energy use, it is not unlikely that cloud providers could start to provide a breakdown of the energy use of individual virtual machines.

Work looking at the energy use of different kind of cloud services, and measuring the energy used to send cloud data around the world, is now being produced (Schien et al., 2012). Our suggested architecture could provide a way to decrease overall energy use in the cloud, by reducing the need to send data over the various hardware and network layers (which for users who redirect huge amounts of data could amount to a significant saving). Even for users who redirect small amounts of data, this could add up to a worthwhile reduction in energy use (especially over time, or from a collection of machines). With energy consumption becoming an environmental concern too, this is the kind of approach regular cloud users may express an interest in too.

5 PROBLEMS

Although our discussed design is intended to reduce issues, there is still the possibility that there will be problems.

5.1 Malware

In theory, malware might be able to use the opportunity to pass data directly to a hypervisor as an ideal opportunity to launch an attack. Despite this worry, hypervisors have so far proven to be well isolated against attack from VMs.

Nevertheless with security threats constantly evolving, there may develop new ways in which cleverly designed malware could use the access to the hypervisor provided here to attack the host, and by extension, the running VMs on the host. As an addi-

tional precaution therefore, it would be prudent to add additional security by writing the transferred data to a space from which nothing should be able to execute.

5.2 Scalability

Although one of the main benefits of the intended system design is to reduce the overall amount of work performed for the delivery of data overall, it is unknown exactly how the system would perform with large amounts of data being sent from multiple instances. These kinds of problems would only be able to be analysed from a prototype, from which any issues could then be properly addressed to make the architecture better able to deal with them.

5.3 Vendor Lock-in

Vendor lock-in is the problem of use of a technology or service by a customer from which they cannot easily transition to a competitor's alternative offering. There are already problems around this issue for various cloud services, such as the inability to move VMs between different cloud providers.

In the architecture we have outlined here, it is clear that users who would want to send data to the hypervisor layer, and hope to store that data within their user account, would have to do it with their VM provider. The only alternative ways of redirecting data would have to involve redirecting over the network.

6 RELATED WORK

Apart from the development of paravirtualisation, work on extended or modified hypervisors making use of alternative architectures have received some attention in the research literature.

Garfinkel *et al*. developed Terra, an architecture designed to help increase VM and application security (Garfinkel et al., 2003). Terra provides users with the ability to run applications within regular "open-box" VMs or specially protected "closed-box" VMs (with tailored security requirements), and uses Trusted Computing to authenticate communications betweeen applications.

Sailer *et al*. developed a secure hypervisor architecture called SHype, specifically to control data flows between VMs (Sailer et al., 2005). This was achieved by implementing Mandatory Access Control to enforce security policies from the hypervisor level in addition to relying only on OS security controls like a regular hypervisor would.

Keller *et al*. propose the NoHype architecture in order to reduce the risk of attack to the hypervisor layer (Keller et al., 2010). Instead of a hypervisor the system uses hardware features to split access to each of the required resources (in a similar way to a VMM) for each guest OS by utilising extensions to existing hardware.

7 CONCLUSION

We have suggested an interface to allow data transfer to take place from VMs to hypervisors, as a way of reducing the overall amount of work associated with data transfer over the cloud. We have looked at the problems associated with transferring data in this way, and demonstrated why it may be preferable to be able to send them directly to the hypervisor layer. By reducing the amount of overhead needed to perform particular jobs, and allowing users to transfer more data in an easier manner, we believe that such an architecture would be able to offer improved security, better utilisation of hardware resources in the cloud, and help to reduce energy consumption.

In order to do this, both operating systems and hypervisors would need to be adapted to take advantage of this property. The success of paravirtualisation suggests that cooperation between OS designers and hypervisors designers can make such interfaces work well.

The next step for this work will be to develop the extended architecture and to test with a suitable operating system. Validation through experimental results could encourage cloud stakeholders that the presented design is worth using to provide cloud services.

ACKNOWLEDGEMENTS

The authors thank BT for their contribution to this work, and the reviewers for their suggested corrections. Mustafa Aydin is sponsored by BT and is an EPSRC funded EngD student, and part of the LSC-ITS project.

REFERENCES

Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M., and Boneh, D. (2003). Terra: A virtual machine-based platform for trusted computing. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 193–206, New York, NY, USA. ACM.

Garfinkel, T. and Rosenblum, M. (2003). A virtual machine introspection based architecture for intrusion detection. In *In Proc. Network and Distributed Systems Security Symposium*, pages 191–206.

Gentry, C. and Halevi, S. (2011). Implementing gentrys fully-homomorphic encryption scheme. In *Advances in Cryptology–EUROCRYPT 2011*, pages 129–148. Springer.

Keller, E., Szefer, J., Rexford, J., and Lee, R. B. (2010). Nohype: Virtualized cloud infrastructure without the virtualization. *SIGARCH Comput. Archit. News*, 38(3):350–361.

Marty, R. (2011). Cloud application logging for forensics. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, pages 178–184, New York, NY, USA. ACM.

Sailer, R., Valdez, E., Jaeger, T., Perez, R., Doorn, L. V., Griffin, J. L., Berger, S., Sailer, R., Valdez, E., Jaeger, T., Perez, R., Doorn, L., Linwood, J., and Berger, G. S. (2005). shype: Secure hypervisor approach to trusted virtualized systems. In *IBM Research Report RC23511*.

Schien, D., Preist, C., Yearworth, M., and Shabajee, P. (2012). Impact of geographic location on the energy footprint of ict services. In *IEEE International Symposium on Sustainable Systems and Technology (IEEE ISSST 2012)*. Conference Organiser: IEEE.

Zawoad, S., Dutta, A. K., and Hasan, R. (2013). Seclaas: Secure logging-as-a-service for cloud forensics. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13, pages 219–230, New York, NY, USA. ACM.

Brokering SLAs for End-to-End QoS in Cloud Computing

Tommaso Cucinotta, Diego Lugones, Davide Cherubini and Karsten Oberle
Bell Laboratories, Alcatel-Lucent, Dublin, Ireland
{*firstname.lastname*}*@alcatel-lucent.com*

Keywords: Quality of Service, SLAs, Cloud Computing, Cloud Broker.

Abstract: In this paper, we present a brokering logic for providing precise end-to-end QoS levels to cloud applications distributed across a number of different business actors, such as network service providers (NSP) and cloud providers (CSP). The broker composes a number of available offerings from each provider, in a way that respects the QoS application constraints while minimizing costs incurred by cloud consumers.

1 INTRODUCTION

Cloud Computing introduces a novel model of computing that brings several technological and business advantages. Customers can rent cloud services on a pay-per-use model, without the need for big investments for resources that have to be designed for peak workloads, whilst being at risk of remaining under-utilized for most of the time. Providers may offer cloud services for rental, hosting them on big multi/many-core machines, where the infrastructure investments may be amortized over thousands of customers.

However, the requirements of cloud customers are evolving quickly, as cloud technology is being massively used worldwide. Many enterprise applications that might take advantage from the cloud model cannot be hosted on current infrastructures due to their stringent performance aspects requiring more and more from the best effort Internet. Think of virtual desktop, Network Function Virtualization (NFV), professional on-line multimedia editing and collaborative tools, and on-line gaming, just to mention a few.

Even though recent standards and research efforts deal with predictable QoS levels in Cloud Computing (Oberle et al., 2013), spanning across different business and operations domains remains a great challenge. For example, a single user request may have to traverse access, metro, core and data center networks, and the top-down provisioning chain across the various cloud layers (SaaS, IaaS, etc.), and still require tight interactivity. Figure 1 depicts the situation. Understanding how to combine all business actors in agreements to deliver end-to-end QoS may become overly difficult.

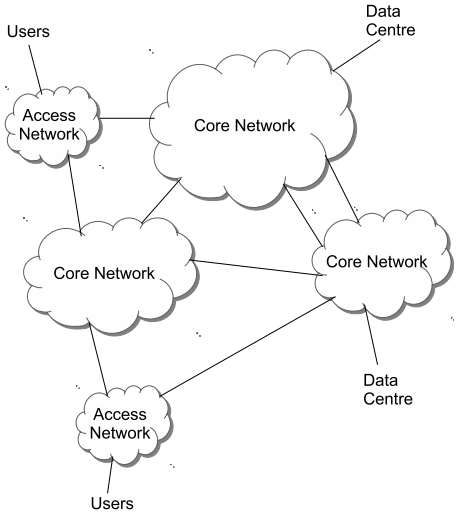


Figure 1: Users crossing multiple network service providers for accessing a cloud data center.

For these reasons, it is becoming increasingly important to have available intermediation services (a.k.a., brokerage) allowing cloud consumers to interface themselves to the multitude of service providers, including network carriers and cloud providers involved in the end-to-end service delivery chain. Brokering of cloud services (Plummer, 2012) allows for dealing with the various aspects of SLAs possibly involved in an end-to-end chain, including legal, economical and technological aspects. However, in this work we deal mainly with the latter aspects. That is, we consider cloud consumers who care about the price of a service and the expected and received end-to-end quality.

In this paper, we propose a brokering approach for

deploying distributed cloud applications with guaranteed end-to-end QoS which is achieved as an end-to-end composition of SLAs to be established along the multiple players participating in the chain. We focus on the mathematical formalization of the brokering logic as a mixed-integer geometric programming program. Note that, albeit a prerequisite to the brokerage of cloud services is a certain level of interoperability and compatibility among the offerings, we do not address these specific issues in this paper. Namely, we assume our brokering logic is managing only services and offerings that can interchangeably be used and composed with each other, for deploying end-to-end applications. For an overview of the issues behind cloud interoperability, we refer the reader to (Petcu, 2011).

2 RELATED WORK

The problem of brokerage of multiple CSP offerings for hosting distributed cloud applications with minimum cost has been addressed in (Houidi et al., 2011). It has been modeled as a MILP, similarly to what is done in this paper. However, Houidi's model is too simple as it does not consider QoS requirements nor capacity constraints. Pawluk et al. defined a cloud broker service (Pawluk et al., 2012) (STRATOS) based on comparing different CSP offerings. However, the presented broker logic is simply based on configuration properties matching (e.g., minimum values to be satisfied on certain properties for the offered services). InterCloud (Buyya et al., 2010) and other architectures (Grivas et al., 2010) (Ferrer et al., 2012) for federation of cloud providers include a broker component that may be an excellent candidate for realizing the brokering logic described in this paper.

The IRMOS European Project has addressed how to enhance execution of real-time multimedia applications in virtualized Cloud infrastructures (Cucinotta et al., 2010; Kyriazis et al., 2011). IRMOS ensures resource allocation for individual hosted applications. However, one of its limitations is that it does not address how to possibly guarantee service levels across domain boundaries (i.e., whenever interacting with other providers and infrastructures).

The SLA@SOI EU Project[1] developed a framework (Wieder et al., 2011) for negotiation, provisioning, monitoring and adaptation of SLAs through the entire cloud service life-cycle. It includes both functional and non-functional characteristics of services, such as QoS constraints, which can be formalized through an XML-based syntax.

The ETICS (Economics and Technologies for Inter-Carrier Services) European Project[2] investigated on the critical issues for the creation of a new ecosystem of innovative QoS-enabled interconnection models between Network Service Providers (NSPs) impacting all of the actors involved in the end-to-end service delivery value-chain. ETICS produced a novel architecture (Zwickl and Weisgrab, 2013) for control and management of automated end-to-end delivery of QoS-enabled services across heterogeneous carrier networks.

Furthermore, many studies exist trying to identify critical factors affecting a market of cloud offerings, including considerations related to QoS-aware provisioning of resources and services. An overview of such approaches can be found in (Breskovic et al., 2013).

Concerning standarization, the European Telecommunications Standards Institute (ETSI) is currently involved in several activities related to the issues mentioned above with the ETSI NFV Industry and Specification group. The Technical Committee CLOUD is also addressing interoperability aspects of end-to-end applications. NIST is also working in this area, with a series of reports being of value to the topic of end-to-end cloud service quality[3], in addition to the well-known reports on the definition of cloud computing (Mell and Grance, 2011) and its reference architecture (Liu et al., 2011), where the importance of having cloud brokerage services is emphasized. Additional details on research and standardization efforts in this area can be found in (Oberle et al., 2013).

3 BROKERING OPTIMIZATION LOGIC

Notation and Assumptions. From an abstract viewpoint, a distributed real-time cloud computing application is modeled as a set of components $A = \{1, 2, \dots\}$, among which we include also client/user-side components. These are interconnected in a logical application topology (limited to be a linear workflow in this paper) expressed as the sequence $S \equiv (s_1, \dots, s_n)$ of n elements of A that activate one after another, whenever triggered by users requests, coming at a minimum inter-arrival time T. Each activa-

[1] More information is available at: http://sla-at-soi.eu/.

[2] More information at: https://www.ict-etics.eu/.

[3] More information can be found at: http://www.nist.gov/itl/cloud/index.cfm.



Figure 2: Simple application sequence graph.

tion i of component s_i is associated with its computing requirements C_i, representing the amount of time needed for (sequential) computations within the component for each request received from the user. After computations, the component sends a message of size M_i to the next component s_{i+1} in the sequence S. For example, consider a cloud application composed of 3 components $A = \{a, b, c\}$, where a might represent the client part of the application running at the consumer premises (e.g., a web browser), b might represent the front-end server (e.g., a web-server), and c might represent the back-end component (e.g., a database). Then, a sequence might look like $S = (a, b, c, b, a)$, where user requests traverse the components forward and backwards.

A cloud application has to be deployed over a set of cloud data centers $\mathcal{P}$ belonging to one or more Cloud Service Providers (CSPs), interconnected by means of network providers (NSPs). Each NSP is characterized by a set of Edge Routers (ERs) $\mathcal{R}$, through which it is connected either to other NSPs or to CSPs or directly to users. From a mathematical perspective, we consider in what follows data center locations as edge routers as well, resulting in $\mathcal{P} \subset \mathcal{R}$. The situation can be depicted as a graph where each node represents an ER which has arcs towards all the other ERs to which it is connected.

CSPs are assumed to be able to host one or more components i by reserving a certain amount of computing resources u_i to it with a minimum VM computing latency of L^C, and we assume an ideal scalability model: if the component is assigned computing capacity $u_i \in (0, U^{max}] \in \mathbb{R}$ with VM wake-up latency of L^C, then the time needed to complete a user request is reasonably approximated or bounded as: $\frac{C_i}{u_i} + L^C$. The VM computing latency may depend on many factors, such as hardware and software configuration of hypervisors and guest OSes, including the CPU schedulers type and their configuration.

Similarly, NSPs are assumed to be able to place the communication flow between each component i and the next one by dedicating a network bandwidth b_i and ensuring a maximum communication latency L^N, in such a way that the time needed to transmit a message among these components (assuming no queuing of messages from the same application/flow is needed) can be reasonably estimated/bounded as: $\frac{M_i}{b_i} + L^N$.

We assume that our brokering logic has available a set of quotations (or equivalently a price list) from all of these providers. More precisely, for each available data center $p \in P$, we have available a few alternate quotations R_p. Each quotation $r \in R_p$ includes the cost for deploying components at the CSP premises that can be made available within conditions of quotation r; this includes $K^C_{p,r}$, the cost of a computing capacity unit up to a maximum computing capacity $U^C_{p,r}$, and $K^N_{p,r}$, the cost of a networking capacity unit up to a maximum networking capacity of $U^N_{p,r}$; the quotation also includes a maximum VM access latency of $L_{p,r}$. For example, the same CSP might provide two different quotations with different latencies, depending on the type of hardware where the VM(s) would be deployed, and/or the hypervisor and guest OS kernel configuration(s) , and/or the level of consolidation of multiple VMs onto the same host pushed by the provider in the context of that quotation; also, the latency might depend on the network configuration within the data center, so the CSP might be willing to offer VMs at particularly chosen locations within the DC, and/or using particular configurations of, e.g., an SDN-enabled data center network, so as to guarantee lower access latencies from the outside, at a conveniently higher unit-cost; different quotations may equally well represent different "sizes" of VMs that can be seen in current CSPs IaaS offerings. Also, for each NSP connecting two data centers or edge routers $p, q \in \mathcal{R}$, we have available a few alternate quotations $R_{p,q}$. Each quotation $r \in R_{p,q}$ is expressed as a cost $K_{p,q,r}$ for deploying, with latency $L_{p,q,r}$, a unit of communication bandwidth between p and q, up to a maximum available bandwidth of $B_{p,q,r}$.

In cases where, among two data centers $p, q \in \mathcal{P}$, there are multiple NSPs connecting them, or there are multiple paths traversing different NSPs for connecting them, we assume that all the corresponding quotations are combined together into a single list $R_{p,q}$ combining together the available multiple quotations to traverse. For example, quotations from two NSPs to be traversed for connecting two locations $p, q \in \mathcal{P}$ can be combined as follows: for each pair of quotations r_1 from the first NSP and r_2 from the second NSP, we consider an equivalent quotation r_{r_1,r_2} in which the bandwidth-unit costs and the latencies of r_1 and r_2 area *added* together, whilst the maximum available bandwidth for the quotation is the *minimum* among the two. After this operation of aggregating quotations from different NSPs, we obtain equivalent end-to-end quotations for all pairs of data center locations $p, q \in \mathcal{P}$, thus in our formalization we can forget about the existence of the intermediate edge routers $\mathcal{R} \setminus \mathcal{P}$ among multiple NSPs carrying the traffic among all CSPs.

Problem Formalization. Now we proceed to formalize the cloud brokering problem as an optimization program. To this purpose, we introduce Boolean variables $x_{i,p,r}$ representing whether the broker will deploy component i onto data center location $p \in \mathcal{P}$ making use of quotation $r \in R_p$, and the Booleans $y_{i,p,q,r}$ representing whether or not traffic between components s_i and s_{i+1} in sequence S is to be deployed between locations $p, q \in \mathcal{P}$ by using quotation $r \in R_{p,q}$. These variables are tied to each other by the constraints:

$$\sum_{p \in \mathcal{P}} \sum_{r \in R_p} x_{i,p,r} = 1 \; \forall s_i \in S \tag{1}$$

$$\sum_{r \in R_{p,q}} y_{i,p,q,r} = \left(\sum_{r_1 \in R_p} x_{s_i,p,r_1} \right) \left(\sum_{r_2 \in R_q} x_{s_{i+1},q,r_2} \right) \quad \forall p, q \in \mathcal{P}, \forall s_i \in S \setminus \{s_n\} \tag{2}$$

where: the first constraint forces each component to be placed onto a single cloud provider, exploiting one among the available quotations; the second one says that, if component s_i is deployed on p, and component s_{i+1} on q, then only one among the available quotations for communications between p and q has to be chosen; otherwise, $y_{i,p,q,r}$ must be 0 for each $r \in R_{p,q}$. Also, the deployment and quotations choice has to satisfy the overall end-to-end deadline D constraint:

$$\sum_{s_i \in S} \left(\frac{C_i}{u_i} + \sum_{p \in \mathcal{P}} \sum_{r \in R_p} L^C_{p,r} x_{i,p,r} + \frac{M_i}{b_i} + \sum_{p,q \in \mathcal{P}} \sum_{r \in R_{p,q}} L^N_{p,q,r} y_{i,p,q,r} \right) \leq D \tag{3}$$

and the allocation constraints

$$\sum_{s_i \in S} u_i x_{i,p,r} \leq U_{p,r} \; \forall p \in \mathcal{P}, \; \forall r \in R_p \tag{4}$$

$$\sum_{s_i \in S} b_i y_{i,p,q,r} \leq B_{p,q,r} \forall p, q \in \mathcal{P} \forall r \in R_{p,q} \tag{5}$$

where u_i and b_i are real problem variables. These are further constrained due to the assumption that requests submitted to the same application workflow at the specified minimum inter-arrival time of T do not queue after each other, namely:

$$u_i \geq \frac{C_i}{T} \tag{6}$$

$$b_i \geq \frac{M_i}{T}. \tag{7}$$

On the other hand, the broker will try to minimize the cost for hosting the application:

$$\min \sum_{s_i \in S} u_i \sum_{p \in \mathcal{P}} \sum_{r \in R_p} K_{p,r} x_{i,p,r} + \tag{8}$$

$$\sum_{s_i \in S \setminus \{s_n\}} b_i \sum_{p,q \in \mathcal{P}} \sum_{r \in R_{p,q}} K_{p,q,r} y_{i,p,q,r}. \tag{9}$$

The formalized problem falls in the class of mixed-integer geometric programming optimization programs, for which there are solvers available, such those found in the GAMS suite[4].

4 IMPLEMENTATION CONSIDERATIONS

Network Service Provider. In order to provide end-to-end cloud services with precise QoS levels, NSPs and CSPs must implement selected routing paths and reserve appropriate resources. On the networking side, we distinguish between *intra-domain routing* – if the end-to-end path is entirely within the same Autonomous System (AS), and *inter-domain routing* if the traffic is sent across multiple NSPs.

In *intra-domain routing*, users and DCs where their applications are deployed are all interconnected via the same NSP. In order to compose different networking offers, the NSP can adopt different strategies to build the end-to-end path. For example, a *best effort latency* offer can be realized by adopting traditional IGP protocols (e.g., OSPF or IS-IS) to route traffic between source and destination on the shortest path. Of course, not always shortest path translates into minimum latency. On the other hand, a *premium* offer (or *minimum latency* offer) can be realized using Multi Protocol Label Switching (MPLS) with its Traffic Engineering extensions (MPLS-TE). In particular, a Label Switched Path (LSP) is a one-way tunnel that can transport the traffic from origin to destination and that can satisfy multiple QoS parameters (e.g., bandwidth, delay, jitter, availability, and loss). The LSP routing path can either be configured automatically (e.g., using Constrained Shortest Path First – CSPF), or manually, e.g., as a result of more sophisticated techniques (Cherubini et al., 2011).

In *inter-domain routing*, when users can reach CSP's DCs across different NSPs, optimal path calculation is inherently more complex. Each router should have a global view of the network. Unfortunately, NSPs "filter" important information needed for the path establishment (e.g., for scalability or confidentiality reasons). The Path Computation Element pre-

[4]More information is available at: http://www.gams.com/solvers/

sented in (Farrel et al., 2006), represents a possible solution to provide an optimal inter-domain routing path (in the form of MPLS-TE LSP) that meets desired QoS requirements and can scale the end-to-end network up to 100,000 MPLS devices (Leymann et al., 2013).

Cloud Service Provider. The centralized design of today's data centers offers advantages in terms of fabric homogeneity and control, in comparison to the best-effort networks describedabove, which usually include various legacy equipment, specialized boxes, multiple protocols and are potentially operated by independent NSPs.

However, meeting the latency guarantees required by demanding Cloud applications in the data center fabric is still a significant challenge. The shared nature of the network in multi-tenant data centers leads to significant variations in the perceived performance. The lack of predictability increases the tenant costs and causes provider revenue loss. Moreover, the Internet-oriented protocols and flow scheduling mechanisms used in data centers are unaware of flow deadlines and application traffic patterns. Instead, they strive to optimize such low-level metrics as network throughput and fairness, ignoring the actual performance requirements associated to traffic flows.

Another reason that makes it difficult to deliver precise QoS levels in data centers is that typical designs focus on bisection bandwidth by overprovisioning the fabric. However, these are clearly not optimized for ultra-low latency applications with predictable delivery of packets. Moreover, overprovisioning today's data centers is prohibitively expensive.

The most common strategy is to implement QoS mechanisms by segregating traffic into different classes to provide isolation and enable traffic engineering. Typically, these mechanisms are implemented in switches and network cards where traffic is prioritized explicitly by marking packets or implicitly by using port ranges. In addition, the cloud services are typically run at low utilization to meet strict SLA demands, which decreases efficiency. However, advanced research in the field proposes different alternatives such as removing the kernel and network stack from the critical path of communication and load balancing requests across application instances (Kapoor et al., 2012). In (Ballani et al., 2011) authors propose to create virtual network abstractions to allow tenants to expose their network requirements. Proposal in (Alizadeh et al., 2012) run the network with near zero queueing and adaptively respond to congestion marks using DCTCP (Alizadeh et al., 2010).

Also, deadline-based scheduling of packets has been proposed as an alternative to TCP. In (Wilson et al., 2011), a control protocol is proposed that uses deadlines to achieve informed allocation of network bandwidth. In (Andrews, 2000), EDF scheduling is leveraged for providing probabilistic end-to-end guarantees to individual streams.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we addressed optimum brokering of cloud and carrier services accounting for end-to-end QoS requirements of cloud applications, and the availability of multiple offerings by CSPs and NSPs with different and heterogeneous quality levels corresponding to different price conditions. The problem has been mathematically modeled as a mixed-integer geometric programming optimization program that can be solved by available standard solvers. This work constitutes a first step towards the realization of evolved end-to-end cloud services offering to consumers precise, stable and reliable QoS levels across the heterogeneous, multi-provider "supply" chain that is involved in the process.

Our planned and ongoing future work in the area includes development of simplified versions of the introduced problem, as well as fast heuristic solvers for trading off accuracy of the solution versus solving time. Also, an implementation of the described technique is under way and its effectiveness will soon be evaluated by simulation, within our CloudNetSim framework (Cucinotta and Santogidis, 2013), and by real prototyping.

REFERENCES

Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M. (2010). Data center tcp (dctcp). *SIGCOMM Comput. Commun. Rev.*, 41(4):–.

Alizadeh, M., Kabbani, A., Edsall, T., Prabhakar, B., Vahdat, A., and Yasuda, M. (2012). Less is more: trading a little bandwidth for ultra-low latency in the data center. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, NSDI'12, pages 19–19, Berkeley, CA, USA. USENIX Association.

Andrews, M. (2000). Probabilistic end-to-end delay bounds for earliest deadline first scheduling. In *INFOCOM*, pages 603–612.

Ballani, H., Costa, P., Karagiannis, T., and Rowstron, A. (2011). Towards predictable datacenter networks. *SIGCOMM Comput. Commun. Rev.*, 41(4):242–253.

Breskovic, I., Brandic, I., and Altmann, J. (2013). Maximizing liquidity in cloud markets through standardization of computational resources. In *Proceedings of the 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*, SOSE '13, pages 72–83, Washington, DC, USA. IEEE Computer Society.

Buyya, R., Ranjan, R., and Calheiros, R. (2010). Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In Hsu, C.-H., Yang, L., Park, J., and Yeo, S.-S., editors, *Algorithms and Architectures for Parallel Processing*, volume 6081 of *Lecture Notes in Computer Science*, pages 13–31. Springer Berlin Heidelberg.

Cherubini, D., Fanni, A., Mereu, A., Frangioni, A., Murgia, C., Scutellà, M. G., and Zuddas, P. (2011). Linear Programming Models for Traffic Engineering in 100% Survivable Networks under Combined IS–IS/OSPF and MPLS-TE. *Computers & Operations Research*, 38(12):1805–1815.

Cucinotta, T., Checconi, F., Kousiouris, G., Kyriazis, D., Varvarigou, T., Mazzetti, A., Zlatev, Z., Papay, J., Boniface, M., Berger, S., Lamp, D., Voith, T., and Stein, M. (2010). Virtualised e-learning with real-time guarantees on the irmos platform. In *Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2010)*, pages 1–8, Perth, Australia.

Cucinotta, T. and Santogidis, A. (2013). Cloudnetsim - simulation of real-time cloud computing applications. In *Proceedings of the 4th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, Paris, France.

Farrel, A., Vasseur, J.-P., and Ash, J. (2006). RFC 4655 – A Path Computation Element (PCE)-Based Architecture.

Ferrer, A. J. et al. (2012). Optimis: A holistic approach to cloud service provisioning. *Future Generation Computer Systems*, 28(1):66 – 77.

Grivas, S., Kumar, T., and Wache, H. (2010). Cloud broker: Bringing intelligence into the cloud. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 544–545.

Houidi, I., Mechtri, M., Louati, W., and Zeghlache, D. (2011). Cloud service delivery across multiple cloud platforms. In *Services Computing (SCC), 2011 IEEE International Conference on*, pages 741–742.

Kapoor, R., Porter, G., Tewari, M., Voelker, G. M., and Vahdat, A. (2012). Chronos: predictable low latency for data center applications. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 9:1–9:14, New York, NY, USA. ACM.

Kyriazis, D., Menychtas, A., Kousiouris, G., Oberle, K., Voith, T., Boniface, M., Oliveros, E., Cucinotta, T., and Berger, S. (2011). A real-time service oriented infrastructure. *GSTF International Journal on Computing*, 1(2).

Leymann, N. et al. (2013). draft-ietf-mpls-seamless-mpls-04 – Seamless MPLS Architecture.

Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L., and Leaf, D. (2011). Nist special publication 500-292 – nist cloud computing reference architecture.

Mell, P. and Grance, T. (2011). NIST SP800-145: The NIST Definition of Cloud Computing.

Oberle, K., Cherubini, D., and Cucinotta, T. (2013). End-to-end service quality for cloud applications. In *Proceedings of the 10th International Conference on Economics of Grids, Clouds, Systems and Services*, Zaragoza, Spain.

Pawluk, P., Simmons, B., Smit, M., Litoiu, M., and Mankovski, S. (2012). Introducing stratos: A cloud broker service. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 891–898.

Petcu, D. (2011). *Towards a Service-Based Internet*, volume 6994 of *Lecture Notes in Computer Science*, chapter Portability and Interoperability between Clouds: Challenges and Case Study, pages 62–74. Springer Berlin Heidelberg.

Plummer, D. (2012). Gartner, inc. – cloud services brokerage: A must-have for most organizations.

Wieder, P., Butler, J., Theilmann, W., and Yahyapour, R. (2011). *Service Level Agreements for Cloud Computing*, springer edition.

Wilson, C., Ballani, H., Karagiannis, T., and Rowtron, A. (2011). Better never than late: meeting deadlines in datacenter networks. *SIGCOMM Comput. Commun. Rev.*, 41(4):50–61.

Zwickl, P. and Weisgrab, H. (2013). Final ETICS architecture and functional entities high level design.

Confidential Execution of Cloud Services

Tommaso Cucinotta, Davide Cherubini and Eric Jul
Bell Laboratories, Alcatel-Lucent, Dublin, Ireland
{*name.surname*}*@alcatel-lucent.com*

Keywords: Security, Cloud Computing, Confidentiality.

Abstract: In this paper, we present *Confidential Domain of Execution* (CDE), a mechanism for achieving confidential execution of software in an otherwise untrusted environment, e.g., at a Cloud Service Provider. This is achieved by using an *isolated execution environment* in which any communication with the outside untrusted world is *forcibly encrypted* by trusted hardware. The mechanism can be useful to overcome the challenging issues in guaranteeing confidential execution in virtualized infrastructures, including cloud computing and virtualized network functions, among other scenarios. Moreover, the proposed mechanism does not suffer from the performance drawbacks typical of other solutions proposed for secure computing, as highlighted by the presented novel validation results.

1 INTRODUCTION

Information and communication technologies (ICT) are undergoing a continuous and steep evolution and the wide availability of high-speed broadband connections is causing an inescapable shift towards distributed computing models. The traditional idea of a personal computer able to process data locally, is giving way to alternative computing paradigms, e.g., cloud computing, where personal computing devices are merely the point of access for data and computing services offered elsewhere. In this context, more and more often virtualization technology plays a major role in enabling new ICT scenarios. For example, in cloud computing it is used to enable deployment of multiple Virtual Machines (VMs) running on the same physical hosts. Network technologies are following the same trend, with the introduction of concepts such as Network function Virtualization (NfV) (NFV Industry Specif. Group, 2012), Software-Defined Networking (SDN) (McKeown et al., 2008; O. M. E. Committee, 2012) and Network Virtualization (Anderson et al., 2005).

Virtualizing (and sharing) network functions and infrastructures allow network operators to (1) minimize capital investments due to less expensive-longer life cycle hardware, (2) decrease operational expenditures due to, among others, reduced energy consumption, maintenance and co-location costs, and (3) speed-up the deployment of services. While *operational confidentiality* in shared network infrastructure has been widely investigated (Fukushima et al., 2011), a similar problem needs to be addressed when multiple operators are co-hosted and implement their network activities within the same system.

For example, in mobile communication, solutions have been presented to create multiple virtual base stations (V-BTS) on a single base station hardware platform (Sachs and Baucke, 2008; Chapin, 2002) or to enable cellular processing virtualization in data centers (Bhaumik et al., 2012). If a physical device is compromised, an attacker can potentially eavesdrop conversations, disrupt normal operations or even steal cryptographic material. Also, malicious or buggy software running on one virtual instance may leverage possible security weaknesses in the isolation mechanisms to compromise the confidentiality of other co-located virtual instances.

Similar attacks can potentially be performed in a SDN infrastructure when multiple controllers (e.g., OpenFlow controllers) are deployed and co-hosted onto remote shared servers.

One of the major problems hindering the uptake of such scenarios is therefore ensuring the proper level of security and isolation for software deployed by independent customers (a.k.a., *tenants*) within a shared physical equipment owned by others. Similarly, in cloud Infrastructure-as-a-Service (IaaS) provisioning, customers deploy Virtual Machines (VMs) within physical infrastructures owned by the provider (see Figure 1). If a user is using the cloud merely as a remote storage provider, confidentiality can readily be guaranteed through client-side encryption. Often, this scenario is obtained when deploying *cloud*

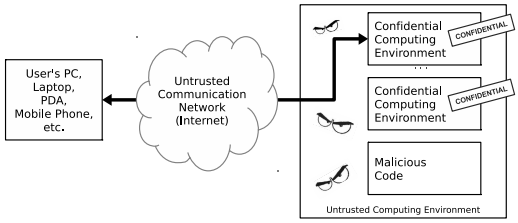


Figure 1: The problem of confidential execution of cloud services.

encryption gateways, often coupled with Software-as-a-Service (SaaS) clouds. However, this way users cannot exploit computing facilities in the cloud. The types of attack described above can be mitigated by using *trusted computing technologies* coupled with a secure boot process, in which a node is allowed to boot and run only after verifying that its software and hardware configuration has not been modified. However, the *Trusted Computing Base* (TCB), i.e., the set of hardware, firmware and software that a user needs to trust to keep data confidential, may be quite large (see Section 2).

In this paper, we illustrate a minimum set of features required to ensure confidential execution in an otherwise untrusted physical infrastructure. Our point is that all we need is a mechanism that completely isolates a user's execution environment, making its data *unreachable* from the outside, and that guarantees that all data exiting the environment is *forcibly encrypted*. In the proposed architecture, called *Confidential Domain of Execution* (CDE) (Cucinotta et al., 2014), the set of functionality to be trusted to ensure confidentiality is so limited that it is straightforward to envision a hardware-only implementation. Therefore, we can reduce the software part of the TCB of the provider to *zero*, and leave to the end-user full control of the code to deploy within the execution environment. Our solution neither allows the owner or administrator(s) of the physical infrastructure to control anyway, nor to spy upon, the software and data being deployed and processed within the CDE, so a CDE is also protected from malicious insiders.

Note that, in addition to summarizing the general architecture of our CDE solution that is going to appear on (Cucinotta et al., 2014), in this paper we also present novel results that validate the approach, showing that the CDE is not affected by the serious performance drawbacks typical of other proposed solutions for secure processing in untrusted environments.

2 RELATED WORK

We provide below a brief overview of solutions for trustworthy execution of software in untrusted environments, in the three areas of OS/run-time robustness, trusted computing, and secure processors.

OS/run-time Robustness – Traditionally, memory protection hardware mechanisms (Memory Management Units—MMUs) guarantee trust and isolation between execution environments. MMUs allow an Operating System (OS) kernel to isolate the execution environments of different users. However, the CPU can still enter a special mode of operation (the so-called *Ring 0*) in which malicious software can bypass normal OS security. Bugs in the OS kernel, in system processes, and in system calls implementations, can be exploited to gain administrator's privileges and subvert any security policy in the system or execute malicious code (Duflot et al., 2006). In addition, a malicious system administrator can overcome any security restriction.

Similarly, in virtualized environments, memory protection (including virtualization extensions (Uhlig et al., 2005; Advanced Micro Devices, Inc., 2008; Abramson et al., 2006)) can be leveraged to isolate the execution of different Virtual Machines (VMs). Still, the hypervisor embeds code that exploits the special mode of operation of the available processor(s) as to perform system management actions, thus bugs in the hypervisor can be exploited to break the VMs isolation. To completely rule out such a possibility, Szefer et al. (Szefer et al., 2011a) propose exploiting multi-core platforms in a way that allows a VM to run without the hypervisor mediation on a subset of the available hardware resources. However, the administrator of the infrastructure can still arbitrarily access any data managed by the hosted VMs. Novel cryptographic mechanisms, such as *homomorphic encryption* (Popa et al., 2011; Brenner et al., 2011), allow a provider to perform computations on encrypted data without being able to understand the data contents. However, applicability and wide adoption of such technique seem limited.

Trusted Computing – Numerous projects (e.g., see (Singaravelu et al., 2006)) have shown how to design a small software TCB, decreasing the likelihood of attacks. In OS designs, the historical idea of microkernels (Liedtke, 1995; Rashid, 1986), proposed as an alternative to monolithic OS architectures, aims to reduce the size of the critical part of a kernel which the system stability relies upon. In (Steinberg and Kauer, 2010), the authors apply a similar idea to hypervisors, developing the concept of *micro-hypervisor*. In this case, however, one still has to trust a non-trivial software TCB, i.e., the micro-hypervisor, albeit it is smaller than regular hypervisors.

A partial remedy to these problems, comes from

the use of *Trusted Platform Modules* (TPMs) (TPM, 2011). For example, in (Correia, 2012), a Trusted Virtualization Environment (TVE) has been proposed, leveraging TPM technologies to allow remote users check the software stack of a remote physical machine before deploying their VMs. However, TPMs provide no guarantees when bugs that can cause leakage of confidential information are present in the TCB (Hao and Cai, 2011). In (Keller et al., 2010), NoHype was presented, an architecture where the hypervisor is removed and the VMs have direct control on system resources that are automatically reserved to each of them. However, NoHype assumes that the server provider and the guest operating system are fully trusted and therefore it does not provide confidentiality of data and code running on the hosted VMs.

Secure Processors — In secure processor architectures (Lie et al., 2000; Suh et al., 2003b; Chhabra et al., 2010), data is kept in the main RAM memory in encrypted form: it is decrypted only within the secure processor when needed, then it is re-encrypted when it is written back to memory. Therefore, a physical attack aiming at spying on the traffic on the bus would only manage to see encrypted data. The main problem resides in the performance penalty incurred by the processor each time data has to be exchanged with the main-memory, i.e., at every cache-miss (for the cache level(s) residing inside the processor chip), causing the processor to stall waiting for data to be fetched and decrypted. To mitigate this, counter-mode encryption has been proposed (Suh et al., 2003a; Yang et al., 2003), in which the encryption scheme is realized as a XOR of the plain-text (cipher-text) and a one-time pad. The degradation of performance can be estimated as the additional cycles required for the XOR operation. However, counter-mode solutions have problems related to the management of the pads used for the encryption processes, and may limit the possibility to move blocks in the memory. Also, for the scheme to be effective, the encryption operations must complete within the memory access cycle, what makes the encryption engine expensive to be realized.

Our CDE solution aims to guarantee confidentiality without requiring expensive encryption and decryption operations at every memory access. It leverages a hardware-based enforced encryption of *the final output only* of the computations performed by the software running within the CDE. The remote user of the CDE has total control over the whole software stack running within the CDE. Furthermore, it is not possible to use any special mode of operation of any processor outside of a CDE, to spy on its memory.

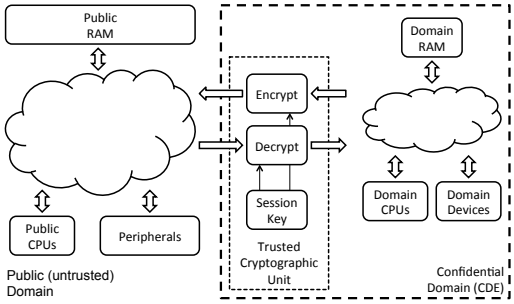


Figure 2: Overview of the Confidential Domains of Execution (CDE).

3 PROPOSED APPROACH

In this section, we briefly describe *Confidential Domain of Execution*(CDE), an abstract architecture that guarantees confidentiality in computing environments. For further details, the reader can refer to (Cucinotta et al., 2014).

In the remaining of the paper, we use the following conventions: the term *owner* (or *provider*, or *administrator*) indicates the entity with *physical* control over the computing machines. The term *user* (or *customer*) is the *remote user* interested in handing over computations to the provider's computing machines.

Informal CDE Description. Figure 1 illustrates the basic idea behind the CDE: to create, on a physical computing machine, one or more protected computing areas (CDE) that can be exclusively accessed via an encrypted channel.

As shown in Figure 2, each CDE includes one or more processing units along with their caches (Domain CPUs), RAM memory (Domain RAM), and peripherals (Domain Devices) used for computations and data movement. The behavior of a CDE is summarized as follows:

1. inside the CDE, confidential data/code is processed/executed in plain-text form (*unencrypted*), at the *native computing speed* of the physical platform;
2. any data flowing out of the CDE to the (potentially untrusted) outside world is *forcibly encrypted* and only the remote user (and other authorized users) can decrypt it; similarly, any data entering the CDE must be in encrypted form, and is decrypted when injected within;
3. all encryption operations are performed by a hardware Trusted Cryptographic Unit (TCU). Its function can neither be disabled nor worked around; not even the administrator of the physical ma-

chines is able to overcome the security features of a CDE;

4. a CDE can be reconfigured (reset) at any time by its owner. Then, the entire contents of the CDE including any memory and all CPU state is *forcibly cleared* by the CDE hardware;
5. the cryptographic material, used by the TCU to encrypt/decrypt any data crossing the CDE boundaries, is established by the remote user by using a cryptographic protocol (Cucinotta et al., 2014), that guarantees that the TCU is the *only* entity having access to it;
6. the CDE is manufactured with a built-in asymmetric key pair, where the private key is embedded in the TCU and is present nowhere else and the public key is made available through a Public Key Infrastructure.

Summarizing, as shown in Figure 1, a user can exploit the proposed mechanism to move confidential code and data for processing purposes onto a remote untrusted server owned by a (either trusted or untrusted provider). Due to the use of the built-in cryptographic material, the remote user can confidentially communicate with the target CDE, despite the traversal of untrusted networks and untrusted computing elements on the same physical machine where the CDE resides (i.e., without the use of secure communication).

CDE Implementation. The CDE architecture as depicted in Figure 2, can be implemented as a single System-on-Chip (SoC) or in more cost effective, but less secure, way by using Commercial-Off-The-Shelf (COTS) hardware elements. For a more exhaustive discussion of the different CDE implementations, including the protocol used to exchange the cryptographic keys, see (Cucinotta et al., 2014).

4 VALIDATION

Our proposed technique may also have a positive performance impact when compared to other solutions discussed in Section 2. Particularly relevant is comparison with secure processor architectures, where the memory is kept in encrypted form in main memory, and it is decrypted on-the-fly at every cache-miss.

We validated our approach using micro-benchmark use-cases that are used to give a rough estimate of the performance advantage we achieve. In the considered micro-benchmarks, the private computations handed over to the remote computing environment consist of: 1) **Matrix Inversion:** this can be a common operation to be used for solving big linear equations problems; 2) **Eigenvalue Computation:** this may constitute a recurrent operation in the context of physical simulations, where systems of integral/differential equations need to be solved; 3) **2D Fast Fourier Transform:** this is a common operation in image and video processing, particularly when applying linear filters. These benchmarks have been realized using the open-source tool GNU Octave[1], and invoking the functions `inv()`, `eig()` and `fft2()` on randomly generated square matrices.

Additionally, we considered a "macro-benchmark" service consisting of a video transcoder useful to change the resolution of a video so as to scale it down to a number of lower resolutions. We used the open-source `ffmpeg` software. All benchmarks have been performed on a laptop equipped with 4 logical cores Intel© Core™ i5-2520M and 4 GB of RAM, under the following conditions: 1) CPU clock frequency locked to 2.50 GHz; 2) 3 logical cores disabled at kernel level; 3) real-time priority (to minimize the impact of other running system services). L2 cache-misses have been measured by running the `stat` command of the Linux `perf` profiler tool. Assuming a strong cipher such as AES-128 (AES, 2001), implemented in CBC mode, and a 32-bit data bus used to transfer a 64-byte data block, the overall encryption and decryption operation delays have been measured to be 97 and 141 CPU cycles, respectively (Szefer et al., 2011b). Therefore, given the above assumptions, the average number of cache misses CM, the CPU clock frequency CPU_{freq} and the average execution time $time_{AV}$, then the percentage encryption overhead of the XOM-type architectures (Lie et al., 2000; Chhabra et al., 2010), can be roughly estimated as:

$$OH_{enc} = 97 \cdot \frac{CM}{CPU_{freq} \cdot time_{AV}} \cdot 100 \qquad (1)$$

and the correspondent percentage decryption overhead is:

$$OH_{dec} = 141 \cdot \frac{CM}{CPU_{freq} \cdot time_{AV}} \cdot 100 \qquad (2)$$

The considered benchmarks, along with the key used parameters and the encryption and decryption overheads, are summarized in Table 1. Results show that the impact on performance of the additional cryptographic operations is significant for most of the performed benchmarks. Nevertheless, in case of CPU greedy computations such as `eig()`, the time required

[1]More information is available at: www.gnu.org/software/octave/.

Table 1: Summary of the used benchmarks and configuration parameters, and obtained performance data.

Benchmark	Matrix Size	Average Time [s]	Cache-miss Average	Cache-miss Ratio	Encryption Overhead	Decryption Overhead
inv()	2048x2048	3.894	21737705.7	16.15%	21.66%	31.48%
inv()	4096x4096	27.907	180510029.5	26.81%	25.10%	36.48%
eig()	512x512	1.278	22529316.4	22.90%	68.40%	99.43%
eig()	1024x1024	11.06	430773713.7	55.37%	151.12%	219.67%
fft2()	1024x1024	0.32	1402260.0	17.73%	17.00%	24.71%
fft2()	2048x2048	0.47	3447163.9	26.57%	28.46%	41.37%
Benchmark	Matrix Size	Average Time [s]	Cache-miss Average	Cache-miss Ratio	Encryption Overhead	Decryption Overhead
ffmpeg	320x240	0.510	5704824.0	49.87%	43.40%	63.09%
ffmpeg	640x480	0.652	6729660.8	51.92%	40.05%	58.21%

by the crypto-computation is larger (by a factor of two) than the time required to perform the whole operation in plain-text. The proposed CDE solution does not have performance degradation due to cryptographic overhead because the whole computation is performed in plain text and cryptographic operations are required only when data or instructions enter or leave the protected area. On the other hand, as mentioned in Section 2, a significant performance improvement can be achieved using counter-mode AES architectures. In this case the encryption/decryption overhead can be estimated in the additional one-cycle XOR operation (Yang et al., 2003) for every cache miss. Still, the CDE presents advantages as compared to this last solution. Indeed, the CDE does not require the additional cycle operation, but more importantly it does not present the memory management issues of counter-mode AES architectures.

5 CONCLUSIONS

Virtualization of network functions onto standard servers creates potential confidentiality issues when the infrastructure provider is either untrusted or trusted but curious. Multi-tenancy adds a dimension of complexity to the confidentiality problem because an attack might actually come from software deployed by other customers.

We have presented a solution enabling confidential processing of data and code in an untrusted environment and shown that it keeps native performance of the underlying hardware at the same time as guaranteeing high security levels. With our proposal, the software TCB that an infrastructure provider must provide is reduced to *zero—we do not have to trust a single line of code provided by the service provider*. What we do have to trust is the hardware manufacturer to accurately and reliably produce the necessary hardware. Our less expensive realization using COTS components requires us to place some extra trust in the provider.

In the future, we plan to realize a working prototype on FPGA, to get concrete overhead data, and compare with other proposals. Also, a firmware-based implementation of the TCU has to be investigated, to provide upgradability of the cryptographic algorithms in use.

REFERENCES

(2001). Federal Information Processing Standards Publication 197 – Specification for the Advanced Encryption Standard (AES). U.S. Governement.

(2011). *TPM Main - Part 1 - Design Principles - Specification Version 1.2 - Revision 116*. Trusted Computing Group, Incorporated.

Abramson, D. et al. (2006). Intel® Virtualization Technology for Directed I/O. *Intel Technology Journal*, 10(3):179–192.

Advanced Micro Devices, Inc. (2008). AMD-V™ Nested Paging. AMD Techincal White Paper.

Anderson, T., Peterson, L., Shenker, S., and Turner, J. (2005). Overcoming the Internet Impasse through Virtualization. *Computer*, 38(4):34–41.

Bhaumik, S., Chandrabose, S. P., Jataprolu, M. K., Kumar, G., Muralidhar, A., Polakos, P., Srinivasan, V., and Woo, T. (2012). CloudIQ: a framework for processing base stations in a data center. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, Mobicom '12, pages 125–136.

Brenner, M., Wiebelitz, J., von Voigt, G., and Smith, M. (2011). Secret program execution in the cloud applying homomorphic encryption. In *Digital Ecosystems and Technologies Conference (DEST), 2011 Proceedings of the 5th IEEE International Conference on*, pages 114–119.

Chapin, J. (2002). Overview of vanu software radio.

Chhabra, S., Solihin, Y., Lal, R., and Hoekstra, M. (2010). An Analysis of Secure Processor Architectures. In

Gavrilova, M. and Tan, C., editors, *Trans. on Computational Science VII*, volume 5890 of *LNCS*, pages 101–121. Springer.

Correia, M. (2012). Software execution protection in the cloud. In *Proceedings of the 1st European Workshop on Dependable Cloud Computing*, EWDCC '12, New York, NY, USA. ACM.

Cucinotta, T., Cherubini, D., and Jul, E. (2014). Confidential Domains of Execution. *to appear in Bell Labs Technical Journal*, 19(1).

Duflot, L., Etiemble, D., and Grumelard, O. (2006). Using CPU System Management Mode to Circumvent Operating System Security Functions. In *CanSecWest*.

Fukushima, M., Hasegawa, T., Hasegawa, T., and Nakao, A. (2011). Minimum Disclosure Routing for network virtualization. In *Proc. of* 14th *Global Internet Symposium (GI) 2011 at IEEE INFOCOM 2011*.

Hao, J. and Cai, W. (2011). Trusted Block as a Service: Towards Sensitive Applications on the Cloud. In *Trust, Security and Privacy in Computing and Communications (TrustCom), Proc. of* 10th *Int. Conf. on*, pages 73–82.

Keller, E., Szefer, J., Rexford, J., and Lee, R. B. (2010). Nohype: virtualized cloud infrastructure without the virtualization. *SIGARCH Comput. Archit. News*, 38(3):350–361.

Lie, D., Thekkath, C. A., Mitchell, M., Lincoln, P., Boneh, D., Mitchell, J. C., and Horowitz, M. (2000). Architectural Support for Copy and Tamper Resistant Software. In *ASPLOS*, pages 168–177. ACM Press.

Liedtke, J. (1995). On micro-kernel construction. *SIGOPS Oper. Syst. Rev.*, 29(5):237–250.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74.

NFV Industry Specif. Group (2012). Network Functions Virtualisation. Introductory White Paper.

O. M. E. Committee (2012). Software-defined Networking: The New Norm for Networks. Open Networking Foundation.

Popa, R. A., Redfield, C. M. S., Zeldovich, N., and Balakrishnan, H. (2011). CryptDB: protecting confidentiality with encrypted query processing. In *Proc. of the* 23rd *ACM Symp. on Operating Systems Principles*, SOSP '11, pages 85–100.

Rashid, R. F. (1986). From RIG to Accent to Mach: the evolution of a network operating system. In *Proc. of 1986 ACM Fall joint computer conference*, ACM '86, pages 1128–1137.

Sachs, J. and Baucke, S. (2008). Virtual radio: a framework for configurable radio networks. In *Proceedings of the* 4th *Annual International Conference on Wireless Internet*, WICON '08, pages 61:1–61:7.

Singaravelu, L., Pu, C., Härtig, H., and Helmuth, C. (2006). Reducing TCB complexity for security-sensitive applications: three case studies. *SIGOPS Oper. Syst. Rev.*, 40(4):161–174.

Steinberg, U. and Kauer, B. (2010). NOVA: a microhypervisor-based secure virtualization architecture. In *Proc. of the* 5th *European Conf. on Computer systems*, EuroSys '10. ACM.

Suh, G. E., Clarke, D., Gassend, B., Dijk, M. v., and Devadas, S. (2003a). Efficient Memory Integrity Verification and Encryption for Secure Processors. In *Proc. of the* 36th *annual IEEE/ACM Int. Symp. on Microarchitecture*, MICRO 36, Washington, DC, USA. IEEE Computer Society.

Suh, G. E., Clarke, D., Gassend, B., van Dijk, M., and Devadas, S. (2003b). AEGIS: architecture for tamper-evident and tamper-resistant processing. In *ICS '03: Proc. of the* 17th *annual Int. Conf. on Supercomputing*, New York, NY, USA. ACM.

Szefer, J., Keller, E., Lee, R. B., and Rexford, J. (2011a). Eliminating the Hypervisor Attack Surface for a More Secure Cloud. In *Proc. of CCS 2011*, Chicago, Illinois, USA.

Szefer, J., Zhang, W., Chen, Y.-Y., Champagne, D., Chan, K., Li, W. X. Y., Cheung, R. C. C., and Lee, R. B. (2011b). Rapid single-chip secure processor prototyping on the OpenSPARC FPGA platform. In *Int. Symp. on Rapid System Prototyping*, pages 38–44.

Uhlig, R., Neiger, G., Rodgers, D., Santoni, A. L., Martins, F. C. M., Anderson, A. V., Bennett, S. M., Kagi, A., Leung, F. H., and Smith, L. (2005). Intel Virtualization Technology. *Computer*, 38(5):48–56.

Yang, J., Zhang, Y., and Gao, L. (2003). Fast Secure Processor for Inhibiting Software Piracy and Tampering. In *Proc. of the* 36th *annual IEEE/ACM Int. Symp.on Microarchitecture*, MICRO 36, pages 351–, Washington, DC, USA. IEEE Computer Society.

Key Completion Indicators

Minimizing the Effect of DoS Attacks on Elastic Cloud-based Applications Based on Application-level Markov Chain Checkpoints

George Kousiouris
Dept. Of Electrical and Computer Engineering, National Technical University of Athens,
9 Heroon Polytechnioy Str, 15773, Athens, Greece
gkousiou@mail.ntua.gr

Keywords: Cloud Computing, elasticity, DoS,eDoS, Markov Chains, DoS Identification as a Service

Abstract: The problem of DoS attacks has significant effects for any computing system available through the public domain. In the case of Clouds, it becomes even more critical since elasticity policies tied with metrics like Key Performance Indicators (KPIs) can lead a Cloud adopter to significant monetary losses. DoS attacks increase the KPIs, which in turn trigger the elastic increase of resources but without financial benefit for the owner of the cloud-enabled application (Economic Denial of Sustainability). Given the numerous scenarios of DoS attacks and the nature of services computing (in many cases involving legitimate automated traffic requests and bursts), networking mitigation approaches may not be sufficient. In this paper, the concept of Key Completion Indicators (KCIs) is provided and an analysis framework based on a probabilistic approach is proposed that can be applied on the application layer in cloud-deployed applications and elasticity policies, in order to avoid the aforementioned situation. KCIs indicate the level of completeness and provided revenue of the requests made towards a publicly available service and together with the KPIs can lead to a safer result with regard to elasticity. An initial architecture of this DoS Identification as a Service is portrayed.

1 INTRODUCTION

In the current IT world, Cloud computing is considered as a revolutionary way to provide software, platform and computational resources as services (SaaS, PaaS, IaaS) in order to reduce IT costs and capital investments for enterprises. The outsourcing of their applications on these infrastructures leads to much simpler management, increased availability and what is more important, elasticity in the needed resources, especially in the case of IaaS/PaaS. Companies do not have to cater for the worst case scenario regarding the usage of their resources but just toggle the amount of reserved virtual servers in a pay-per-use model, following their dynamic need.

In order to identify when this increase/decrease in resources is needed, Key Performance Indicators are used together with service offerings to enable this like AWS Auto Scaling. An example of a KPI may be the response time of a web server to the external requests made by the clients. If this time is over a specific limit, more virtual machines may be deployed in a load balancing way to compensate for the increase in traffic. Due to the fact that Cloud resources are vast, a service can scale up to thousands of servers. This has an effect of course on the overall cost for the owner of the Cloud-based application, but given that along with it the revenues from the usage increase, they are able to make more profit.

The problem in this process is when an application that is available to external clients through the public domain becomes a target for Denial-of-Service (DoS) or Distributed (DDoS) attacks. In that case, the requests generated towards the server do not generate profit for the owner but due to the elasticity policies they increase the cost of deployment, thus leading them to monetary loss and eventually bankruptcy, also known as Economic Denial of Sustainability (Kumar,2012). Due to the severity of the result, not a single false negative detection may be afforded or we must resort to more hard-core solutions, such as budget-based boundaries on elasticity. However the latter greatly cancels the main benefit of Clouds, seemingly

infinite resources to meet demand peaks.

In this paper, we present a design approach that has as a major goal to identify a DoS/DDoS attack against Cloud-enabled elastic applications. This identification must be applied above the networking layer, in order to avoid false negatives, enable web service-based automated implementations or different networking setups and must be based on the single most definitive criterion: "**Is the noticed burst of requests generating the anticipated profit**?"

In order to do so, we define **Key Completion Indicators (KCIs)**. KCIs are checkpoints introduced inside a cloud-enabled application and indicate the level of completion of the external user requests in the application value chain. Thus, KCIs can indicate the gradual fulfilment of the path towards revenue/value creation. The main argument is that this progress and eventual revenue creation is the single most definitive criterion to identify whether a set of requests (despite their origins) corresponds to a DoS/DDoS attack or a legitimate traffic burst, above and beyond any metric based on networking aspects or usage patterns. This is discussed in detail in the Related Work section. The framework may be integrated with an application through API calls to an external service for the maintenance of the KCIs state.

Each request, as it passes through the various stages of the process that generates revenue, raises the KCIs, thus indicating that it is a legitimate transaction. The approach is based on the assumption that a legitimate user will advance through the various stages of the application, until the point where he/she will produce revenue for the application owner (e.g. credit card payment). An illegitimate user or automated bot will be restricted to a specific area of the application, without completing the full application lifecycle.

Based on the level of these KCIs and the proportion of seemingly legitimate requests over the overall ones, a decision can be made as to whether the owner should enable the elasticity policies or not. This of course does not alleviate from the pain of a DoS attack but the scalability effect on the cost will be avoided. The remainder of the paper is structured as follows. In Section 2, similar approaches in the related field are presented, while in Section 3 an analysis is made on the concept of the approach. Section 4 presents requirements necessary for the implementation of the approach while Section 5 provides the overall conclusions from this study and intentions for the future.

2 RELATED WORK

Elasticity is one of the main benefits of Cloud computing. (Kranas et al, 2012) indicate the usage of this feature in a service oriented framework manner, to enable applications to harvest the benefits of it. From a networking point of view, numerous works exist for identifying patterns that are indicative of a DoS attack. CPM (Wang,2004) uses statistical and time series analysis at the network protocol level (e.g. SYN flooding attack detection) in order to abstract from application behaviour. (Kumar et al., 2011) uses neural network classifiers in order to filter traffic messages that are identified as DDoS packets. The characteristics that are taken under consideration include network level attributes such as UDP echo packets, number of connections with SYN errors, type of service etc. (Wang et al., 2011) uses a fuzzy logic based system in order to evaluate the infection of domain names and IP addresses. (Ahmed et al., 2010) use an IP-based approach in order to detect suspicious addresses and the change in the traffic arrival rate.

In a different approach, QoSSoD (Mailloux et al., 2008) caches incoming requests at a proxy and valuates each request. Requests are then scheduled for execution based on their perceived cost or threat. Usage patterns are collected over time and provide a baseline to compare current request behaviour against nominal behaviour.

In (Pinzon,2010) a different approach is presented in order to obtain time bounded Case-Based reasoning conclusions for attacks on SOAP-based web services, based on classifiers and SOAP specific rules for determining whether a set of requests can be categorized as malicious. (Yang et al, 2008) investigate a credit model and flow control policy for minimizing effects of DDoS attacks on P2P systems including malicious nodes. The most similar to our work is (Cheng et al, 2003), in which an application level approach is considered that utilizes specific API injection calls in the code in order to check common rules regarding aspects of DoS attack requests. However this approach also needs detailed knowledge on the types of attacks and their specificities.

An extensive survey on EDoS attacks and countermeasures can be found in (Sandar and Senai, 2012). The main problem with networking approaches is the fact that in many cases false positives or negatives may influence the decision process. For example, corporate gateways that mask all their traffic to be seen as one IP may be mistakenly interpreted for DoS attacks, if their

employees make a burst of requests towards an external cloud-based application. Furthermore, usually malicious approaches may take advantage of various aspects of networking protocols and defensive tools may focus on only a part of them. Moreover, authentication mechanisms may not always be relevant to a specific application deployed on a cloud infrastructure or may hinder the usability of an application by legitimate bots in a more automated application design/usage or web service system.

So in essence, what is needed is a holistic approach that will abstract from the networking layer and will focus only on the global aspect of revenue creation.

3 KEY COMPLETION INDICATORS APPROACH

As seen in the Introduction and Related work section, modern elasticity approaches allow for a feedback loop between the application and the IaaS provider or incorporate an internal mechanism in the applications to automatically trigger the elasticity rules based on the performance metrics. However, in a classic DoS attack, this means that the KPIs of the application will deteriorate, thus trigger the enactment of elasticity. Then, new resources will be deployed, in an attempt to mitigate this effect, driving the cost of the service very high and without getting any revenue due to the false requests.

In order to avoid this situation, we propose the Key Completion Indicators metric to be inserted in the application source code. These API calls may be generic and may only indicate the increase of a global set of counters, external to the program, each of which indicates a part in the process. These multi-level counters are increased each time a request passes through a specific part of the process(Figure 1). By having at each time the amount of requests in each stage we can apply macroscopic statistical analysis compared to a known interval of known operation.

The reason for having multiple levels of KCIs is to reduce the reaction interval. If there is only one KCI in the end of the process, then the comparison must be made in the end of the average time it takes a user to finish the entire process. By breaking the sequence of actions into more elementary ones we can have multiple levels of completeness and thus identify potential threats sooner.

An example of such an implementation could be a multi-page e-shop. Users navigate through the pages, e.g. entering keywords, surfing through catalogues, selecting products, adding to carts and eventually paying for the kart contents. Thus during this lifecycle they create revenue for the e-shop, mainly after the payment step is completed.

Due to the fact that this is in essence a probabilistic process, a similar approach must be followed. Not all users will eventually buy products (they might just have a look on available products) and not all users spend the same time accessing the intermediate steps (some may be more cautious or selective with regard to their added products). However, from a macroscopic point of view, the percentage of users being in each step can be averaged and used in a probabilistic approach. Thus we may conclude that the amount of time spent by people in the selection of products is a cumulative distribution that follows a specific pattern as time proceeds.

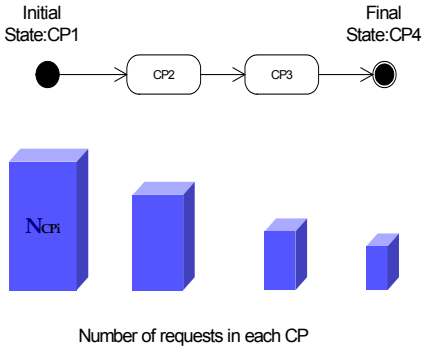


Figure 1: Checkpoints (CPs) through which a request passes until revenue is created in CP4. At each given time, different numbers of requests are in different levels of completion (number of CPs passed).

In order to identify whether a DoS attack is in progress, we can compare the KCI levels of this time interval against the ones from a known normal operation interval. For having the correct coefficients that reflect the normal operation under genuine load, a testing period may be considered, in order to observe application and user behaviour under normal circumstances. By determining the 95% confidence interval of the different KCI levels oscillation, it can be determined during runtime if their observed levels are within the specified intervals.

One problem with this approach is that KCIs will be low also in case of extensive legitimate traffic, due to the low response of the service. We can avoid this false alarm, if we originally enable elasticity for a specific time period (so that the load of the service

is distributed) and then measure the KCI levels. If they do not reach normal levels soon, then the traffic can be considered as illegitimate.

The state diagram for a request across the e-shop appears in Figure 2. A user may navigate through the different pages of the site and these transitions maybe modelled with a probability, as in the case of Markov chains. However, given that this is a stochastic process, the transition probability varies with time, in order to depict different user interactions with the pages (some users may need more than one products, others may be performing other tasks in parallel etc.). Thus, with the use of this method we can determine what would be the probability after a given time for the requests to have reached the intermediate or final stages of the value chain and gradually monitor the anticipated revenue.

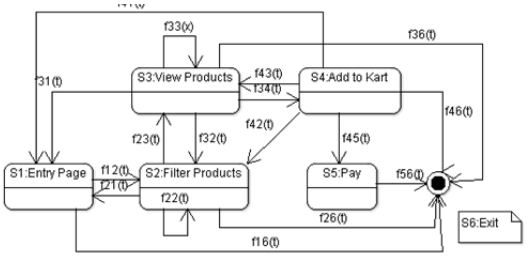


Figure 2: State diagram representing the possible states of a request across the e-shop.

4 APPLICATION AND MODELLING REQUIREMENTS

4.1 Logging Requirements

In order to obtain the necessary data regarding transition statistics, an application should enforce state transition logging. This logging should reflect only the states based on which the Markov model has been designed and it should also support documentation of the source and destination state (e.g. S1->S2, timestamp). The information should be forwarded towards global (to the application threads) counters that will hold the statistics for each transition at any given time.

It is not necessary to have request IDs or similar identifiers since we are not interested in single requests but rather on the macroscopic characteristics in order to detect a potential DoS attack. However it could be helpful to log for example IP addresses in case other countermeasures are jointly applied, in order to reason, isolate and drop packets coming from specific IP addresses that have been identified as malicious through the process. IP address logging could also be helpful during the normal operation to identify a user's path in the e-shop and aid in the modeling analysis.

4.2 Markov Chain Modeling Requirements

In order to create the MC model, we can implement the following process:

- **Step1:** Filter the log files of a known normal operation period in order to extract the probabilities of transitions for each transition case
- **Step2:** Calculate the probability for transitioning from one state to another This would create an array of probability values (Table 1) for the MC analysis that can indicate how many steps are in general necessary (or probable) for a request.
- **Step 3:** Filter the logs of a known normal operation period in order to acquire all the measured timings for a given transition from one state to another
- **Step 4:** Extract the cumulative probability distribution for that given transition and fit a common distribution type (e.g. exponential, normal etc.). This will indicate the probability for a request to go from state A to state B after a specific time interval (Figure 3). By taking the measurements from the normal period of execution we can conclude the time interval after which a transition should have occurred (with a 95% probability).
- **Step 5:** By combining the information from steps 2 and 4, we can macroscopically determine how many requests should have arrived at the final (or even some intermediate) state after a given time interval, based on the initial conditions (e.g. number of pending requests at each stage). This expected value can be compared with the monitored value (potentially incorporating the confidence intervals) in order to conclude whether these requests generate value for the application owner.

It is necessary to stress that steps 1-4, which can be considered in general as time consuming, are needed only in the model preparation phase and not during runtime. Once the model is complete and online, during runtime, only the counters (or KCIs) levels would be needed and probably the calculation of the anticipated value, which is a minor computational task.

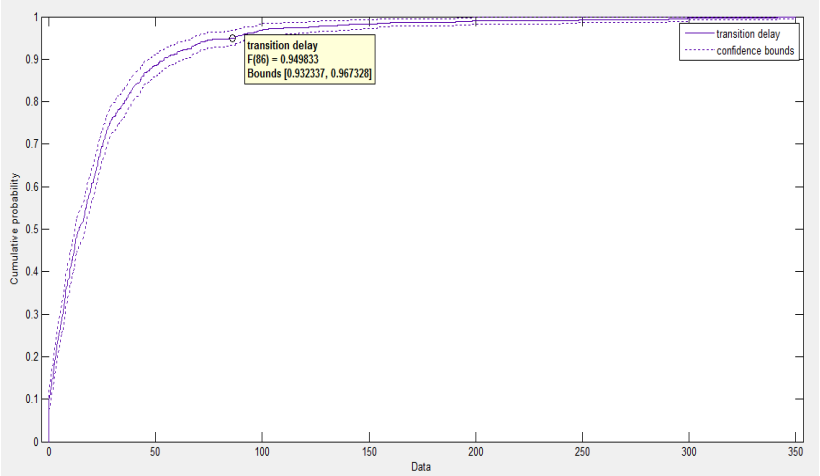


Figure 3: Cumulative Distribution Function of the delay of transition from one given state to another given state (x axis in seconds, y axis the probability for the specific transition to occur within the interval). In the specific example, the 95% probability for this to occur is within 86 seconds.

Table 1: Matrix describing the probability from a transient state to another transient state. From this matrix we can calculate how many steps are probable for a request.

States	S1	S2	S3	S4	S5	S6
S1	0	P_{12}	0	0	0	P_{16}
S2	P_{21}	P_{22}	P_{23}	0	0	P_{26}
S3	P_{31}	P_{32}	P_{33}	P_{34}	0	P_{36}
S4	P_{41}	P_{42}	P_{43}	0	P_{45}	P_{46}
S5	0	0	0	0	0	1
S6	0	0	0	0	0	1

In case one needs to **simplify** this process, then we can reduce the analysis to the absolutely necessary states, which means one state for all cases of surfing through the e-shop and one state that is described as the **"key" decisive state,** the state that generates value. In the aforementioned example the decisive state is S5, in which the user performs the payment. This would however delay the identification process since the interval after which we could check the KCIs level would include all intermediate steps.

4.3 Application Elasticity Logic and Overall Architecture

As mentioned in Chapter 3, the application should enforce elasticity directly in case of a request surge, so that limited resources do not hinder with the progress of the requests and thus contaminate the analysis. For example, if the surge makes the application unresponsive due to scarce resources, valid requests would not be able to reach the end of the process and thus lead to a false positive detection. Thus an elasticity logic should apply the pseudo-code in Figure 4. The sleep interval can be calculated through the model described in Section 4.2 and represent the time after which we would expect requests to have reached the end of the process. The overall system architecture appears in Figure 5.

```
Public class eDoSDetector extends
Thread
.
.
If (elasticity_flag=true)then
    Enable elasticity
    Thread.sleep(interval)
    Check KCIs status
      Run MC model to get anticipated
      KCI values for this interval
      Difference=Predicted-Actual
      If (difference><confidence_limit)
      then
        Disable elasticity
      Else
        Keep elasticity enabled
```

Figure 4: eDoS detector pseudo-code structure.

An interested application will have to inject code at the necessary points, before migrating to the Cloud, that indicate a change of state. That code will be responsible for contacting a web service (DIaaS) to increase the respective KCIs. The probabilistic analyser will have access to the KCI levels and will have created and stored in advance the MC model. Based on the runtime analysis, it may inform the application part (or in a more independent way of communication, the application will query the analyser) that is responsible for enabling elasticity the outcome of the analysis.

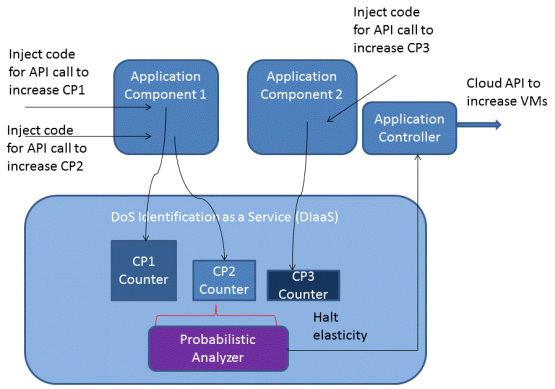


Figure 5: Overall Concept Architecture: the proposed mechanism is offered as a service (DoS Identification as a Service).

5 CONCLUSIONS

DoS attacks on cloud-based applications are more dangerous since they might lead to an unsustainable service and eventual bankruptcy. In order to mitigate this effect and efficiently identifying a DoS attack despite traffic patterns or networking aspects, the Key Completion Indicators concept is proposed in order to model and analyse the transition of requests along the application value chain. The KCIs are accompanied by a probabilistic modelling approach in order to identify whether a surge of traffic is legitimate or not, based only on the generation of wealth by the application. It must be stressed that the approach is for detecting a DoS attack and mitigates only the effect of eDoS, by cutting off the elasticity actions and does not help restore the performance of the server. It is also applicable for DDoS cases, since the analysis is made on the application states transition and not the source of requests.

Given that the approach requires an intervention at the source code level and a similar framework, it can be pursued by PaaS providers as a service offering that ensures the application from such types of attacks. Standard PaaS offering such as Google App Engine already need a certain alteration/adaptation of the source code to the development framework they are using, so it would be easier at this level to support also the concepts presented in this paper. The simplification suggested in Section 4 is also interesting to be investigated in order to minimize the analysis effort but also a potential implementation overhead of the mechanism.

ACKNOWLEDGEMENTS

The research leading to these results is partially supported by the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 317859, in the context of the ARTIST Project.

REFERENCES

Naresh Kumar, M.; Sujatha, P.; Kalva, V.; Nagori, R.; Katukojwala, A. K.; Kumar, M., "Mitigating Economic Denial of Sustainability (EDoS) in Cloud Computing Using In-cloud Scrubber Service," *Computational Intelligence and Communication Networks (CICN), 2012 Fourth International Conference on , vol., no., pp.535,539, 3-5 Nov. 2012.*

Haining Wang, Danlu Zhang, Kang G. Shin, "Change-Point Monitoring for the Detection of DoS Attacks," *IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 4, pp. 193-208, Oct.-Dec. 2004.*

P. Arun Raj Kumar, S. Selvakumar, Distributed denial of service attack detection using an ensemble of neural classifier, *Computer Communications, Volume 34, Issue 11, 15 July 2011, Pages 1328-1341, ISSN 0140-3664.*

Kuochen Wang, Chun-Ying Huang, Shang-Jyh Lin, Ying-Dar Lin, A fuzzy pattern-based filtering algorithm for botnet detection, *Computer Networks, Volume 55, Issue 15, 27 October 2011, Pages 3275-3286, ISSN 1389-1286.*

E. Ahmed, G. Mohay, A. Tickle, and S. Bhatia, "Use of ip addresses for high rate flooding attack detection*," in Proceedings of 25th International Information Security Conference (SEC 2010) : Security & Privacy : Silver Linings in the Cloud, Brisbane, Australia.*

M. Mailloux, H. Naim, T. Wayne, "Application Layer and Operating System Collaboration to Improve QoS against DDoS Attack*", Available at: https://wiki.engr.illinois.edu/download/attachments/68780072/QoSSoD.pdf?version=2&modificationDate=1210044601000.*

Pinzón, C., De Paz, J. F., Zato, C., Pérez, J.: Protecting Web Services against DoS Attacks: A Case-Based Reasoning Approach. In: *Graña Romay, M., Corchado, E., Garcia Sebastian, M.T. (eds.) HAIS 2010. LNCS, vol. 6076, pp. 229–236. Springer, Heidelberg (2010).*

Kranas, P.; Anagnostopoulos, V.; Menychtas, A.; Varvarigou, T., "ElaaS: An Innovative Elasticity as a Service Framework for Dynamic Management across the Cloud Stack Layers," *Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference on* , vol., no., pp.1042,1049, 4-6 July 2012.

Yang J., Li Y., Huang B., Ming J. (2008) Preventing DDoS attacks based on credit model for P2P

streaming system. In*: ATC '08: Proc of the 5th International Conference on Autonomic and Trusted Computing. Springer, Berlin, pp 13–20.*

Sandar S. V., Shenai S. Economic denial of sustainability (EDoS) in cloud services using HTTP and XML based DDoS attacks. *International Journal of Computer Applications 2012;41(20):11–6.*

Cheng Jin, Haining Wang, and Kang G. Shin. 2003. Hop-count filtering: an effective defense against spoofed DDoS traffic. *In Proceedings of the 10th ACM conference on Computer and communications security (CCS '03). ACM, New York, NY, USA, 30-41.*

MOBILE CLOUD COMPUTING AND SERVICES

SHORT PAPERS

Utilizing ICN/CCN for Service and VM Migration Support in Virtualized LTE Systems

Morteza Karimzadeh, Triadimas Satria and Georgios Karagiannis
Department of Computer Science and the Electrical Engineering, University of Twente, Enschede, The Netherlands
{m.karimzadeh, g.karagiannis}@utwente.nl, t.a.satria@student.utwente.nl

Keywords: Virtualized LTE, Service Migration, VM Migration, Service Continuity, ICN/CCN.

Abstract: One of the most important concepts used in mobile networks, like LTE (*Long Term Evolution*) is service continuity. A mobile user moving from one network to another network should not lose an on-going service. In cloud-based (*virtualized*) LTE systems, services are hosted on Virtual Machines (*VMs*) that can be moved and migrated across multiple networks to such locations where these services can be well delivered to mobile users. The migration of the (1) VMs and (2) the services running on such VMs, should happen in such a way that the disruption of an on-going service is minimized. In this paper we argue that a technology that can efficiently be used for supporting service and VM migration is the ICN/CCN (*Information Centric Networking / Content Centric Networking*) technology.

1 INTRODUCTION

Long Term Evolution (*LTE*) is the fourth generation (*4G*) technology, which is standardized by the 3rd Generation Partnership Project (*3GPP*). It is capable of providing high data rates as well as support of high-speed mobility. LTE features low latency in both the control plane and the user plane, which creates new opportunities for real-time applications such as video surveillance, telemedicine, and distance learning. In the LTE system two main network parts can be identified which are called Evolved UMTS Terrestrial Radio Access Network (*e-UTRAN*) and the Evolved Packet Core (*EPC*). The e-UTRAN consists of base stations denoted as Evolved Node-Bs (*eNodeBs*). The EPC is composed of several network elements. The main important ones are the Serving Gateway (*S-GW*), the Packet Data Network Gateway (*P-GW*) and the Mobility Management Entity (*MME*). The P-GW, that is the main mobility EPC anchor point, connects the EPC to other external networks. The S-GW supports the transport of the user data between the User Equipment (*UE*) and the external networks. The MME is the control node that processes the mobility management signalling (*i.e.* handover) between the *UE* and the EPC. Even though LTE promises a faster and more efficient data network, its architecture is still highly centralized, thus it may lead to very high bandwidth requirements on core network equipments. Long communication paths between users and servers can increase delay and waste network resources. The Mobile Cloud Networking (*MCN*) project (EU FP7 MCN, 2013), as one of the EU FP7 projects, integrates the use of cloud computing concepts in LTE mobile networks. This is accomplished with the objective of increasing LTE's performance by building a shared distributed LTE mobile network that can: (1) optimize the utilization of computation, storage and networking resources, (2) minimize communication delays, (3) avoid bottlenecks, and (4) enable multiple network operators to create their own virtual network depending on their requirements and goals, while using a common physical infrastructure. The integration of cloud computing concepts in an LTE system, see (EU FP7 MCN, 2013), can be realized by: (1) extending the cloud computing concept beyond the typical (*macro*) data centers towards new smaller (*micro*) data centers that are distributed within the Radio Access Network (*e.g.,* e-UTRAN) and the Mobile Core Network (*e.g.,* EPC), see Figure 1, and (2) deploying and running cloud-based (*virtualized*) Radio Access Networks, denoted as RAN as a Service (*RANaaS*), and Mobile Core Networks, defined as EPC as a Service (*EPCaaS*). This trend is also in line with the emerging ETSI activities in Network Functions Virtualization (*NFV*).

Service continuity is a critical issue in mobile networks implying access to the requested services without disruption, while the user moves from one network to another.

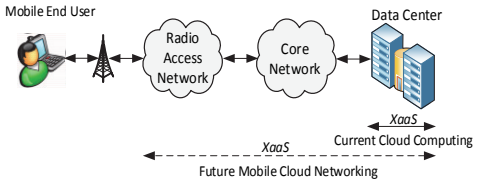


Figure 1: A view of future LTE mobile cellular network.

As illustrated in Figure 2, in cloud based mobile cellular network systems services are hosted on VMs that may migrate across multiple physical networks (*i.e.* datacentres) with the aim of better service delivery. A service could be a content delivery or content generation and manipulation service. The migration of VMs and the services running on such VMs should occur in such a way that the disruption of an on-going service is minimized.

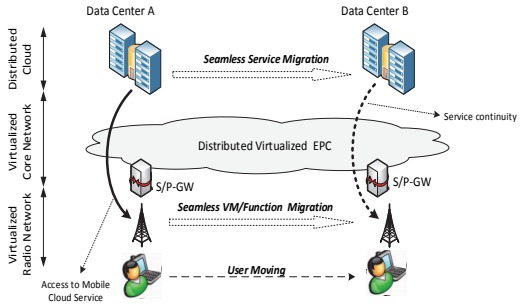


Figure 2: Service continuity in the virtualized LTE system.

A service continuity solution should be able to support the migration of services, implying the support for:

- **IP Address Continuity:** When a user moves to another sub-network, the application will not observe the change of the IP address.
- **Session Continuity:** It is a combination of IP address continuity and service context migration. Service context migration occurs when a user moves to a new location and the service context used by the function in the previous location should be able to be migrated and used by the same function at the new location.
- **Content Continuity:** Refers to migration and delivery of the requested content from a location close to the mobile user. The requested content can be migrated and delivered from a location close to a mobile user.
- **Storage Continuity:** Storage should be able to migrate to a new location close to the mobile user.
- **Function Continuity:** The same function in a new location can be run using context used by the same function in the previous location. Function migration needs to be supported in order to maintain function continuity.

Several technologies can be considered as possible candidates for the support of service and VM migration in virtualized LTE systems. For example, live VM migration solutions have been proposed in the literature, see *e.g.,* (Mandal et al., 2013), but none of these can be used efficiently to support the service continuity requirements listed above.

Moreover, in this paper we argue that the best candidate technology that can efficiently be used for the support of service and VM migration in virtualized LTE systems is the Information Centric Networking/Content Centric Networking (*ICN/CCN*). In particular, this paper answers the following research question:

> *"Can the ICN/CCN technology be used efficiently for the support of service and VM migration in the virtualized LTE?"*

This paper is organized as follows. Section 2 provides an overview of the ICN/CCN concept and explains how it could be exploited to support service continuity in virtualized LTE systems. Section 3, briefly introduces other possible candidate technologies and compares them with the ICN/CCN approach. Section 4 explains by using an example how the ICN/CCN concept can be applied in the virtualized LTE system. Moreover, Sections 3 and 4 are answering the research question listed above. Finally, Section 5 concludes the article and provides recommendations for future work.

2 ICN/CCN

ICN is an Internet architecture approach based on Named Data Objects (*NDOs*), see *e.g.*, (Ahlgren et al., 2012), (Pentikousis et al., 2013). It changes the focal point of the network architecture from the "end host" to "information" (*content or data*). The ICN architecture leverages in-network storage for caching, multiparty communication through replication, and interaction models decoupling senders and receivers (Koponen et al., 2007). The NDO, such as web page, video, document, or other kind of information, is independent of the location, the storage method, the application program, and the

transportation method. A unique name for each NDO is required in ICN in order to identify objects independently of their locations. Information about the source of an object is also useful to associate with the name. ICN has an API, which is used, as an interface for publishing or getting NDOs. By using the ICN API, a producer can publish NDOs to the Internet and a consumer can get them from internet.

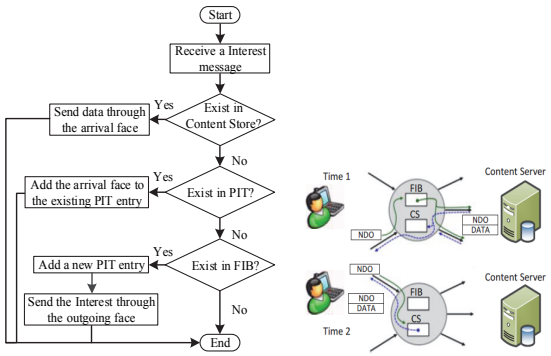
Advantages of the ICN approach could be summarized as follows (Koponen et al., 2007):

- The ICN approach offers scalable and cost-effective content distribution as it leverages in-network caching, so that requests for NDOs can be served by any network node holding a copy of the requested NDOs.
- It benefits from persistent and unique naming of NDOs, and also a service model that decouples senders and receivers, so that it does not have a problem with name-object binding.
- The ICN approach has an interesting security model. It provides name-data integrity and original verification of NDOs, independently of the immediate source. Hence, it enables ubiquitous caching with retained name-data integrity and authenticity.
- The ICN approach supports IP address continuity and multi-homing. A mobile client just needs to send requests for NDOs to a new access and the requests will be served by a network node that might be different from the previous network node.
- The ICN approach provides better reliability and performance compared to the current networks since it leverages optimized hop-by-hop transport and in-network caching.

Currently there are several ICN approaches that have been developed, such as Data-Oriented Network Architecture (*DONA*) (Koponen et al., 2007), Content-Centric Networking (*CCN*) (Jacobson et al, 2009), Publish-Subscribe Internet Routing Paradigm (*PSIRP*) (Fotiou et al., 2012), Network of Information (*NetInf*) (Dannewitz et al., 2013) and Translating Relaying Internet Architecture integrating Active Directories (*TRIAD*) (Gritter and Cheriton, 2000). These ICN approaches undergo the lack of efficient support of session and function continuity. However, the Service-Centric Networking (*SCN*) (Braun et al., 2011) concept, which is a new ICN based concept is able to support session and function continuity. SCN is a new networking paradigm for the future Internet, in which routing and forwarding are based on service identifiers. SCN is an extension of CCN, see Section 2.1, which is designed based on an object-oriented approach, in which the contents and the services are considered as objects. In SCN, the content not only can be retrieved but also can be processed before being delivered to users. In SCN, services are represented as functions to be invoked by users. By using an object-oriented approach, both functions and data are integrated into objects. In SCN, clients can request for both services and contents by using object names.

2.1 CCN Concept

The transport protocol used in CCN (Jacobson et al., 2009) is called CCNx that is used to distribute information related to the location of NDOs published on nodes. There are two types of CCNx messages, namely the *Interest* message, which contains the request for an NDO and the *Data* message, which contains the response for an *Interest* message.



a. Interest message processing *b. Data forwarding*

Figure 3: CCN protocol Overview.

The CCNx protocol, see Figure 3, operates based on three main data structures as follows:

- **Content Store (*CS*):** CS represents a buffer memory used for data retrieval by prefix match lookup on names.
- **Forwarding Information Base (*FIB*):** FIB contains a list of entries with interfaces to where the *Interest* messages should be forwarded. Each entry in the FIB may point to multiple interfaces to where the *Interest* messages could be forwarded.
- **Pending Interest Table (*PIT*):** PIT is used to keep track of *Interest* messages forwarded upstream. It contains information of sources of unsatisfied *Interests*. Each entry in the PIT may point to multiple sources.

3 MOTIVATION: WHY ICN/CCN SHOULD BE USED FOR SERVICE AND VM MIGRATION SUPPORT IN VIRTUALIZED LTE SYSTEMS

This section introduces other possible candidate technologies for service and VM migration support and analyses and compares them with the ICN/CCN approach in order to verify whether they can be applied in a virtualised LTE system.

3.1 Mobile Content Distribution Networks (M-CDN)

The deployment and distribution of cache servers in the Internet, enables the distribution of load in serving the increasing number of requests of content, such as web content, video content or software.

Current caching products can be deployed solely outside a mobile operator's network, such as the solutions offered by Akamai (Nygren et al., 2010), or close to/at the mobile operator's gateways that support mobility anchor functionality. The aim of M-CDN, see *e.g.,* (Saroiu et al., 2002), is to integrate cache servers with the mobile operator's core network or even with its access and backhaul network in order to move these cache servers, as content sources, closer to mobile users.

3.2 Software-Defined Networking (SDN)

Open Networking Foundation (*ONF*) defined SDN as an emerging network architecture where network control is decoupled from forwarding and is directly programmable (ONF, April 2012). The SDN architecture supports network virtualization since the underlying network infrastructure can be abstracted from the applications and network services. It provides a new dynamic network architecture, which changes traditional network platforms into rich service-delivery platforms. In practice, SDN refers more broadly to logically centralized software control. The network appears to the applications and policy engines as a single logical switch since the network intelligence that maintains a global view of the network is logically centralized in software-based SDN controllers.

3.3 Host Identity Protocol (HIP)

HIP introduces a new layer between the network and transport layers, which maps the host identifiers to network addresses and vice versa (Moskowitz and Nikander, 2006). In the HIP architecture, the end-point names and locators, which are both represented by IP addresses in the current Internet architecture, are separated. IP addresses will act as locators, while the host identifiers take the role of end-point identifiers. The existence of Host Identity Layer ensures that the transport layer connections are no more bound to IP addresses, so that if the location of a host changes, the connections will not have to be broken.

3.4 SCTP with Dynamic Address Reconfiguration Extension

T. Dreibholz in (Dreibholz et al., 2003) proposed a new scheme for mobility management for IP-based networks relying on a transport protocol called Stream Control Transmission Protocol (*SCTP*), with the extension for dynamic address reconfiguration, and the Reliable Server Pooling (*RSerPool*) protocol suite. The proposed solution is transparent to application, and does not require changes in the network infrastructure. SCTP, specified in IETF RFC 4960, is a transport protocol that provides a more flexible data delivery by separating reliable transfer of messages between endpoints from the actual delivery to the user process. It supports multi-homed endpoints with more than one IP address, so that it provides an improved network-level fault tolerance.

3.5 Proxy Mobile Ipv6 (PMIpv6)

PMIPv6 is a network-based mobility management protocol (Gundavelli et al., 2008) based on the Mobile IPv6 concept, specified in IETF RFC 3775, which is designed to provide mobility management support to a mobile node without requiring participation of the mobile node in any IP mobility related signalling. There are two core functional entities in PMIPv6 used to track movements of mobile nodes and initiate the mobility signalling and set up the required routing state, namely Local Mobility Anchor (*LMA*) and Mobile Access Gateway (*MAG*). The LMA is the topological anchor point for the mobile node's home network prefixes used to maintain a reachability state of the mobile node, while the MAG is the entity used to perform mobility management for the node attached to its access link.

3.6 Identifier-Locator Network Protocol (ILNP)

Randall Atkinson in (Atkinson et al., 2009) proposed the ILNP, a naming architecture which addresses issues of mobility, multi-homing, and end-to-end IP security. It evolves the naming in the Internet by splitting an address into two different entities, as Identifier (*I*) used for end-to-end identity and Locator (*L*) used for routing and forwarding packets. It provides an integrated solution to the issues mentioned above without changing the core routing architecture, while offering incremental deploy-ability through backward compatibility with IPv6.

3.7 Analysis and Comparison

To support service continuity in virtualised LTE systems, the selected technology needs to support requirements, such as IP address continuity, session continuity, content continuity, storage continuity and function continuity, see Section 1. Table 1 analyses and compares ICN/CCN with other candidate technologies that were briefly described in the previous subsections, taking into account the requirements listed in Section 1.

Table 1: The list of solutions and their ability in supporting service continuity (S: Supported, NS: Not Supported).

Solutions / Approaches	IP Address Continuity	Session Continuity	Content Continuity	Storage Continuity	Function Continuity
ICN	S	S	S	S	S
M-CDN	S	S	S	S	NS
SDN	S	S	S	S	S
HIP	S	NS	NS	NS	NS
SCTP	S	NS	NS	NS	NS
PMIPv6	S	NS	NS	NS	NS
ILNP	S	NS	NS	NS	NS

Based on the analysis and comparison given in Table 1, it can be deduced that the ICN/CCN and SDN solutions can satisfy all requirements for service continuity. This also implies that the ICN/CCN technology is a promising candidate that can efficiently be used for the support of service and VM migration in the virtualized LTE system. The proposed ICN/CCN approach could also offer reliable, scalable and cost-effective content distribution by leveraging in-network caching. This solution may be integrated with the SDN approach, by utilizing OpenFlow switches in order to provide a dynamic network architecture that fosters network virtualization and can improve network scalability, manageability and agility. The centralized network state in the control layer provides simplicity and flexibility for network designers and network operators to configure and improve their networks using automated SDN programs. However, the deployment of SDN requires significant changes on the existing network infrastructure.

4 EXAMPLE OF INTEGRATING ICN/CCN IN A VIRTUALIZED LTE SYSTEM

This section provides an example on how the ICN/CCN concept can be applied in a virtualized LTE system. As shown in Figure 4, the assumption in this architecture is that the eNodeBs, S-GWs and P-GWs are CCN capable (i.e., *able to support the CCNx protocol*) and that the routers deployed in the infrastructure used to interconnect the RAN and EPC components are non CCN capable.

Furthermore, it is also considered that the Content Delivery Network (*CDN*) engine/ repository is CCN capable and could provide the requested content/data to the CCN infrastructure faster than the main server that stores the content. Utilizing the CDN repository could effectively accelerate the content delivery to users and reduce bandwidth consumption in the network.

In this figure Virtualization Controlling Platform (*VCP*) represents the cloud components, which carry out the signalling and configuration of the virtualized LTE elements (*e.g.,* eNodeBs, S-GWs and P-GWs) in order to support service continuity, see (EU FP7 MCN, 2013).

It is also considered that UEs are CCN capable, meaning that they are able to generate, send and receive the *Interest* and *Data* messages.

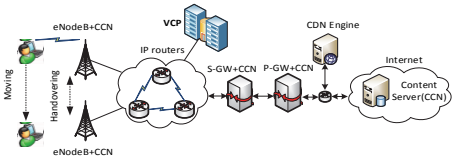


Figure 1: Integrating ICN/CCN in LTE system.

When a UE that uses both a service and a virtualized CCN process hosted on a (*source*) data centre, moves towards another (*destination*) data centre hostsing another virtualised CCN process, the

ICN/CCN approach can be used to migrate the on-going service from the *source* data centre to the new *destination* data centre. The ICN/CCN approach can also be used to migrate a VM running on the *source* data centre to the new *destination* data centre. The latter can be accomplished by enabling the *destination* data centre to request an on-going VM from the *source* data centre, by using one or more *Interest* messages.

5 CONCLUSIONS AND FUTURE WORK

By analyzing and comparing different technologies, this paper argued and verified that the ICN/CCN technology is a promising candidate that can efficiently be used for the support of service and VM migration in the virtualized LTE system.

In order to further validate this statement, the use of the ICN/CCN technology for the support of service and VM migration in the virtualized LTE system will be prototyped and evaluated within the context of the EU FP7 project (EU FP7 MCN, 2013).

ACKNOWLEDGEMENTS

This work is accomplished in the context of the EU FP7 Mobile Cloud Networking (*MCN*) project. Therefore, we would like to acknowledge the European Commission, since the MCN project is an EC funded Integrated Project under the 7th RTD Framework Programme, FP7-ICT-2011-8-grant agreement number 318109.

REFERENCES

EU FP7 MCN, (visited on September 2013),<http://www.mobile-cloud-networking.eu/site/>

Mandal, U., Habib, M., Shuqiang , Z., Mukherjee, B., Tornatore, M., 2013. Greening the cloud using renewable-energy-aware service migration, *IEEE Network,* vol. 27, Iss. 6. 36-43, IEEE.

Ahlgren, B., Dannewitz, C., Kutscher, D., Ohlman, B., July 2012. A survey of information-centric networking. In *IEEE Communication Magazine* vol. 50, pp. 26-36. IEEE.

Pentikousis, K., Ohlman, B., Corujo, D., Boggia, G., Tyson, G. Davies, E., Mahadevan, P., Spirou, S., Molinaro, A., Gellert, D., Eum, S., July 2013. *ICN* Baseline Scenarios and Evaluation Methodology. *IETF Internet draft (work in progress)*, IETF.

Koponen, T., Chawla, M., Chun, B.-G., 2007. A Data-Oriented (and Beyond) Network Architecture. In *Proc. of ACM SIGCOMM '07*, vol. 37, Iss. 3, pp. 181-192, ACM.

Jacobson, V., Smetters, D., Thornton, J., Plass, M., Briggs, N., Braynard, R., 2009. *Networking named content*. In *Proc. of 5th ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT'09)*, ACM.

Fotiou, N., Trossen, D., Polyzos, G.C., 2012. Illustrating a publish-subscribe Internet architecture. *Journal of Telecommunication Systems*, Vol. 51, Iss. 4, pp. 233-245, Springer.

Dannewitz, C., Kutscher, D., Ohlman, B., Farrell, S., Ahlgrene, B., Karla, H., 2013. Network of Information (NetInf)– An information-centric networking architecture. *Journal of Computer Communications*, Vol. 36, Iss. 7, pp. 721–735, Elsevier.

Gritter, M., Cheriton, D.R., Jan 2000. *TRIAD: A New Next-Generation Internet Architecture*, (visited in December 2013) < http://www-dsg.stanford. edu/triad/>.

Braun, T., Hilt, V., Hofmann, M., Rimac, I., Steiner, M., Varvello, M., 2011. Service-Centric Networking. In *Proc. of 4th International Workshop on the Network of the Future*.

Nygren, E., Sitaraman, R.K., Jennifer, S., 2010. The Akamai network: a platform for high-performance Internet applications. *ACM SIGOPS Operating Systems Review*, Vol. 44, No.3, ACM.

Saroiu, S., Gummadi, K.P., Dunn, R.J., Gribble, S.D., Levy, H.M., 2002. An Analysis of Internet Content Delivery Systems. In *Proc. of OSDI '02*.

ONF, April 2012. *White paper,. Open Networking Foundation*, (visited in December 2013) <https://www.Opennetworking.org/>.

Moskowitz, R., Nikander, P., 2006. *Host Identity Protocol (HIP) Architecture*. IETF RFC4423, IETF.

Dreibholz, T., Jungmaier, A., Tuxen, M., 2003. A New Scheme for IP-based Internet-Mobility. In *Proc. of the 28th Annual IEEE International Conference on Local Computer Networks (LCN '03), IEEE.*

Gundavelli, S., Leung, K., Devarapalli, V., Chowdhury, K., August 2008, *Proxy Mobile IPv6*, IETF RFC 5213, IETF.

Atkinson, R., Bhatti, S., Hailes, S., 2009. ILNP: Mobility, Multi-Homing, Localised Addressing and Security Through Naming. *Telecommunication Systems*, vol. 42, no. 3-4 pp. 273-291, Springer.

Applying SDN/OpenFlow in Virtualized LTE to Support Distributed Mobility Management (DMM)

Morteza Karimzadeh, Luca Valtulina and Georgios Karagiannis
Department of Computer Science and the Electrical Engineering, University of Twente, Enschede, The Netherlands
m.karimzadeh@utwente.nl, l.valtulina @student.utwente.nl, g.karagiannis@utwente.nl

Keywords: Virtualized LTE, DMM, IP Address Continuity, OpenFlow Switches.

Abstract: Distributed Mobility Management (*DMM*) is a mobility management solution, where the mobility anchors are distributed instead of being centralized. The use of DMM can be applied in cloud-based (*virtualized*) Long Term Evolution (*LTE*) mobile network environments to (1) provide session continuity to users across personal, local, and wide area networks without interruption and (2) support traffic redirection when a virtualized LTE entity like a virtualized Packet Data Network Gateway (*P-GW*) running on an virtualization platform is migrated to another virtualization platform and the on-going sessions supported by this P-GW need to be maintained. In this paper we argue that the enabling technology that can efficiently be used for supporting DMM in virtualized LTE systems is the Software Defined Networking (*SDN*)/OpenFlow technology.

1 INTRODUCTION

Long Term Evolution (*LTE*) is the fourth generation (*4G*) technology which is standardized by the 3rd Generation Partnership Project (*3GPP*), see *e.g.*, (3GPP Release 10, 2013). It is capable of providing high data rates as well as support of high speed mobility. In the LTE system two main network parts can be identified which are called Evolved UMTS Terrestrial Radio Access Network (*e-UTRAN*) and the Evolved Packet Core (*EPC*). The e-UTRAN consist of base stations that are denoted as Evolved Node-Bs (*eNodeBs*). Each of these eNodeBs are controlling different cells which provide radio coverage and connectivity between the User Equipment (*UE*) and the EPC. The EPC is composed of several network elements. The main important ones are the Serving Gateway (*S-GW*), the Packet Data Network Gateway (*P-GW*) and the Mobility Management Entity (*MME*). The P-GW, that is the main mobility EPC anchor point, connects the EPC to other external networks. A mobility anchor point is mainly in charge of mobility related user data forwarding. Moreover, the P-GW also performs various functions such as IP address/IP prefix allocation or policy control and charging. The S-GW supports the transport of the user data between the UE and the external networks. The MME is the control node that processes the mobility management signaling, (*i.e.,* handover) between the UE and the EPC.

Mobility management provides the mechanisms for maintaining active and seamless session continuity to users across personal, local, and wide area networks without interruption. Most of the current IP mobility solutions standardized by both IETF (*Internet Engineering Task Force*) and 3GPP rely on a centralized mobility anchor entity which is in charge of both mobility-related control plane and user data forwarding. The presence of this centralized network node makes mobility management prone to several problems and limitations such as: (1) suboptimal routing, (2) low scalability, (3) signaling overhead, (4) more complex network deployment, (5) security and reliability issues due to the existence of a potential single point of failure, and (6) a lack of granularity on the mobility management service, see *e.g.* (Bertin et al, 2009), (Chan et al, 2011). Currently, operators and research communities are investigating alternative mobility solutions that are more distributed in nature, by using distributed mobility anchors, denoted as Distributed Mobility Management (*DMM*). The use of DMM allows a cheaper and more efficient network deployment capable to meet their customer requirements. As shown in Figure 1, DMM implements a flatter system in which the mobility anchors are placed

closer to the user, distributing the control and data infrastructures among the entities located at the edge (*access*) of the network. DMM may be partially or fully distributed, where in the former the distribution scheme is applied only to the data plane while in the latter to both the data and control planes. It is important to notice that in the fully distributed approach data and control planes needs to be decoupled although they are both handled by the distributed anchor points.

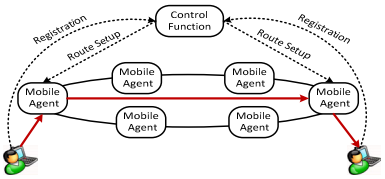


Figure 1: Generic DMM approach scheme.

The Mobile Cloud Networking (*MCN*) project 0(EU FP7 MCN, 2013), as one of the EU FP7 projects, integrates the use of cloud computing concepts in LTE mobile networks in order to increase LTE's performance. This is accomplished by building a shared distributed LTE mobile network that can optimize the utilization of virtualized computing, storage and network resources and minimize communication delays. In particular, the integration of cloud computing concepts in a LTE system, can be realized by: (1) extending the cloud computing concept beyond the typical (*macro*) data centers towards new smaller (*micro*) data centers that are distributed within the e-UTRAN and the EPC, and (2) deploying and running cloud-based (*virtualized*) e-UTRAN, denoted as RAN as a Service (*RANaaS*), and EPC, denoted as EPC as a Service (*EPCaaS*). This trend is also in line with the emerging ETSI activities in Network Functions Virtualization (NFV). The use of DMM can be applied in such environments not only to enhance the LTE mobility management performance and provide session continuity to users across personal, local, and wide area networks without interruption, but also to support traffic redirection when a virtualized LTE entity, like the P-GW running on an virtualization platform (*i.e.,* originating data centre) is migrated to another virtualization platform (*i.e.,* destination data centre) and ongoing sessions supported by this P-GW need to be maintained. In (Chan, 2013), the main IETF requirements for DMM solutions in IPv6 networks deployments are defined as follows:

- **Distributed Deployment:** IP address mobility and routing solutions provided by DMM must enable distributed processing for mobility management so that traffic does not need to traverse centrally deployed mobility anchors and thereby avoid non-optimal routes.
- **Transparency:** DMM solutions must provide transparent mobility support above the IP layer when needed**.**
- **IPv6 Deployment:** DMM solutions should target IPv6 as the primary deployment environment and should not be tailored specifically to support IPv4, in particular in situations where private IPv4 addresses and/or NATs (*Network Address Translations*) are used.
- **Co-existence:** The DMM solution must be able to co-exist with existing network deployments and end hosts. For instance, depending on the environment in which DMM is deployed, DMM solutions may need to be compatible with other deployed mobility protocols or may need to interoperate with a network or mobile hosts/ routers that do not support DMM protocols.
- **Security Considerations:** A DMM solution must not introduce new security risks or amplify existing security risks against which the existing security mechanisms/protocols cannot offer sufficient protection**.**
- **Flexible Multicast Distribution:** DMM should consider multicasting. So the solutions can be developed that, not only to provide IP mobility support when it is needed, but also to avoid network inefficiency issues in multicast traffic delivery (*e.g.,* duplicate multicast subscriptions towards the downstream tunnel entities).

In the context of this paper also the following additional requirements are defined:

- **Dynamicity:** The dynamic use of mobility support by allowing the split of data flows along different paths that may travel through either the mobility anchor or non-anchor nodes, even though no specific route optimization support is available at the correspondent node. This requirement will tackle the lack of fine granularity of the centralized mobility management approaches.
- **Separating Control and Data Planes:** Keeping the control plane centralized while distributing the data plane is a possible solution to minimize the signaling overhead between the mobility anchors due to the lack of knowledge that a distributed anchor point has of its peers and their connected UEs.
- **Network-based:** Not burdening the UE with extra signaling and keeping the user unaware of the on-going handoff procedure within the same

domain are fundamental aspects that need to be provided by the DMM solutions deployed in LTE networks.

Several DMM solutions have been proposed in the IETF and 3GPP contexts. However, it is not yet clear whether these DMM solutions can be applied in virtualized LTE network systems. In this paper we argue that the best candidate enabling technology that can efficiently be used for supporting DMM in virtualized LTE systems is the SDN/OpenFlow technology. In particular, this paper answers the following research question:

"Can SDN/OpenFlow be used efficiently for DMM support in virtualized LTE systems?"

This paper is organized as follows. Section 2 is providing a brief overview of the SDN/OpenFlow technology and explains how it could be exploited to support DMM in virtualized LTE systems. In Section 3 a brief introduction of other possible candidate technologies is provided. Moreover Section 3 analyses and compares these technologies with the SDN/OpenFlow approach. Section 4 shows in an example how the SDN/OpenFlow concept can be applied in the virtualized LTE system. Furthermore, Sections 3 and 4 are answering the research question listed above. Finally, Section 5 concludes and provides recommendations for future work.

2 SDN/OpenFlow

SDN (ONF, 2103) is an architecture that decouples forwarding functions and network control, which become directly programmable. This enables the underlying infrastructure to be abstracted for applications and network services. In particular, SDN capable switches and routers can be configured and programmed using a centralized management entity, denoted as SDN controller. Several SDN based protocols are being developed, such as the IETF FORCES (*Forwarding and Control Element Separation*) and OpenFlow. OpenFlow (OpenFlow, 2013) is the most commonly used SDN based protocol which carries signalling message between SDN controllers and the underlying network infrastructure, bringing network applications to life. With OpenFlow, the forwarding plane of a SDN/OpenFlow capable switch or router can be accessed over the network and reconfigured according to the needs of applications and network services. The vast majority of Ethernet switches and routers used nowadays contain flow-tables to implement firewall, NAT, QoS (*Quality of Service*) and other functionalities. A flow-table of SDN/OpenFlow capable switches or routers can be remotely programmed partitioning the network's traffic into separated flows. Features offered by OpenFlow protocol can be used to deploy a DMM solution offering IP address continuity and traffic redirection in the operator's transport network. This can be achieved by treating each traffic path from the Internet PoPs (*point of presence*) to the mobility anchor points (*e.g.,* P-GWs) as a separated flow. In this way traffic redirection can be supported without involving any IP address translation or modification, when for example a virtualized LTE entity, like the P-GW, running on an virtualization platform (i*.e.,* originating data centre) is migrated to another virtualization platform (*i.e.,* destination data centre) and ongoing sessions supported by this P-GW need to be maintained. Alternatively SDN/OpenFlow switches contain a list of actions that can be applied on every transiting packet that belongs to a specific flow. Example of these actions are: *Drop, Push-Tag, Pop-Tag, Group* and *Set-Field*. The optional *Set-Field* action is the most interesting for the purpose of this proposal providing to SDN/OpenFlow switches the possibility to modify headers of packets and frames, used by *e.g.,* Ethernet, VLAN, and IP. Both flow tables and action lists are added, modified or removed by the SDN/OpenFlow Controller which has a dedicated secure connection with each SDN/OpenFlow switches and routers. The procedures and messages used to support and perform such modifications are specified in the OpenFlow specification document 0.

3 MOTIVATION: WHY SDN/ OpenFlow SHOULD BE USED FOR DMM SUPPORT IN VIRTUALIZED LTE SYSTEMS

This section introduces other possible candidate DMM enabling technologies and analyses and compares them with the SDN/OpenFlow approach in order to verify whether they can be applied for DMM support in a virtualised LTE system.

3.1 IETF Based DMM Enabling Technologies

The IETF DMM working group charter addresses two complementary aspects of mobility management procedures: the distribution of mobility anchors

towards a more flat network and the dynamic activation/deactivation of mobility protocol support as an enabler to distributed mobility management (Chan, 2013). The following DMM solutions are specified within the context of IETF.

3.1.1 Double NAT (D-NAT)

Double NAT DMM solution proposed in (Liebsch, 2012) adopts the concept of an identifier-locator split to solve the routing in the transport network above the mobility anchors. Forwarding downlink packets to the mobile nodes's current mobility anchor can be achieved using tunnels as already done in both Mobile IP and Proxy Mobile IP (*PMIP*) solutions. To avoid encapsulation overhead introduced by tunnelling the use of NAT is proposed at both ends of the operator's transport network. Two new entities, performing address translation from identifier address to locator address and vice-versa, need to be introduced in the network. These entities are referred to as Ingress NAT router and Egress NAT router. Using NAT functionality is required only in the case of downlink traffic, where the Ingress NAT router performs translation of the identifier address into the locator address and it forwards the packets down into the operator's transport network. The Egress NAT router, on the other hand, translates the locator address back to the identifier address in order to forward the packet to the mobile node. The Egress NAT routers will therefore always be placed closer to the southern edge of the operator's transport network than the Ingress NAT routers.

3.1.2 Distributed Mobility Anchoring (DMA)

P. Seite (Seite et al, 2103) proposed a distributed mobility traffic management with dynamic user's traffic anchoring in the networks' access routers (*ARs*). It relies on a flat architecture where a new entity named Mobility capable Access Router (*MAR*) is introduced to provide mobility management functions. The MAR has both mobility anchoring and location update functional capabilities and can acts as a Home-MAR (*H-MAR*) or as a Visited-MAR (*V-MAR*) for a given mobile node. A H-MAR is responsible for the allocation of Home Network Prefix (*HNP*), used in this solution instead of HoA, to mobile node. On the one hand, when a mobile node moves away from the home network, the H-MAR is responsible for tracking the mobile node's location and forwarding packets to the V-MAR where the mobile node is currently attached to. On the other hand a V-MAR manages the mobility-related signalling for a mobile node that is attached to its access link. The architecture of this solution relies on a centralized database storing ongoing mobility sessions for the MNs.

3.1.3 Inter-domain DMM

J.C. Zuniga *et al.* in (Bernardos and Zuniga, 2103) proposed a method which aims to ensure session continuity in an inter-domain roaming scenario. It is based on the partial distributed single operator scenario that uses an entity called Distributed Gateway (*D-GW*) placed at the edge of the network. The D-GW supports two roles: Anchoring D-GW and Serving D-GW. The control plane relies on a central entity called Central Mobility Database (*CMD*). The inter-domain solution uses a centralized Local Mobility Anchor (*LMA*), usually located in the home domain, as top-level anchor to guarantee session continuity when crossing operator borders. It is assumed that the necessary roaming agreement are in place in order to support setting up tunnels between the LMA located at the home domain of the mobile node (*MN*) and the visited D-GWs, which in a 3GPP EPC scenario may correspond to the eNodeBs or Home eNodeBs (*HeNBs*) used for femtocells.

3.2 3GPP Based Solutions IETF Based DMM Enabling Technologies

Currently, in the 3GPP EPC architecture, the mobility management solutions mostly rely on a centralized mobility anchor entity, *i.e.*, P-GW, which is in charge of the control of the network entities involved in the mobility management and the user data forwarding. There are however, two solutions specified by 3GPP that are already introducing the concept of DMM into the 3GPP EPC architecture, *i.e.,* LIPA and SIPTO.

3.2.1 Local IP Access (LIPA) / Selected IP Traffic Offload (SIPTO)

LIPA and SIPTO 0 have been introduced by 3GPP in LTE Release 10. They enable data traffic offload at appropriate points in the Radio Access Network (*RAN*) in a highly cost-efficient manner leading to an increased system scalability and enhance the operators flexibility to cope with the growing mobile data traffic demanded. LIPA allows a UE, connected in a residential or corporate deployment via a HeNB, to directly connect to other devices and services in

the local network, relieving this portion of data traffic from the mobile operator's core network. LIPA breakout takes always place at the newly introduced Local GW entity located in the local/home or enterprise femtocell network. SIPTO, on the other hand, offloads selective IP traffic to the Internet at the local gateway (*L-GW*), similar to LIPA, or above HeNB such as the HeNB gateway located in home and enterprise networks. When the UE is connected to a macro-cellular network, SIPTO offload takes place at or above the RAN. By breaking out selected traffic closer to the edge of the network, operators may avoid overloading their scarce resources, *i.e.* P-GWs and S-GWs, as well as avoid inefficient routing in the mobile backhaul network.

3.3 Analysis and Comparison

In order to provide DMM support in virtualised LTE systems, the selected DMM enabling technology needs to support the DMM requirements listed is Section 1. Table 1 analyses and compares SDN/OpenFlow technology with the other candidate DMM enabling technologies that were briefly described in the previous subsections, by using the DMM requirements listed in Section 1.

Table 1: Comparison (Y:Yes, N:No, P: Partial, N.c:Not considered, N.s:Not specified, C:Considered, O.l:Only local, without support for IP address continuity).

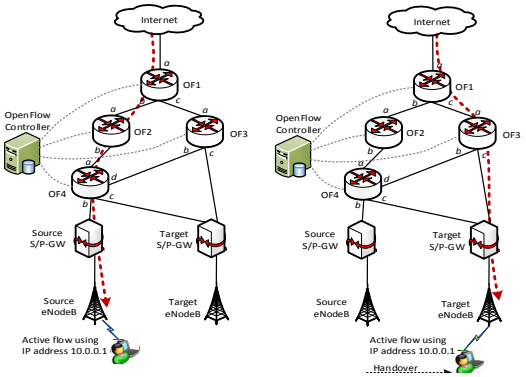
Requirements	SDN/ OpenFlow	D-NAT	DMA	Inter-domain DMM	LIPA/ SIPTO
Distributed deployment	Y	Y	Y	P	O.l
Transparency	Y	Y	Y	Y	O.l
IPv6 deployment	Y	Y	Y	Y	Y
Co-existence	N	N	P	Y	P
Security considerations	N.c	N.c	N.c	N.c	C
Flexible multicast distribution	Y	Y	N.s	N.s	N.s
Dynamicity	Y	N	Y	N	N
Separating control and data planes	Y	Y	Y	Y	Y
Network-based	Y	Y	Y	Y	Y

Based on the analysis and comparison given in Table 1, it can be deduced that the SDN/OpenFlow based DMM technology can satisfy most of the DMM requirements introduced in Section 1. The *Co-existence* requirement is not satisfied due to the fact that SDN/OpenFlow requires the introduction of new entities and features, *i.e.,* SDN/OpenFlow Controller and switches, in the operator's transport network. However, due to the ongoing activities in the SDN area, see (ONF, 2103), it can be assumed that operators will probably introduce and deploy SDN/OpenFlow Controllers and switches in their operator's transport networks. Therefore, it can be deduced that the SDN/OpenFlow technology is a promising candidate that can efficiently be used for DMM support in virtualized LTE systems.

4 EXAMPLE OF INTEGRATING SDN/OpenFlow IN VIRTUALIZED LTE SYSTEMS

In this section, an example is provided on how the SDN/OpenFlow enabling technology could be applied in virtualized LTE systems to support DMM. In a virtualized LTE system, the S/P-GWs are running on Virtual Machines (*VMs*) that are hosted on one or more micro data centres. The DMM framework described in (Liebsch et al, 2013) can be used for this purpose. In particular, 0(Liebsch et al, 2013) defines 4 new functional entities: *FE_I*: Ingress for DMM indirection (*redirection*), *FE_E*: Egress for DMM indirection, *FE_IEC*: Control to establish states for DMM indirection and *FE_MCTX*: Function to transfer/establish context for IP address continuity. In Figure 2, it is considered that: OF1 supports *FE_I* and OF4 and OF3 support *FE_E*. The *FE_IEC* and *FE_MCTX* are supported by the SDN/OpenFlow Controller (*OC*), in cooperation with MME. DMM can be used to provide (1) session continuity to users (*UEs*) that are moving from one mobility anchor, *e.g.,* S/P-GW to another mobility anchor, and (2) traffic redirection when a virtualized LTE entity, like the P-GW running on an virtualization platform is migrated to another virtualization platform and ongoing sessions supported by this P-GW need to be maintained. In this example, DMM is used to provide session continuity to users that are moving from one virtualized S/P-GW to another virtualized S/P-GW. In such an architecture, SDN/OpenFlow switches and SDN/OpenFlow Controllers may be implemented in VMs and the functions can be remotely configured and upgraded using soft-EPC components based on the end-users requirements. With OpenFlow the forwarding plane of a

SDN/OpenFlow switch or router can be accessed over the network and modified according to the service requirements. As shown in Figure 2, a simple OpenFlow network is used as transport network above the EPC. All the routers are OpenFlow capable and their flow tables are managed by the same OC.



(a) before handover *(b) after handover*

Figure 2: OpenFlow approach to support DMM in a virtualized LTE system.

In this example, when a UE changes his EPC mobility anchor point, flow tables of all SDN/OpenFlow switches will be updated by OC to re-route the downlink traffic (*i.e.,* arriving from Internet), from the original S/P-GW, see Figure 2(a), to the new S/P-GW, see Figure 2(b). Due to the fact that no modifications will be performed on the data packets, traffic redirection can be performed only if all routers and switches in the operator's transport network are OpenFlow capable. For uplink traffic (*i.e.,* sent towards Internet), a static forwarding path will be setup when a flow is created. If a path to the specific destination hosts already exists, for instance in case of widely visited end-hosts, no changes will be needed to the setup paths. This static path will be used throughout the whole life of a flow in the operator's transport network.

5 CONCLUSIONS AND FUTURE WORK

In this paper several DMM enable technologies have been analysed and compared. In particular, this paper argued and verified that the SDN/OpenFlow technology is a promising candidate that can efficiently be used for the support of DMM in virtualized LTE systems. In order to further validate this statement, the use of the SDN/OpenFlow technology for the support of DMM in the virtualized LTE system will be prototyped and evaluated within the context of the EU FP7 project 0(EU FP7 MCN, 2013).

ACKNOWLEDGEMENTS

This work is accomplished in the context of the MCN project. We therefore, would like to acknowledge the European Commission, since the MCN project is an EC funded Integrated Project under the 7th RTD Framework Programme, FP7-ICT-2011-8-grant agreement number 318109.

REFERENCES

3GPP Release 10, 2103.*Overview of 3GPP Release 10 V0.1.7*, <www.3gpp.org/Release-10>.

Bertin, P., Bonjour, S., Bonnin, J., 2009. *Distributed or centralized mobility*. In Proc. of IEEE Global Telecommunications Conference, (GLOBECOM 2009). pp. 1–6. IEEE.

Chan, H. A., Yokota, H., Xie, J., Seite, P., Liu, D., 2011. *Distributed and Dynamic Mobility Management in Mobile Internet: Current Approaches and Issues.* Journal of Communications, Vol. 6, Iss, 1, pp. 4–15. Academy Publisher.

EU FP7 MCN, (visited in September 2013) <http://www.mobile-cloud-networking.eu/site/>.

Chan, H. (editor), December 2013. *Requirements for Distributed Mobility Management*. IETF Internet draft (work in progress). IETF.

ONF, (visited in December 2013) <https:// www.opennetworking.org/>.

OpenFlow, 2013. *The OpenFlow Specification*.Version 1.3.0, (visited December) <http:// archive.openflow.org>.

Liebsch, M., 2012. *Per-Host Locators for Distributed Mobility Management* . IETF Internet draft (work in progress). IETF.

Seite, P., Bertin, P., Lee, J. H., 2013. *Distributed Mobility Anchoring*. IETF Internet draft (work in progress). IETF.

Bernardos, C. J., Zuniga, J. C., 2013. *PMIPv6-based distributed anchoring*. IETF Internet draft (work in progress). IETF.

Liebsch, M., Seite, P., Karagiannis, G., 2013. *Distributed Mobility Management-Framework & Analysis*. IETF Internet draft (work in progress). IETF.

Implementation of Asynchronous Mobile Web Services
Implementation and First Usage

Marc Jansen[1,4], Javier Miranda[2] and Juan Manuel Murillo[3]
[1]*Computer Science Institute, University of Applied Sciences Ruhr West, Tannenstr. 43, Bottrop, Germany*
[2]*GLOIN S.L. Cáceres, Spain*
[3]*Software Engineering Department, University of Extremadura, Badajoz, Spain*
[4]*Department of Media Technology, Linnaeus University, Växjö, Sweden*
marc.jansen@hs-ruhrwest.de, jmiranda@gloin.es, juanmamu@unex.es

Keywords: Mobile Devices, Web Services, Asynchronous Services.

Abstract: Mobile devices are nowadays used almost ubiquitously by a large number of users. 2013 was the first year in which the number of sold mobile devices (tablet computers and mobile phones) outperformed the number of PCs' sold. And this trend seems to be continuing in the coming years. Additionally, the scenarios in which these kinds of devices are used, grow almost day by day. Another trend in modern landscapes is the idea of Cloud Computing, that basically allows for a very flexible provision of computational services to customers. Yet, these two trends are not well connected. Of course there exists already quite a large amount of mobile applications (apps) that utilize Cloud Computing based services. The other way round, that mobile devices provide one of the building blocks for the provision of Cloud Computing based services is not well established yet. Therefore, this paper concentrates on an extension of an implementation that allows to provide standardized Web Services, as one of the building blocks for Cloud Computing, on mobile devices. The extension hereby consists of a new approach that now also allows to provide asynchronous Web Services on mobile devices, in contrast to synchronous ones. Additionally, this paper also illustrates how the described implementation was already used in an app provided by a business partner.

1 INTRODUCTION

Nowadays, the success of the Cloud Computing business model cannot be ignored. This is mainly due to the proliferation of small services deployed in the cloud that are consumed massively. Proof of that is the impressive economic growth experimented by companies like Facebook (Facebook, 2013) Instagram (Meijer, 2013), or Dropbox (Dropbox, 2013). Many studies show that, increasingly, this consumption is generated from mobile devices. As an example, Gartner predicts that mobile application development projects targeting smartphones and tablets will outnumber native PC projects by a ratio of 4:1 by 2015 (Gartner, 2012). Undoubtedly this fact is favoured by the effort expended by manufacturers to provide mobile devices with capabilities that were hardly imaginable a few years ago.

However, the growing capabilities of mobile devices and the interest they arouse in the industry and consumers contrast with the secondary role to which they are relegated in cloud applications. Cloud services are conventionally based on pure client-server architectures where mobile devices are always assumed as clients. Thus, although current mobile devices have enough power and capabilities to support architectures in which they could also behave as service providers (SaaS), not many efforts have been done in designing such infrastructures.

Thinking about mobile devices as service providers opens new horizons for the industry of small cloud services (Raatikainen, et al., 2012 and Guillén, et al., 2014). Nevertheless, the roadmap for getting mobile devices providing services is not without difficulties related to important aspects such as security, privacy or mechanisms to deploy services in them. In particular, this paper focuses on the implementation needed to deploy Web Services on mobile devices.

In an effort to get mobile devices capable of providing SaaS some works (McFadden, et al., 2003, Srirama, et al., 2006 and AlShahwan, et al., 2010) have already proposed platforms to deploy Web

Services in them. Although such proposals have made a valuable contribution, they still do not cover specific issues such as the continuous connections and disconnections of mobile devices with the subsequent renewals of IPs. Moreover, while the above proposals cover the essential aspects of the Web Services they still left uncovered other aspects that have proved very useful such as the support for both synchronous and asynchronous invocations.

In previous works (Jansen, 2013a, Jansen, 2013b, Guillén, et al., 2014 and Miranda, et al., 2013) the authors presented an implementation that supports the deployment of Web Services in mobile devices. The use of proxies enabled support for connection and reconnection of devices with IP renewals. Now, this paper focuses on the necessity of both synchronous and asynchronous invocations and how they can be enabled in mobile devices. This equals mobile devices with the potential offered by conventional servers for deployment and invocation of Web Services calls. Thus, mobile devices are provided with capabilities not exploited so far opening the door to a new generation of cloud applications where mobile devices gain relevance. As an example it is shown how the proposed implementation has been used to build a M2M messaging application.

The outline of the paper is as follows. Section 2 presents the context of the work. In section 3 the necessity and benefits of both synchronous and asynchronous Web Services invocations in mobile devices is discussed. Section 4 proposes an implementation of both invocation methods. Section 5 presents an experience of using the proposal for the implementation of a M2M messaging app. Finally, section 6 summarizes the conclusions and future works.

2 STATE OF THE ART

The idea of providing Web Services on mobile devices was probably presented first by IBM (McFaddin, et al., 2003). This work presents a solution for a specific scenario where Web Services are hosted on mobile devices. More general approaches for providing Web Services on mobile devices are presented in (Srirama, et al., 2006) and (AlShahwan, et al., 2010). In (Li and Chou, 2011), another approach, focusing on the optimization of the HTTP protocol for mobile Web Services provisioning, is presented. Importantly, none of the mentioned approaches manages to overcome certain limitations of mobile devices, as demonstrated in the next section.

Yet, these approaches do not overcome certain limitations of mobile devices, e.g., permanently changing networks, IP addresses from networks with network address translation (NAT) or the fact that mobile devices are usually not designed to be always online (might be switched off, might have not network connection, …).

An additional approach that covers these problems, is presented in (Jansen, 2013a). This approach utilizes a central proxy infrastructure, that allows on the one hand to cover the mentioned problems and on the other hand establishes a stable infrastructure for mobile device to provide standardized Web Services. As we will see in the following sections, this central proxy infrastructure could easily be extended in order to provide also asynchronous Web Services on mobile devices.

Additionally, the work presented in (Jansen, 2013b) argues for a new perspective to Web Services especially if those services are deployed to mobile devices.

An alternative proposal for deploying services in Mobile devices has been presented in (Guillén, et al., 2014 and Miranda, et al., 2013). This work focuses not only on technical issues but also on the kind of services that could be served from mobile devices and the new generation of applications they would enable.

All the above works let the authors see the potential of having Web Service in mobile devices and the necessity of having them fully featured, including asynchronous calls.

After contextualizing the presented work in the context of the current research, the next section will start describing the basic difference between synchronous and asynchronous service requests.

3 ASYNCHRONOUS VERSUS SYNCHRONOUS WEB SERVICES

In both, programming and the theory of distributed systems, a differentiation between synchronous and asynchronous tasks is well defined. This section provides a short overview about this differentiation in the first subsection.

Furthermore, based on the description of the differentiation synchronous and asynchronous calls, the necessity of asynchronous calls with respect to Web Services provided on mobile devices is argued.

3.1 Asynchronous versus Synchronous Services

In distributed systems, the differentiation between an asynchronous and a synchronous call to a certain service is well defined. In case of a synchronous service call, the service consumer waits for the answer of the service provider, so the service result, before the service consumer continues its tasks. Figure 1 shows the flow of calls for a synchronous service request between a process p1 (server provider) and a process p2 (service consumer).

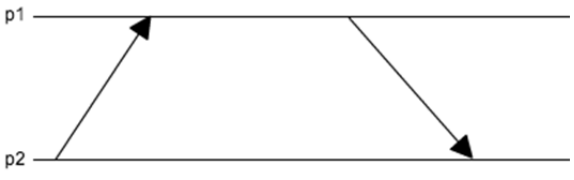


Figure 1: A synchronous service request between a service provider (p1) and a service consumer (p2).

Therefore, the more time the calculation of the service result takes for the provider of the service, the more time the service consumer waits until it continues with its current tasks.

In contrast, by performing an asynchronous service request, the service consumer is not waiting until the service provider answers the request, but just sends the request for the execution of the service to the service provider and immediately continues working on remaining tasks. In a later step, when the service provider has finally calculated the outcome of the service request, the service provider sends the result back to the service consumer. The information flow for an asynchronous service requests between a service provider (p1) and a service consumer (p2) is shown in Figure 2.

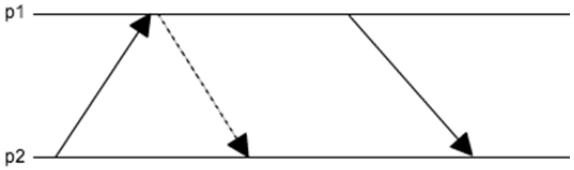


Figure 2: An asynchronous service request between a service provider (p1) and a service consumer (p2).

For retrieving the result of an asynchronous service request, the service consumer usually defines a function that retrieves the result of the requested service. This function then controls also the further actions necessary for handling the result of the service request. The function responsible for handling the result of an asynchronous service request is usually called a callback function.

In programming languages, the difference between an asynchronous and a synchronous call to a method or a function is also well known. If a program performs a usual (synchronous) call to a method, the program continues working in this method and the rest of the program is only executed after the program finished the method call and provided the return value. Of course, programs can outsource computational expensive method calls in sub processes/threads, so that the program continues with the rest of its tasks while the method is executed in parallel.

On the other hand, many programming languages (or frameworks based on usually synchronous programming languages) allow asynchronous requests to methods/functions. One of the probably most prominent examples for this approach is JavaScript. Especially for the development of Web 2.0 applications, the idea of AJAX (Asynchronous JavaScript and XML) (Powell, 2008) came up. Here, method calls to JavaScript methods/functions are performed in an asynchronous manner that allows the browser to continue with its current task. By using this approach, the AJAX application running in a browser, behaves pretty much the same way as a usual desktop application with respect to performance and user feedback. For similar reasons, also the currently most prominent server side implementation for JavaScript, NodeJS (Tilkov and Vinoski, 2010), also integrates an asynchronous approach for method calls. This allows a non-blocking and single threaded approach for the server side implementation of services, that is known for providing increased performance (Tilkov and Vinoski, 2010) in comparison to multi threaded approaches, e.g., this is the major reason that proposed solutions for the C10K problem (Kegel, 2013) almost completely rely on asynchronous approaches.

3.2 The Necessity of Asynchronous Mobile Web Services

Synchronous mobile Web Services represent a valid approach when the required time by a mobile device to process each request is low enough to be solved and responded back to the client before it gets a timeout exception message. In this sense, there are several use cases that could be implemented using this type of mobile Web Service, like those that involve simple information retrieval or processing,

that is, when such information is available immediately (usually, it has been previously calculated and stored in the device, like an image, a contact name, or a text message).

Unfortunately, mobile devices are exposed to continuous context changes that may affect the time lapse required for solving specific types of request, especially those that depend on information provided by external resources or sensors. This implies that the response time for such requests is unpredictable at design time, and a proper behaviour at run-time cannot be ensured using the synchronous approach.

In order to illustrate the necessity of asynchronous mobile Web Services, we have considered, at least, the following potential use cases of mobile Web Services we could deploy in the current state-of-the-art of mobile applications:

- Services that offer information fetched from an external device/sensor. Gathering information from sensors (like GPS or Bluetooth) is usually implemented by mobile operative systems using asynchronous methods with their callbacks.
- Services that depend on device's user interaction. Mobile devices could serve human-based services that allow asking users about anything, including OCR or captcha-like challenges. They can be answered at any time, so the callback response with the user's choice would be sent asynchronously.
- Services offering information that needs to be evaluated gathered or processed during a long period of time. We can include in this category use cases related with measurement and tracking, like "determining the distance the user will travel in the next five minutes" using the accelerometer sensor information for tracking it, or "taking the temperature average for the next 2 hours" tracking it from a temperature sensor.
- Services that depends on a particular context to be processed. This category could include services offering information resulted from some interaction with an external device or resource, like: getting information from the car handsfree system or a domotic interface; heavy processing task that requires the device to be plugged-in to a power supply; or a service that require higher connectivity bandwidth to be properly executed.

These use cases that involve mobile Web Services and require an asynchronous behaviour suggest that synchronous mobile Web Services are not enough for supporting more complex services, and it motivates the proposal of this approach, which implementation is described on the next section.

4 IMPLEMENTATION

Following the requirements formulated in the last section, the approach described in (Jansen, 2013a) was extended by the possibility to also provide asynchronous calls to Web Services deployed to mobile devices. This section describes the implementation of these asynchronous mobile Web Services.

Here, the basic idea for the implementation of these asynchronous mobile Web Services was to be as close as possible at the solution for synchronous Web Services, which basically relies on two major parts:

1) JSR 181 compatible annotations that allow to marker a class as an implementation of mobile Web Services and methods as mobile Web Methods.
2) A proxy infrastructure that dynamically generates implementations of a façade (Gamma, et al., 1995) for the mobile Web Services in order to overcome certain limitations of mobile devices, with respect to the provision of services.

Additionally, from a technical point of view, another requirement for the implementation was to re-use already existing possibilities and not to invent new techniques whenever possible. Therefore, the architecture for synchronous mobile Web Services, as shown in Figure 3, was analysed.

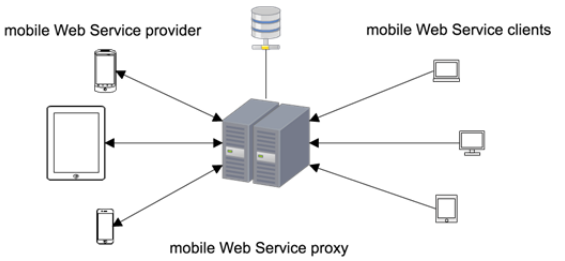


Figure 3: Architecture for providing synchronous mobile Web Services.

In the centre of Figure 3, the mobile Web Service proxy could be seen, which basically provides the façade for the Web Services actually running on the mobile devices in the left part of Figure 3. Therefore, the requests to the performed by the Web Service clients are not directly addressed to the mobile devices, but to the proxy infrastructure.

After the detailed analysis of this architecture, the following strategy was developed for the implementation of asynchronous mobile Web Services. First of all, a new annotation was integrated in the existing framework (@AsynchronousMobileWebMethod) for the

implementation of a method that should be executed on a mobile device as an asynchronous mobile Web Method.

Additionally, the proxy was modified so that it is able to handle also asynchronous requests, next to the synchronous ones. Here, the idea was to provide a callback for every request to an asynchronous mobile Web Service, as shown in Figure 4.

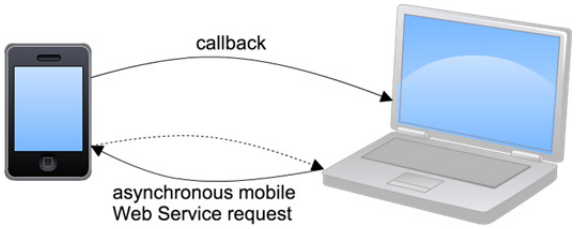


Figure 4: An asynchronous mobile Web Service request.

This callback itself was implemented as a reference to a Web Service that was responsible for handling the result of the asynchronous request to the mobile Web Service. Therefore, the reference to the callback is usually a URL to a WSDL (Web Service Description Language) of the service that handles the result of the asynchronous request.

By this, the method/function handling the result could either be implemented on a mobile device (using the implementation for mobile Web Services described in (Jansen, 2013a)) or it could be implemented as a usual Web Service running on a usual computer/server.

Interestingly, from a developers point of view, the implementation of an asynchronous mobile Web Method does not differ from the implementation of a synchronous mobile Web Method, especially no reference to a callback needs to be provided. All the necessary extensions for the handling of asynchronous requests could automatically be applied to the façade created by the central proxy.

The next section provides a description about a mobile application that already utilizes the described approach for asynchronous mobile Web Services deployed to mobile devices. By the successful integration of this implementation, the mobile application was able to achieve the requirements discussed earlier in this paper.

5 INTEGRATION OF ASYNCHRONOUS MOBILE WEB SERVICES INTO AN EXISTING APPLICATION

The proposed implementation of asynchronous mobile Web Services has been tested in *beeFun*, a particular messaging application.

This mobile app treats to reinvent the traditional messaging concept into a new, different and funny experience. In order to make it attractive for users, *beeFun* combines a simpler interface for sending different preconfigured (or not) messages very quickly with several gaming experiences like weekly-renewed scores and badges obtainment. Different templates of message are available to be sent, including simple text, text with images, location-based messages and surveys. Messages can be sent to a given group of known users, or a specifying set of criteria that define such receivers, like their current location, age, or gender, among others. Unlike other similar apps, this information is stored exclusively on each device, so those criteria are evaluated in every device when this kind of message is sent. This design, in addition to bringing a different perspective on mobile devices as service providers similar to the Software-as-a-Service model (but in a mobile context), allows sending geo-located messages without forcing users to be tracked neither updating their information to a centralized server continuously, solving certain privacy issues in a very elegant way.

Consequently, this architecture requires a fast and flexible way to connect the server-side (which orchestrates every message's request and response) with mobile devices (which evaluate the conditionals and accept or decline every message depending on their current status). Technically, this was implemented through an in-house solution that combines the use of Push Notification Services for "waking up" mobile devices when new messages arrive, and a RESTful-based server API infrastructure for handling both requests and responses to and from mobile devices. Figure 5 shows the main flow of interactions between server and devices when a location-based message is sent.

Due to the asynchronous nature of the location-based messaging of *beeFun*, it represents a useful testbed for integrating the asynchronous mobile Web Services framework. In *beeFun*, devices' location is fetched from the Location Service provided by the mobile operating system, and it is done asynchronously for avoiding deadlocks until a valid

position is obtained from sensors (GPS, WiFi, etc.). Actually, not only location but also any information gathered by *beeFun*, including history of locations, answers to surveys and so, conform the information core component of the app, and can be considered as a mobile Web Service itself with several Web Methods for getting information about the device or its owner.

Following that consideration, the integration of the framework has changed the previous design of the messaging component behaviour from the diagram described in Figure 5 to the new flow illustrated in Figure 6.

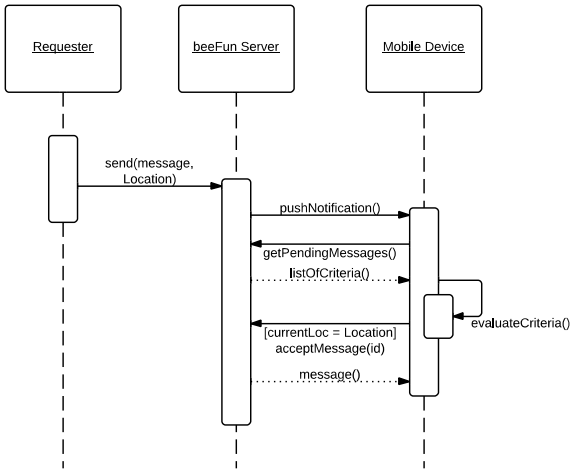


Figure 5: Sequence diagram of the location-based messaging implementation without the asynchronous mobile Web Service integration.

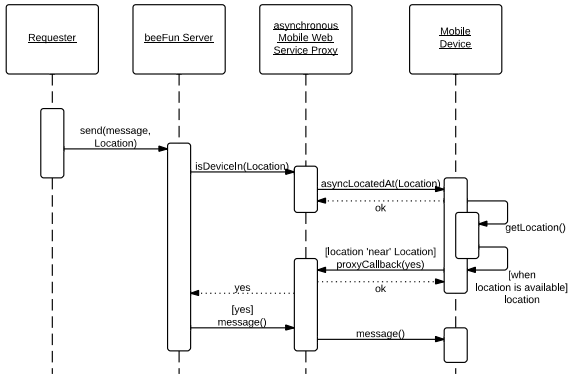


Figure 6: Location-based messaging flow using the asynchronous mobile Web Service approach.

Both figures represent the same request for sending a location-based message that is evaluated and accepted by the device. However, the integration of the asynchronous mobile Web Services framework greatly simplifies the complexity of the outgoing protocol between the *beeFun* server and mobile devices in comparison with the previous non-standardized approach.

Regarding the code, the implementation of the mobile-side is also clearer thanks to the separation of concerns derived from the adoption of the proposed class hierarchy and its annotations, both required for implementing mobile Web Services and Web Methods properly.

Additionally, the use of the asynchronous mobile Web Services platform can easily be extended to other parts of *beeFun* services with minimal effort due to the standardization of the business rules between server and mobile.

6 DISCUSSION AND OUTLOOK

This paper has presented a proposal to enable Web Services deployment in mobile devices allowing both synchronous and asynchronous calls. It has also described the experience of using the implementation to build *beeFun,* a M2M messaging application. *beeFun* is currently being commercially exploited by the startup GLOIN[1].

Having the possibility of deploying web services to mobile devices raises the concept of Mobile Clouds to a higher level. It not only opens the door to a new generation of cloud applications in which mobile devices acquire the capabilities of providing SaaS. In addition, it also enables new business flows benefiting the device owners.

Currently the authors are releasing new applications based on the principles presented in the article[2]. The new application areas are allowing them to detect new features to be incorporated into the technological platform.

Additionally, the described implementation also allows to implement new scenarios, e.g., in the area of mobile health applications, in which aspects like security and privacy, but also de-central storage of sensual personal data, could be handled much easier in comparison if other technologies are used. Therefore, the approach described in this paper provides rich potential for further development of new Cloud Computing based applications.

[1] http://www.gloin.es/en/
[2] http://www.nimbees.com/

REFERENCES

F. AlShahwan, K. Moessner, “Providing SOAP Web Services and REST Web Services from Mobile Hosts”, In: *Fifth International Conference on Internet and Web Applications and Services (ICIW 2010)*, pp. 174-179.

Dropbox. Dropbox FAQ, 2013. https://www.dropbox.com/help/7/en.

Facebook, Do you know What's Up?. https:// www.facebook.com/notes/up-creative-inc/do-you-know-whats-up-check-out-these-2013-social-media-statistics/470970089631080.

E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Pattern – Elements of Reusable Object-Oriented Software, Addison-Wesley. 1995.

Gartner. Market Trends: Application Development Software, Worldwide, 2012-2016. Technical report, Gartner, 2012. http://www.gartner.com/ resId=2098416.

J. Guillén, J. Miranda, J. Berrocal, J. García-Alonso, J. M. Murillo, C. Canal, *“People as a Service: a mobile-centric model for providing collective sociological profiles”*. IEEE Software SWSI: Next Generation Mobile Computing (2014) [ACCEPTED, PENDING PUBLICATION]

M. Jansen. Analysis and Improvement of Energy Consumption for Providing Mobile Web Services. In. *International Journal of Soft Computing and Software Engineering*, DOI: 10.7321/jscse. 2013a.

M. Jansen. About the Necessity to Change the Perspective for Mobile Web Services. In: Proceedings of the 15th *IEEE International Symposium on Web Systems Evolution*, 2013b.

D. Kegel. The c10k problem, http://www.kegel.com/ c10k.html, last visited: 18.12.2013.

L. Li, W. Chou, “COFOCUS – Compact and Expanded Restful Services for Mobile Environments”, In: *Proceedings of the 7th International Conference on Web Information Systems and Technologies*, pp. 51-60, Noordwijkerhout, The Netherlands. 2011.

G. Meijer. Instagram: A Case Study for the Clouds, 2013. http://www.cloudproviderusa.com/instagram-a-case-study-clouds/.

S. McFaddin, C. Narayanaswami, M. Raghunath, “Web Services on Mobile Devices – Implementation and Experience”, In: *Proceedings of the 5th IEEE Workshop on Mobile Computing Systems & Applications (WMCSA’03)*, pp. 100-109, Monterey, CA.

Miranda, J., Guillén, J., Berrocal, J., Garcia-Alonso, J., Murillo, J., & Canal, C. (2013). Architecting Infrastructures for Cloud-Enabled Mobile Devices. In C. Canal & M. Villari (Eds.), Advances in Service-Oriented and Cloud Computing SE - 23 (Vol. 393, pp. 277–287). *Springer Berlin Heidelberg*. doi:10.1007/978-3-642-45364-9_23.

T. A. Powell. AJAX – The Complete Reference, *McGraw-Hill Osborne Media*, 2008.

M. Raatikainen, T. Mikkonen, V. Myllarniemi, N. Makitalo, T. Mannisto, and J. Savolainen. Mobile content as a service a blueprint for a vendor-neutral cloud of mobile devices. *IEEE Software*, 29(4):28{32, 2012.

S. Srirama, M. Jarke, W. Prinz, “Mobile Web Service Provisioning”, In: *Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW 2006)*, p. 120, Guadeloupe, French Caribbean.

S. Tilkov, S. Vinoski. Node.js: Using JavaScript to Build High-Performance Network Programs, *IEEE Internet Computing*, vol. 14, no. 6, pp. 80-83, November/December, 2010.

Semantic Data Integration for Ubiquitous Logistics
An Approach Supporting Autonomous Logistics in Urban Environments

Stefan Wellsandt, Konstantin Klein, Marco Franke, Karl Hribernik and Klaus-Dieter Thoben
BIBA – Bremer Institut für Produktion und Logistik, Hochschulring 20, Bremen, Germany
{wel, kle, hri, tho}@biba.uni-bremen.de

Keywords: Autonomous Logistics, Ubiquitous Computing, Semantic Data Integration, Interoperability.

Abstract: This paper introduces an approach for seamless plug & play data integration in a novel urban logistics concept. The logistics concept is called ubiquitous logistics and contains an agent-based perspective on shared logistics resources. Each private and public vehicle participating in the concept, as well as parcels, features an intelligent agent that may request (or offer) information services to other agents or legacy systems. The technical approach of this paper suggests that each of the heterogeneous data sources delivers additional information that is used to virtually integrate the data in an automated way. This additional information concerns, the authentication, data structure and sequence – information that have to be provided manually nowadays. The technical approach is explained using a typical situation from the future urban logistics concept. This situation represents an intelligent agent trying to deliver small goods along a stream of urban commuters.

1 INTRODUCTION

The exchange of goods between companies and the delivery of final products to consumers requires considerable logistics capacity. The planning and realisation of both the transport and storage of goods need to take into account time, cost and quality constraints. Changes in customer behaviour, such as an increase in importance of individualised/ personalized products compared to mass-produced ones (Kumar, 2007), challenge these activities. This demand for individualised products, combined with the growing popularity of online shopping and the customers' desire for same-day delivery cause a reduction of delivery lot size, in turn increasing the overall amount of deliveries (Ickert et al., 2007). This growth of delivery volumes puts urban traffic and logistics systems under additional stress causing congestion, limited parking space, pollution and infrastructure deterioration (Müller et al., 2006).

An approach to reduce these negative impacts caused by increasing delivery volumes is the ubiquitous logistics concept as presented in (Wellsandt et al., 2013). Cornerstone of the approach is to describe urban freight transport as a system of shareable logistics resources managed by collaborating intelligent agents. Agents continuously request and offer information services while each service is connected to a data source.

One of the most significant problems in this environment is the seamless exchange of information from distributed, heterogeneous data sources. Intelligent agents and data sources like traffic management systems, logistic infrastructure and other systems have a need for actual information about relevant system states to make a sustainable decision. Nowadays, there are several approaches to establish the communication between different data sources featuring different information structures and information meaning. One of these techniques is the approach of the *virtual data integration*. The main advantage of this approach is the delivery of information on demand without the need for an additional data storage device like a data base.
The paper aims to describe an approach enabling automated virtual data integration in urban logistics. For this purpose, section 2 will explain existing research on the fundamental topics of this paper. Section 3 will cover the technical approach that is explained using a guiding example. Section 4 concludes the described approach.

2 RELATED WORK

This section covers existing work related to the core

ideas of this paper. The section will explain the concept of ubiquitous logistics and its preceding background. Furthermore, the section will cover aspects of data integration that are reused in the introduced approach of this paper.

2.1 Autonomous Logistics

Interpretation of logistics as a multi-agent system (MAS) is discussed in literature for several years (Moore et al., 1997). In 2005, Davidsson et. al. published a survey about the use of MAS in transportation and traffic management (Davidsson et al., 2005). Findings indicate that many logistics problems have characteristics similar to those of agent-based systems. A similar argumentation is suggested for the collaborative research centre 637 for autonomous logistics in Bremen, Germany (Scholz-Reiter et al., 2004). According to (Gehrke et al., 2010), autonomous logistics can be enabled by *situation-aware agents*, actively gaining information about their surroundings. Based on an agent's observations, decisions can be made about logistics processes. An example for the implementation of situation-aware agents in logistics is the intelligent container that is described in (Lang et al., 2011). A comprehensive study on multi-agent coordination enabling autonomous logistics is provided by (Schuldt, 2012).

The argumentation of autonomous agents in logistics offers far reaching possibilities for future urban freight transport. In order to utilize possibilities, the focus of planning and control of urban transport needs to shift from a centralized perspective towards a decentralized one. At the same time, efficiency in urban freight transport is increased by taking into account all available logistics resources an urban environment provides.

2.2 Ubiquitous Logistics

In the ubiquitous logistics concept, public and private transportation means that are able to transport goods (parcels) within a city are considered as *logistics resources* of that same city (i.e. private vehicles, trucks, trams, buses, etc.). Primary objective of the concept is to avoid additional traffic for freight transport by utilizing existing logistics resources best.

In order to manage the multitude of logistics resources within a city, transported parcels feature an autonomous, situation-aware agent that can be human or artificial. The agent collects information and takes decisions required to complete the scheduled transportation task for a certain parcel. The decision process each agent has to perform is based on a multi-criteria problem. The problem concerns the selection and realization of an optimum route from the current location to a target destination within a dynamic environment. Potential routes are created and destroyed dynamically based on the availability of local logistics resources. A situation-aware agent picks logistics resources in order to make progress against its scheduled transportation task. Completion of a transportation task might require taking different logistics resources (i.e. vehicles or transportation modes).

The assignment of parcel and resource depends on criteria like transportation time, load efficiency, service level, stock level, as well as transportation and storage cost (Nyhuis and Wiendahl, 2013). Furthermore, environmental dynamics need to be taken into account including for instance traffic condition, delivery order changes, weather conditions, and availability of logistics resources.

The decision process requires large amounts of most recent information from distributed sources like other agents, traffic management systems and logistics providers. Therefore, the underlying information infrastructure needs to be capable of handling several thousands of decentralized information requests and offers at the same time. In this infrastructure, each information offer is represented by a specific *service*. Services can be provided by different organizations with different data source structures. Complex decisions require that several information services are requested and integrated.

2.3 Data Integration

Information services are based on data sources. From a technical point of view, significant data sources in logistics are data processing systems, data storage systems and sensors deployed in the field. The *data processing systems* are decision making agents, infrastructure management systems and other necessary systems to handle processes in logistics. *Data storage systems* are used to aggregate data in logistics. This includes for instance generic data bases, data warehouse systems, warehouse management systems and product lifecycle management systems. *Sensors* are typically used to collect field data. This data may contain measurements like temperature, light intensity and mechanical forces. The described data is allocated in different ways, for instance as a CSV file, text file or database. It can be accessed by using HTTP (hyper

text transfer protocol), web services or industrial standards like EDIFACT or EANCOM. The data is transmitted as a stream or in a structured way by, for instance services, cloud applications, data gathering modules and data warehouses.

The concept of the *semantic mediator* developed to fetch data on demand based on the mentioned virtual data integration approach is used to retrieve data from sensors and data storage systems (Hribernik et al., 2010). This concept abrogates the persistence of the data from each data source in one huge data warehouse. The implementation of the virtual data integration concept retrieves only the needed data on demand. The binding module to resolve a query is implemented by a wrapper, depicted in Figure 1.

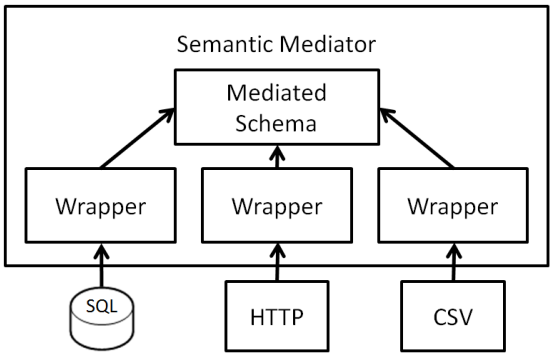


Figure 1: Integration module concept (semantic mediator).

The wrapper contains the communication interface between the data sources and a semantic mediator. Nowadays, the wrapper modules have to be adapted for new data sources manually. The mediated schema remains untouched by this adaption. The wrapper modules abstract the data sources and all users query the needed information over the wrapper. Here, the data from the particular data sources will be transformed into a global representation. The data sources are represented in the figure by SQL, HTTP and CSV modules.

The following information in the data integration domain need to be implemented manually: (1) the access method; (2) the sequencing of the data access and exchange; (3) the adaption to the structure of the data sources. Related to the seamless integration of information service in ubiquitous logistics, there is a high demand for the automated handling in the mentioned three points. Otherwise, the dynamic integration of, for instance logistics resources (and their related agents), is too time and cost intensive.

3 APPROACH

The technical approach of the proposed solution aims for seamless (plug & play) integration of data from different sources, in order to facilitate a collaborative service infrastructure in urban logistics. For this purpose, automated binding of heterogeneous data sources that leaves only a few activities up to the user is targeted in this paper. The technical foundation of the approach is explained in the next section, followed by a guiding example consisting of a small situation.

3.1 Technical Foundation

In order to realize an automated binding of different data sources, each source is able to describe itself. Key information concerns the authentication, sequence and structural characteristics of a data source. Each of these three descriptors is explained briefly in the following:

- **Authentication** – concerned with the security aspects of the access process of the particular service. The information ensures subsequent access authentication to eligible machines, services and persons.
  The authentication is typically based on username and password. More sophisticated methods are also possible. Moreover, the protocol for the authentication can differ from the essential data exchange protocol. It can happen in one or more steps.
- **Sequence** – typically, available data exchange processes are conducted in multiple steps. Up to now, implemented approaches don't consider information flows in more than one step. Especially the data access over web services proceeds during several steps. Moreover, the most data accessing procedures happen in one session with a recurrent sequential pattern of "data access" and "data processing". This pattern is common in authenticated sessions in which the subsequent data query has to carry the secure token.
- **Data structure** – describes what information and what kind of data model the data source uses. Furthermore, it contains information how certain integration problems related to the data structure can be solved. The data structure problems that need to be solve are: (a) different names for attributes with the same meaning; (b) different data representation (describes how data is conceived, manipulated and recorded); (c) use of different data model; (d) data sources cover

different attributes.
The data sources send the information about their authentication process, sequencing information and the data structure information to the virtual data integration module (semantic mediator) described in section 2.3. The authentication and data structure information relies on ontology models, located in the mediated schema, describing the needed information semantically. Ontologies lack the ability to describe sequencing. According to this fact, the description of the sequencing information is modelled by means of sequence diagrams which are part of the Unified Modelling Language (UML).

3.2 Guiding Example

The following example (see Figure 2) will be used to explain how the integration of data might work in the ubiquitous logistics concept.

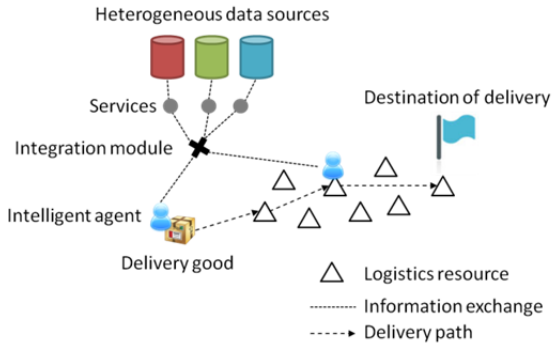


Figure 2: Illustration of the guiding example for data integration in ubiquitous logistics.

A hypothetical stream of daily commuters (logistics resources) will be exploited to transport small sized goods within a city. In order to keep the delivery time and cost low, delivery paths (routes) are decided by an intelligent. The agent utilizes based the information extracted from heterogeneous data sources. These sources concern for instance weather and traffic data from the field (sensor data). In addition, information about the delivery order needs to be taken into account. These information concern for instance the delivery destination (address), expected delivery day and pricing information. Each commuter's agent offers a service for its location and delivery status. Due to the underlying agent-based network in ubiquitous logistics, each parcel's agent has to integrate the distributed information for decision-making.

The necessary data exchange process for the decision making happens through the proposed approach. According the described situation the following data integration process flow is conceivable:

1. An intelligent agent needs additional traffic information because a scheduled delivery route collapsed. This might happen, for instance if a commuter leaves her daily travel route and another logistics resource has to complete the delivery.
2. The intelligent agent sends a request for additional routing information. The integration module (e.g. semantic mediator) looks up for the particular service(s) which can deliver the requested information from a data source.
3. The integration module asks for the particular authentication data and sequence models the intelligent agent can use. After that the authentication process between the requested service and the intelligent agent proceeds.
4. The next step is the exchange of the existing data structure model and the sequencing model for the data exchange. After that, the data exchange process proceeds.
5. The final logout process is part of the authentication process. The necessary information for this step has been exchanged in the third step.

According to this brief example the data from different sources can be integrated by the intelligent agent automatically. The agent can request the needed information services in order to manage the delivery of the parcel. In case an additional service provider (and the related data source) becomes available, the new wrapper for the data source does not need to be implemented manually but receives relevant information from the data sources.

4 CONCLUSIONS

This paper presents an approach allowing intelligent agents within an urban logistics scenario to integrate heterogeneous information services and their related data automatically. The paper starts with a brief overview of autonomous and a future urban logistics concept, as well as an overview of data integration. In the following, the technical foundation of the approach is outlined and explained through an example. The example represents a typical situation potentially occurring in ubiquitous logistics.

Benefit of the proposed approach is the reduction of time and cost for manual adaption of data integration tools. Additionally, it allows the integration of data for distributed agents, reducing

the need for maintaining local data bases. This will become especially useful in the mentioned concept for ubiquitous logistics that is grounded on agent-based decision making based on numerous distributed data sources. Dynamically changing sources, especially related to logistics resources, can be seamlessly integrated through the introduced approach.

ACKNOWLEDGEMENTS

The paper is based on research that is done in the research project “CyProS”, funded by the German Federal Ministry of Education and Research (BMBF) within the Framework Concept ”Research for Tomorrow’s Production” (fund number 02PJ2461).

REFERENCES

Davidsson, P., Henesey, L., Ramstedt, L., Törnquist, J., Wernstedt, F., 2005. An analysis of agent-based approaches to transport logistics. *Transp. Res. Part C Emerg. Technol. 13*, 255–271.

Gehrke, J. D., Herzog, O., Langer, H., Malaka, R., Porzel, R., Warden, T., 2010. An agent-based approach to autonomous logistic processes. *KI-Künstl. Intell. 24*, 137–141.

Hribernik, K. A., Kramer, C., Hans, C., Thoben, K.-D., 2010. A Semantic Mediator for Data Integration in Autonomous Logistics Processes, in: Enterprise Interoperability IV. *Springer*, pp. 157–167.

Ickert, L., Matthes, U., Rommerskirchen, S., Weyand, E., Schlesinger, M., Limbers, J., 2007. Abschätzung der langfristigen Entwicklung des Güterverkehrs in Deutschland bis 2050 (Study for the German Ministry of Traffic, Construction and Urban Development). Prognosen und Strategyberatung für Transport und Verkehr.

Kumar, A., 2007. From mass customization to mass personalization: a strategic transformation. *Int. J. Flex*. Manuf. Syst. 19, 533–547.

Lang, W., Jedermann, R., Mrugala, D., Jabbari, A., Krieg-Brückner, B., Schill, K., 2011. The “Intelligent Container” — A Cognitive Sensor Network for Transport Management. Sens. *J. IEEE 11*, 688–698.

Moore, M. L., Reyns, R. B., Kumara, S. R., Hummel, J. R., 1997. Distributed intelligent agents for logistics (DIAL), in: Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 I*EEE International Conference on. IEEE*, pp. 2782–2787.

Müller, M., Görnert, S., Volkamer, A., 2006. Gueterverkehr in der Stadt - Ein unterschätztes Problem. Verkehrsclub Deutschland e.V.

Nyhuis, P., Wiendahl, H.-P., 2013. Logistische Kennlinien: Grundlagen, Werkzeuge und Anwendungen. *Springer DE*.

Scholz-Reiter, B., Windt, K., Freitag, M., 2004. Autonomous logistic processes: New demands and first approaches, in: *Proceedings of the 37th CIRP International Seminar on Manufacturing Systems*. pp. 357–362.

Schuldt, A., 2012. Multiagent Coordination Enabling Autonomous Logistics. KI - Künstl. Intell. 26, 91–94.

Wellsandt, S., Werthmann, D., Hribernik, K., Thoben, K.-D., 2013. Ubiquitous logistics: a business and technology concept based on shared resources. Dublin, Ireland.

SPECIAL SESSION ON FEDERATED CLOUD GOVERNANCE

FULL PAPERS

On-demand Cloud Architecture for Academic Community Cloud
Another Approach to Inter-cloud Collaboration

Shigetoshi Yokoyama and Nobukazu Yoshioka
National Institute of Informatics, Tokyo, Japan
{yoko, nobukazu}@nii.ac.jp

Keywords: Inter-cloud, Community Cloud, Cluster as a Service, Bare-metal Provisioning, Academic Cloud.

Abstract: This study describes a new approach to cloud federation architecture for academic community cloud. Two basic approaches have been proposed to deal with cloud burst, disaster recovery, business continuity, etc., in community clouds: standardization of cloud services and multi-cloud federation. The standardization approach would take time; i.e., it would not be effective until there are enough implementations and deployments following the standard specifications. The federation approach places limitations on the functionalities provided to users; they have to be the greatest common divisor of the clouds' functions. Our approach is "cloud on demand", which means on-demand cloud extension deployments at remote sites for inter-cloud collaborations. Because we can separate the governance of physical resources for cloud deployment and the governance of each cloud by this approach, each organization can have full control on its cloud. We describe how the problems of the previous approaches are solved by the new approach and evaluate a prototype implementation of our approach.

1 INTRODUCTION

Private clouds get some benefit from the consolidations made possible by using virtualization technology. However an individual organization cannot reduce IT costs significantly through the use of its own private cloud because it must have on hand the maximum IT resources needed to deal with peak traffic.

In order to better utilize IT resources, a hybrid cloud solution is feasible in some situations. A hybrid cloud consists of a private cloud and public cloud; the private cloud deals with flat traffic and the public cloud covers peak traffic. However, when security matters, it is not feasible to send all the peak traffic to the public cloud.

It is important to think about sharing IT resources among private clouds to ensure better utilization and security at the same time. This idea can be viewed as a private cloud hosting service.

Table 1 describes the characteristics of public, private and hybrid clouds. Cloud users have to decide what kind of cloud they want to use, depending on their applications. A hybrid cloud integrates private and public clouds vertically. It assigns peak traffic of applications that do not necessarily need strong security to the public cloud. It cannot fit the situation in which all peak traffic have to be dealt with securely.

On the other hand, a community cloud is a way to keep clouds independent from one another while getting flexibility and security at the same time. In fact, there has been a lot of activity on ways to establish community clouds. The approaches can be categorized into two kinds. One is standardization of cloud services and the other is multi-cloud federation. The standardization approach would take time; i.e., it would not be effective until there are enough implementations and deployments following the standard specifications. The federation approach places limitations on the functionalities provided to users; they have to be the greatest common divisor of the clouds' functions.

We propose a new approach, called "cloud on demand", which integrates many private clouds horizontally and shares IT resources among them to accommodate peak traffic. By applying this solution, users can get good IT resource utilization like in a public cloud and have the level of security of a private cloud.

In this paper, we introduce our cloud on demand solutions called dodai and colony and describe a real cloud on demand service that was recently deployed as the research cloud of our research institute, National Institute of Informatics (NII).

This paper is organized as follows. Section 2 describes the previous approaches. Section 3 introduces the cloud on demand solution. Section 4 shows a prototype implementation. We summarize our evaluation of case studies in Section 5 and conclude in Section 6.

Table 1: Characteristics of cloud solutions.

	Cost	Security	Ease of Application Development
Public Cloud	Strong for peak traffic pattern	Depends on public cloud provider policy and management	Has public cloud architecture constraints
Private Cloud	Strong for flat traffic pattern	Depends on controllable private cloud management	Can choose application architecture
Hybrid Cloud	Strong for flat + peak traffic pattern	Depends on controllable deployment architecture and private cloud management	Has hybrid cloud architecture constraints
Community Cloud	Strong for small to big and flat to peak traffic	Depends on controllable private cloud management	Can choose application architecture

2 PREVIOUS APPROACHES

In this section, we describe the previous approaches , which are cloud standardization and cloud federation.

2.1 Cloud Standardization

One of the famous cloud standardization activities is the Global Inter-Cloud Technology Forum (GICTF) (GICTF). Its mission is as follows:

- Promote the development and standardization of technologies to use cloud systems.
- Propose standard interfaces that allow cloud systems to interwork with each other.
- Collect and disseminate proposals and requests regarding the organization of technical exchange meetings and training courses.
- Establish liaisons with counterparts in the U.S. and Europe, and promote exchanges with relevant R&D teams.

GICTF has produced a number of white papers, including "Use Cases and Functional Requirements for Inter-Cloud Computing", “Technical Requirements for Supporting the Intercloud Networking”, "Intercloud Interface Specification Draft (Intercloud Protocol)" and "Intercloud Interface Specification Draft (Cloud Resource Data Model)."

There are other similar standardization activities like the Open Cloud Standards Incubator, Cloud Storage Technical Work Group, Open Cloud test bed and Open Cloud Computing Interface Working Group (Hiroshi Sakai, 2011).

The use cases they deal with are as follows:

U1.Guaranteed performance during abrupt increases in load

U2.Guaranteed performance regarding delay

U3.Guaranteed availability in the event of a disaster or a large-scale failure

U4.Service continuity

U5.Market transactions via brokers

U6.Enhanced convenience by service cooperation

The clouds maintain independence from one another and collaborate with each other through standard interfaces. This approach seems to be the ultimate solution for community clouds but it will take time to get a consensus from all the communities on the standard.

2.2 Cloud Federation

Cloud federation is the practice of interconnecting the cloud computing environments of multiple service providers for the purpose of load balancing traffic and accommodating spikes in demand. Cloud federation requires one provider to federate the computing resources of the cloud providers. Cloud federation consists of the following components:

Application: a set of virtual machines and data volumes connected by a virtual network to be deployed at the IaaS level.

Portal: a common entry point for multiple cloud providers. A user submits an application to the portal. The portal selects providers to run the application. Usually, the portal can only offer functionalities that are the greatest common divisor of the providers.

This approach tries to cover use cases U1, U2, U3, U4 and U5. In contrast, we assume that the main purpose of establishing community clouds is to accommodate use cases U1, U2, U3 and U4. That is, our cloud on demand solution focuses on these use cases.

3 CLOUD ON DEMAND

Our approach is different from the previous ones. Figure 1 is an overview of our cloud on demand solution. There are two service components. One is called Cluster as a Service (Yokoyama and Yoshioka, 2012a), and the other is called the inter-cloud object storage service (Yokoyama and Yoshioka, 2012b), (Yokoyama et al., 2012a). Cluster

as a Service is a service by which users create clusters consisting of physical servers, and it can deploy software components for building an IaaS.

The inter-cloud object storage service lets users store objects, like machine images, as if they were using a local cloud object storage service. Physically, each cloud is connected to a high-speed wide area network, such as SINET-4 (SINET). The network connections are made by using network functionalities like L2VPN and VPLS. The physical servers can be located in the same L2 network segment if the same VLAN–ID is assigned to them. The physical servers that are assigned different VLAN-IDs are securely separated from the other network segments.

Through this design, we can generate physical machine clusters in inter-cloud environments on which we can deploy IaaS software like OpenStack, Eucalyptus, and others in our favourite configurations for each.

In addition, we configure a distributed inter-cloud object storage service using open source software like OpenStack swift for storing machine images.

To allocate the application execution environments, we deploy an IaaS cluster on demand on physical servers and deploy the application virtual machine cluster on it. In this case, the IaaS cluster uses the inter-cloud object storage service to launch virtual machines from machine images that have been prepared for the application cluster. IaaS clusters themselves are not necessarily destroyed after each the application execution. The life cycle of the IaaS is independently controlled by the application execution environment managers.

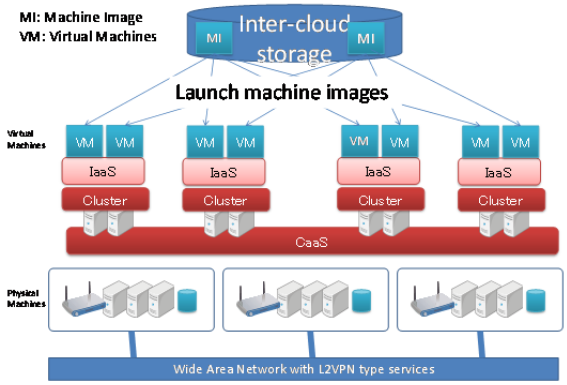


Figure 1: Architecture overview.

3.1 Cluster as a Service (CaaS)

Cluster as a Service is designed as follows:

1) Two-layer implementation

The lower layer takes care of physical machine cluster management. The upper layer handles virtual machine cluster management. Moreover, each layer is programmable with web APIs.

2) The lower layer

The lower layer handles the operating systems of each node composing a cluster. Nodes can be allocated to clusters dynamically from software and securely separated by using network technology, like virtual LAN in the allocation.

3) The upper layer

The upper layer deals with deploying IaaS software such as OpenStack and Eucalyptus. It also can deploy PaaS software. The layer has configuration management tools to ease deployment on the nodes of clusters.

An actual deployment example is depicted in Figure 2.

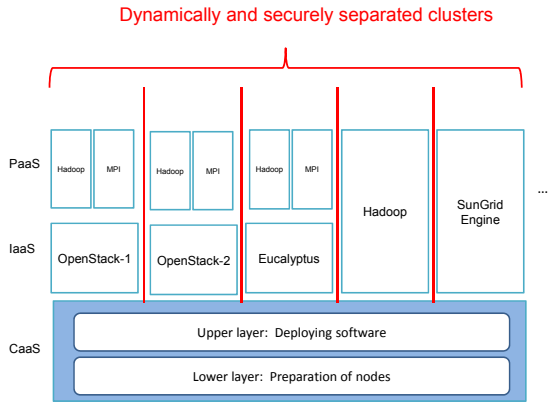


Figure 2: Cluster as a Service.

3.2 Inter-cloud Object Storage Service

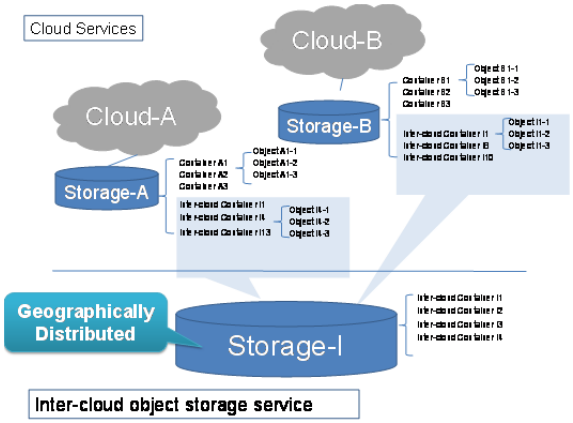


Figure 3: Inter-cloud storage service.

Figure 3 depicts the service from the user's view point. Users of these clouds can share objects,

simply by dropping objects in inter-cloud-containers. Users explicitly specify the locations where they want to store objects.

4 PROTOTYPE

4.1 CaaS Overview

We developed Cluster as a Service by which a private cloud can be deployed from common computer resources.

The cloud on demand solution has a resource pool from which each private cloud allocates IT resources as they need them and releases them when they are not using them (Figure 4). The security is guaranteed by separating the network segments for each private cloud. When servers are released, the cloud on demand solution erases the storage before it allocates it to the other private clouds. For rapid elastic allocation, some servers in the resource pool have to be ready to run. These servers are moved from the resource pool network segment to the target private cloud network segment by changing the network configuration.

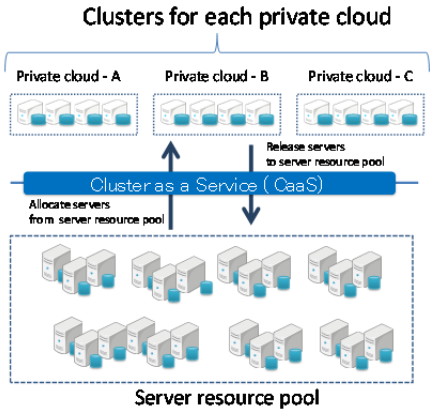


Figure 4: Cluster as a Service.

4.2 Requirements of CaaS

Req.1) Computer resources must be dynamically allocated to the clusters of different private clouds.
Req.2) Clusters must be securely separated.
Req.3) Software components of the cloud must be easily deployed on the clusters.

4.3 Design of CaaS

CaaS is designed to satisfy these requirements:

1) Two-layer implementation

The lower layer takes care of Req. 1) and Req. 2). The upper layer handles Req. 3). Moreover, each layer is programmable with web APIs.

2) Lower layer

The lower layer handles the operating systems of each node composing a cluster for using machine images. Nodes can be allocated to clusters dynamically from software and securely separated by using network technology, like virtual LANs, in the allocation. The lower layer also deals with erasing storage when servers are released. A prototype of this layer is dodai-compute (Dodai-compute).

3) Upper layer

The upper layer deals with deploying IaaS/PaaS software such as Hadoop, Grid Engine, OpenStack, and eucalyptus. Configuration management tools make it easy to deploy software on the nodes of clusters. A prototype of this layer is dodai-deploy (Dodai-deploy).

4.4 Dodai-compute

The lower layer dodai-compute is a system based on OpenStack nova to control operations (such as run instances from an image) on physical machines instead of VMs. Figure 5 illustrates the architecture of dodai-compute. The run instances, terminate instances, start instances, stop instances, reboot instances and associate address operations on physical machines can be done via EC2 APIs. The architecture of dodai-compute is as follows. Dodai-compute uses PXEboot via a cobbler library to bootstrap physical machines corresponding to run instance API calls. It also uses an OpenFlow controller to assign network segments to the physical machine. IPMI is used to control physical machines corresponding to the start instance, stop instance and reboot instance. The terminate instance operation is used to move physical machines to the machine pool network segment. The OpenFlow controller does this operation. The disks are physically cleaned up and become ready for the next launch. The associate address operation is done by an agent in each physical machine instance.

VLANs are used on some private clouds. When IaaS is deployed on them, we use OpenFlow technology, instead of VLAN, in order to separate network segments for each private cloud.

4.5 Dodai-deploy

The upper layer of dodai-deploy's specification and its prototype are described in this section. Dodai-

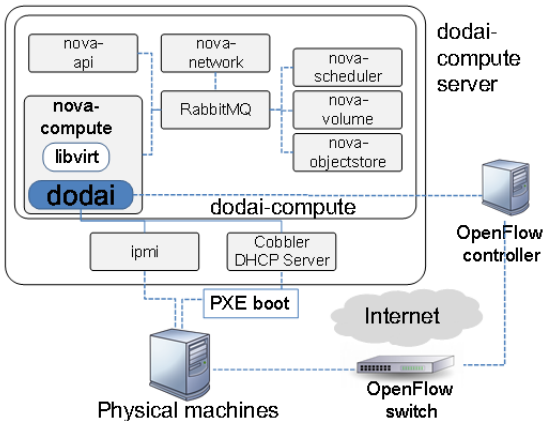


Figure 5: Dodai-compute architecture.

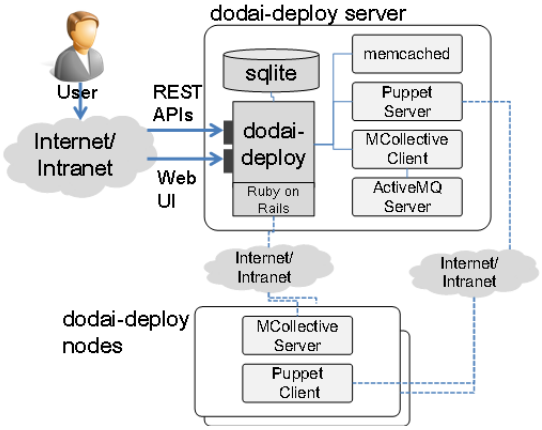


Figure 6: Dodai-deploy architecture.

deploy has the following functionalities:

1) Installation configuration proposal creation

Dodai-deploy runs according to a user installation plan called a 'proposal.'

2) Installation and un-installation

Software components are installed according to a proposal on target physical machine nodes and virtual machine nodes. Dodai-deploy can un-install software components, as well.

3) Test installation result

Automatic testing of the deployed IaaS and PaaS is an important functionality of dodai-deploy. Users can use these functionalities through a Web GUI and CLI. Figure 6 illustrates the architecture of dodai-deploy. The dodai-deploy server generates manifest files for the puppet configuration tool (Puppet) when users submit proposals requesting installations. The architecture was designed with fast deployment in mind to cope with the growing number of target machines. Parallel deployment is the key to achieving this goal but dependencies among software components have to be used to make usable deployment strategies. The actual parallel deployment procedure uses MCollective (Mcollective) to control many puppet clients.

4.6 Colony

We describe how to implement a geographically distributed inter-cloud storage service. Storage-I in Figure 7 should be a network-aware object storage service in order to make the remote application deployment rapid. The prototype uses OpenStack Swift as the base software. A prototype of this inter-cloud storage service is colony (Colony).

OpenStack Object Storage (code-named Swift is open source software for creating redundant, scalable data storage using clusters of standardized servers to store peta-bytes of accessible data. It is not a file system or real-time data system, but rather a long-term storage system for large amounts of static data that can be retrieved, leveraged, and updated. Object Storage uses a distributed architecture with no central point of control, providing greater scalability, redundancy and permanence.

Objects are written to multiple hardware devices, with the OpenStack software responsible for ensuring data replication and integrity across the cluster. Storage clusters scale horizontally by adding new nodes. Should a node fail, OpenStack works to replicate its content from other active nodes. Because OpenStack uses software logic to ensure data replication and distribution across different devices, inexpensive commodity hard drives and servers can be used in lieu of more expensive equipment.

Swift has proxy nodes and auth nodes acting as the front-end and storage nodes acting as the back-end for accounts, containers, and object storage.

The internal software components of the service are shown in Figure 7. The caching component makes the machine image launch fast. The dispatcher selects the nearest object replica in object storage service-I, even if there is no copy in the local cache.

4.6.1 How the Original Swift Works

The basic mechanism of downloading and uploading objects in the original Swift is as follows:

1) GET

The proxy server randomly chooses a replica from the ring and asks the storage server to send the object in which the replica resides.

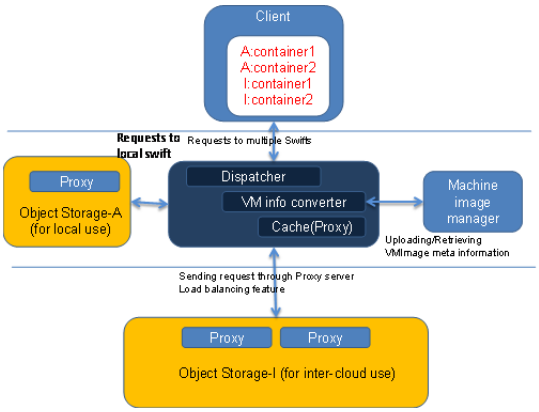


Figure 7: Software components of colony.

2) PUT

The proxy server knows the storage servers to which object replicas from the ring should be put and sends the objects to the all storage servers. The PUT operation ends when the all replica writes finish.

This implementation is based on an assumption that the replicas are concentrated in the network, for example, in the same data center. However, in our context, this assumption is not valid. Actually, if we apply the original OpenStack swift to storage-I, the GET and PUT operations take time when the randomly selected replica is far away. This is the reason why we have to make the Swift software network-aware.

4.6.2 Network-aware OpenStack Swift

1) How to make Swift network-aware

In the put operation, all replicas are written in the same site as the proxy server instead of writing them to the location the ring specifies. The replicas of the original positions are made asynchronously by the object replicator. After confirming the replication, the local copies corresponding to the replicas are deleted. In the get operation, the 'nearest' replica, instead of a random one, is chosen by the mechanism described in the next section. The proxy server works with the cache mechanism as well.

2) How to measure network distance

We use the zone information in the ring for the network distance measurement. The zone information consists of fixed decimal numbers, and we can allocate them freely. Therefore, we can use these decimal numbers to specify actual locations. Let's say the nodes in data center #1 are from zone-100 to zone-199, the nodes in data center #2 are from zone-200 to zone-299, and so on. By using this sort of convention, the software can know the network distance without our having to modify the ring structure or code related to it.

5 EVALUATION

In order to evaluate the prototype in a real context, we deployed and evaluated our cloud on demand solution as NII's research cloud (called gunnii). We also evaluated the prototype in a number of user scenarios.

5.1 Evaluation Environment

We deployed cloud on demand solution as our research cloud providing bare-metal cloud service to NII researchers on July, 2012. An overview of gunnii from the users' viewpoint is shown in Figure 8. NII researchers can extend their existing research clusters to this research cloud on demand.

Figure 9 shows how we use OpenFlow technologies with dodai-compute in this configuration. Dodai-compute provisions bare-metal clusters by using PXEboot and IMPI interfaces and allocates the bare-metal machines in OpenFlow closed networks, regions, on demand. It also connects these regions to corresponding existing closed networks of research groups, which are assigned individual VLAN-IDs by setting up suitable flow tables in OpenFlow switches.

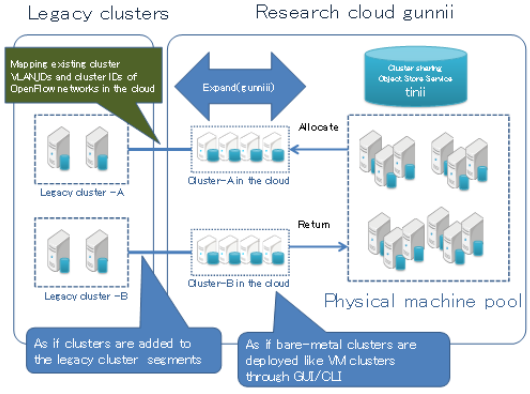


Figure 8: Overview of the NII research cloud, gunnii.

5.2 User Scenario According to U1 (Guaranteed Performance against a Abrupt Increase of the Load)

We also set up two private clouds. One was an OpenStack (OpenStack) IaaS private cloud (private cloud-A), on which web services of a simulated e-

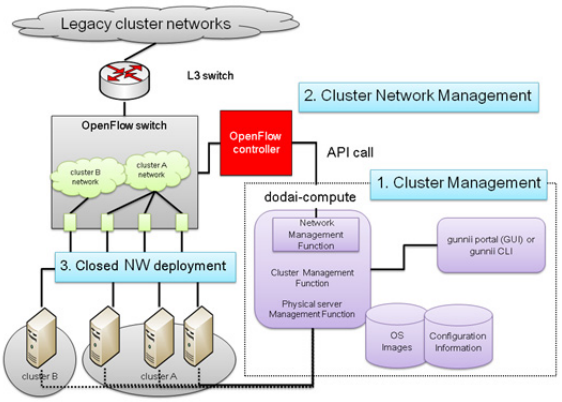


Figure 9: gunnii's architecture.

commerce company were hosted. The other was a Hadoop PaaS private cloud of a business intelligence company (private cloud-B), which was used for analyzing big data like web service usage logs.

The traffic of private cloud-A decreased during the period from 2 am to 5 am. To increase the utilization of IT resources, allocations to these two clouds changed depending on the amount of traffic in private cloud-A.

The business intelligence company was supposed to give daily feedback to the e-commerce company by using the big hadoop cluster on demand.

1) Cost evaluation

We verified the cloud on demand operations according to the user scenarios in gunnii. The verification points were as follows:

1. Can we change the size of the two clouds dynamically?

2. Can the services run continuously on the clouds even during the size change? Figure 10 shows the verification environment. Private cloud-A consisted of two servers: a master node which had master software components of OpenStack diablo software and included OpenStack nova, glance, swift, keystone and horizon and slave software. The other server only had slave software like nova compute and swift object servers.

First, two servers were allocated to private cloud-A by using dodai-compute and OpenStack software components were deployed with dodai-deploy. In order to check verification point 2, a virtual machine was launched on one of the OpenStack nova-computes.

When private cloud-A's traffic reached a peak, dodai-compute allocated another server to it and dodai-deploy configured the cloud to have three servers (Figure 11).

During this rerun, the application connection to the virtual machine was not interrupted. Nova-api

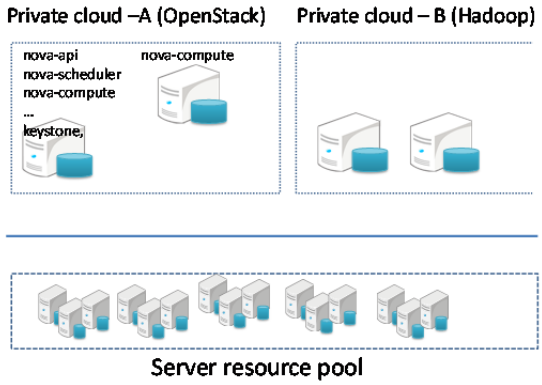


Figure 10: Verification environment.

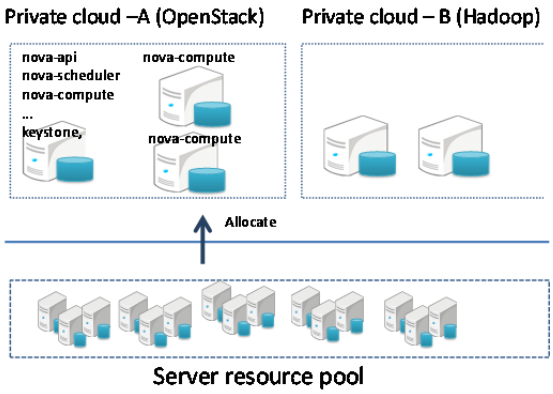


Figure 11: Private cloud expansion.

and other software were continuously available to users. This was possible because dodai-deploy can notice that software components are deployed and services are running already on the two pre-existing servers. It deploys software components only to the newly allocated server. Moreover, through the OpenStack nova mechanism, the nova-scheduler automatically recognizes the new nova-compute.

On the other hand, when a bigger hadoop cluster is needed, private cloud-A should release a server. In this experiment, we released the most recently allocated server, because it was not the server on which the virtual machine was running. In a real situation, however, we would need to monitor the allocations of virtual machines by nova-compute and need to live migrate some of them to servers that will not be released (Figure 12).

In the experiment, the newly allocated server was released by using a dodai-compute terminate-instance call. OpenStack nova detected the loss of one server for nova-compute, and it did not try to launch virtual machines on that server later.

1) Security

We verified that the network separation of the

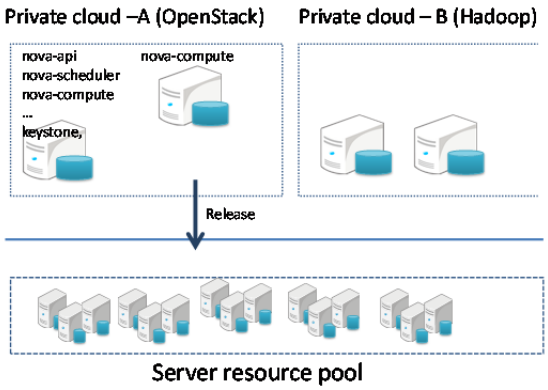


Figure 12: Private cloud reduction.

OpenFlow controller and the disk cleanup process in machine pool segment maintained the security of the user information. It was impossible to get into other clusters through the network and impossible to retrieve any information of the previous user from the physical machines.

2) Ease of application development

We verified that the cluster networks did not have restrictions on broadcast or multicast. Users can develop applications with network multicasting functionalities on elastic private environments. Moreover, we evaluated the deployment performance of dodai-deploy for OpenStack and Hadoop. Because of the concurrent deployments to the target nodes, the performance was almost flat regarding the number of nodes. However, for Hadoop, there was an 8% increase in deployment time in going from n=7 to n=8. The increase was due to the CPU constraints of the dodai-deploy server(Figure 13). We should be able to avoid this by scaling up or scaling out the dodai-deploy server.

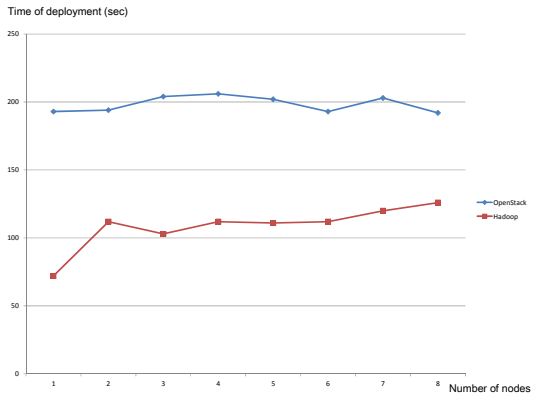


Figure 13: Deployment performance of dodai-deploy.

5.3 User Scenario According to U2 (Guarantee regarding Delay)

In this scenario, a user of a service provided by a cloud system goes on a business trip to a remote location. Because the longer physical distance causes a longer network delay from the site where the service is provided, the user may experience performance degradation as far as the response time goes.

1) Extension to wide area network configuration

The evaluation environment of gunii was in a data center configuration. However, our cloud on demand solution architecture allows for an easy extension to a wide area network. Figure 14 shows how we can make this extension.

2) Delay

The delay stays practically small because the cloud on demand solution can deploy the corresponding service in a data center nearer to the user.

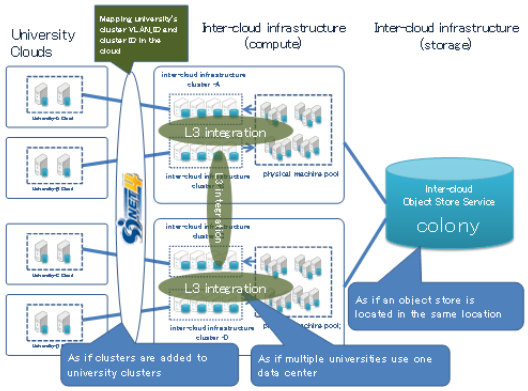


Figure 14: Cloud on demand in a wide area network configuration.

5.4 User Scenario According to U3 (Guaranteed Availability in the Event of a Disaster or a Large-Scale Failure)

In this scenario, the cloud system of a municipality is damaged in a natural disaster and cannot continue to provide its services.

The disaster recovery operations used the resources of the remote municipalities (Such measures would be pre-arranged) .

1) Cloud migration

We developed a cloud migration tool which can migrate OpenStack IaaS from site-A to site-B by using dodai-compute, dodai-deploy and colony, and we demonstrated it in public. It stored the OpenStack user database and snapshots as well as

configuration information for dodai regularly at site-A and restored them after reconstruction of the OpenStack at site-B using dodai (Figure 15).

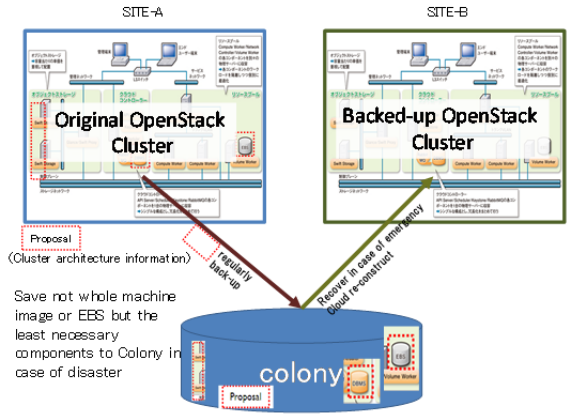


Figure 15: Cloud Migration.

Moreover, we could migrate any software supported by dodai-deploy, i.e., OpenStack, Hadoop, GridEngine and Eucalyptus.

2) Performance of inter-cloud object storage PUT and GET

By making the object storage service OpenStack swift network-aware, the inter-cloud object storage is almost equal in performance to local object storage for PUT and GET, which is described in (Yokoyama et al., 2012b).

5.5 User Scenario According to U4 (Service Continuity)

Normally, if a provider suspends its business, its customers need to re-register with different providers for similar services. To avoid such a situation, resources, applications, and customer ID data for the services provided by one provider can be transferred to the cloud systems of other providers in advance. Then, if its business is suspended, its consumers can use similar services provided by the other providers.

1) Cloud migration

As described in the previous section, however it not necessary for dodai to regularly store a user database, snapshots, or configuration information.

2) Performance of inter-cloud storage PUT and GET performance

Same as in the previous section.

6 CONCLUSIONS

We proposed a solution called cloud on demand and described a prototype implementation based on the dodai and colony projects. The cloud-on demand was proved to be feasible in the actual user scenarios in one data center. This architecture can be extended to wide area networks using SINET L2VPN and VPLS services if we plug the upper link from the OpenFlow switches into the SINET directly.

We are now constructing a new prototype of cloud on demand upon SINET, and we will evaluate its performance in this wide area network environment.

ACKNOWLEDGEMENTS

We would like to thank all the contributors to the dodai project, especially Shin-ichi Honiden, Yoshitaka Kuwata, Masashi Ochida, Osamu Habuka, Takahiro Shida, Guan Xiaohua, Motonobu Ichimura, Takahiko Yuzawa and Daishi Kimura.

REFERENCES

GICTF: http://www.gictf.jp/index_e.html.

Hiroshi Sakai, "Standardization Activities for Cloud Computing", NTT Technical review, Vol. 9 No. 6, June 2011.

Shigetoshi Yokoyama, Nobukazu Yoshioka, “Cluster as a Service for self-deployable cloud applications”, pp.703-704, Cluster, Cloud and Grid Computing (CCGrid), *2012 12th IEEE/ACM International Symposium*, 2012a.

Shigetoshi Yokoyama, Nobukazu Yoshioka, "An Academic Community Cloud Architecture for Science Applications," pp. 108-112, *2012 IEEE/IPSJ 12th International Symposium on Applications and the Internet (SAINT)*, 2012b.

Shigetoshi Yokoyama, Nobukazu Yoshioka, Motonobu Ichimura, “Intercloud Object Storage Service: Colony”, pp. 95-98, CLOUD COMPUTING 2012, The Third International Conference on Cloud Computing, GRIDs, and Virtualization, 2012a.

SINET: http://www.sinet.ad.jp/index_en.html?lang=english.

Dodai-compute: https://github.com/nii-cloud/dodai-compute.

Dodai-deploy: https://github.com/nii-cloud/dodai-deploy.

Puppet: http://puppetlabs.com/

Mcollective: http://puppetlabs.com/mcollective/

Colony: https://github.com/nii-cloud/colony.

OpenStack: http://openstack.org/

Shigetoshi Yokoyama, Nobukazu Yoshioka and Motonobu Ichimura, "A Network-aware Object Storage Service", pp.556-561, The 2nd International Workshop on Network-aware Data Management to be held in conjunction with SC12, 2012b.

Virtual Infrastructure Management in Private and Hybrid Clouds, Computer, 2009.
Sky Computing, Internet Computing, 2009.
Nimbus, www.nimbusproject.org, last access on December 24th 2012.
Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center, NSDI, 2011.
EGI-FedCloud: http://www.egi.eu/infrastructure/cloud/

A New Cloud Computing Governance Framework

Ahmed Shaker Saidah and Nashwa Abdelbaki
School of Information and Communication Technology, Center for Informatics Science, Nile University, Cairo, Egypt
ahmed.shaker@nileu.edu.eg, nabdelbaki@nileuniversity.edu.eg

Keywords: Security Framework, Governance Model, Cloud Computing.

Abstract: Nowadays, most service providers adopt Cloud Computing technology. Moving to Cloud creates new risks and challenges. The Cloud era is to outsource our services to Cloud Service Provider (CSP). However, we have to develop a strong governance framework to review the service level, to manage risk effectively and to certify that our critical information is secure. In this paper, we develop an innovative governance model. It is based on the theoretical Guo, Z., Song, M. and Song, J governance model for Cloud computing. We distribute Cloud Control Matrix (CCM) on the Guo's model categories. This turns the theoretical Guo's model to a practical model. Governance model alone will not allow us to bridge the gap between control requirements, technical issues and business risks. As a result, we introduce a new Cloud governance framework using the processes on the new Cloud governance model.

1 INTRODUCTION

Cloud Computing is a new term for an old service with new features. Many of us used to have an e-mail account during the last two decades. Data location, storage and processing are usually unknown to the user. In fact, this was a kind of Cloud service. Cloud was known as on demand infrastructure in the 90s and as Grid/Utility computing in the 2000s. Clouds and Grids are common in their vision, architecture and technology, but they differ in security, programming model, business model, compute model, data model and applications (Foster and Zhao et al., 2008, pp. 1-10). Earlier in these days, it was too risky to store our data outside organization premises; safety was a concern.

Data is the most valuable asset in any organization. It can be categorized as PI (personal information) or organizations' data. Nowadays, all internet users intensively process and store data on the Cloud. Cloud Computing depends on sharing of resources to gain economies of scale. It focuses on maximizing the effectiveness of the shared resources. Despite the benefits promised by Cloud computing, we see that essential improvement on technologies and operations governance are needed to enable widely adoption of Cloud services (Popovic and Hocenski, 2010, pp. 344-349). The best way to protect data outside organizationpremises is to define a policy to organize the relation between the owner and service provider. Policy definition requires well-developed information security governance framework (Borgman and Bahli et al., 2013, pp. 4425-4435).

It is mandatory for any organization to follow a framework for establishing information security governance environment. The framework will be utilized by the business across the organization (Mukherjee and Sahoo, 2010, pp. 31-34). We create a new Cloud governance framework for helping organization to govern the Cloud services. It is a measurable, sustainable, continually improving and cost effective framework on an ongoing basis (Li and Zhou et al., 2010, pp. 2843-2848) (Ahmad and Janczewski, 2011, pp. 372-379).

The rest of this paper is organized into six sections. Section II discusses related works in governance and Cloud computing. We will go through existing Guo's Model and show the gap between its theoretical model and practical world, and will go through the pros and cons of the model. We propose our new Cloud Computing definition in section III. Section IV illustrates our proposed new model of Cloud Governance. Our new governance framework is introduced in section V. Finally, conclusion and future work are presented in section VI.

2 RELATED WORK

Cloud Computing is a relatively new term in the computing world. The definition of Cloud Computing from NIST (2009) is very common and almost all other definitions are part of this definition (Mell and Grance, 2011).

Cloud Computing becomes a huge market. Relations between services inside the Cloud are complicated. Virtualization vendors use different APIs. This creates many obstacles and challenges when moving between Clouds. Infrastructure inside the Cloud contains many layers of shared resources. Software licensing and end users license and agreement have many parameters and stages (Li and Chinneck et al., 2009, pp. 33-40). Federations and access control between service provider premises and end user premises become vague (Copie and Fortis et al., 2013, pp. 1229-1234).

The Cloud services become a self-service through websites. Customers can customize orders by themselves, which mean that they need to access the Cloud via all connectivity facilities. The user can increase or decrease the usage of the resources that is distributed across all provider premises.

Cloud Computing service models (SaaS, PaaS, IaaS) can be deployed in public, private or mixed model. User Control is varying from model to other and increasing or decreasing depending on the features and capabilities provided by the service provider or needed by the customer, (Figure 1) (*NIST Cloud Computing Security Reference Architecture*, 2012).

In SaaS model as an example, the Cloud user accesses the web service through any type of connectivity via web browsers, and he does not have control to the infrastructure or applications running in the Cloud.

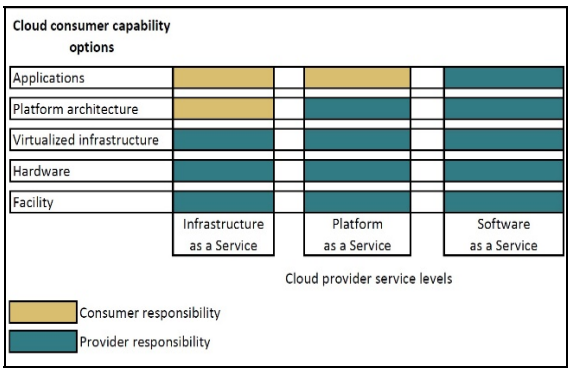


Figure 1: Control Level of Cloud Computing.

All of these features and facilities maximize security risks on the Cloud, open many doors for hijacking, and increase possible system vulnerabilities. Risks will be eliminated or mitigated by a robust governance framework (Furht and Escalante, 2010, pp. 3-21) (Buyya and Broberg et al., 2011, pp. 573-593).

Governance consists of policies, guidance, processes and decision-rights for a given area of responsibility. Corporate governance is to align processes and policies with business to ensure arrival to the business objectives. IT governance is part of the corporate governance and focuses on IT decisions and policies to ensure that IT assets are used according to the approved policies and procedures (workshop 116, Security, Openness and Privacy – Cloud Governance, 2011).

IT assets are huge and distributed between customer and service provider premises. They are classified into many types like people, policies, and equipment. It may be inside the organization or outsourced. Here, governance is required to control and maintain assets.

In many organization success stories, there is a harmony between managing the IT assets and decisions made by management. DELL Supply chain success story is an example of this harmony (Mcwiliams and White, 1999).

IT governance is responsible for aligning the IT assets with the business goals and strategy to deliver values to the entity. Cloud Computing Governance is part of the IT governance in the organization's governance hierarchy.

Governance is to control and secure our data outside our organization premises. It will align business speed to the Cloud and will cope with market demands. It helps also to initiate a new IT operating model (Mather and Kumaraswamy et al., 2009, pp. 176-202).

Organizations must ensure that the level of access they request is guaranteed into the Service-Level Agreement (SLA); uptime must be audited regularly to ensure that it conforms to the SLA (*IT Control Objectives for Cloud Computing: Controls and Assurance in the Cloud,* 2011). There are many ways to mitigate risks in the Cloud using technologies and policies (*Guidelines on Security and Privacy in Public Cloud Computing,* 2011). Cloud governance makes the decision easier and balances the investments and risks while gaining the Cloud benefits (Enisa.europa.eu, 2014) (Morin and Aubert et al., 2012, pp. 5509-5514).

Processes, policies, tools and even organization personnel will be unified under one framework that makes the workflow easier and give the business

some elasticity on applying the framework.

A Cloud governance should contain processes to apply Cloud Computing inside the organizations and applied controls to facilitate it. Moreover, it must adopt the organizational roles and responsibilities to ensure better support of implementing Cloud Computing governance. Finally, it should use all available technology tools that will help to apply the governance framework.

When implementing security governance, we need well-articulated policies and procedures including controls. Security controls is the key to apply security governance. CSA CCM is a well-defined industrial security control list (Cloudsecurityalliance.org, 2014). We will distribute these controls on the theoretical Guo's model for aligning the model with the Cloud market. We will demonstrate CSA CCM and Guo's model in the next two subsections (Guo and Song et al., 2010, pp. 1-6).

2.1 Cloud Control Matrix

The Cloud Security Alliance (CSA) Cloud Controls Matrix (CCM) is an initiative from CSA to determine the baseline of security. It leads the Cloud market and helps customers assessing the risks of all Cloud domains.

The CSA CCM provides a controls framework that covers almost all Cloud security domains. CSA relates it to the standards already in the market for IT Governance like COBIT. They mapped controls to the industry and practical life that help during the process of transferring the Guo's Model from theoretical model to practical model.

As a framework, the CSA CCM offers to the organization the required structure and details related to information security tailored to the Cloud industry. It covers all Cloud aspects and controls. Some controls are covered in IT governance models and other controls, related to the Cloud system, are brand new.

By mapping these controls to security standards that are already implemented in the market like COBIT and HIPPA, it helps in pointing to information security control required by business and management strategy (Sahibudin and Sharifi et al., 2008, pp. 749-753).

The main target of CSA CCM is to provide a standard management to security and operational risk that will face any organization implementing Cloud Computing in its infrastructure. This matrix is mitigating and minimizing security threats and vulnerabilities in the Cloud by providing controls to each domain that covers almost all Cloud security related topics.

CSA CCM contains eleven domains that cover all security issues related to Cloud computing. They divide it by function. It means that controls related to legal issues will be a domain and controls related to data governance will be a domain and so on.

Compliance is the first domain. It has six controls that cover audits, regulations, and intellectual property. It also reviews legislative, regulatory and contractual requirement. Data governance has eight controls that manage data objects containing information. It classifies and assigns responsibilities, communication, labelling, policies, and data destruction. Facility security has eight controls that secure working environment like physical access, site authorization and asset management. Human resource security has three controls that cover aspects related to humans like background screening and employment termination.

Information security is the largest domain in CSA CCM; it has thirty-four controls that take care of security management, policy, user access, training, benchmarking, encryption, security incidents, infrastructure and auditing. Legal is the domain that controls agreements and reviews contracts with the national and international laws. It has only two controls. Operation management is taking care of resources planning and managing procedures and equipment. Risk management is a very important part of the matrix. It predicts all risks happening in the Cloud or the project. It delivers a plan to control and mitigate risks. Release management controls planned changes in production environment and set policies and procedures to apply the new changes. It has five controls.

Resiliency is responsible for business continuity planning and environmental risks mitigation. It has eight controls. Security architecture is the last domain containing fifteen controls that address all regulatory requirements for customer access, data security, network security including infrastructure and applications.

2.2 Guo's Governance Model

The Guo's Governance Model can be identified as the first proposed academic governance model to our knowledge (Figure 2). It outlines the necessary components for Cloud governance. It was created based on four objectives of Cloud governance, which are service, policy, risk, and compliance management. It classifies the components of Cloud governance into three categories; policy, operational

and management activities.

There is a gap between the model and the real world, which we cover in this paper. We contribute in this paper to close this gap. The gap in the model can be identified after we apply controls in the CCM to the Guo's model. The CCM is a list of controls extracted from real Cloud business. We can apply it to any Cloud Computing system and be sure that most security aspects are covered. It helps transferring the theoretical model to an applicable one.

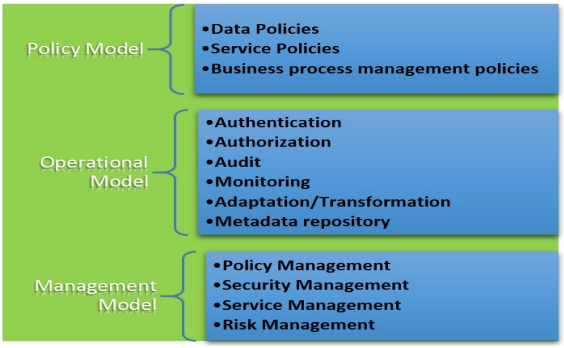


Figure 2: Guo's Governance Model.

First missing corner in the model is the aligning with business strategy. The gap between IT and organizational alignment obstructs the adoption of Cloud computing. In Cloud Computing system, an organization's IT team has to be upgraded from being only technologists towards being also information and business experts. Therefore, the organization should determine how Cloud Computing could best serve its business needs while addressing how it may affect its current IT organization and governance.

In the Cloud, the traditional roles of CIO, IT support, service provider and even user are changing dramatically. Organization applying security roles should align it with the whole organization's roles. It has to assure the harmony between controls and roles governing the organization. This integration makes implementation easier and changes the way employees can accept and apply these roles and responsibilities. This is the second missing corner.

Roles and responsibilities may change during or after implementation of the Cloud system. Therefore, change management should have a well-defined strategy because of the nature of Cloud. It changes periodically and rapidly more than any other fixed systems. Replacing defective items, applying patches, or upgrading firmware are a few examples of the change procedures needed in Cloud environments. Taking resources down for change, applying efficient change management techniques is a key to survive in the Cloud. Change management is the third missing corner in the model.

Feedback process in a successful system improves the efficiency and reliability. Using Cloud feedback process gives all parties the ability to ensure that the system performs as expected. Guo's model does not clearly state this type of feedback. Service feedback is the fourth missing point in the model.

Due to asset distribution in the Cloud environment, asset management will be an important part of Cloud governance. It should be stated clearly. Assets management changes depending on the type of implementation and the agreements between the parties. Asset management is another missing point in the model.

Last missing point is the exit strategy. It contains contract ending, data and systems maintenance and it manages assets before and after exit. A Cloud exit strategy should be as simple as putting data in the Cloud, but this is far from the case, especially in case of proprietary public Clouds.

3 OUR PROPOSED DEFINITION OF CLOUD GOVERNANCE

Cloud governance definition is still in the developing mode. Cloud Computing Use Discussion Group (2010) defines Cloud governance as "the controls and processes that make sure policies are enforced" (Cloudusecases.org, 2010). Many organizations and groups define the Cloud Computing governance in a different ways.

According to our definitions, defining policies is important, but defining processes to apply these policies is more important. Cloud governance model should be aligned with corporate governance and IT governance. Moreover, it has to comply with organization strategy to accomplish business goals.

In our experience, Cloud governance has to support business strategy and to ensure service value, service quality and security irrespective of the control and locations of the services. Therefore, we define Cloud governance as:

"Cloud governance is a framework applied to all related parties and business processes in a secure way, to guarantee that the organization's Cloud supports the goals of organization strategies and objectives."

4 NEW MODEL PRESPECTIVE

Cloud governance is challenging. Technology is faster than the standards. We have to take into consideration the future expansion and update.

Building Cloud governance increase the ability to its technology to grow not to hinder it (workshop 116, Security, Openness and Privacy – Cloud Governance, 2011). The governance process guarantees the rights of all stakeholders.

The challenge is the trade-off to achieve a governance model's implementation plan agreed by all parties. The plan should be elastic and customizable to all models and business cases. The plan has to tolerate moving between the Service Providers (SP) and their customers.

The vague nature of information interchange, the ubiquitous connectivity and the old static controls, all require new thinking with regard to Cloud computing. How can we implement the governance model without knowing the practical controls from real world and its implementation?

Therefore, what we already did is transforming the Guo's model to an applicable framework. We distribute controls under each model and its components to illustrate the practical implementation of the governance. We categorize the controls into two main categories, normal controls and key controls. We reserve developing the criteria to measure each control for future publications.

As we have seen, the Guo's model is not a process oriented. To overcome the problem, we redefine its three models (policy, operational, and management) to be processes. Then, we correlate the different CCM controls to each relevant process. Thereafter, we create new processes for the controls that are not relevant to any existing process.

We have to go deeply inside each model to determine the related controls to achieve the goal of this model. The model should be understandable and the structure of the model should be logical and reasonable.

To solve these issues we add, modify and update few categories of the Guo's model (Table 1). In the Management Model, we define clearly the Roles and Responsibilities under the Security Management. We use it in aligning Cloud system roles with the organization's roles and responsibilities. In addition, we have added Service improvement to the Service Management to be used as a key of the feedback to increase system reliability and efficiency.

Table 1: New Cloud Governance model.

New Governance Model		
Policy Model	Operational Model	Management Model
Data Policy	Authentication	**Policy Management**
Service Policy	Authorization	• Generic Policy Ontology
Business Process Management Policy	Audit	• Application Specification Ontology
Exit Policy	Monitoring	• Policy Repository
	Adaptation/ Transformation	• Policy Specification Service
	Metadata repository	**Security Management**
	Asset Management	• Integration
	• Human resources	• Privacy
	• IT Assets	• Access
	Configuration management and documentation	• Jurisdiction
	Capacity planning	• Roles and responsibilities
		Service Management
		• Service Discovery
		• Service Delivery
		• SLAs management
• Note		• Errors and exceptions management
Grey cell is the identified gap.		• Auditing and Logging
Bullet is a sub items		• Service improvement
		Risk Management
		Change management

Change Management will be part of the Management Model due to the rapid changes in the

Cloud service either from the customer side or from the provider side.

Under the Operational Model, we define the asset management, configuration management and capacity planning. It supports the organization to operate its own Cloud or the Cloud services they use. Moreover, we have added Capacity planning to enforce changing the way of thinking inside the organization regarding the Cloud service. It helps in the planning phase and it guides the organizations to meet future changing demands of its services. Moreover, it supports the organization to take the right decision about Cloud service.

Finally, we have added the exit policy to be stated clearly and be defined in any contract separately to well define the procedures to be done to maintain user systems and data after ending the Cloud service or moving to a new provider. It supports both sides to be secure before or after service contracting.

Now we have processes in the new model and each process has its own controls. Each control has inputs and outputs. Control's measures and tools depend on the deployment model. We create a framework and put each process in its suitable stage. The new framework is a conceptual structure to serve and guide organizations in Cloud Computing adoption process.

5 NEW CLOUD GOVERNANCE FRAMEWORK

The changes being driven by Cloud Computing and the growing sophistication of attackers do represent new challenges. We solve these challenges by creating the Cloud Governance Framework to control people, data, applications and infrastructure. Our security framework provides a more integrated, intelligent approach to Cloud Governance.

An intelligent framework must improve itself continuously; it has to have a feedback and service improvement process. We develop a new framework with five stages to achieve this goal (Figure 3).

It also solves the weakness of organization strategy alignment. The stages are:

- Strategic trigger
- Define and align
- Build and implement
- Deliver and measure
- Operate and feedback

Strategic trigger is the first stage. It is the event that initiates the need to use the Cloud computing.

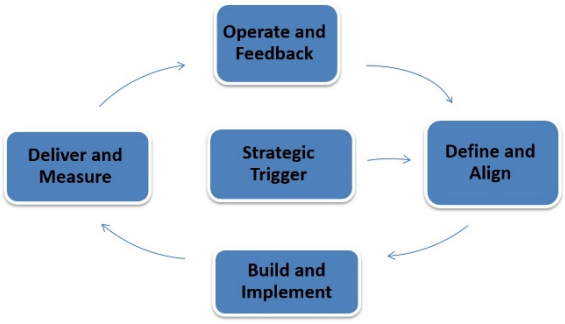


Figure 3: New Cloud Governance Framework.

Business need is the main trigger for using the Cloud services. Other trigger may be gaining market share due to strong competition in market. The company needs a competitive edge. We use Cloud services to comply with a standard or a government rule. The major trigger is the technical need. An SP delivering services needs technically a Cloud service (Heier and Borgman et al., 2012, pp. 4982-4991); for example, E-mail services.

This stage contains four processes. Business process management policy defines interrelations between Cloud-based services. It analyses the business and considers the service process reuse. Service discovery finds and discovers the existing services and available technologies for new services. Capacity planning reviews the existing environment and future business extensions to plan the best way technically and financially to achieve business goals. Exit policy is mandatory. Business needs changes to cope with the market. It may require ending the Cloud service. Exit the Cloud service is more complicated than joining and entering it. A well-defined plan is mandatory before starting to use Cloud service.

Define and align stage is the planning phase of adopting the Cloud service or transforming the existing environment to the Cloud. It ensures that the Cloud services are aligned to the business needs and actively supports them. Organizations using a Cloud require their service to be successful. If processes and services are implemented, managed and supported in the right way, the business will be more successful. This means cost reduction, revenue increase, and achieving its business objectives. It is the most important phase helping the decision makers with the economic and technical preparations for Cloud services.

This stage contains six Processes. Data Policy defines data's physical and logical model, in addition to data performance and stability. Service policy builds a service dictionary. It analyses the

integration and separation of the service based on deployment model. Policy management determines and reviews the service policy. Moreover, it reviews the violation and solves the policy conflicts in order to prevent further problems. Risk management defines risks when moving to the Cloud. It plans a mitigation process and determines residual risks. Risk plan has to be reviewed with the organization and provider policies. Jurisdiction is an important process. Law and regulations vary from country to another. Organizations must review country laws where data is to be stored and processed. Integration is a mandatory process if you have an existing infrastructure. It plans the integration between the existing environment and the Cloud service.

Build and implement stage covers issues related to people, processes and infrastructure technology. It ensures cost-effective and the high quality provision of Cloud service necessary to meet business needs. The blurred lines between the traditional technology and Cloud services management means that an updated approach to managing Cloud implementation is needed. This stage contains eight processes. Authentication determines the authentication mechanism that will be used in the Cloud and between organization systems and Cloud. Authorization is the level of access that will be granted to users from the organization side and from the provider side as well. Metadata repository is the storage of policy. It considers the location of polices and roles. Asset management monitors and maintains things of value to an organization. It manages the logical and physical assets and even human assets. Configuration management and documentation establishes and maintains performance, functional and physical attributes. It also establishes and maintains configurations within Cloud service throughout its life. Roles and responsibility is a dictionary, which determines the roles and the responsibility of each contributor in the Cloud service. Privacy considers the data encryption and the location privacy. Access takes care of the access policy in the Cloud because of using shared resources.

Deliver and measure stage ensures that the implemented service is aligned with the planned services. It measures and compares the outputs with the references that were determined before. This stage contains four processes. Service delivery is moving the service to the execution environment. SLA Management ensures that all service levels are met. It reviews contract for penalties. Errors and expectation management reviews the current environment with the planned one. It analyses the running systems and reports the existing errors. Auditing and logging track all the activities and define whom, when and where this activity was done. It helps during external and internal auditing.

Operate and feedback stage is the final stage in the framework. Feedback for many organizations becomes a temporary project recalled only in case of malfunction or failure that affects the business. After resolving the issue, the concept is forgotten until the next failure occurs. The most important task starts after implementation. How do we gain benefits of using the new service? How do we measure, report and operate the new service to improve the service delivery? This requires wise decisions to operate and control feedback. It clearly defines goals, documented procedures, and identified roles and responsibilities.

This stage contains four processes. Monitoring collects transaction and access data to present a service statistics. It helps the management to review the existing environment and to plan for the future expansion. Adaptation/transformation manages the unavoidable consequences and changes in the running service. Service improvement assesses measures and improves everything in the system. It uses all the data collected in the execution phase. Change management transforms the service to a desired future state. Due to rapid changes in technology, the organization must cope with these changes. All changes have to be approved from all parties.

We can apply this framework to any Cloud system. We need controls and tools that can activate each process inside the framework. We have to state controls under each model and its components. We classify the controls into two types, key control and normal control. Key control is the control that will be mandatory and necessary to apply this process into the framework. Normal control is the control that has some inputs but is not mandatory to achieve the main goals of the process. We have distributed ninety-eight controls from CSA-CCM on each process in the framework. Then, we determine the key and normal control. Due to limited space, we will publish the details at many future publications.

6 CONCLUSION AND FUTURE WORK

A Cloud system has different deployment models and architecture. Although it offers an economy of scale solution to the market, it creates new risks and

challenges in the IT environment. In this paper, we introduce our new Cloud Computing governance model that represents a perspective combination of theoretical and practical implementation. We turn the Guo's theoretical model to a practical model to enable applying it to the industry. We identify the gap using CCM, and then identify controls related to each process and its effect using CCM. We add, modify and update the missing corners in the model. We create a new governance framework. It is a five stages framework with a service feedback. Each stage has few processes. Each process contains controls. Each control has inputs, outputs, and tools to activate and measure it. The framework is suitable for all Cloud deployment models. In the future, we will apply the new governance model and framework to all Cloud models (SaaS, PaaS, and IaaS). We will specify inputs and outputs to each control. We will define the RACI (Responsible, Accountable, Consulted, and Informed) Model and identify persons that must be informed and accountable based on the deployment model. In addition, we will extract and develop SLA from the new Cloud governance model. We will relate controls effect directly the SLA.

REFERENCES

Ahmad, R. and Janczewski, L. 2011. Governance Life Cycle Framework for Managing Security in Public Cloud: From User Perspective. pp. 372-379.

Borgman, H. P., Bahli, B., Heier, H. and Schewski, F. 2013. Cloudrise: Exploring Cloud Computing Adoption and Governance with the TOE Framework. pp. 4425-4435.

Buyya, R., Broberg, J. and Goscinski, A. 2011. *Cloud computing*. Hoboken, N.J.: Wiley.

Cloudsecurityalliance.org. 2011. *Cloud Controls Matrix (CCM): Cloud Security Alliance*. [online] Available at: https://Cloudsecurityalliance.org/research/ccm/.

Cloudusecases.org. 2010. *Cloud Computing Use Cases group*. [online] Available at: http://Cloudusecases.org.

Copie, A., Fortis, T., Munteanu, V. I. and Negru, V. 2013. From Cloud Governance to IoT Governance. pp. 1229-1234.

Enisa.europa.eu. 2014. *Cloud Computing Risk Assessment — ENISA*. [online] Available at: http://www.enisa.europa.eu/activities/risk-management/files/deliverables/Cloud-computing-risk-assessment.

Foster, I., Zhao, Y., Raicu, I. and Lu, S. 2008. Cloud Computing and grid computing 360-degree compared. pp. 1-10.

Furht, B. and Escalante, A. 2010. *Handbook of Cloud computing*. New York: Springer.

Guidelines on Security and Privacy in Public Cloud Computing. 2011. [e-book] USA: NIST. Available through: http://csrc.nist.gov/publications/PubsSPs.html http://csrc.nist.gov/publications/nistpubs/800-144/SP800-144.pdf.

Guo, Z., Song, M. and Song, J. 2010. A governance model for Cloud computing. pp. 1-6.

Heier, H., Borgman, H. P. and Bahli, B. 2012. Cloudrise: Opportunities and Challenges for IT Governance at the Dawn of Cloud Computing. pp. 4982-4991.

IT Control Objectives for Cloud Computing: Controls and Assurance in the Cloud. 2011. [e-book] ISACA. http://www.isaca.org/Knowledge-Center/Research/ResearchDeliverables/Pages/IT-Control-Objectives-for-Cloud-Computing-Controls-and-Assurance-in-the-Cloud.aspx.

Li, J. Z., Chinneck, J., Woodside, M. and Litoiu, M. 2009. Deployment of services in a Cloud subject to memory and license constraints. pp. 33-40.

Li, X., Zhou, L., Shi, Y. and Guo, Y. 2010. A trusted computing environment model in Cloud architecture. 6 pp. 2843-2848.

Mather, T., Kumaraswamy, S. and Latif, S. 2009. *Cloud security and privacy*. Beijing: O'Reilly.

Mcwiliams, G. and White, J. 1999. Dell to derail: Get into gear online. *Wall Street Journal*.

Mell, P. and Grance, T. 2011. *The NIST definition of Cloud computing*. Gaithersburg, MD: Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology.

Morin, J., Aubert, J. and Gateau, B. 2012. Towards Cloud Computing SLA risk management: issues and challenges. pp. 5509-5514.

Mukherjee, K. and Sahoo, G. 2010. Cloud Computing: Future Framework for e-Governance. *International Journal of Computer Applications*, 7 (7), pp. 31-34.

NIST Cloud Computing Security Reference Architecture. 2012. [e-book] USA: NIST. Available through: http://csrc.nist.gov/publications/PubsSPs.html, http://collaborate.nist.gov/twiki-Cloud-computing/pub/CloudComputing/CloudSecurity/NIST_Security_Reference_Architecture_2013.05.15_v1.0.pdf.

Popovic, K. and Hocenski, Z. 2010. Cloud Computing security issues and challenges. pp. 344-349.

Sahibudin, S., Sharifi, M. and Ayat, M. 2008. Combining ITIL, COBIT and ISO/IEC 27002 in order to design a comprehensive IT framework in organizations. pp. 749-753.

Workshop 116, Security, Openness and Privacy – Cloud Governance. 2011. *Internet Governance Forum*. [online] Available at: http://igf.wgig.org/cms/component/chronocontact/?chronoformname=WSProposals2011View&wspid=116.

Cloud Governance by a Credit Model with DIRAC

Víctor Méndez Muñoz[1], Adrian Casajús Ramo[2], Ricardo Graciani Diaz[2] and Andrei Tsaregorodtsev[3]
[1]*Computer Architecture and Operating Systems Department, Universitat Autònoma de Barcelona (UAB), Edifici Q, Campus UAB, ES-08193 Bellaterra, Spain*
[2]*Department of Structure and Constituents of Matter, University of Barcelona, Diagonal 647, ES-08028 Barcelona, Spain*
[3]*Centre de Physique des Particules de Marseille, 163 Av de Luminy Case 902 13288 Marseille, France*
vmendez@caos.uab.es

Keywords: Cloud Computing, Federated Cloud, Cloud Governance.

Abstract: Nowadays, different eScience actors are assuming the Federated Cloud as a model for the aggregation of distributed cloud resources. In this complex environment, service delivery and resource usage is an open problem, where multiple users and communities are committed to particular policies while using federated resources. In the other hand, DIRAC has become a multi-community middleware fully interoperable in Distributed Computing Infrastructures (DCI), including several cloud managers. Furthermore, DIRAC is able to federate Infrastructure as a Service (IaaS) to provide Software as a Service (SaaS) in a transparent manner to the final user. This paper defines a credit model for the resource usage providing automatic management in federated cloud. It is presented a prototype of this model, which is implemented with DIRAC, describing a test for the model assessment and drawing up conclusions for a production level federated cloud governance.

1 INTRODUCTION

The benefits of cloud computing are related with a set of key features. Cloud uses Internet access to remote services or resources supported by virtual storage and computing. It is provided on-demand and depending on user workload it can scale up or down without affecting the service level. A main asset of these features is a virtual running environment deployed for particular user requirements, instead of a bare-metal platform under provider specifications. By this mean Cloud is saving operational costs and simplifying software maintenance. Moreover, the cloud scaling features are of great importance for an efficient use of the resources.

The cloud key features can be implemented in different ways, depending on the particular physiognomy of the cloud deployment model. In this sense, federated cloud model is the evolution of private clouds, aggregating not only IaaS, but also additional services for eScience communities such as monitoring, authentication, authorization, marketplace and accounting (Simon et al., 2012).

DIRAC (Tsaregorodtsev et al., 2008) has evolved from single community to multi-community framework. The cloud management of DIRAC follows Rafhyc architecture (Méndez et al., 2013b) for clouds of hybrid nature. DIRAC is using Cloud resources in the LHCb production (Ubeda et al., 2014), federating private clouds supported by OpenStack (OpenStack, 2014) and OpenNebula (Llorente et al., 2011). Another production level use of cloud was in Belle Monte Carlo simulation campaign with AWS (Lerner, 2006) using Amazon EC2 interface (Graciani Diaz et al., 2011).

This paper is focused in the next challenge of providing federating cloud in multi-community environments. For this purpose it is necessary to leverage the use of the resources between different communities. This is achieved by a governance between the policies of IaaS and the policies of communities. The main idea is to assign resource credits in terms of CPU, memory or IO usage (temporal and permanent storage as well as network resources), to the running campaigns of the communities. Such credits are consumed on-demand, with a pay-per-use approach, but instead of legal tender resource credits are used. This pay-per-use concept can be transparently adopted in the case of commercial clouds prices, where the credits are simply pre-payment. Main cloud managers are providing billing systems which can be enabled for pricing. However, non commercial clouds are coming from capital expenditure (Capex) model in the use of the resources. In this model user invests capital

in ownership infrastructures, or delegates invest to a computing hosting institution. Then, resources are allocated without a pay-per-use on-demand scheme, just relating investment negotiations with available power and storage capacity. A first step to provide an on-demand price is to have a total cost breakdown of the resources in an updated system, then to apply a commercial margin considering the market competence. Significant advances in the last years (Cohen et al., 2013) (Konstantinou et al., 2012) are targeting a costs management level. In the meantime, the presented model based in resource credit metrics is an affordable approach to offer on-demand resources in non commercial IaaS sites. Furthermore, it is compatible with future metrics, when price management will reach maturity in non commercial clouds.

An additional important point of the provided governance is about keeping the key features of cloud computing, with elastic allocation of resources as well as Virtual Machine (VM) configuration to run any scientific software.

Section 2 describes a previous approach of federated cloud with DIRAC. The contribution of the paper is a credit model for cloud governance in Section 3. Additional contribution in Section 4, describes a prototype implementation with DIRAC and test results for model assessment is shown in Section 5. Conclusions and future work can be found in Section 6.

2 PREVIOUS APPROACH OF FEDERATED CLOUD WITH DIRAC

DIRAC has a cloud extension integrating different IaaS providers which need to fulfil some requisites (Méndez et al., 2013a). In addition to the mentioned Amazon, OpenNebula and OpenStack clouds, CloudStack (CloudStack, 2014) has also been integrated with DIRAC (Méndez et al., 2012) (Albor et al., 2011). Beyond providing the aggregation of different cloud managers, DIRAC includes federated services for eScience communities like monitoring, authorization or image metadata catalog.

Regarding the key features of cloud computing, DIRAC cloud extension allows VM scheduling policies for different possibilities of VM horizontal auto scaling set-up. The scale up can be more aggressive and scale down softer, to reach better job response times. On the other hand, soft scale up and hard scale down can be set-up to reduce VM overheads and therefore a better efficiency, when job response time is not a constrain.

Another key feature is the image management by generic contextualization to configure the VMs depending on the SaaS requirements and the IaaS endpoint within to be deployed. I.e. DIRAC can use a golden image which is configured for different purposes and IaaS environments. This image management saves operational costs and minimizes complexity. Two contextualization approaches are contemplated, HEPiX contextualization for *cernvm* images (Bunic et al., 2011), and *ssh* contextualization for generic images, which require an available public IP and *sshd* service running on boot. Both of them can be combined with *cvmfs* software repository (Jakob Blomer, 2010), simplifying the software distribution in the VMs.

Fig. 1 shows the main part of VM allocation and job brokering. User submits jobs to the DIRAC Workload Management System (WMS), which are initially stored in the Task Queue (TQ), with an associated SaaS workload. Then, VM Scheduler is checking each of the queues and depending on workload it is instantiating new VMs with specific requirements for the queued jobs. Latter, when the VM is created, contextualized to run DIRAC VM agents and configured with SaaS requirements, then, Job Agent starts and secured connects to the Matcher to get the corresponding SaaS payload.

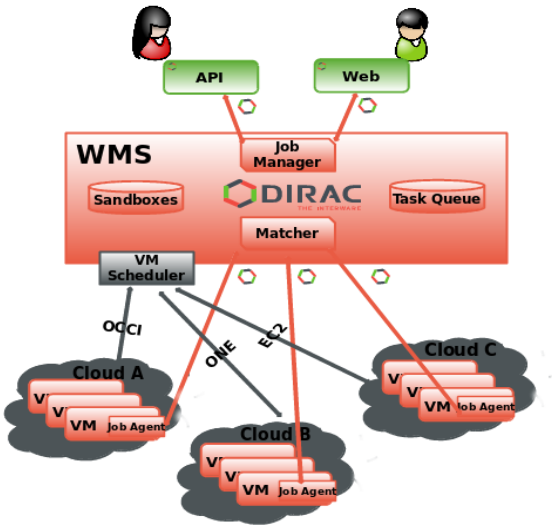


Figure 1: DIRAC VM schedulig with job brokering.

For simplicity, further VM features are not drawn in Fig. 1. It is worth to mention the VM stoppage and status management. A VM runs many jobs until some stoppage conditions are reached. VM Monitor agent is in charge of VM status update, reporting statistics to the VM Manager, as well as VM stoppage control. VM Monitor is a client of the VM Manager

service at the DIRAC server side. Stop command can be launched from VM when no jobs are running in the VM for a certain period, or it can also be requested to stop by an authorized external request from VM Manager, then, VM Monitor is doing a graceful stoppage after current job is finished.

3 CREDIT MODEL FOR CLOUD GOVERNANCE

This section presents a credit model for cloud governance of multi-community environments for the use of federated cloud resources. Fig. 2 shows such model scheme. In the bottom of the diagram supply is supported by Resource Providers (RP). In the top of the diagram User Communities (UC) demand services. Cloud Governance Actors (CGA) are found in the mediation between RP and UC.

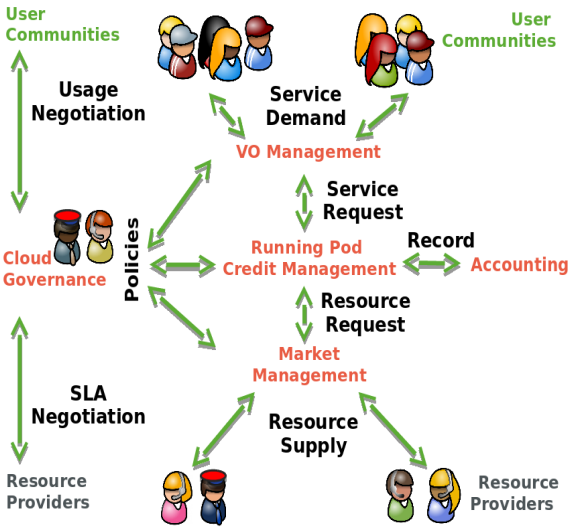


Figure 2: Credit model for cloud governance.

Previously to matching resources and services, it is necessary to negotiate the governance conditions. For this purpose each actor cover basic actions. GCA takes the central part of the governance management. To this concern it is necessary to have an overview of the hold production, providing scientific directions for the UC, including technical guidance of how to run scientific software in the Cloud. GCA also require to have an overview of the different RP technologies as well as a know-how in federated cloud management technologies. Cloud governance process is starting in the left of the diagram, from bottom-up flow, Service Level Agreements are continuously negotiated with every RP to obtain an overall resource supply power. Considering this power as a constrain of the model, usage is negotiated with every UC attending to scientific directions for the different use cases, as well as the UC financing contribution to the overall system.

Governance specifications are updated through a continuous negotiation process. Once supply and demand are identify, it is possible to define governance policies to be apply in three levels of the credit model, which are shown in the central part of Fig. 2. Usually, a user community is a Virtual Organization (VO) in terms of authorization in the use of resources. Eventually, if a single VO is composed of many UCs, it can be broken down into virtual groups. VO policies describe which *running pods* are assigned for each UC or virtual group, to be able to run their SaaS request. The running pod is an abstraction of the requisites to be included in a VM which can be submitted to certain IaaS providers. Secondary, VO policies can define external links to third party VO authorization for user and group management. Therefore, user service requests can be matched with specific running pods.

Running pod policies define how many overall resource credits are assigned for an specific period of usage. Furthermore, policies also describe which IaaS RP can be used in the running pod, since UC can use a subset of the federated resources to meet with particular SLA running requisites. Image requirements for a SaaS running pod are also defined, including the contextualization parameters for the image configuration to deploy the necessary VMs to provide the requested SaaS. Thus, a SaaS request produces several resource request attending to the workload.

Market management policies define maximum available resources in each RP, including total computing power, as well as the maximum VCPU, memory and disk per VM. It is a market with trades between resources and credit. In addition, it is possible to define some opportunistic use of the resources which can be allocated for those running pods without credit left. When the overall power demand is larger than the power supply, then, it is required to define overflow policies for the running pods, between several options as follows:

- To increase the power supply with third party RP, for example using cloud bursting. Another option in hybrid clouds is to include in the federation the use of commercial providers, delegating to such IaaS providers the responsibility of providing as much power as needed. This is the ideal solution in terms of elasticity features, but it requires flexible budget to acquire public cloud resources on-demand and imposes additional financial issues.
- To enable priorities between running pods so that whether power demand is larger than supply, run-

ning pods with less priority will fall into larger response times. In this manner, higher priority running pods are less affected when no more computing power is available. This option can be applied simultaneously with the first option.

- A variant of the priority approach is the option of stopping VMs when demand is larger than supply. This feature can be configured for the running pods which have non elastic requisites. This does not resolve all the possible power requirements, but it can be applied to first and second option for a sensible use of the resources.

The second and third options are constraints to the elasticity which is one of the assets of cloud computing. Furthermore, it is necessary to consider additional stoppage policies to avoid resource blocking of some running pods with others of the same priority. Grateful stoppage avoids the loses of the current payload when stopping a VM to free the resource for other running pod.

The right part of the Fig 2 is the necessary accounting of the resource and credits usage. An example of such federated cloud accounting system could the one promoted by EGI Fedcloud (EGI-FedCloud, 2014) using APEL (APEL-EMI, 2014) system. APEL was originally developed for grid infrastructures, and currently, it has available plug-ins for OpenStack and OpenNebula, while more cloud managers are coming. APEL is considering the resource usage per VO basis, but a direct metrics on credit accounting. Credit accounting could be separated system, but will fall into inconsistencies between the two systems. For this reason double accounting is something to avoid, so it would be mandatory to integrate RP accounting per VO basis with credit model accounting which is per running pod basis.

4 A PROTOTYPE OF CLOUD GOVERNANCE WITH DIRAC

Cloud governance prototype is a modification of previous DIRAC cloud extension, including an implementation of the credit model described in Section 3. Only VCPU hours are considered for the prototype metric leaving aside other metrics like memory or IO usage. New configuration sections have been included in the DIRAC configuration service (CS) to support governance parameters. Core functionalities has been included in the VM Manager to attend credit status request of the VM Scheduler and credit updates from the VM Monitor of each VM.

Governance process starts defining CS parameters to associate a running pod with a particular group of users. A user is part of many groups corresponding to the running pods which is able to use. The prototype adds the running pod governance parameters:

- Maximum number of deployed VCPUs
- Running pod campaign start and end dates
- Running pod total VCPU hours of credit for the campaign.

Deployed VCPUs are the VM which have been submitted even if they have not reached a running status jet. They are corresponding to VMs which are in status of booting, contextualizing or running. So maximum running VMs can be lower than maximum deployed, or equal, but never greater. For governance purposes two new parameters are included at each IaaS configuration section:

- Maximum number of deployed VCPUs
- Maximum number of opportunistic deployed VCPUs

VM Scheduler has a submission logic based in previous features as well as in the credit model. For each active running pod, VM Scheduler checks Task Queue jobs and VM horizontal auto scaling policy. Whether TQ and VM horizontal auto scaling policy indicate the possibility of a new VM creation, further governance conditions have to be checked with the following algorithm:

```
If runningPod(deployed VCPUs < max VCPUs):
  If there is credit left:
    list = IaaS(deployed VCPUs < max VCPUs)
  else:
    list = IaaS(deployed VCPUs <
                  max oportunistic VCPUs)
  if list not null:
    candidate = random(list)
    submit(VM, candidate)
```

- Algorithm: Credit checking -

Algorithm checks first if the maximum number of deployed VCPUs for a running pod in a particular moment has been reached. It the limit has not been reached, then it checks if there is credit in VCPU hours for the running pod. If there is credit left an IaaS site candidate list is created with those sites which have not reached the maximum number of deployed VCPUs of each site. If there is no credit left, then the IaaS site candidate list is created with those sites which have not reached the maximum number of deployed VCPU for opportunistic purposes of each site. Finally, a random selection from the candidate list is chosen to submit a new VM.

In parallel to the VM Scheduler process there is a continuous credit update. Each VM is running a VM Monitoring agent, which is reporting periodical heart beats with statistical information to the VM Manager. For governance purposes, heart beat includes a parameter of the number of VCPUs of the VM. On the server side, the VM Manger sets instance history record in a database, calculates the gap between the last heart beat of the instance and the current one, and adds the VCPU hours to the used VCPU hours in the running pod database. The VCPU is wall-time since the resource is allocated even if not real CPU is used, same as commercial clouds scheme.

5 PROTOTYPE TEST

This section presents a test designed in order to evaluate elasticity key feature, including scale-up and down. It also evaluates the overall system response when some of the running pod is active but without credit left, so it has to run the payload using opportunistic resources. The test is not aiming a stage with a power demand larger than power supply. In this sense it is a test with no job response time constrains, the target is to validate the core features of the credit model for cloud governance, particularly about elasticity.

5.1 Test Set-up

Test set-up top down description defines two user groups: A and B. Each group has two running pod because it is uses different instance types, *m1.small* flavour with 1 VCPU, 2GB of memory and 20 GB of disk, and *small* instance type with 1VCPU, 1GB of memory and 10 GB of disk. The job submissions of a group can run in both instance type running pods, but these jobs can not match with the VMs submitted by the other running pod.

The RP of the test are part of the EGI Fedcloud. CESGA cloud with OpenNebula and BIFI cloud with OpenStack. Maximum number of VCPUs by each IaaS provider is 25 VCPUs, with *small* instances for CESGA and *m1.small* for BIFI. Both of them have a maximum of 5 VCPUs for opportunistic use.

Submission pattern is composed by 10 series of 100 jobs each. The jobs of a single series are submitted about the same moment using DIRAC parametric jobs. The sequence is to start to submit the 100 jobs series of a group, then the other group series, and repeating the period up to 10 series. Using this pattern it is possible to evaluate the adaptability of the resources for VMs which can be used by a particular group.

Job workload is running a fractal plotting software (*mandelbrot*), with high *CPU/IO* ratio. The binary is allocated in the VMs using the job input sandbox and the fractal plot is put in the output sandbox. A single job process has been previously tested in the clouds, with very similar execution time between different providers. Group B has been configured with a maximum VCPU hours to run 3 series of 100 jobs, then opportunistic use of resources will apply.

VM horizontal auto scale policy is configured for a compromise between VM efficiency and total wall time. The VM is configured with 5 minutes of margin time before halt. I.e. previously to stop a VM is necessary a margin time without any workload running. If more workload of the next series is matched within this halting margin time, then the VM is not stopped. Only jobs of the same group can match the payload of a VM. Series pattern time gaps are taken as reference a normal distribution with average in the half of the time of the larger job response time, obtained from a previous processing of 100 jobs from cold system. The idea is not to simulate real user behaviour, but to test the elastic scale up and down of the system, producing series which uses previous VMs of the same group, as well as new VMs, depending on the workload gap times and the available power for a group in a specific moment.

The contextualization is done by ssh with an agent is polling the VMs in *waiting for context* status. This method is needing synchronization between submission time and contextualization time, which it is a disadvantage compared with context methods integrated in the cloud managers, like *cloudinit* or *prolog and epilog*. In the other hand *ssh* contextualization is not requiring additional libraries in the images neither particular implementations of the cloud managers. With a public IP and a *sshd* running in the VM, the same context scripts can be launched independently of the IaaS site.

5.2 Test Result

This section presents different results of the test, starting from general plots then breaking down in more detailed metrics.

Fig. 3 shows the running VMs, those VMs which has been submitted and booted, then contextualized and finally declared running by the VM Monitor when Job Agent starts to match jobs. During the VM running, the VM Monitor is sending periodical heart beats. If a VM is not changing status or sending running heart beat in 30 minutes then it is declared stalled. Two VMs were stalled in the test, without reaching the running status the VM were in error sta-



Figure 3: Overall Running VMs.

tus in the BIFI OpenStack dashboard. Additionally, three VMs at CESGA an other three VMs at BIFI were booted without problem, but they were not contextualized because they reached the 30 minutes limit and declared stalled by DIRAC, when actually they were not stalled.

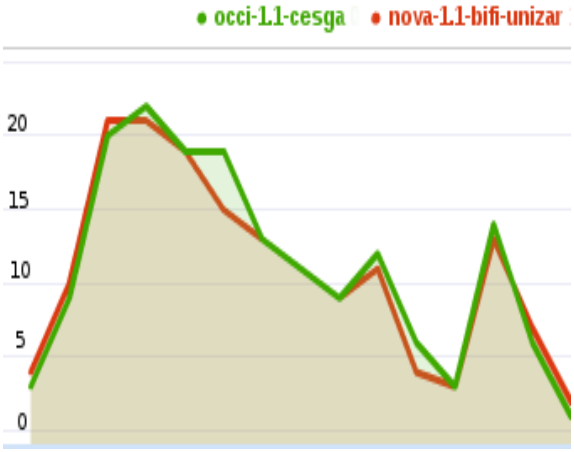


Figure 4: Running VMs by IaaS endpoint.

Peaks and valleys in Figs. 3, 4 and 5 are demonstrating the ability of DIRAC to scale up and down adapting to different workloads patterns in multi-community environment. Let see next plots for further details about scalability.

Running VMs by IaaS endpoint is shown in Fig. 4. The running VMs shape is about the same than overall running VMs of previous plot. CESGA and BIFI clouds, which are supported by different cloud managers, with the same DIRAC configuration are obtaining very close results in terms of running VMs. However, a workload analysis by endpoint shows that 63.5% of the workload has run in CESGA cloud, while BIFI has run 36.5%. This is quite significant in terms of power features. CESGA was providing for this test instance types with 1VCPU, 1GB of memory and 10GB of disk, while BIFI was giving instances of 1VCPU, 2GB of memory and 20GB of disk. With the same VCPU number per instance type, BIFI instances provide more memory and disk, but it is able to process about the half of the workload with near the same running hours of VMs. Thefore VCPU provided by CESGA have more power by near a factor two. It is clear that VCPU/h is not an appropriate metric for power, and therefore neither for credit.

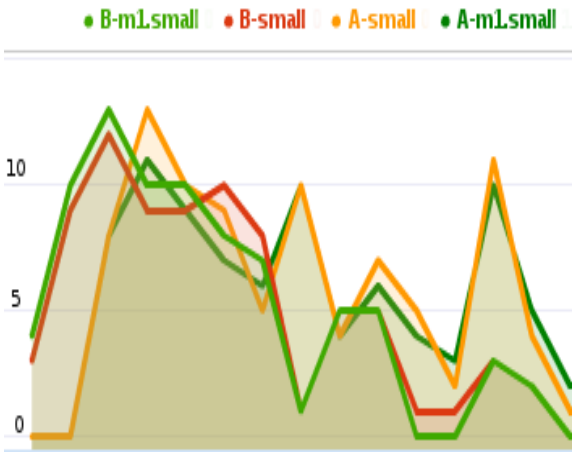


Figure 5: Running VMs by running pod.

Fig. 5 shows running VMs by running pod. As it was mentioned above in the test set-up, the jobs of group A can run in *A-small* or *A-m1.small*, and equivalent with group B. A group is divided in two running pods to evaluate instance types differences and therefore IaaS provider differences in multi-community stage. However, the response is equivalent in the level of running pods of group A, and also equivalent for group B running pods. This is demonstrating that high level governance policies are working transparently in multiple IaaS providers. At the same time, the scalability is independent between groups, following separated workload patterns.

Fig. 5 also shows opportunistic features. When there is no credit is left for group B, running VMs reach a plateau without trespassing the maximum number of opportunistic resources per IaaS endpoint.

Fig. 6 is a job histogram which provides some information. The first series of each group are in a cold system, therefore, they have the first jobs response time near 1000 seconds, while in the rest of series the first jobs are below 500 seconds. The opportunistic features are producing larger serie 4 and even larger in serie 5 of the group B.

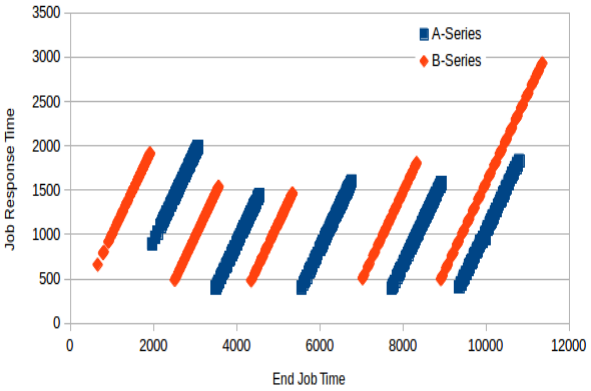


Figure 6: Jobs histogram.

6 CONCLUSIONS

The first conclusions are about elasticity. The prototype result shows a good scaling features for multi-community workloads which are using different VMs. The credit model has been demonstrated in the test, restricting the scale up to opportunistic resources when no more credit is left, but it is not affecting the scale down. Thus, the model is responding as expected by the larger job response time for series of the opportunistic cases.

The prototype has shown a secondary constrain about scalability, which could be resolved in future work. Complex scale up and down environments, with multi-community support, are producing high rates of creation and deletion. In this environment, a lack of synchronization has been detected between the submission rate and the *ssh* contextualization rate, which has to be tuned in production stages to avoid false positives in stalled machines. At the end of the day, booting time is constrained by cloud manager capabilities so there is no definitive solution from DIRAC side. Furthermore, error status VMs are also going to stalled as expected. Anyway, it is necessary an additional DIRAC agent trying to halt the VMs registered in stalled status, not only to clarify false positives, but also to free resources when possible.

Test results have shown that VCPU is not a proper credit metric, because it is not a power metric. For a governance based in credit model it is important to uses credits clearly related with the workloads. In this sense CPU power can be calculated by benchmarking the VMs power, as suggested in Rafhyc architecture (Méndez et al., 2013b). At the same time not only CPU power is consuming resources, memory footprint requisites, permanent and temporal disk requirements as well as network resources should also be considered in the credit metric of cloud computing.

There are some conclusions about governance policies. Notice that resource blocking stages have not been considered in the prototype. Future work shall consider to include a method to avoid resource blocking for the cases of power supply lower than demand. An tool for this method implementation can be the VM graceful stoppage option of DIRAC. Moreover, test has demonstrated that high level governance policies are working transparently in multiple IaaS providers. Further, each group workload is corresponding to separated elasticity pattern. Simultaneously, opportunistic policies are responding as expected in the credit model, not only terms of resource use limits, but also in job response times.

ACKNOWLEDGEMENTS

This work was supported by projects FPA2007-66437-C02-01/02 and FPA2010-21885-C02-01/02, assigned to UB. We are greatly in debt with EGI Fedcloud, in particular with BIFI and CESGA for providing the cloud resources for the test.

REFERENCES

Albor, V. F., Silva, J. J. S., Gómez-Folgar, F., López-Cacheiro, J., and Diaz, R. G. (2011). Dirac integration with cloudstack. In *Proceedings of 3rd IEEE International Conference on Cloud Computing Technology and Science (IEEE CloudCom 2011)*.

APEL-EMI (2014). http://www.eu-emi.eu.

Bunic, P., Aguado-Sanches, C., Blomer, J., and Harutynunyan, A. (2011). Cernvm: Minimal maintenance approach to the virtualization. *Journal of Physics Conference Serires*.

CloudStack (2014). http://www.cloud.com.

Cohen, S., Karagiannis, F., Courcoubetis, C., Iqbal, K., Andreozzi, S., and Heikkurinen, M. (2013). Computing einfrastructure cost estimation and analysis. pricing and business models. Technical Report Deliverable D2.3, e FISCAL : Financial Study for Sustainable Computing e Infrastructures.

EGI-FedCloud (2014). https://wiki.egi.eu/wiki/Fedcloud-tf:Blueprint.

Graciani Diaz, R., Casajus Ramo, A., Carmona Aguero, A., Fifield, T., and Sevior, M. (2011). Belle-dirac setup for using amazon elastic compute cloud. *Journal of Grid Computing*, 9:65–79. 10.1007/s10723-010-9175-7.

Jakob Blomer, T. F. (2010). A fully decentralized file system cache for the cernvm-fs. In *Proceedings of 19th International Conference*, pages 1–6.

Konstantinou, I., Floros, E., and Koziris, N. (2012). Public vs private cloud usage costs: the stratuslab case. In *Proceedings of the 2nd International WorkShop on Cloud Computing Platforms*.

Lerner, R. (2006). Amazon web services. *Linux journal*, 143:20.

Llorente, I. M., Montero, R. S., and Milojicic, D. (2011). Opennebula: A cloud management tool. *IEEE Internet Computing*, 15:14.

Méndez, V., Albor, V. F., Diaz, R. G., Ramo, A. C., a, T. F. P., Arévalo, G. M., and Silva, J. J. S. (2012). The integration of cloudstack and occi/opennebula with dirac. *Journal of Physics Conference Serires*.

Méndez, V., Ramo, A. C., Albor, V. F., and Diaz, R. G. (2013a). How to run scientific applications with dirac in federated hybrid clouds. In *ADVCOMP 2013 : The Seventh International Conference on Advanced Engineering Computing and Applications in Sciences*, pages 73–78. IARIA. ISBN: 978-1-61208-290-5.

Méndez, V., Ramo, A. C., Albor, V. F., Diaz, R. G., and Arévalo, G. M. (2013b). Rafhyc: an architecture for constructing resilient services on federated hybrid clouds. *Journal of Grid Computing*, 11(4):753–770.

OpenStack (2014). http://openstack.org/.

Simon, A., Freire, E., Rosende, R., Diaz, I., Rey, P., Lopez-Cacheiro, J., and C.Fernandez (2012). Egi fedcloud task force. In *IBERGRID, 6th Iberian Grid Infrastructure Conference Proceedings*.

Tsaregorodtsev, A., Bargiotti, M., Brook, N., Casajus Ramo, A., Castellani, G., Charpentier, P., Cioffi, C., Closier, J., Graciani Diaz, R., Kuznetsov, G., Li, Y. Y., Nandakumar, R., Paterson, S., Santinelli, R., Smith, A. C., Miguelez, M. S., and Jimenez, S. G. (2008). Dirac: a community grid solution. *Journal of Physics: Conference Series*, 119(6):062048.

Ubeda, M., Méndez, V., andlippe Charpentier, F. S., and closier, J. (2014). Integration of cloud resources in the lhcb distributed computing. *Journal of Physics Conference Serires*.

SPECIAL SESSION ON MULTI-CLOUDS

FULL PAPERS

Multi-dimensional Resource Allocation for Data-intensive Large-scale Cloud Applications

Foued Jrad[1], Jie Tao[1], Ivona Brandic[2] and Achim Streit[1]
[1]*Steinbuch Centre for Computing, Karlsruhe Institute of Technology, 76128 Karlsruhe, Germany*
[2]*Information Systems Institute, Vienna University of Technology, 1040 Vienna, Austria*
{*foued.jrad, jie.tao, achim.streit*}*@kit.edu, ivona@infosys.tuwien.ac.at*

Keywords: Cloud Computing, Multi-Cloud, Resource Allocation, Scientific Workflow, Data Locality.

Abstract: Large scale applications are emerged as one of the important applications in distributed computing. Today, the economic and technical benefits offered by the Cloud computing technology encouraged many users to migrate their applications to Cloud. On the other hand, the variety of the existing Clouds requires them to make decisions about which providers to choose in order to achieve the expected performance and service quality while keeping the payment low. In this paper, we present a multi-dimensional resource allocation scheme to automate the deployment of data-intensive large scale applications in Multi-Cloud environments. The scheme applies a two level approach in which the target Clouds are matched with respect to the Service Level Agreement (SLA) requirements and user payment at first and then the application workloads are distributed to the selected Clouds using a data locality driven scheduling policy. Using an implemented Multi-Cloud simulation environment, we evaluated our approach with a real data-intensive workflow application in different scenarios. The experimental results demonstrate the effectiveness of the implemented matching and scheduling policies in improving the workflow execution performance and reducing the amount and costs of Intercloud data transfers.

1 INTRODUCTION

With the advantages of pay-per-use, easy-to-use and on-demand resource customization, the Cloud computing concept was quickly adopted by both industry and the academia. Over the last decade, a lot of Cloud infrastructures have been built which distinguish themselves in the type of offered service, access interface, billing and SLA. Hence, today Cloud users have to make manual decisions about which Cloud to choose in order to meet their functional and non-functional service requirements while keeping the payment low. This task is clearly a burden for the users because they have to go through the Web pages of Cloud providers to compare their services and billing policies. Furthermore, it is hard for them to collect an maintain the all needed information from current commercial Clouds to make accurate decisions.

The raising topic of Multi-Cloud addresses the interoperability as well the resource allocation across heterogeneous Clouds. In this paper we focus from a user perspective on the latter problem, which has been proved to be NP-hard. A challenging task for the resource allocation is how to optimize several user objectives like minimizing costs and makespan while fulfilling the user required functional and non-functional SLAs. Since in Multi-Cloud data transfers are performed through Internet between datacenters distributed in different geographical locations, another challenge is how to distribute the workloads on these Clouds in order to reduce the amount and cost of Cloud-to-Cloud (Intercloud) data transfers.

Today, related work on Cloud resource allocation is typically restricted to optimizing up to three objectives which are cost, makespan and data locality (Deelman et al., 2008). Attempts to support other objectives (e.g. reliability, energy efficiency) (Fard et al., 2012) cannot be applied directly to Multi-Cloud environments. However, the support for more SLA constraints like the Cloud-to-Cloud latency and Client-to-Cloud throughput, which both have high importance for data-intensive Mutli-Cloud applications, is still missing.

Motivated by the above considerations, we propose a multi-dimensional resource allocation scheme to automate the deployment of large scale Multi-Cloud applications. We facilitate the support of multi-

ple generic parameters by applying a two stage allocation approach, in which the target Clouds are selected using an SLA-based matching algorithm at first and then the application workloads are distributed to the selected Clouds using a data locality driven scheduling policy. For the SLA-based matching, we adopted from the economic theory a utility-based algorithm which scores each non-functional SLA attribute (e.g. availability, throughput and latency) and then calculates the user utility based on his payment willingness and the measured Cloud provider SLA metrics. Overall, our optimization is done with four objectives: makespan, cost, data locality and the satisfaction level against the user requested non-functional SLA.

In order to validate our resource allocation scheme, we investigate in this paper the deployment of data-intensive Multi-Cloud worlflow applications. Our idea of supporting Multi-Cloud workflows comes from the observation of following scenario: A customer works on several Clouds and stores data on them. There is a demand of jointly processing all of the data to form a final result. This scenario is similar to the collaborative work on the Grid. For example, the Worldwide LHC Computing Grid (WLCG) [1] involves more than 170 computing centers, where the community members often work on a combined project and store their data locally on own sites. A workflow application within such a project must cover several Grid resource centers.

We evaluated our allocation scheme using a Mutli-Cloud workflow framework, we developed based on the CloudSim (Calheiros et al., 2011) simulation toolkit. The reason for applying a simulator rather than real Clouds is that our evaluation requires different Cloud platforms with various properties in infrastructure Quality of Service (QoS). In addition, the simulator allows us to fully validate the proposed concept with various scenarios and hence to study the developed resource matching and data locality optimization schemes.

The experimental results show that our multi-dimensional allocation scheme offers benefits to users in term of performance and cost compared to other policies. Overall, this work makes the following contributions:

1. A multi-dimensional resource allocation scheme for large scale Multi-Cloud applications.
2. An efficient utility-based matching policy to select Cloud resources with respect to user SLA requirements.
3. A data locality driven scheduling policy to minimize the data movement and traffic cost.
4. An extensive simulation-based evaluation with a real workflow application.

The remainder of the paper is organized as follows: Section 2 presents the related work on data locality driven workflow scheduling. Section 3 describes the Multi-Cloud workflow framework used to validate our approach. Section 4 and Section 5 describe the functionality of the matching and scheduling algorithms implemented in this work. Section 6 presents the simulation environment and the evaluation results gathered from the simulation experiments. Finally, Section 7 concludes the paper and provides the future work.

2 RELATED WORK

Over the past decade, the scheduling of data-intensive workflows is emerged as an important research topic in distributed computing. Although the support of data locality have been heavily investigated on Grid and HPC, only few approaches apply data locality for Cloud workflows. A survey of these approaches is provided in (Mian et al., 2011). In this section we focus on works dealing with data locality driven workflow scheduling in Multi-Cloud environments.

The authors in (Fard et al., 2013) adopted from the game theory a dynamic auction-based scheduling strategy called BOSS to schedule workflows in Multi-Cloud environments. Although their conducted experiments show the effectiveness of their approach in reducing the cost and makespan compared to traditional multi-objective evolutionary algorithms, the support for data locality is completely missing in their work. Szabo et al. (Szabo et al., 2013) implemented a multi-objective evolutionary workflow scheduling algorithm to optimize the task allocation and ordering using the data transfer size and execution time as fitness functions. Their experimental results prove the performance benefits of their approach but not yet the cost effectiveness. Although the authors claim the support for Multi-Cloud, in their evaluation they used only Amazon EC2-based [2] Clouds. Yuan et al. (Yuan et al., 2010) proposed a k-means clustering strategy for data placement of scientific Cloud workflows. Their strategy clusters datasets based on their dependencies and supports reallocation at runtime. Although their simulation results showed the benefits of the k-means algorithm in reducing the number of data movements, their work lacks the evaluation of the execution time and cost effectiveness. In (Pandey et al., 2010) the authors use particle swarm

[1] http://lcg.web.cern.ch

[2] http://aws.amazon.com/ec2

optimization (PSO) techniques to minimize the computation and traffic cost when executing workflows on the Cloud. Their approach is able to reduce execution cost while balancing the load among the datacenters. The authors in (Jin et al., 2011) introduced an efficient data locality driven task scheduling algorithm called BAR (Balance and reduce). The algorithm adjusts data locality dynamically according to the network state and load on datacenters. It is able to reduce the completion time. However, it has been tested only with MapReduce (Dean and Ghemawat, 2008)-based workflow applications.

The examination of the previous mentioned works shows that the support of data locality in the scheduling improves the performance and minimizes the cost of workflow execution on Cloud. A more clever solution allowing the support of more generic workflow applications is to implement task scheduling as part of a middleware on top the Cloud infrastructures. In such way, it would be possible to support more user-defined SLA requirements, like latency and availability in the task scheduling policies. Since, a proper SLA-based resource selection can have also significant impact on the performance and cost, we investigate in this work the effect of different resource matching policies on the scheduling performance of data-intensive workflows. The cost effectiveness of such matching policies in Multi-Cloud environments have been investigated in (Dastjerdi et al., 2011), but they have not been evaluated with scientific workflows.

In contrast to the previous works, the resource allocation scheme introduced in this paper allows the execution of workflows on top of heterogeneous Clouds by supporting an SLA-based resource matchmaking combined with a data locality driven task scheduling. In addition to a cost evaluation, we study the impact of the matching and scheduling on data movement and makespan using a real scientific application.

3 BACKGROUND

In order to validate and evaluate our multi-dimensional resource allocation scheme we use a broker-based workflow framework, we implemented in a previous work (Jrad et al., 2013b) to support the deployment of workflows in Multi-Cloud environments. In this section we describe briefly the main components of the framework.

Figure 1 depicts the architecture of the Multi-Cloud workflow framework developed on top of Cloudsim. The Cloud Service Broker, as shown in the middle of the architecture, assists users in finding the suitable Cloud services to deploy and execute their workflow applications. Its main component is a Match-Maker that performs a matching process to select the target Clouds for the deployment. A scheduler assigns the workflow tasks to the selected Cloud resources. The architecture includes also a Data Manager to manage the data transfers during the workflow execution. The entire communication with the underlying Cloud providers is realized through standard interfaces offered by provider hosted Intercloud Gateways. A Workflow Engine deployed on the client side, delivers the workflow tasks to the Cloud Service Broker with respect to their execution order and data flow dependencies.

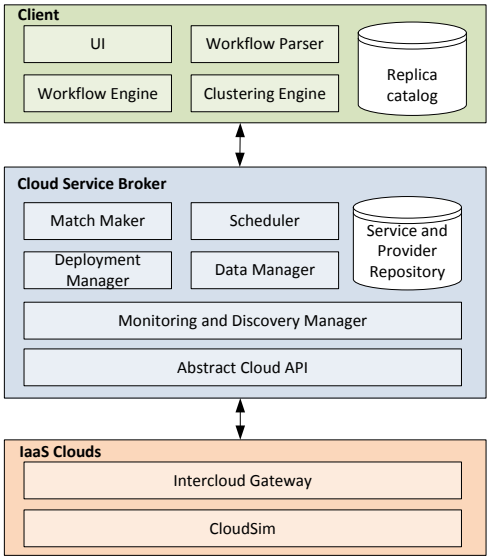


Figure 1: Multi-Cloud workflow framework architecture.

In order to execute workflows using the framework, the Workflow Engine receives in a first step a workflow description and the SLA requirements from the user. After parsing the description, the Workflow Engine applies different clustering techniques to reduce the number of workflow tasks. The reduced workflow and the user requirements are then forwarded to the Broker. In the following, the Match-Maker selects the Cloud resources that can fit the user given requirements by applying different matching policies. After that all the requested virtual machines (VMs) and Cloud storage are deployed on the selected Clouds, the Workflow Engine transfers the input data from the client to the Cloud storage and then starts to release the workflow tasks with respect to their execution order. During execution, the scheduler assigns each task to a target requested VM according

to different scheduling policies while the Data Manager manages the Cloud-to-Cloud data transfers. A Replica Catalog stores the list of data replicas by mapping workflow files to their current datacenter locations. Finally, the execution results are transferred to the Cloud storage and can be retrieved via the user interface. The requested VMs and Cloud Storage are provisioned to the user until the workflow execution is finished. However, if the same workflow should be executed many times (e.g. with different input data), a long-term lease period can be specified by the user.

4 SLA-BASED MULTI-CLOUD MATCHING POLICIES

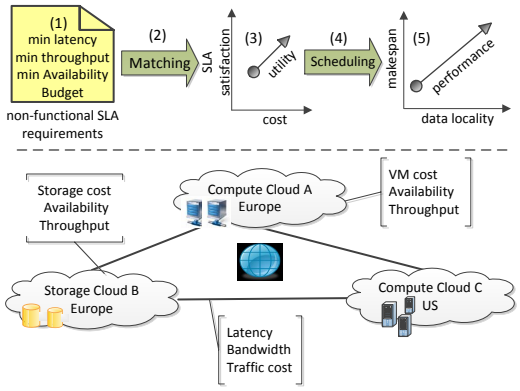


Figure 2: Multi-dimensional resource allocation scheme.

As illustrated in Figure 2, our multi-dimensional resource allocation is performed in five steps. After that the user gives his SLA requirements and budget (step 1), a matching process is started (step 2), where the functional and non-functional SLA requirements are compared to the measured Cloud SLA metrics as well as their service usage costs. The selection of the optimal Clouds (step 3) is then performed using a utility-based matching algorithm with the goal to maximize the user utility for the provided QoS. In the following step, The application workloads are distributed to the requested compute resources using a scheduling policy. For this purpose, a data-locality scheduling scheme is used to achieve a minimal data movement and improve the overall application performance (step 5). In the following subsections we describe in details the used utility-based matching algorithm. In addition, we describe another simple matching algorithm, we used for a comparative study with the utility-based algorithm. The used scheduling policies are described in the next Section.

4.1 Sieving Matching Algorithm

On the search for efficient matching algorithms, we implemented a simple matching policy called Sieving. Given a service list forming the requested Mutli-Cloud service composition and a list of candidate datacenters, the Sieving matching algorithm iterates through the service list and selects randomly for each service a candidate datacenter, which satisfies all functional and non-functional SLA requirements. Therefore, for each selected datacenter the measured SLA metrics and his usage price should be respectively within the ranges specified by the user in his SLA requirements and budget. In addition, the algorithm checks if the current datacenter capacity load allows the deployment of the requested service type. However, it may be possible that the result set is empty in case that none of the Cloud providers fulfill the requested criteria. The following pseudo code describes in detail the functionality of the Sieving algorithm:

Algorithm 1: Sieving matching.

```
Input: requiredServiceList, DatacenterList
For each service S in requiredServiceList Do
 For each datacenter D in DatacenterList Do
  If S in offers(D) and isDeployableIn(D) Then
    If price(D) <= budget(S) &
       availability(D)>=requiredAvilability(S) &
       throughput(D)>=requiredThroughput(S) Then
       add D to CandidatesList
    Endif
  Endif
 Endfor
 If sizeof(CandidatesList) >0 Then
   Choose random datacenter D from CandidatesList
   CloudCompositionMap.add(S,D)
 Else return null
 Endif
Endfor
Output : CloudCompositionMap
```

4.2 Utility-based Matching Algorithm

A major issue of the above described Sieving algorithm is the lack of flexibility in the matching of non-functional SLA attributes. Hence, it cannot handle use cases like availability is more important than throughput or select well qualified providers while keeping the total costs low. In addition, the network connectivity between the Clouds and traffic costs are ignored in the matching. Therefore, we adopted from the Attribute Auction Theory (Asker and Cantillon, 2008) a new economic utility-based matching algorithm, which takes the payment of customers as the focus. In a previous work (Jrad et al., 2013a),

we compared the utility algorithm with Sieving and validated its cost benefits in matching single Cloud services. This work, is our first attempt in using the utility-for matching Multi-Cloud service compositions.

The main strategy of the utility-algorithm is to maximize the user profit for the requested service quality by using a quasi-linear utility function (Lamparter et al., 2007). The user preferences for the non-functional SLA attributes are modeled by weighted scoring functions, whereas all functional requirements must be fulfilled similar to Sieving. Let Q be the set of m required non-functional SLA attributes q with $q \in Q = \{1,...,m\}$. The utility of a customer i for a required service quality Q with a candidate Cloud composition j is computed as follows:

$$U_{ij}(Q) = \alpha_i F_i(Q) - P_j, \tag{1}$$

where α_i represents the *maximum willingness to pay* of consumer *i* for an "ideal" service quality, and $F_i(Q)$ the customer's *scoring function* translating the aggregated service quality attribute levels into a relative fulfillment level of consumer requirements. P_j denotes the total price that has to be paid for using all Cloud services in the Cloud composition j with:

$$P_j = T * C^j_{VM} + D_{st} * C^j_{st} + D_{tr} * C^j_{tr} \tag{2}$$

where C^j_{VM}, C^j_{st} and C^j_{tr} denote respectively the charged compute, storage and traffic cost with composition j, whereas D_{st} and D_{tr} denote respectively the amount of data to be stored and transferred. The lease period T is defined as the time period in which the Cloud services are provisioned to the user. The scoring function $F_i(Q)$ is defined as follows:

$$F_i(Q) = \sum_{q=1}^{m} \lambda_i(q) f_i(q) \rightarrow [0,1], \tag{3}$$

where $\lambda_i(q)$ and $f_i(q)$ denote respectively the relative assessed weight and the fitting function for consumer *i* regarding the SLA attribute *q*, where $\sum_{q=1}^{m} \lambda_i(q) = 1$. The fitting function maps properly to the user behavior each measured SLA attribute to a normalized real value in the interval [0,1] with 1 representing an ideal expected SLA value. An example for three non-linear fitting functions is provided in Section 6. The aggregated SLA values for the Cloud composition are calculated from the SLA values of the component services by applying common used aggregation functions as in (Zeng et al., 2003) presented in Table 1.

A candidate Cloud service composition j is optimal if it is feasible and if it leads to the maximum utility value with:

$$U_{ioptimal}(Q) = \max_{j=1}^{n} U_{ij}(Q) \tag{4}$$

Table 1: Aggregated SLA attributes of N component services S.

SLA attribute	Aggregation function
Throughput	$\min_{i=1}^{N} Th(S_i)$
Latency	$\overline{Lat(S_{ij})} \begin{cases} i \in \{1,\ldots,N-1\} \\ j \in \{i+1,\ldots,N\} \end{cases}$
Availability	$\prod_{i=1}^{N} Av(S_i)$

where n is the number of possible Cloud service compositions. Hence, the match-making problem can be formulated with a search for the Cloud composition with the highest utility value for the user. As this kind of multi-attribute selection problems is NP-Hard, we apply a single objective genetic algorithm combined with crossover and mutation operations to search for the optimal candidates. To evaluate each candidate we use as objective function the utility function in equation 1. Additionally, we use a death penalty function to penalize candidates, which not satisfy the service constraints and to discard Cloud compositions with a negative utility. In order to include the Cloud-to-Cloud latency and the traffic cost in the utility calculation, we model each candidate service composition as a full connected undirected graph (see Figure 2), where the nodes represent the component services and the edges the network connectivity between them.

5 DATA LOCALITY DRIVEN WORKFLOW SCHEDULING

To support the data locality during the workflow execution, we implemented two dynamic greedy-based scheduling heuristics, which distribute the workflow tasks to all the user requested VMs allocated by the previous described matching policies (see Section 4). In the following subsections we describe in details the functionality of the implemented policies.

5.1 DAS Scheduler

We implemented a Data-Aware Size-based scheduler (DAS) capable of scheduling tasks to the provisioned VMs running in different Cloud datacenters with respect to the location of the required input data. In a first step, the algorithm iterates through the workflow tasks and calculates for each task the total size of the required input files found in each matched datacenter and store the result in a map data structure called task data size affinity $T_{sizeaff}$. If we assume that task T has m required input files $freq_i$, $i \in \{1,...,m\}$ and there are k matched datacenters, the task affinity of

the datacenter dc_j, $j \in \{1,...,k\}$ is calculated using the following equation:

$$T_{sizeaff}(dc_j) = \sum_{freq_i \in dc_j} size(freq_i), \quad (5)$$

After sorting the task affinity map by the data size values in descending order, the policy assigns the task to the first free provisioned VM running on the datacenter dc_{cand} containing the maximum size of located input files, where:

$$T_{sizeaff}(dc_{cand}) = \max_{j=1}^{k} T_{sizeaff}(dc_j) \quad (6)$$

In case that all the provisioned VMs in the selected datacenter are busy, the algorithm tries the next candidate datacenters in the sorted map to find a free VM. So that, a load balancing between the datacenters is assured and an unnecessary waiting time for free VMs can be avoided. The implemented data-aware scheduling policy is described using the following pseudo-code:

Algorithm 2: DAS/DAT Scheduler.

```
Input: requestedVMList, TaskList, schedulingPolicy
For each task T in TaskList Do
  processAffinity(T)
  If schedulingPolicy=DAS Then
     Taff=sizeAffnityMap
     sort Taff by size in descending order
  Elseif schedulingPolicy=DAT Then
     Taff=timeAffnityMap
     sort Taff by time in ascending order
  Endelse
  For each entry in Taff Do
    site=entry.getkey()
    For each vm in requestedVMList Do
      If vm.getStatus()=idle
         & vm.getDatacenter()=site Then
         schedule T to vm
        scheduledTaskList.add(T)
        Break
      Endif
    Endfor
  Endfor
Endfor
Output: scheduledTaskList
```

Algorithm 3: Function processAffinity(Task T).

```
Input: matchedDatacenterList, Replica catalog R
For each datacenter D in matchedDatacenterList Do
  Time=0; Size=0;
  inputFileList = T.getInputFileList()
  For each file in inputFileList Do
      maxBwth=0
      siteList = R.get(file)
      If siteList.contains(D) Then
         size=size+file.getSize()
      Else
        For each site in siteList Do
         If region(site)=region(D) Then
            bwth=Cloud-to-Cloud Intra-continental
         Else
            bwth=Cloud-to-Cloud Inter-continental
         Endelse
         If bwth>maxBwth Then
            maxBwth=bwth
        Endfor
      Endelse
      time=time+file.getSize()/maxBwth
  Endfor
  sizeAffnityMap.add(D, size)
  timeAffnityMap.add(D, time)
Endfor
Output sizeAffnityMap, timeAffnityMap
```

5.2 DAT Scheduler

The second scheduling policy called Data-Aware Time-based (DAT) scheduler has a strong similarity with the previous described DAS policy except in the method of calculating the task affinity. Instead of calculating the maximum size of existing input files per datacenter, the algorithm computes the time needed to transfer the missing input files to each datacenter and stores the transfer time values in a Map data structure called $T_{timeaff}$, which is calculated using the following equation:

$$T_{timeaff}(dc_j) = \sum_{freq_i \notin dc_j} transfertime(freq_i), \quad (7)$$

As target datacenter, the algorithm chooses the datacenter, which assures the minimum transfer time by sorting the affinity map in the ascending order.

$$T_{timeaff}(dc_{cand}) = \min_{j=1}^{k} T_{timeaff}(dc_j) \quad (8)$$

For each workflow task the algorithm (see function processAffinity() above) iterates through the matched datacenters and checks the existence of local input files. For each missing input file, it calculates the needed time to transfer the file from a remote location to that datacenter. For this, it fetches the replica catalog and selects a source location, which assures the maximum bandwidth and consequently the minimal transfer time to that datacenter.

6 EVALUATION

In order to evaluate our proposed multi-dimensional resource allocation scheme with a real large scale workflow, we conducted a series of simulation experiments. The simulation setup and results are presented in the following subsections.

6.1 Simulation Setup

We implemented the matching and scheduling policies presented in Section 4 and 5 as part of the Multi-Cloud simulation framework. As workflow engine, we use WorkflowSim (Weiwei and Ewa, 2012), a CloudSim-based version of the Pegasus WfMS [3]. In addition, we use the opt4j (Lukasiewycz et al., 2011) genetic framework to perform the utility-based matching. For all the conducted simulation experiments, we configured 20 compute and 12 storage Clouds located in four world regions (Europe, USA, Asia and Australia). Each compute Cloud is made up of 50 physical hosts, which are equally divided between two different host types with respectively 8 and 16 CPU cores. In order to make the simulation more realistic, we use real pay-as-you-go prices for computation, storage and network traffic for each modeled Cloud. Traffic inside the same datacenter is free. The real Cloud SLA metrics values (average for last 3 months) for availability and Client-to-Cloud throughput were acquired through CloudHarmony[4] network tests from the same client host. In order to consider the network traffic and latency in the matching and scheduling, we defined based on the location of the datacenters three constant bandwidth and latency values, which are presented in Table 2. The use of synthetic values is justified by the lack of free accessible Cloud-to-Cloud network metrics from current Cloud benchmarking tools.

Table 2: Defined Cloud-to-Cloud latency and bandwidth values.

Cloud-to Cloud	local transfer	intra-continental	inter-continental
Bandwidth	100 Mbit/s	30 Mbit/s	10 Mbit/s
Latency	10 ms	25 ms	150 ms

With the help of the framework, we modeled the following use case for a workflow deployment on Multi-Cloud. A user located in Europe requests 10 VMs of the type *small* and 10 VMs of the type *medium* and one storage Cloud to store the workflow input and output data. The configuration of each VM type is reported in Table 3. We assume that all the VMs located in the same datacenter are connected to a shared storage. The Workflow Engine transfers at execution start the input data from the Client to the Cloud storage. The output data is stored in the Cloud storage when the execution is finished. The Data Manager fetches the Replica catalog and transfers all the missing input files before each task execution from their source datacenters with the maximal possible bandwidth. We configured the Workflow Engine to release maximal 5 tasks to the broker in each scheduling interval (default value used in Pegasus).

Table 3: VMs Setup; 1 CPU Core: 1GHZ Xeon 2007 Processor of 1000 MIPS; OS: Linux 64 bits.

VM Type	Cores	RAM (GB)	Disk (GB)
small	1	1.7	75
medium	2	3.75	150

It is worth to mention that if the user executes the same workflow multiple times, our implemented data locality scheduling scheme reuses the existing replicated input data in order to save on data transfers and costs. For simplicity, we consider in our evaluation only the first run of the workflow. For collecting the simulation results we repeated each of the experiments ten times from the same host and then computed the average values.

6.2 Montage Workflow Application

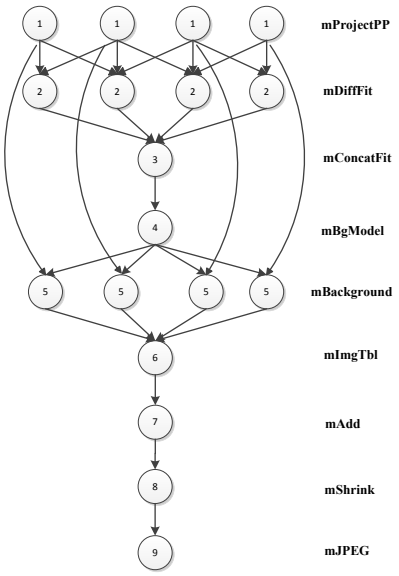


Figure 3: A sample 9-level Montage workflow.

Montage (Berriman et al., 2004) is a data-intensive (over 80% I/O operations) workflow application used to construct large image mosaics of the sky obtained from the 2MASS observatory at IPAC [5]. A sample directed acyclic graph (DAG) of a 9 level Montage workflow is illustrated in Figure 3. The tasks at the same horizontal level execute the same

[3] http://pegasus.isi.edu/

[4] http://www.cloudharmony.com

[5] http://www.ipac.caltech.edu/2mass/

Table 5: User preferences expressed using fitting functions f and relative weights λ; $\gamma = 0.0005$; $\beta = 1 - \gamma$.

	Availability		Latency		Throughput	
Scenario	λ_{av}	$f(av)$	λ_{lat}	$f(lat)$	λ_{th}	$f(th)$
EU	$\frac{3}{10}$	$\frac{\gamma}{\gamma+\beta e^{(-0.9(av-84))}}$	$\frac{6}{10}$	$\frac{\gamma}{\gamma+\beta e^{(0.2(lat-100))}}$	$\frac{1}{10}$	$1-\beta e^{-0.2th}$
EU-US	$\frac{2}{5}$	$\frac{\gamma}{\gamma+\beta e^{(-0.9(av-84))}}$	$\frac{2}{5}$	$\frac{\gamma}{\gamma+\beta e^{(0.2(lat-150))}}$	$\frac{1}{5}$	$1-\beta e^{-0.6th}$

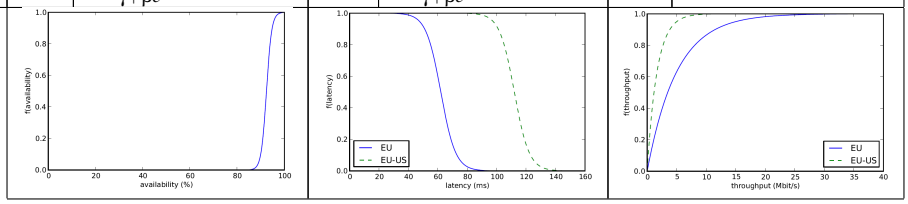


Table 4: User non-functional SLA Requirements; Availability (av); Latency (lat); Throughput (th).

deployment scenario	min av (%)	max lat (ms)	min th (Mbit/s)
EU	95	50	12
EU-US	95	100	2

executable code (see right side of the figure) with different input data. For all our conducted experiments we imported with the help of the WorkflowSim Parser a real XML formatted Montage trace executed on the FutureGrid testbed using the Pegasus WfMS. The tasks runtime and files' size information are imported from separate text files. The imported trace contains 7463 tasks within 11 horizontal levels, has 3 GB of input data and generates respectively about 31 GB and 84 GB of intermediate output and traffic data.

6.3 Simulation Scenarios

For the purpose of evaluation, we modeled two simulation scenarios. In the first scenario, named "EU-deployment", all the requested VMs and storage Cloud are deployed in the same user region (Europe). In the second scenario, named "EU-US deployment", all the 10 VMs of type *small* are deployed on Clouds located in the US region, while the 10 VMs of type *medium* and the storage Cloud are located in Europe. The non-functional SLA requirements for both scenarios, given in Table 4, express the user desired ranges for availability, latency and throughput in order to deploy the workflow with an acceptable quality. These values are consumed by the Sieving matching algorithm described in Section 4 to select the target Clouds for the workflow deployment.

The fitting functions and the relative weight for each SLA parameter, both required for the utility-based matching, are given in Table 5. As we can see from the table, the user prefers for the first scenario Clouds with low Cloud-to-Cloud latency values to advantage the data transfer time, while for the second scenario availability is equally weighted with the latency and the Client-to-Cloud throughput has more significance.

The maximum user payment willingness for each VM type as well as for storage and network traffic are given in Table 6.

Table 6: Maximal payment for VMs, storage and traffic.

VM small ($/hour)	**VM medium** ($/hour)	**storage** ($/GB)	**traffic** ($/GB)
0.09	0.18	0.1	0.12

6.4 Impact of Clustering

As mentioned before we use clustering to reduce the scheduling overhead of large scale workflows on Multi-Cloud. A well suited clustering technique for the Montage workflow structure is horizontal clustering. Herewith tasks at the same horizontal level are merged together into a predefined number k of clustered jobs. Table 7 shows the resulted total number of clustered jobs for each used k.

Table 7: Total Number of clustered jobs with different cluster numbers k.

k	20	40	60	80	100
Jobs number	92	152	212	272	332

In order to evaluate the impact of clustering with respect to increasing cluster number k, we measured in a first experiment the total time needed to execute a single run of the sample Montage workflow and the total consumed time for data transfer in minutes for the "EU-deployment" scenario. For this experiment, we configured the Cloud Service Broker to use utility-based matching together with DAS as scheduling policy. For an accurate calculation of the execution time, we extracted from the workflow trace the real delay

overhead resulted from clustering, post-scripting and queuing. Figure 4 illustrates the results achieved.

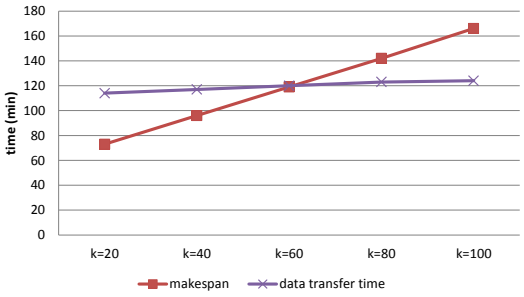


Figure 4: Workflow makespan and total data transfer time with utlity+DAS EU for different cluster numbers k.

As depicted in the figure, the continually increase of k results in a steady increase of the workflow execution time. This demonstrates that our allocation scheme scales well with increasing number of the clustered workflow jobs. Although the transfer time is nearly constant, as file transfers are performed with a high throughput and low latency values because the matched Clouds are close to the user, we observed a slow decrease of the transfer time for small numbers of k. In the latter case more files are merged in a clustered job, so that the amount of Intercloud transfers is heavily reduced. For all the next conducted experiments, we fixed the cluster number to k=20, as it gives us the best results in term of makespan and consequently execution cost.

6.5 Makespan Evaluation

We repeated the previous experiment with the "EU" and "EU-US" scenarios with the different matching and scheduling policies presented in Section 4 and 5. For a comparative study we executed the workflow using a simple Round Robin scheduler, which schedules tasks to the first free available VMs regardless of their datacenter location and the from the literature well established "Min-Min" scheduler (Freund et al., 1998), which prioritizes tasks with minimum runtime and schedule them on the *medium* VM types. The results with k=20 for all the possible combinations are shown in Figure 5.

It can be seen from the figure that for both scenarios the use of utility-based matching combined with the DAS or DAT scheduler gives the lowest execution time compared to Sieving. This result approves how the efficiency of the utility matching affects the scheduling performance. For the EU scenario, the utility algorithm have a tendency to deploy all the requested VMs on the cheapest datacenter to save cost

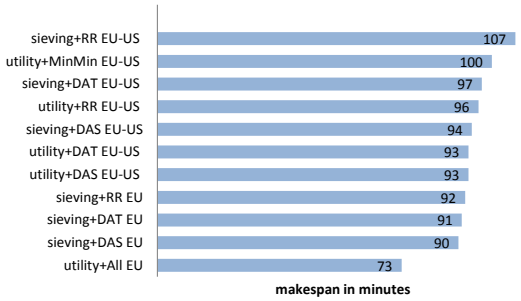


Figure 5: Workflow makespan with different scheduling and matching policies for k=20.

and minimize latency. So that, the user is charged only for the costs to transfer input/output data from/to storage Cloud. This explains the same makespan and Intercloud transfer obtained with different scheduling policies. We observed also that our implemented DAS and DAT scheduler outperform Round Robin and MinMin, especially for the EU-US scenario.

6.6 Intercloud Data Transfer

In order to assess the impact of our multi-dimensional scheme on reducing the Intercloud data transfers, we measured for the previous simulation experiment the size of Intercloud transfers and the ratio of transfer time over the total consumed processing time. The results for different combinations of matching and scheduling policies with k=20 are depicted respectively in Figure 6 and Figure 7. Note that for the Intercloud transfer time calculation, we use the previous defined "Inter-Continental" and "Intra-Continental" bandwidth values from Table 2.

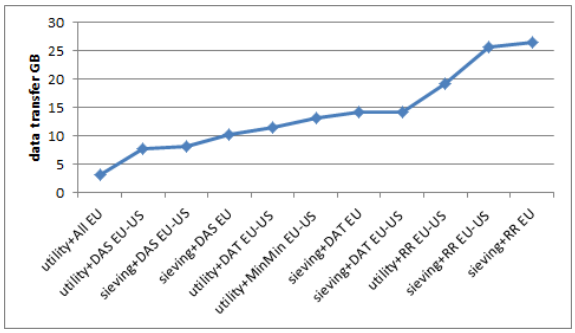


Figure 6: Total amount of Intercloud transfers with different scheduling and matching policies for k=20.

It can be seen from Figure 6 that the DAS scheduler keeps the Intercloud transfer size under 10 GB for both scenarios with utility and Sieving policies. Next to DAS on saving Cloud-to-Cloud data movements is DAT, whereas MinMin and Round robin occupy the last places. The evaluation results for the

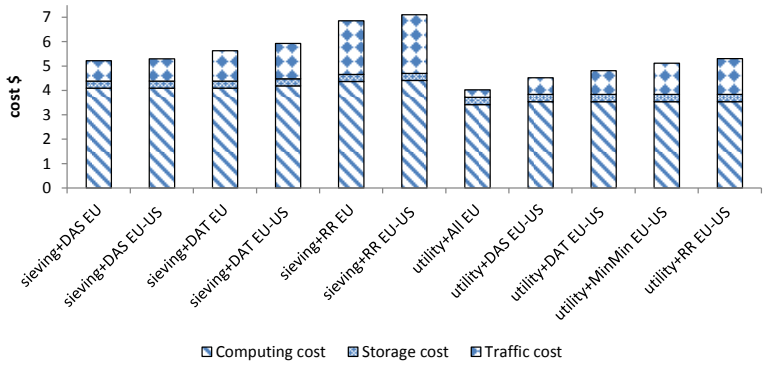


Figure 8: Total execution cost with Sieving (left) and utility (right) matching for k=20.

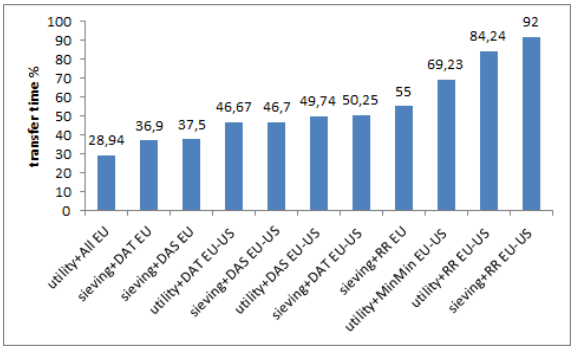
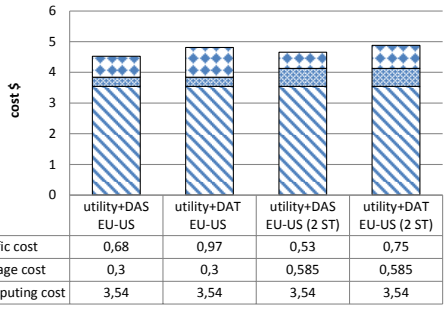


Figure 7: Percentage of data transfers with different scheduling and matching policies for k=20.

transfer time ratio approve also that DAS and DAT are able to reduce the transfer time ratio up to 37% and 50% respectively for the EU and EU-US scenario. On the contrary, the use of Round Robin and Min-Min scheduling regardless of the matching policy results in very high transfer ratios in particular for the EU-US scenario, which disadvantages consequently the workflow makespan. Therefore, data locality has more impact when Clouds are not close to user, as in this case latency and throughput affect more the transfer time.

6.7 Cost Evaluation

In this subsection, we evaluate the impact of used matching and scheduling policies on the traffic costs for both simulation scenarios. The previous experimental results show that the use of utility-based matching and data-aware scheduling heavily reduces the amount of Intercloud transfers and consequently the data traffic costs. This result is approved in Figure 8, in which the amount of compute, storage and traffic costs for different use cases is illustrated. Please note that for the costs calculation the VMs are charged on hourly basis. Therefore, the makespan of each scenario is rounded up to the nearest next hour. Also, we do not consider the additional license and VM images costs which can be charged by some Cloud providers. We found that the utility-based matching combined with the DAS scheduler benefits more the compute and traffic costs compared to the other use cases. For example, comparing utility and Sieving with DAS, respectively up to 25% and 15% cost-saving can be achieved with the EU and EU-US deployment.



	utility+DAS EU-US	utility+DAT EU-US	utility+DAS EU-US (2 ST)	utility+DAT EU-US (2 ST)
Traffic cost	0,68	0,97	0,53	0,75
Storage cost	0,3	0,3	0,585	0,585
Computing cost	3,54	3,54	3,54	3,54

Figure 9: Total execution cost with one and two storage Clouds (ST) for k=20.

We conducted another experiment by adding another storage Cloud located in the US region for the EU-US scenario. The gathered cost results are depicted in Figure 9. For the purpose of comparison, we not consider the time needed to transfer the input files from the Client to the US located storage Cloud. It can be seen from the figure that the total costs for the two storage Clouds use-case are very close to the one storage use-case, even with the doubled storage cost. This is due to the tendency of the utility algorithm to deploy the US located VMs and the added storage Cloud on the same provider to save on traffic costs.

7 CONCLUSIONS

This work presented a two-stage multi-dimensional resource allocation approach for running data-intensive workflow applications on a Multi-Cloud environment. In a first phase, a utility-based matching policy selects the suitable Clouds for users with respect to their SLA requirements and payment willingness. In a second phase, a data locality driven scheduler brings the computation to its data to reduce the Intercloud data transfer.

We evaluated our approach using an implemented simulation environment with a real data-intensive workflow application in different usage scenarios. The experimental results show the benefits from utility-based matching and data locality driven scheduling in reducing the amount of Intercloud transfers and the total execution costs as well improving the workflow makespan.

In the next step of this research work, we will evaluate our multi-dimensional approach with other Mutli-Cloud real large scale applications like MapReduce. In addition, we will automate the collection of the newest SLA metrics from real public Clouds by extending the simulation framework to fetch them from third-party Cloud monitoring services. Also, we will include more accurate network models to make the simulation more realistic. Furthermore, we will investigate the use of adaptive matching and scheduling policies like in (Oliveira et al., 2012) in order to deal with resource and network failures on Clouds.

REFERENCES

Asker, J. and Cantillon, E. (2008). Properties of scoring auctions. *The RAND Journal of Economics*, 39(1):69–85.

Berriman, G. B., Deelman, E., Good, J. C., Jacob, J. C., Katz, D. S., Kesselman, C., Laity, A. C., Prince, T. A., Singh, G., and Su, M.-H. (2004). Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand. In Quinn, P. J. and Bridger, A., editors, *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 5493 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 221–232.

Calheiros, R. N., Ranjan, R., Beloglazov, A., Rose, C. A. F. D., and Buyya, R. (2011). Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50.

Dastjerdi, A. V., Garg, S. K., and Buyya, R. (2011). QoS-aware Deployment of Network of Virtual Appliances Across Multiple Clouds. *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, pages 415–423.

Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113.

Deelman, E., Singh, G., Livny, M., Berriman, B., and Good, J. (2008). The cost of doing science on the cloud: The montage example. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, pages 50:1–50:12, Piscataway, NJ, USA. IEEE Press.

Fard, H. M., Prodan, R., Barrionuevo, J. J. D., and Fahringer, T. (2012). A Multi-objective Approach for Workflow Scheduling in Heterogeneous Environments. *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 300–309.

Fard, H. M., Prodan, R., and Fahringer, T. (2013). A Truthful Dynamic Workflow Scheduling Mechanism for Commercial Multicloud Environments. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1203–1212.

Freund, R., Gherrity, M., Ambrosius, S., Campbell, M., Halderman, M., Hensgen, D., Keith, E., Kidd, T., Kussow, M., Lima, J., Mirabile, F., Moore, L., Rust, B., and Siegel, H. (1998). Scheduling resources in multi-user, heterogeneous, computing environments with smartnet. In *Heterogeneous Computing Workshop, 1998. (HCW 98) Proceedings. 1998 Seventh*, pages 184–199.

Jin, J., Luo, J., Song, A., Dong, F., and Xiong, R. (2011). Bar: An efficient data locality driven task scheduling algorithm for cloud computing. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 295–304.

Jrad, F., Tao, J., Knapper, R., Flath, C. M., and Streit, A. (2013a). A utility-based approach for customised cloud service selection. *Int. J. Computational Science and Engineering, in press*.

Jrad, F., Tao, J., and Streit, A. (2013b). A broker-based framework for multi-cloud workflows. In *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds*, MultiCloud '13, pages 61–68, New York, NY, USA. ACM.

Lamparter, S., Ankolekar, S., Grimm, S., and R.Studer (2007). Preference-based Selection of Highly Configurable Web Services. In *Proc. of the 16th Int. World Wide Web Conference (WWW'07)*, pages 1013–1022, Banff, Canada.

Lukasiewycz, M., Glaß, M., Reimann, F., and Teich, J. (2011). Opt4J - A Modular Framework for Meta-heuristic Optimization. In *Proceedings of the Genetic and Evolutionary Computing Conference (GECCO 2011)*, pages 1723–1730, Dublin, Ireland.

Mian, R., Martin, P., Brown, A., and Zhang, M. (2011). Managing data-intensive workloads in a cloud. In Fiore, S. and Aloisio, G., editors, *Grid and Cloud Database Management*, pages 235–258. Springer Berlin Heidelberg.

Oliveira, D., Ocaña, K. a. C. S., Baião, F., and Mattoso, M. (2012). A Provenance-based Adaptive Scheduling

Heuristic for Parallel Scientific Workflows in Clouds. *Journal of Grid Computing*, 10(3):521–552.

Pandey, S., Wu, L., Guru, S., and Buyya, R. (2010). A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 400–407.

Szabo, C., Sheng, Q. Z., Kroeger, T., Zhang, Y., and Yu, J. (2013). Science in the Cloud: Allocation and Execution of Data-Intensive Scientific Workflows. *Journal of Grid Computing*.

Weiwei, C. and Ewa, D. (2012). Workflowsim: A toolkit for simulating scientific workflows in distributed environments. In *The 8th IEEE International Conference on eScience*, Chicago. IEEE, IEEE.

Yuan, D., Yang, Y., Liu, X., and Chen, J. (2010). A data placement strategy in scientific cloud workflows. *Future Gener. Comput. Syst.*, 26(8):1200–1214.

Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., and Sheng, Q. Z. (2003). Quality driven web services composition. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 411–421, New York, NY, USA. ACM.

Multi-cloud and Multi-data Stores
The Challenges Behind Heterogeneous Data Models

Marcos Aurélio Almeida da Silva and Andrey Sadovykh
Research and Development, Softeam, 8 parc Ariane Immeuble Le Jupiter, SOFTEAM, 78284 CEDEX, Guyancourt, France
{marcos.almeida, andrey.sadovykh}@softeam.fr

Keywords: Data Stores, NoSQL, Data Migration.

Abstract: The support to cloud enabled databases varies from one cloud provider to another. Developers face the task of supporting applications living in different clouds, and therefore of supporting different database management systems. To them, the challenge lies in understanding the differences in expressivity between different data stores and their impact on the application. The advent of the NoSQL movement increased the complexity of this task by leveraging the creation of a large number of cloud enabled database management systems employing slightly different data models. In this paper, we will present a model the will allow us to compare the differences in expressivity of the features supported by different databases and consider the impact of these features to different concrete deployment scenarios in multiple clouds. This model is based on the underlying data models adopted by the most used cloud database management systems. It has been developed on the FP7 JUNIPER project and will be the basis of our approach for dealing with these issues.

1 INTRODUCTION

A decade after the advent of the first cloud based solutions, it is clear to companies that migrating to cloud platforms is cost effective (Rackspace, 2013). The success of the cloud lead to the advent of multiple cloud provider offerings. As in any nascent market, there are no established standards. Economically speaking, on the one hand, the multiplication of offerings reduces the prices and makes cloud and multi-cloud applications more and more interesting to companies. On the other hand, the consequent fragmentation of the market, makes the life of cloud developers harder, since it increases the complexity of the development and maintenance of applications.

In this paper we focus on the challenges related to data stored on the cloud. The problem stems from the fact that different cloud providers support different database management systems (DMS). Developers have a variable degree of flexibility, ranging from the one they have on Infrastructure As a Service providers, in which virtually any DMS can be installed; to the one they have in the Platform as A Service providers, usually supporting a very specific subset of DMSs. Finally, in the Software as a Service providers, developers are usually only able to store data opaquely and this data is only later accessible by means of a provider specific APIs.

The consequences of this fragmentation of the support of DMSs by cloud providers are amplified by the so-called NoSQL "movement". This "movement" consists of a series of DMSs that strip the well-known SQL based relational DMSs from some for their characteristics in order to increase their performance. The problem is that different applications usually have different performance bottlenecks, which leads to different sets of optimizations that need to be applied to SQL DMSs to make them adapted to each application. This lead to the existence of a myriad of NoSQL DMSs, each one based on a slightly different set of optimizations on the SQL DMSs or even on completely different data models, fine-tuned to specific applications.

For a developer, building and maintaining a cloud application means dealing with all this fragmentation. **The main hypothesis of this paper is that in order to deal with these concerns, one, first of all, needs to understand the differences in expressivity between the data models provided by different DMSs and the potential difficulties in migrating data from one model to another.** In this paper we are going to present the main concepts behind most used cloud DMSs, and the semantic gaps between them. We are then going to present a

classification of the DMSs according to these concepts and we'll use this classification in a case study in which we identify the best DMSs to support parts of the data manipulated by an application.

The present work is a first step towards what we intend to achieve in the EU FP7 projects MODAClouds and JUNIPER. They intend to tackle such challenges by means of the model driven approach to allow developers to model their application on a very high level. Developers can then have their models analysed by automated tools and to have code automatically generated from it. The MODAClouds project focuses on public clouds and in providing automated tools to help data design. The JUNIPER project focuses on private clouds and in providing analysis tools to make sure that a particular set of DMSs respects a given set of real time constraints.

This paper is structured as follows. Section 2 details the context of data management in multi cloud applications and introduces the fragmentation of databases in this domain. Section 3 presents the most important concepts used in multi-cloud DMSs and a classification of the most important DMSs. Section 4 presents the trade-offs in migrating to and from different classes of DMS. Finally, in Sections 5 and 6 we present related works and conclusions of this paper.

2 MULTI-CLOUD APPLICATIONS AND THE NOSQL "MOVEMENT"

2.1 Overview

The cloud started as a way to offer all this as a service, in a "pay as you" go way. That means that the cloud provider would create a big data centre, and would use it to offer virtual machines to clients. On top of purely infrastructure driven cloud solutions, platform driven ones came into being. In this case, cloud providers do not offer virtual machines and storage device, but instead, they offer software platforms. The customer then is only responsible for installing the necessary software on the platform while the cloud provider will dimension the needed machines, storages and load balance strategies for the user's application (Khajeh-Hosseini, Greenwood, & Sommerville, 2010).

The main disadvantage of clouds is that users have much less flexibility than in a "on premises" solution. Each cloud provider provides only a limited set of configurations of machines, storage and platforms, while on premise solutions allow for unlimited sets of configurations. Each cloud is also optimized for a limited range of applications, i.e. some clouds are optimized to running applications involving fast running queries and long running background processes; while others may also accept long running queries over data. One way to mitigate this heterogeneity problem is to use multiple cloud providers, putting parts of the application on each provider, trying to find the best match between the cloud and the application (Liu, Katsuno, Sun, & Li, 2011) (Singh, Kandah, & Zhang, 2011).

As one could expect, the data storages supported by each cloud provider vary from one offering to another. This is so, because data storage is nowadays a much complicated matter than it was years ago. It doesn't consist anymore of choosing between traditional relational databases or home grown file based data formats. Now, developers have a myriad of Data Management Systems (DMS), each of them optimized to a particular set of data structures. This is the result of the NoSQL "movement", which in fact intends to improve the efficiency of relational DMSs by constraining the data structures they support and the queries that they can answer.

The downside for the programmer is that designing a cloud application is not only a matter of choosing the "cheapest cloud provider", but choosing the provider that supports the DMSs backed by the best data structures to represent the application data. One still needs to think about the cost and performance costs involved in transferring data from one cloud to another, and consequently from one DMS and backed data structure to another.

The main objective of this paper is helping developers in choosing the best DMSs for their data and in understanding the **performance and expressiveness trade-offs** involved in moving data from one DMS to another. In order to do so, we intend to provide a model of the data structures and queries supported by existing NoSQL and SQL

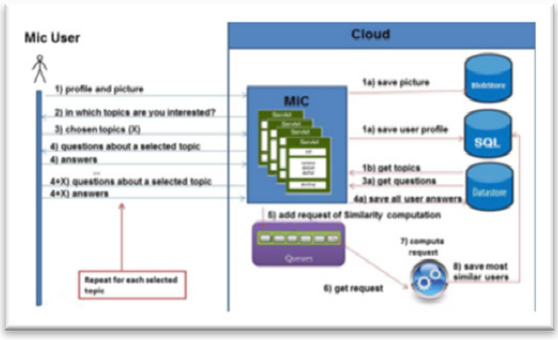


Figure 1: Overview of the MiC Application.

based DMS. Developers will then be able to develop high performance applications, losing the least as possible when moving data from one DMS to another.

2.2 Motivating Example: The MiC Application

The MiC (Meeting in the Cloud) application (F., D., M., D., & E., 2013) is a social network which allows users to maintain user profiles in which they register they topics of interests. The MiC application then groups users by similarity, allowing users to interact with their "best contacts", based on the answers given by each user in their profiles. Figure 1 overviews the main workflow of use of the MiC application.

Figure 3 presents a simplified view of the data model behind the MiC application. It stores, `Messages` posted by `UserProfiles` in `Topics` associated to `Questions`. `UserRatings` store ratings given by `UserProfiles` to `Topics`. `UserRatings` also include `Pictures` of users. Finally, and `UserSimilarity` stores pairs of similar users.

When developing this application, developers need to decide on using an infrastructure or platform as a service solution; and then on which specific provider the application is going to be deployed. When it comes to designing the data layer of the application, the developer has to decide on which DMSs will be reused and which part of the data is going to be stored on each DMS.

In order to illustrate the complexity of these choices, let us suppose the developers want to use platform as a service cloud providers, in order to reduce the cost of managing the infrastructure and to focus on the application design. Suppose they want to choose between Microsoft Azure, Heroku and Google App Engine.

Provider	DMS
Microsoft Azure	Table Service, Blob Service
Heroku	Postgres, Cloudant add-on
Google App Engine	Datastore, Blobstore

Figure 2: Comparing possible platform as a service providers for the MiC application.

Without going into the details on each DMS, Figure 3 shows that each provider includes a variety of different data stores. Each DMS supports slightly different kinds of data, with different levels of details.

For example, on the one hand, blob services support hash like structures that associate binary data to unique keys. On the other hand, table services, Google Datastores and the Cloudant add-on store multiple pieces of data associated to a single key. The former are optimal for queries on leys, while the later may support filters and more complex queries on the values associated to each key.

On top of that, each provider has different pricing strategies. The developer then needs to understand the trade-offs when designing the MiC application, in order to eventually store part of the data in one cloud and part of the data on another.

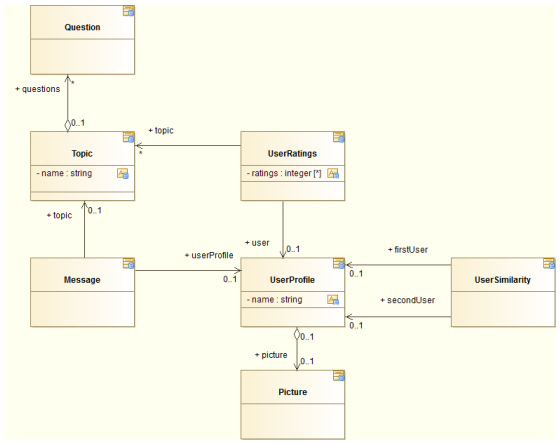


Figure 3: The data model of the MiC Application.

This paper focuses on the trade-offs involved in store different kinds of data in different DMSs, eventually in different cloud providers. The cost optimization involved in this task **is out of the scope of the present paper.**

3 CLOUD BASED DMS CONCEPTS AND COMPARISSON

In this section we present the main concepts concerning the data design in a Big Data Real-time system. These concepts are going to be presented in Section 3.1 and used to compare the most popular DMSs in Section 3.2.

3.1 Main Concepts

The main concepts related to **data structures** are summarized in Figure 5. They are based on an

extensive review of relational and NoSQL DMSs initially published in (SOFTEAM; University of York, 2013). The first aspect to be dealt with at this section is the **Underlying Data Abstraction** supported by the database. That is important because storing the same data under different data abstractions may lead to data loss and/or increase the complexity of the application code.

The next aspect is the one of how each system **uniquely identifies** data stored in it. This is usually done by means of a piece of data called **Key**. Notice that keys are dispensable in object oriented databases; because objects are unique by themselves, no matter the data they contain. Keys may be atomic or composed of many pieces of data (they are then called **Composite Keys**). Additionally, **File Paths** are special kinds of keys that uniquely identify documents in file systems. Finally, keys may be **Ordered** or not.

The **Values** stored in the database are represented differently from one system to another. They can represent **Single** or **Multiple** columns containing primitive types only or **Documents,** which stand for non-structured blobs of information. Finally, columns may be **Single** or **Multi-valued**

Links between different pieces of information are established differently in different kinds of database. In tuple based ones, **Foreign-Keys** are generally used, while in object oriented ones **Relationships** are used**.** A relationship is a direct link from an object to another, allowing navigation usually in constant time. Foreign-keys link two tuples by adding the key from one tuple as part of the columns represented in another. Lookups from tuples using foreign-keys may vary from logarithm time complexity in single primitive ordered keys, to linear time in non-ordered keys.

Different tools also provide different strategies for **Aggregating** data. **Tuple Spaces** and **Regions** group objects or tuples in different containers, so that items that are most accessed together (from a single region) can be retrieved more quickly. Tuple spaces differ from regions by the fact that they are also a concurrent programming mechanism: processes can **put** and **take** tuples from the tuple space, i.e. no two processes can take the same tuple at the same time. The third aggregation technique is called **Column Families.** In this case, the columns that form each tuple are grouped into families of columns that should be stored together, accelerating analysis over the whole column (e.g. summing all values).

3.2 Comparing Cloud based DMSs

Figure 5 presents and compares the main kinds of Big Data databases based on the concepts presented in the previous section and presents the main implementations for each category of database.

Distributed File Systems represent data as an association between file paths (used as keys) and documents (that represent file content). The underlying data abstraction paradigm is the object oriented one, i.e. files are not uniquely identified by their content, but only by their paths (usually, several paths may point to a single file).

The **Key-Value Stores** represent data as simple tuples containing simple primitive keys and a single column of data. **Ordered Key-Value** stores support

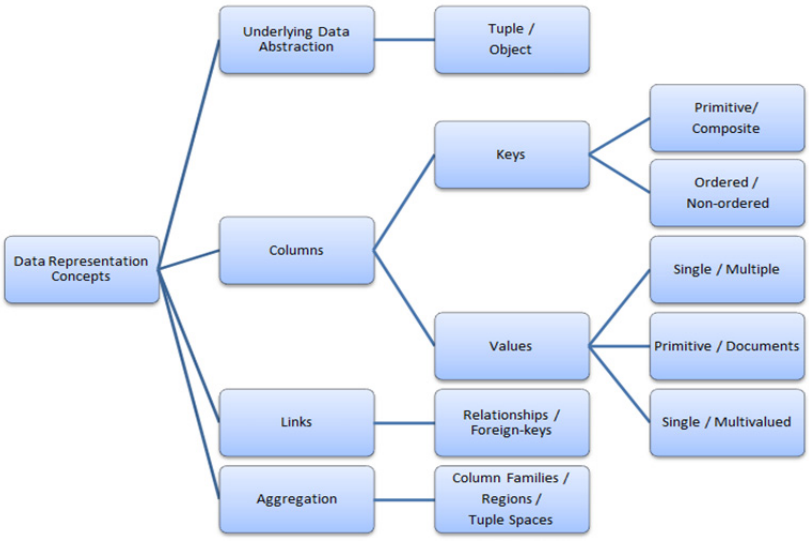


Figure 4: Kinds of Data Structure related concepts.

ordered keys, and therefore allow retrieval of ranges in linear time on the length of the range, whereas in Key-value systems this operation may be quadratic. **Document Stores** are also a special case of key-value stores in which the single column in each row (apart from the key) can store a arbitrarily complex document. Notice that these four kinds of data stores may be referred to generally as **Key-Value Stores**.

Big Tables are tuples with a primitive key and multiple columns aggregated in families and rows that can be grouped into regions. **Object Databases** represent objects which contain multiple columns (or fields) and are connected by means of relationships. **Multivalued Databases** are systems that allow more than one value to be stored at a time for a column. Expressiveness and PErformance Trade-offs.

Tuple Stores are databases that support tuple spaces. Finally, **Relational Databases** are tuple based databases supporting composite keys and foreign keys.

This section presents the trade-offs in the different data structures used by different DMSs and in the underlying limitations of these structures. Notice that complexity estimates are given relative to abstract algorithms needed to back such structures on the general case. Specific versions of sets of configuration parameters of some DMSs may achieve **better** time or space complexity for specific classes of input, which are out of the scope of this paper.

Category	Underlying Data Abstraction	Keys	Values	Links	Aggregation	Examples
Distributed File Systems	Object	Primitive (File Path)	Document	-	-	HDFS, Lustre
Key-value Store	Tuple	Primitive	Single Column	-	-	Amazon DynamoDB
Ordered Key-value Store	Tuple	Ordered	Single Column	-	-	Memcache DB, Redis
Document Store	Tuple	Primitive	Document	-	-	MongoDB, CouchDB, Riak SimpleDB
Big Table	Tuple	Primitive	Multiple Columns	-	Column Families, Regions	Google BigTable, Cassandra,
Object Database/RDF Store	Object	-	Multiple Columns	Relationships	-	Neo4j, RavenDB, FlockDB, InfiniteGraph
Multivalued databases	Tuple		Multiple Multivalued Columns		-	jBASE, Caché
Tuple store	Tuple	-	Multiple Columns	-	Tuple Spaces	Gigaspace, Javaspaces, Tarantool
Relational Database	Tuple	Composite Primitive	Multiple Columns	Foreign Keys	-	MySQL

Figure 5: Comparing cloud enabled databases.

3.3 Underlying Data Abstraction: Tuples x Objects

In practice, tuple based data models are the best suited for **data aggregation**, while the object based ones are the best suited for **navigation**. To illustrate that, let us consider a social network with tenths of millions of users. On the one hand, when storing data for computing statistical information about these users, e.g. they average age, one should prefer a tuple based data model (probably on top of a vertical partitioning scheme). On the other hand, when storing data for computing properties of the graph of friend-to-friend connections, e.g. computing a series of suggestions of products to buy for each user based on his/her friends, one should prefer an object oriented data model (probably using horizontal partitioning).

Expressiveness trade offs

Tuples are uniquely defined **by their contents**, while objects are unique **by themselves**, the main consequence of that is that two objects with the same contents will be interpreted as the same tuple, unless an internal identifier field is created to ensure uniqueness of objects.

Performance trade offs

- Navigating between tuples is a **linear time operation**[1], while it is a **constant time** operation for objects. Hashing techniques can be used to reduce this time, but they imply **extra memory cost**.
- Operations on all tuples (e.g. filtering, aggregating and bulk updates) are **cheap** on tuples but may be **expensive** on objects.

3.4 Keys

3.4.1 Primitive x Composite

Keys are used for uniquely identifying elements and for retrieving them from the database.

Expressiveness trade offs

- Migrating from composite to primitive keys has the disadvantage that the uniqueness constraint on the multiple parts of the key will not be enforced by the DMS, it thus **needs to be enforced by the application**.

Performance trade offs

- Furthermore, the DMS **may not be able** to retrieve an element my multiple keys, this may increase the cost of element retrieval if it **needs to be implemented by the application**.

3.4.2 Non-Ordered x Ordered

This feature is mainly used to **speed up retrieval operations**, it however slows down insert and deletion operations.

Expressiveness trade offs

None

Performance trade offs

- This has a great impact on the query patterns that are natively supported by database systems and on their computational cost. If keys are ordered, one can retrieve a range of keys in **linear time**, while for an unordered set, the worst case of this operation has a **quadratic time complexity**.

3.5 Values

3.5.1 Single x Multiple

The main advantage of supporting single columns is that the schema of the database normally doesn't need to be defined in advance.

Expressiveness trade offs

- Migrating from single to multiple DMSs is trivial. Conversely, multiple columns can easily be simulated as metadata attached to documents, however, in most databases, the schema associated to multiple DMSs **may be lost in this translation**.
- Single DMSs normally **do not support** queries other than given a key returning or changing the value associated to it.

Performance trade offs

- The ability of storing multiple columns for a single value is a mere convenience offered by the DMSs. Migrating from multiple to single valued databases will certainly **increase the complexity** of the application code decoding the values stored in the database or encoding sets of objects into a single value.

3.5.2 Primitive x Documents

Documents are complex structured elements, ranging from a binary blob annotated with metadata to complex trees of elements. They are mainly used to de-normalize the data model, and then increasing retrieval speed of large amounts of data.

Expressiveness trade offs

- As for single x multiple values, documents can be represented as primitive values, but

this will **increase the complexity** of the application code.

Performance trade offs

- Going from primitive to document based may be trivial, but will **certainly sub utilise the resources** of the DMS.

3.5.3 Single x Multivalued

Multiple values are usually a syntactic sugar provided by DMSs as they can usually be implemented at application side without too much increase in complexity.

Expressiveness trade offs

- The ability of storing multiple values can be seen as a mere convenience offered by the DMSs, but migrating from multiple to single valued databases will certainly **increase the complexity** of the application code decoding the values stored in the database or encoding sets of objects into a single value.

Performance trade offs

None

3.6 Links: Relationships x Foreign-Keys x No Relationships

To put it simply, relationships are links between objects whereas foreign keys represent links between tuples. In order to understand the trade-offs between both cases, refer to the subsection on objects and tuples.

Expressiveness trade offs

- Migrating from a DMS without relationships to one with relationships or foreign keys is easy, but the other way **is not**. In this case one needs to **simulate relationships on the application code**.

Performance trade offs

- The trade-off here will be between finding elements in linear time or updating and deleting elements in linear time.
    - On tuple based DMSs, navigating will take linear time, which may **become a bottleneck on the long run**.
    - On Document based DMSs, one can try to de-normalize the data model, but that **will increase the complexity of the code** associated to update and delete operations.

3.7 Aggregation: Column Families X Regions or Tuple Spaces

Column families and regions are best suited to completely different use cases. The former target queries that aggregate information on a subset of columns while the later target queries that collect information on subsets of elements. These approaches imply completely different partitioning strategies: the former allocates **columns** to different nodes while the later allocates **elements** to different nodes.

Expressiveness trade offs

None

Performance trade offs

- Running queries that are not appropriate to a particular kind of database may result in **huge bottlenecks** since the DMS needs to look up data in potentially all nodes in order to answer to queries.

4 APPLICATION TO THE MIC MULTI-CLOUD CASE STUDY

In this section we use our concepts and trade off analysis to compare six DMS from three different platform as a service providers on the MiC case study presented in Section 2.2.

4.1 Method

The main objective of such comparison is to understand the trade-offs involved in representing data in one of the target DMSs. Notice that in this case study we limited ourselves to comparing the DMSs provided by large-scale generic providers addressing any kind of application as they would be the first clouds to consider for application developers. As explained in Section 2.2, the motivation for this analysis to developers, is to understand the hidden costs involved into storing parts of the application on different cloud providers.

The targeted use cases are mainly the choice of initial DMS to store data, and besides to eventually carry a migration from one data store to another. In order to do that, our method consists in classifying the data we need to represent and the data supported by the target DMSs. The comparison of the needed and available support will hopefully guide the developer into writing the code of the application and to avoid pitfalls involving eventual incompatibilities between DMSs.

			Underlying Data Abstraction	*Keys*	*Values*	*Links*	*Aggregation*
Application Data							
	User Profile		Objects	Non-ordered	Multiple, Primitive, Single	Yes	Regions
	UserSimilarity		Tuple	Non-ordered	-	Yes	Regions
	Picture		Tuple	Non-ordered	Single, Primitive, Single	No	Regions
PaaS under consideration		**DMS**					
	Azure						
		Table Storage	Tuple	Primitive, Ordered	Multiple, Primitive, Single	-	Regions
		Blob Storage	Tuple	Primitive	Single, Primitive, Single	-	Regions
	Heroku						
		Postgres	Tuple	Primitive	Multiple, Primitive, Single	Foreign Keys	-
		Cloudant	Tuple	Primitive	Single, Document, Single	-	-
	Google App Engine						
		Datastore	Tuple	Primitive	Multiple, Primitive, Single	-	Regions
		Blob Storage	Tuple	Primitive	Single, Primitive, Single	-	-

Figure 6: Case study: comparing storage options for the MiC application.

Figure 6 presents the result of this comparison for both the application data model and the target cloud DMSs (detailed respectively in Sections 4.2 and 4.3.

4.2 Analysing the Application Data Model

We divide the data model presented in Figure 3 into three parts, presented on the top part of Figure 6. For the sake of simplicity we didn't include all parts of the data model in this comparison. The three parts are: (i) the `UserProfile`, concerning the user profiles and related pieces of information; (ii) the `UserSimilarity`, concerning the data store that stores the users that are similar to a given user profile; and (iii) the `Picture`, concerning the picture linked to each user profile.

We classify each part of the data model using the concepts presented in Section 3.1. User profiles are object oriented information, because two profiles may refer to the same pieces of information and still represent different users. `UserSimilarity` and Picture are different, because they should refer or belong to a specific user profile. In all cases, no key ordering is necessary in the MiC application. However, the ability to group pieces information by "region" is important in all cases: user profiles and related data a very geographically specific, and should all be located in the same geographic region to speed up computation. When it comes to values and links, user profiles should contain primitive values to represent the pieces of information that compose a user profile (e.g. name, gender, location, data of birth etc.). Pictures only contain a binary blob representing the picture, and the user similarity only contain references to similar user profiles.

4.3 Analysing Target Cloud DMSs

At the bottom part of Figure 6, we present the platform as a service providers presented in Figure 2 and their respective DMSs. We then classify the DMSs according to the same criteria used to classify the parts of the data model on the top of the table. All DMSs represent tuples, i.e. they do not support objects directly. They also enforce the use of primitive keys and most of them (except for Postgres, a relational database) do not support links or foreign-keys between tuples. Only Azure DMSs and the Google Data store support grouping elements that need to be accessed in the proximity of a geographic area. The main difference between the supported DMSs is in the supported expressiveness of columns/values: Azure Table Storage, Postgres and the Google Datastore all support multiple primitive single valued values. The Azure Blob storage, Heroku cloudant and Google Blob storage all support one single valued value, which may be a document in Cloudant, or a single value in the other ones.

4.4 Analysing Cloud Migration Scenarios

Figure 7 overviews the trade-offs that need to be faced by a developer intending to develop the MiC application and host it on the three target platform as a service providers. Notice that different DMSs have different trade-offs that need to be taken into account during application development and future maintenance and eventual migration. Let us show how this table may be useful in two eventual migration scenarios.

Based on the analyses provided in Figure 6 and Figure 7, in the next section we will analyse to specific hypothetical deployment and migration

Trade offs:	Azure		Heroku		Google App Engine	
Data x DMS	**Table Storage**	**Blob Storage**	**Postgres**	**Cloudant**	**Datastore**	**Blob storage**
User Profile	Objects x Tuples No links	Objects x Tuples Multiple = Single value No links	Objects x Tuples FK for links No regions	Objects x tuples Multiple => Single value No regions No links	Objects x Tuples No links	Objects x Tuples, Multiple x Single value, No links, No regions
UserSimilarity	No links	No links	No regions	No regions	No links	No links, No regions
Picture			No regions	No regions		No regions

Figure 7: Trade-offs between DMSs.

scenarios.

4.4.1 Migration Scenario 1

Let us suppose that the developer chooses the Azure blob storage to store the user profile pictures since there are no significant differences between the required and provided expressiveness. If later the company decides to migrate to Heroku Cloudant or Google App Engine Blob Storage, the developer needs to notice that these DMSs do not support the aggregation of tuples by geographic region/access frequency. Both databases provide a generic algorithm that distributes data and then balances query answering resources. As described in Section 3.7 this may generate a performance bottleneck if data is not properly distributed. Developers should be therefore aware of this potential limitation.

4.4.2 Migration Scenario 2

In this scenario, let us suppose that the developer decided to deploy the `UserSimilarity` part of the data model in an instance of Azure Table Storage and that in the future, she or he decides to deploy part of this information in Heroku Postgres (e.g. as part of a spin off social network).

When initially deploying data on Azure, the developer needs to handle the fact that links between elements are not supported at this database. This therefore needs to be implemented in the application code (cf. Section 3.6). Since the target DMSs supports links, this does not need to be supported by the application any more. However, special care needs to be taken during the data migration with the application provided implementation of links. This should be done in order to avoid data loss during the migration or loss of functionality when part of the old application code will be fulfilled by the DMS itself.

5 RELATED WORK

The main problem addressed by this paper is the one of understanding the trade-offs between different cloud DMSs, in order to optimize the deployment of application data in multiple clouds. Past work has tried to address this problem but in different ways. We classify these works into two categories: (i) the ones that try to hide this complexity from the developer, (ii) the ones that allow the developer to work on surpassing such complexity.

We consider that approaches in the first category are not best suited to developers that need to extract the most from cloud data stores, since any black box that hides the real complexity of the DMSs is going to be efficient only in a restricted set of situations. The present work falls in the second category, but differently from other works, that try to provide tools under which the developer can himself try to bridge the semantic gap between different tools, we show explicitly the gap and the involved gaps to the developer.

In the **first category** we would put the systems that try to automatically bridge the gap between different database categories. This group starts out by the tools that facilitate the use of relational data stores by object oriented applications (DB-UML Database Modeling Tool) (Hibernate: Relational Persistence for Java and .NET) (DeMichiel, 2009).

In the non-relational word some tools try to do the same. A first set of tools (Acid House) (Kundera) (PlayORM) (DataNucleus Access Platform) (Hibernate Object/Grid Mapper) (Morphia) reuses the concepts defined by JPA, which is a very popular system of annotations over Java code (i.e. an object oriented model of data), to translate an object oriented model represented by a set of Java classes into a non-relational databases. Other tools do the same thing for relational models (Toad for Cloud) (eobjects.org MetaModel). They provide a relational SQL-based interface to non-relational NoSQL databases, allowing existing

relational modelling approaches to be reused to model non-relational databases. Finally, Spring Data (Spring Data), provides different interfaces for different NoSQL databases.

This comes with the drawback of the inherent loss of information in the translation process or the loss of *"object-orientedness"* in the object oriented model in some corner cases.

Other approaches do not try to hide the non-relational concepts behind relational ones, but instead, propose unified abstract modelling languages. These languages try to represent the common concepts that are present in many different non-relational stores in a uniform way. Two examples of such languages are FQL (Federated Unfied Query Language, FunQL) and UnQL (UnQL Specification). The former received this name because it was created to support "federations" of databases. A federation of databases is a set of data stores, possibly storing data under different paradigms (relational or non-relational). The FQL language is then based on SQL but is able to query non-relational data bases. Its main drawback is that it supports only data retrieval, i.e. it provides no Data Definition Language. A similar approach for dealing with federated databases can be found in (JBoss Teiid). The UnQL language stands for Unstructured Query Language. It follows a similar approach, but is limited to unstructured (and therefore non-relational databases). It is targeted only to data stores containing JSON documents.

On the **second category** we will find tools such as such as Pentaho (Pentaho) and Yahoo! Pipes (Yahoo Pipes), which are Data Integration tools. They offer visual editors that allow one to describe how data coming from different sources, following different schemas and data types can be mapped into different data types and then fed to other systems. The semantic gap between different DMSs needs to be understood and filled by the developer.

In scientific literature, some papers also discuss the differences between the offerings of cloud providers and their supported DMSs. A good example of this kind of work is (Rimal, Sch. of Bus. IT, Choi, & Lumb, 2009). In this work, the different cloud providers are described along with their features and storage solutions. However the referred paper focuses on runtime characteristics (security, load balancing, fault tolerance etc.) and not on the impact of the design time storage choices to the cloud application.

More recent works such as (Cattell, 2010), (Hecht & Jablonski, 2011) and (Moniruzzaman & Hossain, 2013) go into the concepts behind different DMSs, their runtime properties, preferred use cases and supported queries. However these works are usually restricted to some specific kinds of cloud storage (usually variations of key-valued stores), and compare tools mostly based on runtime characteristics instead of design time ones.

6 CONCLUSION

The multiplication of cloud providers has both positive and negative impacts on industrial applications. On the one hand, the increasing availability and multiplicity of cloud providers allows for the existence of clever applications profiting from the best of different providers. On the other hand, the fragmentation of the market makes developing such applications much harder. In particular, maintaining them (fixing bugs and eventually moving to other clouds) becomes much harder than for regular non-cloud applications.

In this paper we investigated this problem in the point of view of the developer that needs to design data structures that will be potentially deployed on different clouds and on different data management systems (DMS). More specifically, we investigated the main concepts behind the different DMS and the semantic gap between different databases.

The present work is a first step on the direction of providing some automated support to developers and is going to be extended as part of the FP7 projects MODAClouds and JUNIPER. As future works, we are currently working on providing automated tools for analysing data models and proposing better data structures, and verifying if they respect a given set of real-time constraints on a multi-cloud setting. The extension of the model presented in this paper with other concerns unrelated to data structures (i.e. support to transactions, programming language integration etc.) is also under consideration.

ACKNOWLEDGEMENTS

The research reported in this article is partially supported by the European Commission grant no. FP7-ICT-2011-8- 318484 (MODAClouds).

The research reported in this article is partially supported by the European Commission grant no. FP7-ICT-2011-8- 318763 (JUNIPER).

REFERENCES

Acid house. (n.d.). Retrieved november 8, 2013, from https://github.com/eiichiro/acidhouse

Cattell, R. (2010). Scalable SQL and NoSQL data stores. *ACM SIGMOD Record* , 12-27.

DataNucleus Access Platform. (n.d.). Retrieved November 8, 2013, from http://www.datanucleus.org/

DB-UML Database Modeling Tool. (n.d.). Retrieved November 8, 2013, from http://argouml-db.tigris.org/

DeMichiel, L. (2009). *JSR 131:Java Persistence API, Version 2.0.* Sun Microsystems.

eobjects.org MetaModel. (n.d.). Retrieved November 8, 2013, from http://metamodel.eobjects.org/index.html

F., G., D., L., M., S. Y., D., A., & E., D. N. (2013). An Approach for the Development of Portable Applications on PaaS Clouds. *Proceedings of the 3rd International Conference on Cloud Computing and Service Science (CLOSER 2013)*, (pp. 591-601).

Federated Unfied Query Language, FunQL. (n.d.). Retrieved November 8, 2013, from http://funql.org/

Han, J., Haihong, E., Le, G., & Du, J. (2011). Survey on NoSQL database . *Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on* , (pp. 363 - 366). Port Elizabeth .

Hecht, R., & Jablonski, S. (2011). NoSQL evaluation: A use case oriented survey . *Cloud and Service Computing (CSC), 2011 International Conference on* , (pp. 336-341). Hong Kong .

Hibernate Object/Grid Mapper. (n.d.). Retrieved November 8, 2013, from http://www.hibernate.org/subprojects/ogm.html

Hibernate: Relational Persistence for Java and .NET. (n.d.). Retrieved November 8, 2013, from http://hibernate.org

JBoss Teiid. (n.d.). Retrieved November 8, 2013, from http://www.jboss.org/teiid/

Khajeh-Hosseini, A., Greenwood, D., & Sommerville, I. (2010). Cloud Migration: A Case Study of Migrating an Enterprise IT System to IaaS . *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on* , (pp. 450 - 457). Miami, FL .

Kundera. (n.d.). Retrieved November 8, 2013, from https://github.com/impetus-opensource/Kundera

Liu, T., Katsuno, Y., Sun, K., & Li, Y. (2011). Multi Cloud Management for unified cloud services across cloud sites . *IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS)*, (pp. 164-169). Beijing.

Moniruzzaman, A. B., & Hossain, S. A. (2013). NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison. *International Journal of Database Theory and Application*, 1-14.

Morphia. (n.d.). Retrieved November 8, 2013, from http://code.google.com/p/morphia/

Pentaho. (n.d.). Retrieved November 8, 2013, from http://www.pentaho.com/

PlayORM. (n.d.). Retrieved November 8, 2013, from https://github.com/deanhiller/playorm

Rackspace. (2013, February 13). *88 per cent of cloud users point to cost savings, according to Rackspace Survey*. Retrieved June 2013, from http://blog.rackspace.co.uk/in-the-industry/88-per-cent-of-cloud-users-point-to-cost-savings-according-to-rackspace-survey/

Rimal, B., Sch. of Bus. IT, K. U., Choi, E., & Lumb, I. (2009). A Taxonomy and Survey of Cloud Computing Systems. *Fifth International Joint Conference on INC, IMS and IDC, 2009. NCM '09.*, (pp. 44-51). Seoul.

Singh, Y., Kandah, F., & Zhang, W. (2011). A secured cost-effective multi-cloud storage in cloud computing . *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, (pp. 619-624). Shanghai.

SOFTEAM; University of York. (2013). *D5.1 – Foundations for MDE of Big Data Oriented Real-Time Systems.*

Spring Data. (n.d.). Retrieved November 8, 2013, from http://www.springsource.org/spring-data

Toad for Cloud. (n.d.). Retrieved November 8, 2013, from http://toadforcloud.com/index.jspa

UnQL Specification. (n.d.). Retrieved November 8, 2013, from http://www.unqlspec.org

Yahoo Pipes. (n.d.). Retrieved November 8, 2013, from http://pipes.yahoo.com/pipes/

A Multi-Cloud Framework for Measuring and Describing Performance Aspects of Cloud Services Across Different Application Types

G. Kousiouris[1], G. Giammatteo[2], A. Evangelinou[1], N. Galante[2], E. Kevani[1], C. Stampoltas[1], A. Menychtas[1], A. Kopaneli[1], K. Ramasamy Balraj[2], D. Kyriazis[1], T. Varvarigou[1], P. Stuer[3] and L. Orue-Echevarria Arrieta[4]

[1]*Dept. Of Electrical and Computer Engineering, NTUA, 9 Heroon Polytechnioy Str, 15773, Athens, Greece*
[2]*Research and Development Laboratory, Engineering Ingegneria Informatica S.p.A., V. R. Morandi, 32 00148 Roma, Italy*
[3]*Spikes Research dept., Spikes, Mechelsesteenweg 64, B-2018 Antwerpen, Belgium*
[4]*TECNALIA, ICT-European Software Institute Division, Parque Tecnológico Ed #202, E-48170 Zamudio, Spain*
gkousiou@mail.ntua.gr, gabriele.giammatteo@eng.it, aevang@mail.ntua.gr, nunzioandrea.galante@eng.it, eltonkevani@hotmail.com, stampoltaschris@gmail.com,{ameny, alikikop}@mail.ntua.gr, kanchanna.ramasamybalraj@eng.it, dimos@mail.ntua.gr, dora@telecom.ntua.gr, peter.stuer@spikes.be, leire.orue-echevarria@tecnalia.com

Keywords: Benchmarking, Cloud Services, Multi-Cloud, Performance.

Abstract: Cloud services have emerged as an innovative IT provisioning model in the recent years. However, after their usage severe considerations have emerged with regard to their varying performance due to multitenancy and resource sharing issues. These issues make it very difficult to provide any kind of performance estimation during application design or deployment time. The aim of this paper is to present a mechanism and process for measuring the performance of various Cloud services and describing this information in machine understandable format. The framework is responsible for organizing the execution and can support multiple Cloud providers. Furthermore we present approaches for measuring service performance with the usage of specialized metrics for ranking the services according to a weighted combination of cost, performance and workload.

1 INTRODUCTION

Performance of Cloud environments has started to gain significant attention in the recent years (Hauck, 2010). After promises for infinite resources and on-demand scalability, the issues of Cloud environments instability with regard to performance issues of the allocated resources have begun to arise (Kousiouris et al., 2012). Thus, for a successful Cloud migration process, the issue of provider performance should be taken very seriously, in order to save money but also guarantee a (as much as possible) stability in the migrated application. As identified in our previous work (ARTIST Consortium D7.2, 2013) a significant gap in existing research is the lack of such descriptions in current metamodels regarding Cloud infrastructures. However, different providers may have their own metrics and strategies for guaranteeing Cloud QoS. Thus a more abstracted and common way should be found for identifying performance aspects of Cloud environments.

The main performance aspects of Cloud computing as analysed by the ARTIST approach can be summarized as:

a) Heterogeneous and unknown hardware resources: the computing resources offered by the Cloud providers are unknown to the external users. Available information may be limited to number of cores for example, memory sizes or disk quotes. However this level of information is far from sufficient in order to characterize the provider's hardware capabilities that may depend also on architecture, interconnection, RAM speeds etc. According to a study on Amazon platform conducted by Aalto University (Zhonghong Ou et al., 2012), the variation between the fast instances and slow instances can reach 40%. In some

applications, the variation can even approach up to 60%.

b) Different configurations: even in the existence of the same hardware however, the way this resource is configured plays a significant role in its performance. The same applies for software configurations (e.g. a DB instance over a virtual cluster) or variations in the software development.

c) Multi-tenancy and obscure, black box management by providers: Cloud infrastructures deal with multiple different users that may start their virtual resources on the same physical host at any given time. However, the effect of concurrently running VMs for example (Kousiouris et al., 2011) significantly degrades the actual application performance. This is even more affected by the usage patterns of these resources by their virtual owners or their clients. Furthermore, consolidation decisions made by providers and that are unknown to the users may group virtual resources on the same physical node at any given time, without informing the owner.

d) VM interference effects. In (Koh et al., 2007) an interesting research investigates the performance interference for a number of applications in experimental virtual environments that were selected for classifying their behaviour using different metrics. The result from the research shows that combined performance varies substantially with different combinations of applications. Applications that rarely interfere with each other achieve performance to the standalone performance. However, some combinations interfere with each other in an adverse way. Furthermore, virtualization is a technology used in all Cloud data centers to ensure high utilization of hardware resources and better manageability of VMs. Despite the advantages provided by virtualization, they do not provide effective performance isolation.

All these aspects plus the fact that Cloud providers are separate entities and no information is available on their internal structure and operation, makes it necessary to **macroscopically** examine a provider's behaviour with regard to the offered resources and on a series of metrics. This process should be performed through benchmarking, by using the suitable tools and tests. One of the key aspects is that due to this dynamicity in resource management, the benchmarking process must be iterated over time, so that we can ensure as much as possible that different hardware, different management decisions (like e.g. update/reconfiguration/improvement of the infrastructure) are demonstrated in the refreshed metric values, but also observe key characteristics such as performance variation, standard deviation etc. Finally, the acquired information should be represented in a machine understandable way, in order to be used in decision making systems.

The aim of this paper is to provide such mechanisms to address the aforementioned issues. A benchmarking framework designed in the context of the FP7 ARTIST project is presented in order to measure the ability of various Cloud offerings to a wide range of applications, from graphics and databases to web serving and streaming. The framework has defined also a number of templates in order to store this information in a machine understandable fashion, so that it may be used by service selection mechanisms. What is more. we define a metric, namely *Service Efficiency* (SE), in order to rank different services based on a combination of performance, cost and workload factors.

The paper is structured as follows. In Chapter 2, an analysis of existing work is performed. In Chapter 3 the description of the ARTIST tools for mitigating these issues is presented, while in Chapter 4 a case study on AWS EC2 resources is presented. Finally, conclusions and future work are contained in Chapter 5.

2 RELATED WORK

Related work around this paper ranges in the fields of performance frameworks, available benchmark services and description frameworks and is based in the according analysis performed in the context of the ARTIST project (ARTIST Consortium D7.2, 2013). With regard to the former, the most relevant to our work is (Garg, 2012). In this paper, a very interesting and multi-level Cloud service comparison framework is presented, including aspects such as agility, availability, accountability, performance, security and cost. Also an analytical hierarchical process is described in order to achieve the optimal tradeoff between the parameters. While more advanced in the area of the combined metric investigation, this work does not seem to include also the mechanism to launch and perform the measurements. Skymark (Iosup et al., 2012) is a framework designed to analyze the performance of IaaS environments. The framework consists of 2 components – Grenchmark and C-Meter. Grenchmark is responsible for workload generation and submission while C-Meter consists of a job scheduler and submits the job to a Cloud manager

that manages various IaaS Clouds in a pluggable architecture. Skymark focuses on the low level performance parameters of Cloud services like CPU, Memory etc. and not on elementary application types.

CloudCmp (Li, 2010) provides a methodology and has a goal very similar to our approach to estimate the performance and costs of a Cloud deployed legacy application. A potential Cloud customer can use the results to compare different providers and decide whether it should migrate to the Cloud and which Cloud provider is best suited for their applications. CloudCmp identifies a set of performance metrics relevant to application performance and cost, develop a benchmarking task for each metric, run the tasks on different providers and compare. However CloudCmp does not seem to define a common framework for all the benchmark tasks.

With regard to benchmarking services, the most prominent are CloudHarmony.com and CloudSleuth.com. The former utilizes a vast number of benchmarks against various Cloud services, offering their results through an API. However, there are two aspects that can be improved with relation to this approach. Initially it is the fact that too many benchmarks are included in the list. We believe that a more limited scope should be pursued in order to increase the focus of the measurements. Furthermore, the measurement process is not repeated on a regular basis, in order to investigate aspects such as deviation. For CloudSleuth, the focus is solely on web-based applications and their response time/availability. Their approach is very worthwhile, by deploying an elementary web application across different providers and monitoring it constantly, however it is limited to that application type.

With regard to description frameworks, a number of interesting approaches exist. According to the REMICS project (REMICS Consortium Deliverable D4.1 v2.0, 2012) PIM4Cloud, which is focused in both private and public Clouds, has been defined to provide support to model the applications and also to describe the system deployment on the Cloud environment. PIM4Cloud is implemented as a profile for UML and a meta-model which is capable to describe most of the features of a system that will be deployed in a Cloud environment. It is organized in four packages (Cloud Provider domain, Cloud Application domain, Physical Infrastructure domain and Resource domain).

FleRD (Schaffrath et al., 2012) is a flexible resource description language for inter-provider communication in virtual networks architectures. It appears enhanced with regard to realism and generality (ability to describe real world topologies), extensibility, grouping and aggregation. FleRD is mainly focused around networking elements, however its concepts of modeling more information for QoS of networks has influenced our approach.

EDML (Charlton, 2009) defines a XML syntax for declaring internal and external general parsed entities. VXDL(Koslovski et al, 2008) is an XML-based language that describes Virtual Private eXecution Infrastructure (ViPXi) which is a time-limited organized aggregation of heterogeneous computing and communication resources. VXDL can describe resources, networks' topology that are virtual but are also, to some extent, adapted to physical ones and finally to represent timeline.

The implementation of DADL (Mirkovic et al., 2010) is based on the prediction that future businesses will use allocated resources from different Clouds such as public or private to run a single application. DADL was developed as an extension of SmartFrog (framework for deploying and managing distributed software systems based on java) and it is used to specify application architecture and Cloud resources that are necessary for an application to run. There are elements to describe QoS features such as CPU speed, number of cores etc.

The main issue with the aforementioned approaches, which most of them support description of QoS terms, is the fact that in many cases the standard ways (CPU cores, frequency etc.) of describing capabilities are not sufficient to demonstrate the actual performance of virtualized resources. Thus a new approach based on benchmark scores should be pursued that would indicate the direct capability of a resource service to solve a particular computational problem. The descriptions defined in this paper are centered around this test-based approach.

3 BENCHMARKING APPROACH IN ARTIST

The benchmarking approach followed in the context of ARTIST has the following targets:

- Identify a set of common application types and the respective benchmarks to measure the performance of Cloud services
- Create a framework that is able to automatically install, execute and retrieve the benchmark

results, with the ability to support multiple providers
- Investigate aspects of Cloud service performance with regard to variation and ability across a wide range of potential applications
- Define a machine understandable way of representing this information and improved metrics that will characterize more efficiently the services.

The use case diagram for the benchmarking suite appears in Figure 1. We envisage that the capabilities of the toolkit will be exploited by an entity ("Benchmarks Provider") that will be responsible for performing and obtaining the tests, similar to the role of CloudHarmony.com. This entity will utilize the various aspects of the toolkit in order to create provider models that have concrete results and metrics per service offering, that are stored on the ARTIST repository, so that an external entity ("Model User") may be able to retrieve and consult them. More details on each part of the process are presented in the following paragraphs.

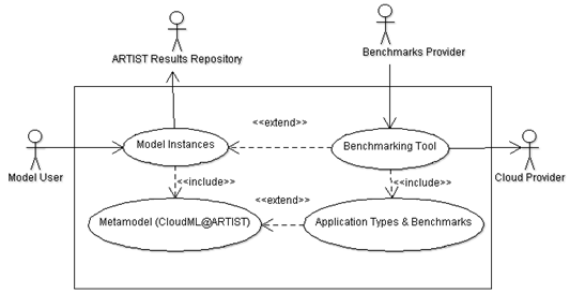


Figure 1: Use case diagram for the ARTIST Benchmarking process and tools.

3.1 Application Benchmark Types

The main target of the application benchmark types is to highlight a set of common and popular application tests that can be used in order to benchmark provider offerings. Thus each offering may have a performance vector indicating its ability to solve specific problems or cater for a specific type of computation. The set of application benchmarks used in ARTIST appears in Table 1.

Table 1: Benchmark Tests used in the ARTIST platform.

Benchmark Test	Application Type
YCSB	Databases
Dwarfs	Generic Applications
Cloudsuite	Common web aps like streaming, web serving etc.
Filebench	File system and storage

For the future these tests may be extended with other specialized tests like DaCapo benchmarking suite (Blackburn et al., 2006) for measuring JVM related aspects.

3.2 Models of Necessary Information

In order to exploit the information from the benchmark execution, a suitable machine understandable format should be in place in order to store results and utilize them in other processes like service selection. For achieving this, suitable model templates have been designed. These templates include all the relevant information needed, such as measurement aspects (number of measurements, statistical information like standard deviation etc.), test configurations and workload profiles. Initially these templates are defined as an XML schema and in later stages they will be incorporated into a suitable metamodel developed in the context of the ARTIST project (CloudML@ARTIST). Examples of types defined in the schema are portrayed in Figure 2.

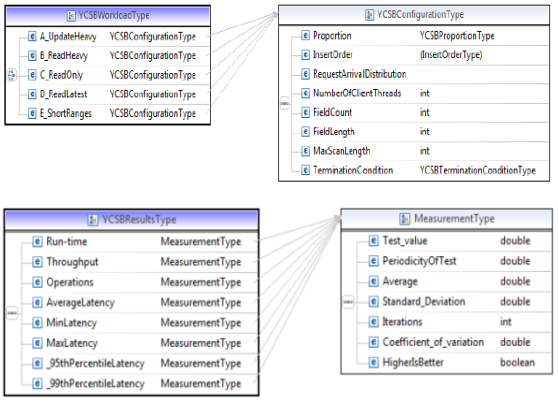


Figure 2: Examples of templates for describing benchmarking information in a machine understandable fashion. Each test is represented by an according template.

3.3 Benchmarking Suite Architecture

The Benchmarking Suite Architecture appears in Figure 4. The user through the GUI (Figure 3) may set the conditions of the test, selecting the relevant benchmark, workload conditions, target provider and service offering. This information is passed to the Benchmarking Controller which is responsible for raising the virtual resources on the target provider and executing the tests. The former is based on the incorporation of Apache LibCloud project, in order to support multiple provider frameworks. The latter needs to install first the tests, through the utilization

of an external Linux-like repository that contains test executables. Once the tests are installed (through a standard repo-based installation), the workload setup scripts are transferred to the target machines and the execution begins. Results are transferred back and processed in order to be included in the model descriptions, following the template definition highlighted in Section 3.2.

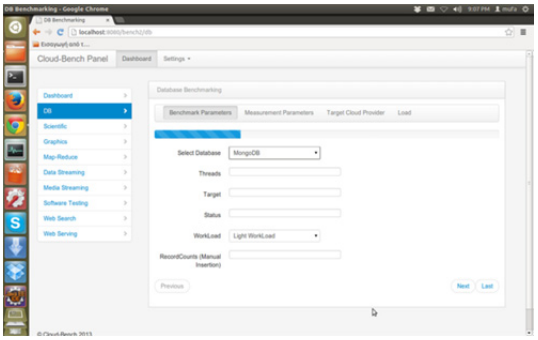


Figure 3: GUI for selecting and configuring tests on provider offerings.

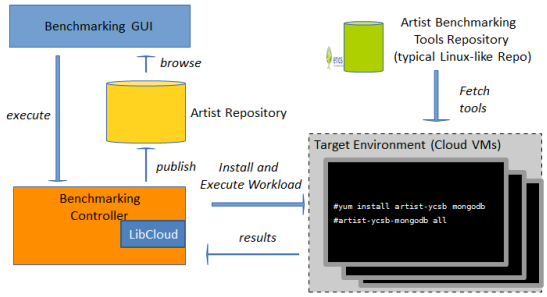


Figure 4: Overall System Architecture.

At the moment the two parts (GUI and Controller) are not integrated, however the benchmarking controller which is the main backend component can also be used standalone to perform measurements on the various application types. Installation instructions can be found in (ARTIST Consortium D7.2.1, 2013).

3.4 Service Efficiency Metric Description

In order to better express the performance ability of a service offering, we considered the usage of a metric that would fulfill the following requirements:

- Include workload aspects of a specific test
- Include cost aspects of the selected offering
- Include performance aspects for a given workload
- Give the ability to have varying rankings based on user interests
- Intuitively higher values would be better

Following these points, we considered that positive factors (e.g. workload aspects) should be used in the numerator and negative on the denominator (cost and Key Performance Indicators that follow a "higher is worse" approach). Furthermore, normalization should be used in order to have a more generic rating. Usage of sum would be preferable over product, since the former enables us to include weight factors. Thus such a metric can answer a potential question of the sort: "what is the best service to run my web streaming application, when I am interested more on a cheap service?".

The resulting formula of Service Efficiency is the following (Equation 1):

$$SE = \frac{\sum_i s_i l_i}{\sum_j s_j w_j f_j} \quad (1)$$

Where s scaling factor
l: workload metric
f: KPI or cost metric
w: weight factor

The resulting metric can be compared between different offerings, potentially of different providers **but on the same workload basis.** However the incorporation of workload is necessary since it affects the performance and thus ranking of the services. Usage of different workloads may display a different optimal selection, based on the anticipated workload conditions for the application that needs to be deployed.

4 METRIC CASE STUDY ON AMAZON EC2

In order to experiment with the metrics definitions and initially investigate differences in VM performance, we utilized the service described in (Kousiouris et al, 2013). This is a service implementation offering time series prediction, by utilizing a backend Matlab implementation of prediction models. The instance on which the service was running was changed in order to test multiple types of Amazon EC2 VM instances (micro, small, c1.medium, m1.medium).

The number of concurrent clients was set to 1, 5 and 10, with 10 representing the maximum workload (it is a computationally heavy service and it was observed that after 10 clients the delay made it

unusable). Each client launches constantly requests against the service in a serial manner, so that at all times there are that many pending requests equal to the client number. The number of calls made by a client thread is regulated according to the number of clients workload in order to accumulate 500 values for each combination 500. From these the average value and standard deviation were extracted for each case (Table 2).

Table 2: Average Measurements and Cost for combinations of EC2 instances and workload.

Instance-Clients	Avg Delay (msec)	Std Dev (msec)	Cost/Hour
micro-1	2.95E+04	1.39E+04	0.02
micro-5	1.27E+05	1.17E+05	0.02
micro-10	3.37E+05	3.38E+04	0.02
small-1	1.11E+04	558.5749	0.06
small-5	4.75E+04	3.10E+03	0.06
small-10	1.00E+05	8.92E+03	0.06
m1.medium-1	6.64E+03	106.5448	0.12
m1.medium-5	2.20E+04	2.27E+03	0.12
m1.medium-10	4.36E+04	1.27E+03	0.12
c1.medium-1	5.92E+03	61.5125	0.145
c1.medium-5	1.06E+04	954.8628	0.145
c1.medium-10	2.23E+04	1.41E+03	0.145

Then the metric SE described in Section 3.4 was applied with the following form (results appear in Figure 5):

$$SE = \frac{\#Clients}{w_1 * \text{delay} + w_2 * Cost}$$

Different weights were given to the performance and cost aspects (50-50, 90-10,10-90) and different normalization intervals were considered (1-2,1-10) in order to check the metric's sensitivity. We avoided using a normalization interval including 0 since it may lead to infinite values for some cases. One should compare between same color bars, indicating similar workloads. From the graphs it is evident how the ranking of a potential VM rating can change based on the aspect that we are most interested in. For example, for a weighted decision (50-50) with high workload, the best choice would be EC2 small instance, while for a performance-biased selection (90-10) one should focus on c1.medium instances. For cost-biased selection, one should focus on micro instances. While this may seem the obvious choice (cheaper VM for the cost-biased and more expensive for the performance-biased), this is only evident in the extreme cases. If we need an intermediate trade-off then the choice is not that obvious and the defined metric could help in this selection process.

In the future the measurements will be based on the elementary application types and benchmarks that are described in Section 3.1, in order to obtain more generic and reusable (not case specific) ratings.

Another variation that was pursued was the incorporation of the standard deviation measurements as a KPI, in order to include the interest to rank services based on their stability. Thus the metric equation was altered as follows:

$$SE = \frac{\#Clients}{w_1 * \text{delay} + \text{w}\,2 * \text{deviation} + w_3 * Cost}$$

The contents of Table 2 were then used for the combinations of equal interest (0.33 weight for all

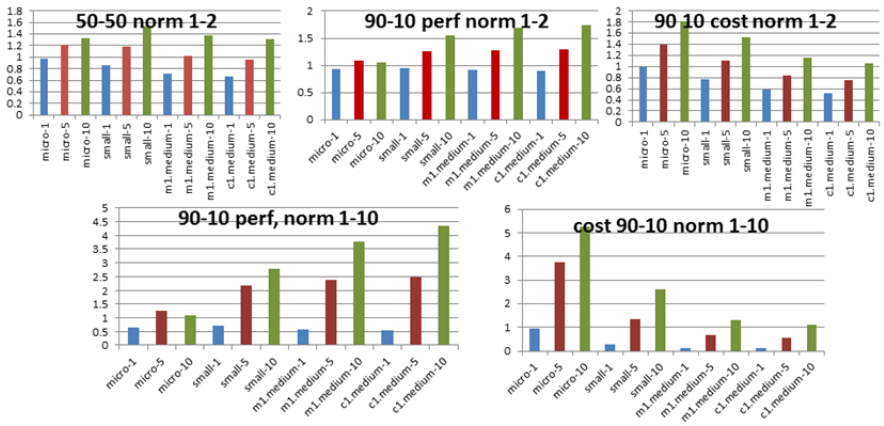


Figure 5: Application of the metric on an example service response times across different type of Amazon EC2 VM instances and variable workload (higher score is better). Comparison should be made between columns with similar color (identical workload conditions).

parameters) and performance biased interest including deviation (0.45 for performance, 0.45 for deviation and 0.1 for cost) and were compared to the standard 50-50 graph for performance and cost in Figure 6. The deviation affects the ranking (for example in the m1.medium.10 case for high workload). This would be especially helpful in cases where we have applications that are very sensitive to deviation phenomena like multimedia streaming applications.

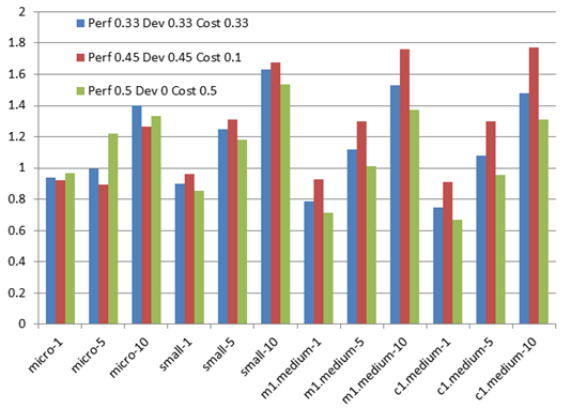


Figure 6: Inclusion of deviation as a KPI in the formula and comparison for various combinations in the normalization interval 1-2.

5 CONCLUSIONS

The emergence of Cloud computing has led to a plethora of various offerings by multiple providers, covering different needs of the consumer. However significant questions emerge for the stability of these pay-per-use resources, mainly with regard to performance, in this highly dynamic management environment. In this paper a multi-Cloud measurement framework has been presented, that has the aim of investigating these performance issues of the virtualized offerings. The framework utilizes a variety of benchmark tests in order to cover a wide range of application types and it mainly focuses on investigating aspects such as performance variations.

A combined metric (Service Efficiency) is also proposed in order to combine workload, performance and cost aspects in a single rating for comparing Cloud offerings across different providers. A case study on the Amazon EC2 Cloud has indicated the application of such a metric to characterize the offerings based on this combination.

For the future, we intend to complete the integration of the framework (currently missing the integration between GUI and Benchmark Suite Controller) and investigate the addition of tests with extended focus (e.g. JVM aspects, availability measurement aspects etc.). However the Benchmarking Controller for the execution of the tests can be used also as standalone, following the instructions in (ARTIST Consortium D7.2.1, 2013). Another interesting aspect would also be the incorporation of other non-functional aspects such as availability in the main SE metric and a comparison across different provider offerings.

ACKNOWLEDGEMENTS

The research leading to these results is partially supported by the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 317859, in the context of the ARTIST Project.

REFERENCES

ARTIST Consortium (2013), Deliverable D7.2 v1.0-PaaS/IaaS Metamodelling Requirements and SOTA, Available at: http://www.artist-project.eu/sites/default/files/D7.2%20PaaS%20IaaS%20metamodeling%20requirements%20and%20SOTA_M4_31012013.pdf.

REMICS Consortium (2012), Deliverable D4.1 v2.0 - PIM4Cloud, Available at: http://www.remics.eu/system/files/REMICS_D4.1_V2.0_LowResolution.pdf.

Kousiouris, George; Vafiadis, George; Varvarigou, Theodora, "Enabling Proactive Data Management in Virtualized Hadoop Clusters Based on Predicted Data Activity Patterns," P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), *2013 Eighth International Conference on*, vol., no., pp.1,8, 28-30 Oct. 2013.

doi: 10.1109/3PGCIC.2013.8.

George Kousiouris, Dimosthenis Kyriazis, Andreas Menychtas and Theodora Varvarigou, "Legacy Applications on the Cloud: Challenges and enablers focusing on application performance analysis and providers characteristics", in *Proceedings of the 2012 2nd IEEE International Conference on Cloud Computing and Intelligence Systems (IEEE CCIS 2012)*, Oct. 30th ~ Nov. 1st,Hangzhou, China.

Hauck, M., Huber, M., Klems, M., Kounev, S., Muller-Quade, J., Pretschner, A., Reussner, R., and Tai, S. Challenges and opportunities of Cloud computing. Karlsruhe Reports in Informatics 19, Karlsruhe Institute of Technology - Faculty of Informatics, 2010.

Zhonghong Ou, Hao Zhuang, Jukka K. Nurminen, Antti Ylä-Jääski, and Pan Hui. 2012. Exploiting hardware heterogeneity within the same instance type of Amazon EC2. In *Proceedings of the 4th USENIX*

conference on Hot Topics in Cloud Ccomputing (HotCloud'12). USENIX Association, Berkeley, CA, USA, 4-4.

George Kousiouris, Tommaso Cucinotta, Theodora Varvarigou, "The Effects of Scheduling, Workload Type and Consolidation Scenarios on Virtual Machine Performance and their Prediction through Optimized Artificial Neural Networks , *The Journal of Systems and Software (2011)*,Volume 84, Issue 8, August 2011, pp. 1270-1291, Elsevier, doi:10.1016/ j.jss.2011.04.013.

Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu. An analysis of performance interference effects in virtual environments. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 200–209, April 2007.

Schaffrath, Gregor and Schmid, Stefan and Vaishnavi, Ishan and Khan, Ashiq AND Feldmann, Anja, 2012. A Resource Description Language with Vagueness Support for Multi-Provider Cloud Networks. Munich, Germany: *International Conference on Computer Communication Networks (ICCCN '12)*.

Charlton, S. (2009). Model driven design and operations for the Cloud. In OOPSLA09, *14th conference companion on Object Oriented Programming Systems Languages and Applications*, pages 17–26.

Koslovski, G., Primet, P.V.-B., Charao, A. S.: VXDL: Virtual Resources and Interconnection Networks Description Language. In: GridNets 2008 (October 2008).

Mirkovic, J., Faber, T., Hsieh, P., Malayandisamu, G., Malavia, R.: DADL: Distributed Application Description Language. USC/ISI Technical Report ISI-TR-664 (2010).

Stephen M. Blackburn, Robin Garner, Chris Hoffmann, Asjad M. Khang, Kathryn S. McKinley, Rotem Bentzur, Amer Diwan, Daniel Feinberg, Daniel Frampton, Samuel Z. Guyer, Martin Hirzel, Antony Hosking, Maria Jump, Han Lee, J. Eliot B. Moss, Aashish Phansalkar, Darko Stefanović, Thomas VanDrunen, Daniel von Dincklage, and Ben Wiedermann. 2006. The DaCapo benchmarks: java benchmarking development and analysis. SIGPLAN Not. 41, 10 (October 2006), 169-190. DOI=10.1145/1167515.1167488 http://doi.acm.org/10.1145/1167515.1167488.

Saurabh Kumar Garg, Steve Versteeg, Rajkumar Buyya, A framework for ranking of Cloud computing services, Future Generation Computer Systems, Volume 29, Issue 4, June 2013, Pages 1012-1023, ISSN 0167-739X, http://dx.doi.org/10.1016/j.future.2012.06.006.

A. Iosup, R. Prodan, and D. Epema, "Iaas Cloud Benchmarking: Approaches, Challenges, and Experience," *Proc. of the Int'l Conf. on High Performance Networking and Computing (SC)*, MTAGS 2012 . IEEE/ACM, pp. 1–8.

Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. 2010. CloudCmp: comparing public Cloud providers. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement (IMC '10)*. ACM, New York, NY, USA, 1-14. DOI=10.1145/1879141.1879143 http://doi.acm.org/10.1145/1879141.1879143.

ARTIST Consortium (2013), Deliverable D7.2.1 v1.0-Cloud services modelling and performance analysis framework, Available at: http://www.artist-project.eu/sites/default/files/D7.2.1%20Cloud%20services%20modeling%20and%20performance%20analysis%20framework_M12_30092013.pdf.

HS4MC
Hierarchical SLA-based Service Selection for Multi-Cloud Environments

Soodeh Farokhi[1], Foued Jrad[2], Ivona Brandic[1] and Achim Streit[2]
[1]*Institute of Information Systems, Vienna University of Technology, Vienna, Austria*
[2]*Steinbuch Centre for Computing, SCC Karlsruhe Institute of Technology, KIT Karlsruhe, Karlsruhe, Germany*
{*s.farokhi, ivona*}*@infosys.tuwien.ac.at,* {*foued.jrad, achim.streit*}*@kit.edu*

Keywords: Multi-Cloud, Service Level Agreement (SLA), Service Selection, Software-as-a-Service (SaaS), Infrastructure-as-a-Service (IaaS), InterCloud-SLA.

Abstract: Cloud computing popularity is growing rapidly and consequently the number of companies offering their services in the form of Software-as-a-Service (SaaS) or Infrastructure-as-a-Service (IaaS) is increasing. The diversity and usage benefits of IaaS offers are encouraging SaaS providers to lease resources from the Cloud instead of operating their own data centers. However, the question remains for them how to, on the one hand, exploit Cloud benefits to gain less maintenance overheads and on the other hand, maximize the satisfactions of customers with a wide range of requirements. The complexity of addressing these issues prevent many SaaS providers to benefit from the Cloud infrastructures. In this paper, we propose HS4MC approach for automatic service selection by considering SLA claims of SaaS providers. The novelty of our approach lies in the utilization of prospect theory for the service ranking that represents a natural choice for scoring of comparable services due to the users preferences. The HS4MC approach first constructs a set of SLAs based on the given accumulated SaaS provider requirements. Then, it selects a set of services that best fulfills the SLAs. We evaluate our approach in a simulated environment by comparing it with a state-of-the-art utility-based algorithm. The evaluation results show that our approach selects services that more effectively satisfy the SLAs.

1 INTRODUCTION

Using various services from multiple Clouds to have a wide range of choices with various cost and quality of services (QoS) can be viewed as a natural evolution in Cloud computing. There are several reasons to utilize multiple Clouds such as: improving the quality of service, while optimizing service cost; migrating among various providers; avoiding vendor lock-in; and the need of particular Cloud services which are not provided elsewhere. There are two types of delivery models for multiple Clouds: Federated Cloud and Multi-Cloud (Petcu, 2013), which differ in the degree of collaborations between the involved Clouds and the way that the user interacts with them. In the Federated model, there is a need for an agreement between the various involved Cloud providers transparent for the user. While in the Multi-Cloud model, which is the focus of this paper, there is no need for such agreement. In the latter, a user is aware of various Clouds, and usually a third party is responsible to deal with these variation in the service provisioning phase.

Software-as-as-Service (SaaS) providers, as potential Cloud infrastructure service users, are being encouraged to profit from hosting their offered software service in the Cloud instead of establishing their own data-centers. Therefore, SaaS providers are looking into solutions that minimize their overall infrastructure leasing cost without adversely affecting the customers (Wu et al., 2011). To achieve this goal, it is essential to have a clear definition of user requirements and then provide a solution by which these requirements are met. In the Cloud context, the exact requirements (both functional and non-functional) under which the services are or should be delivered are specified in SLA. In addition, service delivery systems are managed through the SLA management process to meet the QoS objectives specified in the SLA. In the SaaS provider scenario, it is necessary to specify and manage SLAs in two layers: an SLA between the SaaS customer and the SaaS provider that reflects the QoS objectives of the offered services to the customer, and an SLA between the SaaS and the Infrastructure-as-a-Service (IaaS) provider, which

implicitly affects the customer satisfaction. Because of the aforementioned advantages of Multi-Cloud and the various QoS requirements of the SaaS providers, they have begun to exploit this fertile environment.

There have been a number of studies exploring service selection in the Cloud such as (Bellavista et al., 2013), (Clark et al., 2013) and (Wu et al., 2013). However, they have mostly focused on maximizing the profit of either the customer or the IaaS providers by proposing solutions in a single Cloud without thoroughly investigating SLAs. Thus, barriers relevant to the service allocation for SaaS providers while managing SLA issues in Multi-Cloud has not been studied well. As a recent investigation of SLA interoperability issues in Multi-Cloud, an IEEE working group called InterCloud Working Group (ICWG)[1], has been established recently to focus on developing a standard for interCloud interoperability.

Since in Multi-Cloud dependent components of a single SaaS application can be distributively deployed in diverse Cloud infrastructures with various QoS attributes, from the SaaS provider perspective, the deployment can be considered as a *Composite Infrastructure Service* with a set of functional and non-functional requirements for each included application component. As a result of service diversity in Multi-Cloud, the selection of a set of suitable services is a challenging task. The question remains how to, on the one hand, score and select services for each single component and on the other hand, optimize this allocation to satisfy the requirements of the composite service. Moreover, these concerned QoS parameters can be conflicting or have differing importance degrees for the SaaS provider.

Stimulated by these challenges, in this paper we address the problems surrounding SLA management and service selection for SaaS providers in Multi-Cloud to assist them in fulfilling their objectives in an effective way. We propose an SLA-based Multi-Cloud service allocation approach that sits between the SaaS and IaaS provider and includes two main phases: (1) the *SLA Construction* and the (2) *Service Selection*. In the first phase, the *Composite Infrastructure Service* of SaaS provider is broken down to a set of functional and non-functional requirements called sub-SLAs and a meta-SLA. Meta-SLA includes the requirements of a more abstract level related to the whole composite service, while sub-SLAs express requirements of each infrastructure service included in this composite service.

The first phase forms SLAs named InterCloud-SLAs that are provider-independent. These SLAs include SaaS provider desired QoS requirements for the application deployment on the Cloud. The focus of this paper is the second phase which uses an algorithm to score services based on the user satisfaction. As the user satisfaction has been a subject of great interest to Cloud providers, the principle of prospect theory (Kahneman and Tversky, 1979) is used in the scoring and selection algorithm to model the user satisfaction as a function of quality aspect of service and the importance of each parameter for the user. This theory is an alternative descriptive model of decision making under risk for utility theory and is said to be more realistic in calculating the user satisfaction. Based on this theory the changes in specific quality aspects of a service is sensed more by users who have assigned higher weights to these aspects. Namely, satisfaction is subjective and may be different for different users even for the exact same service with the same quality aspects. To the best of our knowledge, this is the first application of prospect theory in the Cloud service selection problem.

As prospect theory is an alternative model for utility theory, we evaluate our work by comparing the result with a utility-based matching algorithm using a real-world SaaS provider scenario based on a simulation environment. As it will be presented later in the paper, the evaluation result shows the flexibility and effectiveness of our selection algorithm as well as justification of our SLA specification in a Multi-Cloud.

The remainder of this paper is organized as follows. In Section 2 we discuss the related work. Section 3 presents an overview of HS4MC approach and outlines its main phases, while Section 4 focuses on the selection algorithm and SLA specification by considering a concrete example. Section 5 is dedicated to the evaluation of our approach, and finally, in Section 6 we conclude the paper and express the directions of our future work.

2 RELATED WORK

The issues of SLA-based service selection has been widely investigated in both web service domain and Cloud computing in recent years. In this section we explore approaches dealing with this issue. An extensive comparison of existing approaches working on service selection for composite Web services is given in (Moghaddam and Davis, 2014). In web service domain, the author of (Yau and Yin, 2011) proposes a web service QoS-based ranking and selection approach. Their approach calculates the satisfaction score of the user for each QoS parameter based on the basis of prospect theory and then aggregates the scores in order to select the service with the highest

[1] http://grouper.ieee.org/groups/2302/

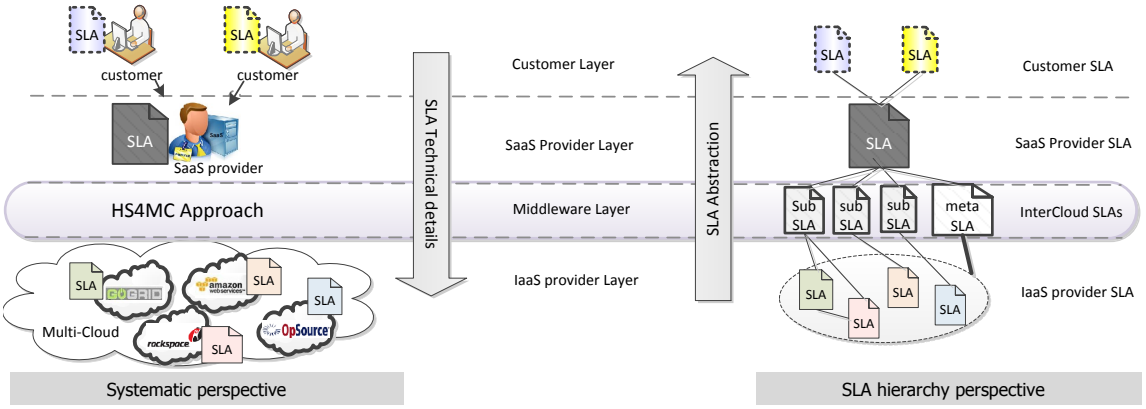


Figure 1: SLA hierarchy and service diversity in Multi-Cloud.

aggregated score. Their approach has some significant advantages over existing work, such as selecting the service that best satisfies QoS requirements concerning by the user, instead of the service with the best QoS, which may lead to over-qualification and can improve utilization of services. Moreover, by using prospect theory, they modeled the relation between service QoS parameters and the user satisfaction more precisely. However, in their work, they only focus on a single service selection. In our work, we also use the principle of prospect theory to rank both single Cloud services as well as composite Cloud services.

For the service selection in Multi-Cloud, a methodology is needed to compare Cloud services based on the various criteria such as the cost and QoS parameters for different user profiles (Petcu, 2013). In addition, because of the SLA heterogeneity in this environment, SLA management from both customer and provider perspectives is challenging.

Most of the works that focus on the SLA-based service selection and allocation in Clouds are focused only on maximizing the customer profit (Dastjerdi, 2013) and (Clark et al., 2013) or IaaS provider profit (Lee et al., 2010). The work in (Wu et al., 2011) is one of the first dealing with resource allocation from the SaaS provider perspective. This work was also enhanced to support both customer and SaaS provider profits in (Wu et al., 2013). The authors propose an allocation strategy for SaaS providers to maximize their profit and customer satisfaction level when deploying their applications in Cloud infrastructure services. Their approach also supports the dynamic changing of customer requests with the goal to minimize the number of used VMs. However, in their SLA, they consider just response time and service initiation time as service selection parameters for SaaS provider. In addition, their evaluation is performed in one Cloud with a single requested VM per Service.

Similar works have investigated service allocation in the Cloud (Son et al., 2013) and our previous work (Emeakaroha et al., 2011) by providing an SLA-driven resource allocation scheme that selects a proper data center among globally distributed centers operated by a provider. In contrast, we support composite Multi-Cloud services where the SLA of the component services can even be conflicting, by supporting more parameters such as availability, latency, reputation, throughput and cost.

The concept of sub-SLA and meta-SLA were first mentioned in (Ouelhadj et al., 2005) in Grid computing domain. The authors use these concepts to schedule jobs in Grids by utilizing a multi-agent system and an SLA negotiation protocol. Our usage of these concepts on Multi-Cloud significantly differs from this one due to the differences between Grid and Cloud business models.

Compared to the previous works, we propose an approach to assist SaaS providers to select the most suitable Cloud infrastructure services while handling the SLA hierarchy and heterogeneity (in abstraction and technical details) and variety of services in Multi-Cloud. Our goal is to maximize the SaaS provider profit by maximizing its SLA satisfaction level, which we believe can be achieved by applying prospect theory in the computation of user satisfaction

3 HS4MC APPROACH

The systematic perspective of Figure 1 represents the position of HS4MC approach. It lies between the SaaS and IaaS provider layers and handling the SLA heterogeneity and service selection. HS4MC proposes the concept of sub-SLA and meta-SLA as InterCloud-SLAs to be able to cover the requirements of the SaaS provider for the *Composite Infrastructure*

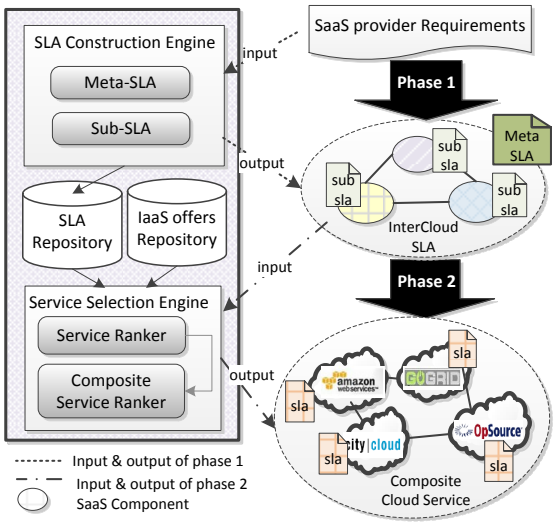


Figure 2: HS4MC architecture and phases.

Service as well as each included service. In HS4MC approach, we focus on the SLA between a SaaS and an IaaS provider which includes both functional and non-functional parameters. Each non-functional parameter in an SLA can be considered as hard or soft. Hard parameters must be satisfied, e.g the infrastructure cost *must* be less than a specific amount, while satisfaction of soft parameters is not mandatory but preferred, e.g response time *is preferred to be* less than 5s. By SLA satisfaction, we mean providing functional and hard non-functional parameters as well as trying to find the most suitable services for the soft ones.

In HS4MC there are two phases: *SLA Construction* and *Service Selection*, which are described in this section. Figure 2 depicts the HS4MC architecture and the included phases along with the input and output of each phase.

SLA Construction Phase. As the input of this phase, the SaaS provider submits its Cloud infrastructure requirements to the *SLA Construction Engine* as a single XML file. These requirements contain two abstract parts: one including the requirements for each infrastructure component (Cloud virtual machine (VM) or storage), and the other enfolding the requirements of the whole set of requested infrastructures. The *SLA Construction Engine* extracts the data related to these two parts and constructs a set of sub-SLAs as well as a meta-SLA by utilizing an SLA ontology. The main purpose of constructing such SLAs, named InterCloud-SLA[2] is to address the SLA interoperability issue in Multi-Cloud. Figure 1 presents this process by considering the SLA hierarchy perspective.

[2]Inspired by a sub-group of aforementioned IEEE Intercloud Working Group (ICWG), named InterCloud-SLA.

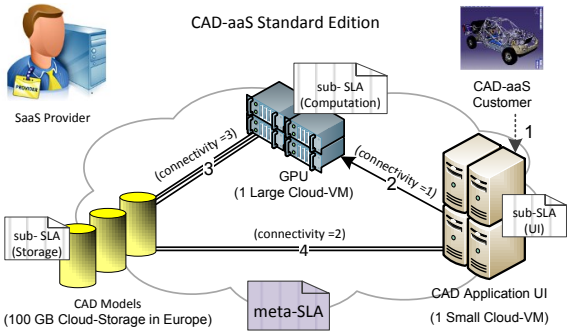


Figure 3: Composite infrastructure service for a CAD-aaS standard edition.

Namely, the *SLA Construction Engine* needs some sort of data-connection and reasoning ability in order to break down the SaaS requirements in a way that they can be mapped to the current service offers. Moreover, it should be able to combine current offers in order to provide new value-added services for the user requirements. The further steps of this phase on SLA extraction are according to our previous work (Breskovic et al., 2013) and (Redl et al., 2012).

Service Selection Phase. The InterCloud-SLAs and the IaaS providers' offers are two inputs of this phase, as shown in Figure 2. The *Service Ranker* component is responsible for creating a ranking list of services for each sub-SLA, which are then used by the *Composite Service Ranker* component to score the combinations of services for the meta-SLA. The output of this phase is a *Composite Cloud Service*, which has the best satisfaction score among the other candidates. Before describing the details of this phase in the following section, we present a concrete example to provide the intuition of our approach.

3.1 Example Scenario

As a running example, a CAD[3] software provider aims to deploy its CAD software in Cloud and offers it as a public Cloud SaaS, called CAD-as-a-Service (CAD-aaS), in three software editions: enterprise, professional and standard. Each edition has a specific set of requirements. The CAD-aaS provider wants to deploy the various components of its CAD service, presented in Figure 3 in Cloud, so it request several Cloud infrastructure services: one small VM for the application user interface (UI), one large VM with GPU features for the computation component and 100GB of storage to store the CAD models. Each requested service has a sub-SLA. Non-functional parameters related to the whole CAD composite ser-

[3]Computer-Aided Design

vice form the meta-SLA. The edge numbers represent the execution sequence of the CAD-aaS. The components communicate with each other based on the CAD software application topology. The degree of data communications between components are depicted as connectivity value for each edge and it will be described later. The rationale behind choosing this example is that deploying CAD applications in the Cloud has been investigated widely; for example it is used in the Cloudflow project [4] which has the aim to make the Cloud infrastructures a practical solution for manufacturing by automatic provisioning of SaaS applications over Cloud infrastructure services.

4 SERVICE SELECTION PHASE

The cornerstone of the *Service Selection Phase* is a selection algorithm that works based on prospect theory to compute the user satisfaction score for a certain service. This theory is proper for describing user decisions among various choices with uncertainty, and considers human behavior in computation of the user satisfaction. We adopted the idea of the algorithm presented in (Yau and Yin, 2011) and developed it to support service selection for a Cloud composite service, and to cover all functional and non-functional parameters of InterCloud-SLAs. In this section, we first introduce the SLA specification used in the *Service Selection Phase*, then we elaborate the steps of the selection algorithm.

4.1 SLA Specification

A set of m infrastructure services can be defined as follows where F_i is the functional parameter of service S_i and NF_i is its non-functional parameter.

$$Service\ Offers = \{S_1, \cdots, S_m\},$$
$$S_i = \{F_i, NF_i\} \quad 1 \leq i \leq m \tag{1}$$

The InterCloud-SLA contains a set of n sub-SLAs and one meta-SLA and is defined by

$$InterCloud\ SLA =$$
$$\{\{subSLA_1, \cdots, subSLA_n\}, metaSLA\} \tag{2}$$

where each $subSLA_j$ and *metaSLA* includes a set of functional parameters F and non-functional parameters NF as follows:

$$subSLA_j = \{F_j, NF_j\} \quad 1 \leq j \leq n \tag{3}$$

$$metaSLA = \{F, NF\} \tag{4}$$

[4] http://eu-cloudflow.eu/project/concept.html

Non-functional parameters of *subSLA* or *metaSLA*, NF_k are represented as tuple

$$NF_k = \{Min_k, Max_k, W_k, T_k\} \quad 1 \leq k \leq l \tag{5}$$

where Min_k and Max_k determine the accepted boundaries for parameter k. W_k represents the user preference on parameter k in the service selection, and it is valued within (0, 1]. A larger W_k shows that parameter k is more important for the user, so this parameter will have more impact on the service ranking. T_k specifies the type of parameter k, which can be hard or soft. In the selection algorithm, non-functional parameters are treated in the same way as functional parameters.

If the user specifies no value for each of the introduced factors, the default values are assigned, $T_k = soft$ and $W_k = 0.5$ that show the medium importance of parameter k. Moreover, for Min_k and Max_k, the default values are defined as the smallest and largest values for parameter k within a corresponding set of $Service\ Offers$. For example, if $k = availability$, then $Min_k = 90\%$ and $Max_k = 100\%$ can be assigned as the default values.

The last part in our modeling is a graph which describes the degree of connectivity among sub-SLAs (a number between 1 to 3, larger means higher connectivity). In this graph, the nodes present the sub-SLAs and the edge shows to which degree these two corresponding components are going to transfer data at runtime, which influence the traffic cost and latency of the composite deployment. As depicted in Figure 3, the connectivity value for each edge shows the amount of data transfer between the involved nodes. For example, the computation node transfers the highest amount of data with the storage node, among other edges in the CAD-aaS graph.

4.2 Service Selection Algorithm

The selection algorithm, as depicted in the following pseudo code, can be described in six steps.

Step (1) A set of services that satisfies the functional and hard non-functional parameters is chosen for each sub-SLA (line 4-5). We assume that the filtered list in not empty at this step, indeed the negotiation with the user in case that no service is found for the given requirements can be considered as a future work.

Step (2) Due to the variety of differing non-functional parameters in metrics and scales for a given service set, they are needed to be normalized before being used in service ranking. In Equation (6), we normalize service quality parameters in such a way that the higher value always means better. This equation calculates the normalized result, N_{ik}, for values

$$Norm(NF_{ik}) = N_{ik} = \begin{cases} \frac{NF_{ik} - Min_{jk}}{Max_{jk} - Min_{jk}} & if\ larger\ NF_{ik}\ is\ desirable \\ \\ \frac{Max_{jk} - NF_{ik}}{Max_{jk} - Min_{jk}} & if\ smaller\ NF_{ik}\ is\ desirable \end{cases} \quad (6)$$

between the accepted boundaries, *Min* and *Max*. For values which are better than accepted boundaries, the result is 1 and for the worse values the result would be 0. For parameters like availability where larger value is desirable, we use the first case, and for parameters like cost, where minimization is the goal, the second case is used for the normalization (line 6-8).

Listing 1: Pseudo-code of the selection algorithm.

```
1  Input: ServiceOfferList, subSLAList, metaSLA
2  Output : Selected CompositeService
3
4  for each subSLA in subSLAList
5    Filter ServiceOfferList by F and Hard-NF
6    for each NF_i of S_i in ServiceOfferList
7      for each NF_j in subSLA_j
8        Normalize Value NF_ik by Norm(NF_ik)
9        Compute satisfaction score N_ik by SSF(N_ik)
10     endfor
11     Compute FinalSatisfactionScore S_i by CF(S_i)
12   endfor
13 endfor
14 Store all Combinations of services of Service-
15 OfferList in CompositionServiceList
16 for each Composition in CompositionServiceList
17   for each NF in metaSLA
18     Compute AggregatedValue
19   endfor
20 endfor
21 Filter CompositionServiceList by F and Hard-NF
22 for each Composition in CompositionServiceList
23   for each NF in metaSLA
24    Normalize AggregatedValue of NF by Norm(NF)
25    Compute satisfaction score Composition
26      by SSF(N)
27   endfor
28   // based on included services and subSLAs
29   Compute average FinalSatosfactionScore
30   // based on subSLAs and metaSLA
31   Compute FinalCompositionScore by Score_final
32   // based on calculated FinalCompositionScore
33   Sort CompositionServiceList
34 endfor
35 return Selected CompositeService
```

Step (3) Let N_{ik} be the normalized value of NF_{ik} of service S_i, the Satisfaction Scoring Function *SSF* can be defined as Equation (7). This formula computes the user satisfaction score of the normalized value N_{ik} of each non-functional parameter in S_i based on the W_j in $subSLA_j$ (line 6-10).

$$SSF(N_{ik}) = \begin{cases} 0.5(2N_{ik} - 1)^{1-W_{jk}} + 0.5 & N_{ik} > 0.5 \\ \\ -0.5(-2N_{ik} + 1)^{1-W_{jk}} + 0.5 & N_{ik} \leq 0.5 \end{cases} \quad (7)$$

The rationale behind the *SSF* is based on prospect theory. This theory implies that changes in a specific quality aspects of a service is sensed more by users who have assigned higher weights to those quality parameters. Indeed, the satisfaction of user for a service is based on the gains and losses relative to the reference point (normalized value of 0.5) instead of absolutely determined by the normalized QoS parameters of that service. Moreover, satisfaction is influenced by the importance weight of user assigned to a specific quality parameter. According to this theory, the satisfaction function should be concave for gains, and convex for losses. To clarify the behavior of the *SSF*, we present the graphs in Figure 4. Each curve shows the behavior of *SSF* based on different weights for various non-functional parameters supported either in sub-SLA or meta-SLA. For example, in the second diagram which the accepted boundaries are 99.5% as minimum and 100% as maximum, then the normalized value of 0.4 (value=99.7%) has different satisfaction scores for standard, professional and enterprise users based on their specified weights. Other diagrams also show the influence of weights in *SSF* for differing non-functional parameters. The weights can also be different based on the goal of the user for a service. For example, the first diagram shows varying weights of a user for response-time of different infrastructure services. As depicted response-time for VM which is dedicated to UI[5] is more important for user (higher weight) than VM for computation.

Step (4) Until now, we have calculated the satisfaction score for each NF_i of service *Si*, so we have a satisfaction score set $\{SC_{i1}, \cdots, SC_{ik}, \cdots, SC_{il}\}$ calculated based on the $\{NF_{j1}, \cdots, NF_{jk} \cdots, NF_{jl}\}$. At this point, it is needed to compute the final satisfaction score of each service by the following Combination Function *CF* (line 11)

$$CF(S_i, subSLA_j) = \frac{\sum_{k=1}^{l} SC_{ik}.W_{jk}}{\sum_{k=1}^{l} W_{jk}} \quad (8)$$

Step (5) The previous steps (line 4-13) are executed on the sub-SLA level, while from this step on,

[5] User Interface

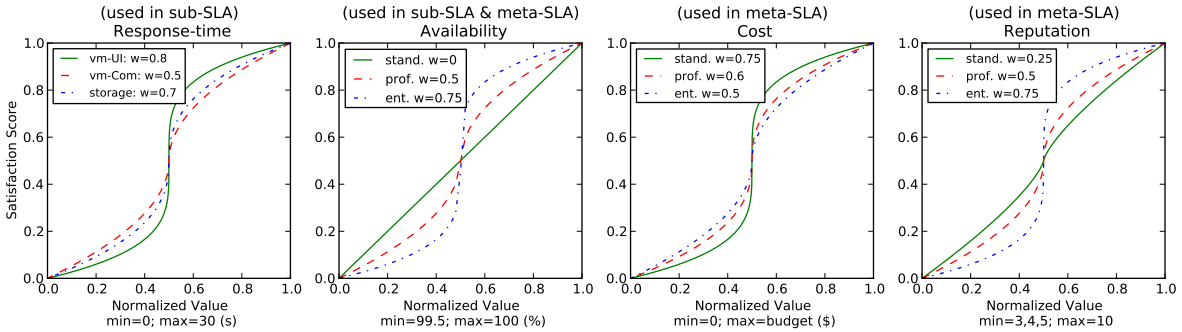


Figure 4: The effect of weighting and min-max boundaries in satisfaction score for parameters of sub-SLA and meta-SLA.

the meta-SLA will be processed. First, for all possible combinations of selected services corresponding to each *subSLA*, the aggregated non-functional values are calculated by using their correlated aggregated functions, defined in Table 3 (line 14-20). Then, the algorithm follows the same steps (1 to 4) by considering the aggregated values, meta-SLA and the possible service combinations, named *Composition* (line 21-26).

Step (6) For the final selection, the algorithm considers the influence of both sub-SLAs and meta-SLA. To this aim, the average $FinalSatisfactionScores$ of services included in the *Composition* (line 27-28) is computed. Then, by the Equation (9) in which $W_{metaSLA}$ is the importance weight of meta-SLA and W_{subSLA} the weight of sub-SLA, the $FinalCompositionScore$ is calculated for the final selection (line 29-30).

$$Score_{final} = W_{metaSLA}.Score_{agg} + W_{subSLA}.Score_{ave} \quad (9)$$

The first ranked *CompositeService* would be the output of *Selection Algorithm* (line 31-34). The top ranked composite service is the one with the largest satisfaction score which means that the service QoS can best satisfy the user requirements. Hence, in the proposed algorithm the services whose QoS parameters are not the best in comparison with other services can still be ranked on the top as long as they perfectly satisfy the user QoS requirements. This is beneficial in improving utilization of services, and also saving the resources to satisfy other users.

5 EVALUATION

In this section, we evaluate the proposed algorithm using the scenario described in Section 3.1 in comparison with a state-of-the-art utility-based matching algorithm (Jrad et al., 2013). For this purpose, we implemented both algorithms in Java language and then conducted a set of simulations. The experimental result will be presented in this section.

Table 1: Functional and non-functional parameters of simulated IaaS providers (*S=Small*, *M=Medium*, *L=Large*).

Parameters	Definition (unit/range)
$IaaS_{Type}$	$VM_{S/M/L}$, Storage
Location	data-center region (1–4)
Reputation	provider reputation rank (1–10)
Availability	up-time of service (%/3 *Month*)
ResTime	time to fully receive an answer (*s*)
Throughput	download a few large files (*Mb/s*)
$Cost_{VM}$	lease $VM_{S/M/L}$ ($/*hour*)
$Cost_{Storage}$	store ($/*GB/Month*)
$Cost_{Traffic}$	download ($/*GB/Month*)

Table 2: Latency matrix (ms).

	USA	Europe	Asia	Australia
USA	25	150	250	100
Europe	150	25	150	200
Asia	250	150	25	500
Australia	100	200	500	25

5.1 Simulation Setup

In order to model heterogeneous infrastructure services in a Multi-Cloud environment, we simulate 12 commercial IaaS providers, each with one or more data centers distributed along different geographical locations (four continents). In our simulation environment, each provider has a set of functional and non-functional parameters as well as pricing models, as depicted in Table 1. Each provider either provides Cloud storage or Cloud VMs presented in three sizes: small, medium, and large. The VM budget is given based on the computation units. one, two and four computation units are respectively assigned to small, medium and large VM sizes. We assume that compu-

Table 3: Meta-SLA functional and non-functional parameters and the corresponding aggregated functions.

Parameter	Definition	Unit/Range	Aggregation function
$Contract_{duration}$	leasing period	hour	-
Budget	$VM(small/hour)$, $Storage(GB/month)$, $Traffic(GB/month)$	$	-
Availability	up-time of composite service	%	$Ava_{agg} = \prod_{i=1}^{n} Ava(S_i)$
Throughput	downloaded data from composite service	Mbit/s	$Th_{agg} = min_{i=1}^{n} Th(S_i)$
Latency	latency of composite service	ms	$Lat_{agg} = \frac{\sum_{i,j=1}^{n} W(S_{ij}).Lat(S_{ij})}{\sum_{i,j=1}^{n} W(S_{ij})}$
Reputation	popularity of included providers	1 - 10	$Rep_{agg} = \frac{\sum_{i=1}^{n} Rep(S_i)}{n}$

tational units of different IaaS providers for the same VM size are similar. The pricing model for each simulated provider has been acquired by the pricing model of the corresponding real-world IaaS provider, while the values of QoS attributes have been gathered by running a set of Cloud tests via *CloudHarmony*[6] on the considered public Cloud providers using a single client hosted in Europe. This specification allows us to model a realistic simulation environment.

Furthermore, the reputation value for each provider is simulated considering the reputation ranking model introduced in (Itani et al., 2011). Due to lack of free access to information related to the latency between different data centers (available on *CloudHarmony)*, a latency matrix has been defined with synthesized values by considering the geographical distances, shown in Table 2, while the latency between services within the same data center is assumed to be 10ms.

Supported meta-SLA parameters in our simulation are presented in Table 3. In this Table, budget is the willingness of the user's investment in VM and storage renting as well as the data traffic based on the defined units. For the VM a unit of cost expresses the value of renting a small VM per hour, while for the storage and traffic it represents the value of storing or transferring 1GB of data per month. The calculation of the total budget in our simulation is based on these three introduced units as well as the $Contract_{duration}$. In addition, the estimated size of the transferred data also influences the calculation of traffic cost. For other non-functional parameters in the meta-SLA, as depicted in Table 3, there is a corresponding aggregation function according to the ones introduced in (Zeng et al., 2003). These functions calculate the aggregated values of composite service non-functional parameters based on its constituent services. In the aggregated latency formula $W(S_{ij})$ is the connectivity value of each edge in the graph introduced in Section 4.1. While $Lat(S_{ij})$ is gathered from the latency matrix, showed in Table 2, based on the data center geographical location of included services in the composition. In order to show the different aspects of our approach, we run the experiments by three sets of SLAs for three CAD software editions (Standard, Professional and Enterprise), each with different meta-SLAs and sub-SLAs as depicted in Table 4 and Table 5, respectively.

[6]http://cloudharmony.com/

Table 4: Meta-SLA values for different software editions.

Parameter (unit)	Stan.	Prof.	Ent.
$Contract_{duration}$ $(hour)$	720	720	720
Budget-VM ($/hour)	0.07	0.1	0.14
Budget-Storage ($/month)	0.1	0.12	0.15
Budget-Traffic ($/month)	0.1	0.12	0.15
Availability (%)	96	98	99.8
Throughput $(Mbit/s)$	5	5	15
Latency (ms)	100	100	40
Reputation (1−10)	3	3	5

Table 5: Sub-SLA values for different software editions (St=$Standard$, Pr=$Professional$, En=$Enterprise$).

	$SubSLA_{UI}$	$SubSLA_{Comp.}$	$SubSLA_{Stor.}$
Edit.	St / Pr /En	St / Pr /En	St / Pr /En
Type	vm	vm	$storage$
Size	$S/M/L$	$L/L/L$	100/500/1000
Num.	1 / 1 /1	1 / 2 /5	1 / 1 /2
Loca.	–	–	$Europe$

As shown in Figure 3, CAD-aaS example, we divided the sub-SLAs into three logical groups. The first group of requested services belongs to the infrastructures which provide the application's user interface, called $subSLA_{UI}$. The second group, is related to the infrastructures for computing the application's business logic which is rendering the images in CAD-aaS, called $subSLA_{Computation}$. Finally, data persistence group belongs to the infrastructures which store and maintain the data (CAD models), called $subSLA_{Storage}$. Although our approach is flexible enough to receive three different sets of sub-SLAs $\{subSLA_{UI}, subSLA_{Computation}, subSLA_{Storage}\}$ for each software edition, we consider the same set for all editions. The rationale behind it is that each

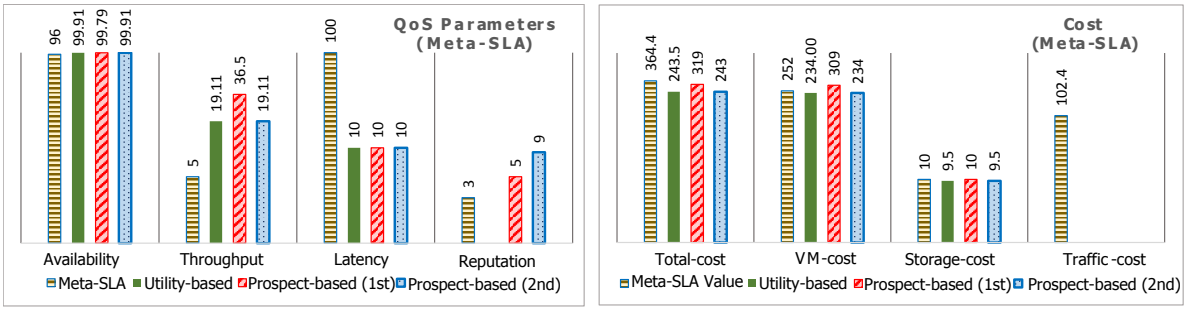


Figure 5: Simulation results: aggregated QoS values of selected services and meta-SLA for the standard edition scenario.

sub-SLA expresses the functional and non-functional requirements related to the application components (UI, Computation, Storage). While it is possible to have different functional needs such as type, size and number of requested infrastructure services for each component, the non-functional requirements of various components in different software editions are almost the same.

5.2 Utility-based Matching Algorithm

The utility-based algorithm, which the proposed algorithm is compared to, uses a quasi-linear utility-function adopted from the multi-attribute auction theory (Asker and Cantillon, 2008) to calculate the user utility, taking the user payment willingness in focus. The utility for each composite service is calculated by subtracting the total service usage costs (include VM, traffic, storage) from its monetized usage benefit. The former is calculated for each user by multiplying his overall score for the SLAs with his maximal payment for a perfect service. The overall SLA scores (a normalized value between 0 and 1), defined as the sum of the weighted single SLA parameter scores, express the overall user satisfaction for a certain service quality. For the calculation of the SLA score, the algorithm uses so called fitting functions that map each SLA metric value to a satisfaction level between 0 and 1.

5.3 Theoretical Comparison

The key differences of the proposed algorithm, named prospect-based, with the utility-based algorithm can be summarized as follows.

(1) The two algorithms have a different perspective on the cost. In our algorithm, the cost has a flexible impact with a specified weight, similar to the other non-functional parameters, while in the utility-based algorithm, the cost has a fixed influence with a predefined impact. (2) The weighting model of these two algorithms are different. In the prospect-based algorithm, the weight of each non-functional parameter is chosen within (0,1] independently of other parameters. However, in the utility-based algorithm the weights are dependent to each other within (0,1] and the sum of them should be 1. (3) The utility-based algorithm selects as the target a *Composite Infrastructure Service* which the one offering the maximal utility value for the user. While, our algorithm selects the service that best satisfies both the meta-SLA and the sub-SLAs. In both algorithms, functional parameters must be fulfilled and both aim to find the best set of non-functional parameters. However, contrary to the utility-based algorithm, our algorithm can also support hard non-functional parameters that can be treated as the same way as functional parameters. (4) The fitting functions used in the utility-based algorithm are quite similar to *SSF* used in our algorithm. However, our algorithm is more flexible as it is based on the weights, while the utility-based algorithm needs to define separated fitting function for each QoS. Therefore, supporting more SLA parameters in our algorithm needs less effort. Furthermore, these fitting functions in utility-based algorithm are only used to score the aggregated QoS based on the SLA for the *Composite Infrastructure Service*, while our algorithm supports scoring each single infrastructure service based on the sub-SLA as well as the composite service based on the meta-SLA using the *SSF*.

5.4 Experimental Results

In this section, the service selection results of both algorithms are analyzed based on several evaluation scenarios in which the inputs are as shown in Table 4 and Table 5, described earlier. The scenarios in the following sections have been designed in a way to highlight different aspects of our work.

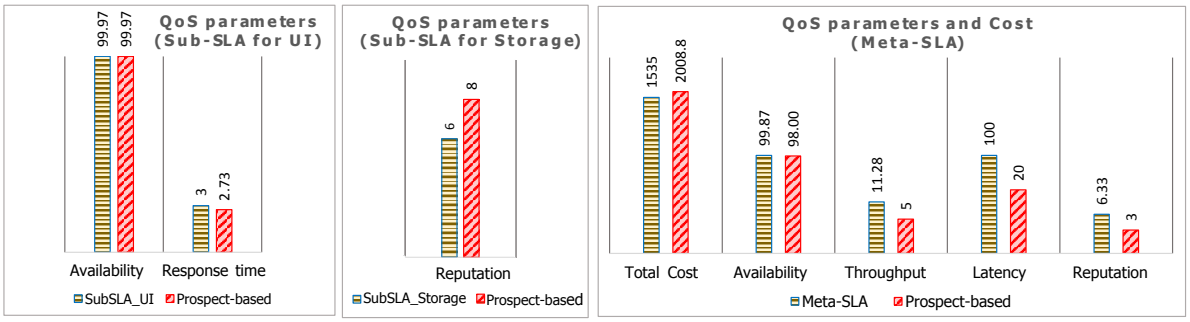


Figure 7: Simulation results: QoS values of selected services, meta-SLA and sub-SLAs for the professional edition scenario.

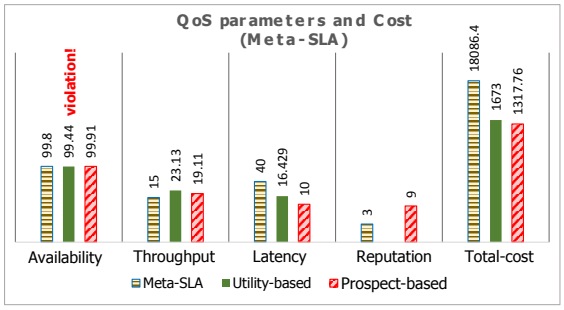


Figure 6: Simulation results: aggregated QoS values of selected services and meta-SLA for the enterprise edition scenario.

5.4.1 Impact of Cost

At the first scenario, minimizing the cost is the main goal of the user and the composite service QoS is more important than the quality of individual services. To simulate this scenario, we assigned the weights as $W_{subSLA} = 0.1$ and $W_{metaSLA} = 0.9$, to dominate the influence of meta-SLA on sub-SLAs in service selection. Furthermore, in meta-SLA, $W_{Cost} = 0.8$ and for other non-functional parameters $W_{NF} = 0.25$ to make cost as the most effective factor on service selection.

Figure 5 depicts the results of aggregated QoS values of chosen services for the both algorithms. In this scenario, the selected services (the 1st ranked or the 2nd rank) of the prospect-based algorithm are the exactly the same as the selected result of the utility-based algorithm, and both algorithms select the services of the same Cloud provider data center. For this reason, the aggregated values of latency are 10ms. Furthermore, both algorithms are successful in satisfying the requested non-functional parameters of meta-SLA including cost (cost of VM, Storage and Traffic), as illustrated in Figure 5.

The results of the first scenario, demonstrate the tendency of both algorithms to select services from the same Cloud provider data center, if this does not lead to SLA violations. The first ranked services of the prospect-based algorithm were chosen from AmazonEC2[7] (Europe[8]) while the utility-based algorithm and the second ranked of our algorithm selected all services from the Voxel[9] (Europe). This is reasonable, since the data traffic is free of charge and latency is negligible when all the services are located at the same data center. Note that the experiments with the professional and enterprise editions achieved similar results to the experiments with the standard edition and will, therefore, not be herewith discussed in detail.

5.4.2 Impact of Meta-SLA

At the second scenario, we evaluate the proposed algorithm in a condition in which the importance of another QoS parameter, availability, for the user is equal to cost and higher than the ones of other non-functional parameters. We achieve this by setting the weight of availability more than the weights of other meta-SLA parameters (incl. throughput, latency, reputation) in the enterprise software edition scenario.

Moreover, in the utility-based algorithm, we set the weight equal to 0.5 for the availability and 0.25 for both the latency and throughput. By applying these settings, we gain the results as depicted in Figure 6. Our algorithm chose AmazonEC2 (Europe), while the utility-based algorithm selected Voxel (Europe) for all requested infrastructure services. The graphs in this figure show that our algorithm performs well by first not violating any SLA parameters and then by giving priority to the parameters that are more important for the user (cost and availability) and finally trying to choose the services with suitable quality values (closer to the user request) for other parameters. This can justify the idea of applying prospect theory in the service selection, in which the best score is for a service that more closely satisfy the requested SLA

[7] http://aws.amazon.com/ec2/

[8] It means data center is located in Europe.

[9] http://www.voxel.net

parameters from the higher priority ones to the lower ones, contrary to the services that have the best quality. The throughout graph of this figure demonstrates that the utility-based algorithm tries to maximize the values of throughput, while the requested availability is violated. This scenario shows that the utility-based algorithm focuses more on keeping the cost under the requested budget as the cost impact is not configurable in the algorithm. While in the prospect-based algorithm by assigning suitable weights and accepted boundaries for each SLA parameters, we can obtain the ranking score that maximizes the user's satisfaction.

5.4.3 Impact of Sub-SLA

At the last scenario, we consider that the SaaS provider has specific non-functional requirements for some of its components. In the other word, the CAD-aaS provider wants to provide a professional software edition in which, for the security reasons, it is required to use Cloud storage service that stores the CAD models only in Europe, and has a good reputation ($location = EU$ and $reputation = 6$ in $subLSA_{Storge}$). Furthermore, the CAD-aaS provider wants to keep the response-time of its deployed UI under 3s, and makes it highly available ($responseTime = 3$s and $availability = 99.98\%$ in $subSLA_{UI}$) with high importance weights (0.75) for the professional software edition. we consider the importance weight as default (W=0.5) for other meta-SLa parameters.

The graphs depicted in Figure 7 depict the comparison results between the QoS parameters requested in meta-SLA and sub-SLAs, and the QoS values of the selected services (Rackspace[10] for UI and Storage, Citycloud[11] for Computation both in Europe) by the prospect-based algorithm. As shown, the algorithm found a suitable set of services that satisfies not only the requested meta-SLA for the composite service, but also all the requested sub-SLAs' parameters have been satisfied.

5.4.4 Discussion

From the first scenario, we conclude that although the impact of cost in the prospect-based algorithm is not a predetermined like the utility-based algorithm, by assigning proportional weights to the service selection criteria, the proposed algorithm can behave the same as a utility-based method which is widely accepted Cloud service selection from the efficiency point of view.

The second scenario, in which another QoS parameter is as important as cost for the user, shows that the prospect-based algorithm outperforms the utility-based algorithm. It selects more suitable services which first do not violate any requested meta-SLA parameters, and then chooses the services with better quality values for the parameters that are more important for the user.

The third scenario highlights one of the main features of the proposed algorithm. The proposed algorithm not only tries to find an optimum composite service with accepted aggregated QoS values to satisfy the meta-SLA, but also consider the sub-SLA satisfaction for each included service. While, existing service selection algorithms that support composite service selection, only focus on finding the set of services which satisfies the aggregated QoS parameters and neglect to view the included services in this composite service separately. We can cover this need easily by the sub-SLA concept, as represented in the third scenario.

To sum up, according to the experimental results, when we are dealing with the quality parameters of a service as a whole, any algorithms may be adequate to find the most suitable services among the candidates. The main problem here is that in this situation, services provided by the same provider are preferred. This diminishes the benefits of a multi-cloud environment. Our algorithm comes into play when users want to take the advantage of being able to combine services offered by different providers available in a multi-cloud. By introducing the sub-SLA concept, now the user can defines more details regarding the individual services inside a composite service. This, as can be seen in the results, leads to a better user satisfaction. It is worth mentioning that supporting parameters such as reputation is significant in our work.

However, as also hinted in (Yau and Yin, 2011) regarding to the application of prospect theory in web service domain, although this theory has been widely accepted and used in economics, the evaluation of its effectiveness and accuracy in regarding to its influence on Cloud service ranking and selection needs more studies and investigation.

6 CONCLUSION

Diversity of services in Multi-Cloud is encouraging more SaaS providers to move towards using the infrastructures provided by the IaaS providers instead

[10] http://www.rackspace.com

[11] https://www.citycloud.com

of running their own private data centers. However, there are still some issues that impede this evolutionary process such as the lack of an efficient service selection and SLA management. In this paper, we introduced HS4MC approach an Hierarchical SLA-based Service Selection for Multi-Cloud Environments. This approach contains two phases: *SLA Construction* and *Service Selection*. In the former, we investigated the SLA hierarchy issue and tackled it by proposing the sub-SLA and meta-SLA concepts. In the latter, we used a selection algorithm based on prospect theory to score the infrastructure services based on the given SLAs and the degree of user satisfaction. Our simulation-based evaluation and a comparison with a utility-based matching algorithm showed that our approach effectively selects a set of services for the composition that satisfy both meta-SLA and sub-SLA parameters. For this purpose, we implemented both algorithms and modeled a realistic simulation environment for a CAD software provider scenario.

In our future work, we will investigate more on the construction of InterCloud-SLAs by utilizing the model driven principles. We will also focus on mapping these SLAs to the selected cloud infrastructure services at runtime. Finally, we will explore the SLA violation detection issue and develop a penalty model in case of violations as the next steps in the Multi-Cloud SLA management.

ACKNOWLEDGEMENTS

Soodeh Farokhi is financed through the doctoral college "Adaptive Distributed Systems", Holistic Energy Efficient Management of Hybrid Clouds (HALEY) project of the Vienna University of Technology, the Austrian National Research Network S11403 and S11405 (RiSE) of the Austrian Science Fund (FWF), and by the Vienna Science and Technology Fund (WWTF) grant PROSEED. Foued Jrad's research stay abroad at the Vienna University of Technology was funded by the Karlsruhe House of Young Scientists (KHYS).

REFERENCES

Asker, J. and Cantillon, E. (2008). Properties of scoring auctions. *The RAND Journal of Economics*, 39(1):69–85.

Bellavista, P., Corradi, A., Foschini, L., and Pernafini, A. (2013). Automated provisioning of saas applications over iaas-based cloud systems. In *Advances in Service-Oriented and Cloud Computing*, pages 94–105. Springer.

Breskovic, I., Altmann, J., and Brandic, I. (2013). Creating standardized products for electronic markets. *Future Generation Computer Systems*, 29(4):1000–1011.

Clark, K., Warnier, M., and Brazier, F. M. (2013). Automated non-repudiable cloud resource allocation. In *Cloud Computing and Services Science*, pages 168–182. Springer.

Dastjerdi, V. (2013). *QoS-aware and semantic-based service coordination for multi-Cloud environments*. PhD thesis, University of Melbourne.

Emeakaroha, V. C., Brandic, I., Maurer, M., and Breskovic, I. (2011). Sla-aware application deployment and resource allocation in clouds. In *Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual*, pages 298–303. IEEE.

Itani, W., Ghali, C., Kayssi, A. I., and Chehab, A. (2011). Accountable reputation ranking schemes for service providers in cloud computing. In *CLOSER*, pages 49–55.

Jrad, F., Tao, J., Knapper, R., Flath, C. M., and Streit, A. (2013). A utility-based approach for customised cloud service selection. *Int. J. Computational Science and Engineering*.

Kahneman, D. and Tversky, A. (1979). Prospect theory: An analysis of decision under risk. *Econometrica: Journal of the Econometric Society*, pages 263–291.

Lee, Y. C., Wang, C., Zomaya, A. Y., and Zhou, B. B. (2010). Profit-driven service request scheduling in clouds. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 15–24. IEEE Computer Society.

Moghaddam, M. and Davis, J. G. (2014). Service selection in web service composition: A comparative review of existing approaches. In *Web Services Foundations*, pages 321–346. Springer.

Ouelhadj, D., Garibaldi, J., MacLaren, J., Sakellariou, R., and Krishnakumar, K. (2005). A multi-agent infrastructure and a service level agreement negotiation protocol for robust scheduling in grid computing. In *Advances in Grid Computing-EGC 2005*, pages 651–660. Springer.

Petcu, D. (2013). Multi-cloud: expectations and current approaches. In *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds*, pages 1–6. ACM.

Redl, C., Breskovic, I., Brandic, I., and Dustdar, S. (2012). Automatic sla matching and provider selection in grid and cloud computing markets. In *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing*, pages 85–94. IEEE Computer Society.

Son, S., Jung, G., and Jun, S. C. (2013). An sla-based cloud computing that facilitates resource allocation in the distributed data centers of a cloud provider. *The Journal of Supercomputing*, pages 1–32.

Wu, L., Garg, S. K., and Buyya, R. (2011). Sla-based resource allocation for software as a service provider

(saas) in cloud computing environments. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 195–204. IEEE.

Wu, L., Kumar Garg, S., Versteeg, S., and Buyya, R. (2013). Sla-based resource provisioning for hosted software as a service applications in cloud computing environments. *Journal of IEEE Transactions on Services Computing*.

Yau, S. S. and Yin, Y. (2011). Qos-based service ranking and selection for service-based systems. In *Services Computing (SCC), 2011 IEEE International Conference on*, pages 56–63. IEEE.

Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., and Sheng, Q. Z. (2003). Quality driven web services composition. In *Proceedings of the 12th international conference on World Wide Web*, pages 411–421. ACM.

SHORT PAPERS

Cloudifying Applications with ARTIST
A Global Modernization Approach to Move Applications onto the Cloud

Leire Orue-Echevarria[1], Juncal Alonso[1], Hugo Brunelière[2], Andreas Menychtas[3], Philip Langer[4] and Manuel Wimmer[4]
[1]*TECNALIA, ICT-European Software Institute Division, Parque Tecnológico Ed #202, E-48170 Zamudio, Spain*
[2]*AtlanMod Team Inria, Mines Nantes & LINA, Nantes, France*
[3]*National Technical University of Athens, Athens, Greece*
[4]*Vienna University of Technology, Vienna, Austria*
{leire.orue-echevarria, juncal.alonso}@tecnalia.com, hugo.bruneliere@inria.fr, ameny@mail.ntua.gr, {langer, wimmer}@big.tuwien.ac.at

Keywords: Software Modernization, Software Migration, Cloud Computing, Cloud Business Model, Feasibility Assessment, Reverse Engineering, Forward Engineering, Cloud Provider Benchmarking, CLOUDML@ARTIST.

Abstract: Cloud computing is still considered a disruptive technology in spite of being part of our lives for several years now. However, cloud computing is much more than a technology; it is also a business model. Many companies that have sold software in a traditional way are now attending to this revolution, wondering if that new technological and business shift is adequate for them, if they would be able to move their application towards the cloud, transforming alongside the company in a service oriented company and how they could do that. The European Project ARTIST aims to guide companies in this transition by providing them with methods, techniques, and tools, from when the migration is just a thought, until it can be provisioned as a service, taking into account technical, business and organizational aspects.

1 INTRODUCTION AND MOTIVATION

Much more than the technology that supports it, cloud computing is the latest step in the evolution of the IT industrialization process. Most CIOs wish to be on the "cloud train", and therefore cloud adoption is growing in popularity for enterprises and independent software providers. One way to quickly move to the cloud is by developing new, cloud-ready software applications that can be delivered through PaaS solutions like Google App Engine and Azure. But when cloudifying existing software applications, significant re-engineering and adaptation is needed. Only after those steps have been performed, existing applications can be delivered and offered effectively in the software-as-a-service (SaaS) model.

However, this transition can be complex, time-consuming and expensive, especially when not all applications can be moved to cloud and each application has its own business specific migration requirements that derive on the necessity of different problem approximation strategies. Questions such as *How do you know which applications are best candidates for fitting the cloud? Are there any tools that can help with application cloudification? How can enterprises take advantage of the flexibility and scalability of the cloud while avoiding application migration frustrations?* arise.

Due to these arising needs, several ongoing projects (e.g., cf. ARTIST (ARTIST, 2012), ModaClouds (MODAClouds, 2012), PaaSage (PaaSage, 2012)) explore automation possibilities for moving to the Cloud from a modeling perspective.

Besides, cloud providers' compatibility issues, vendor lock in (e.g. being non interoperable) and performance concerns still keep cloud-wary IT managers from getting more comfortable with the idea of moving applications to the cloud.

This paper proposes a global modernization framework developed in the context of the ARTIST European project which understands the cloudification of an application as a global concept involving technical, business and organizational aspects and provides methods, techniques, and tools to guide companies in this transition.

2 THE ARTIST APPROACH

2.1 ARTIST Methodology Overview

ARTIST Migration and Modernization Methodology (ARTIST WP6, 2013) has been implemented to enable the effective migration of legacy applications to cloud environments. Legacy applications have some unique characteristics which introduce many challenges for their modernization and migration to cloud environments. On one hand, there are technical issues related with the nature of the specific application and on the other, the business aspects that need to be considered in offering an application "as a service". Often the legacy applications are not cloud-enabled, following monolithic architecture design approaches and implemented in technologies which may be deprecated. The modernized version of the applications needs the equivalence of functionality and performance as well as business continuity. This does not only require adaptation of the software and integration of modern services of cloud solutions on a technical level (monitoring, security etc.), but also changing the business processes and models based on which the application is offered to the customers so as to exploit the strategic advantages of clouds.

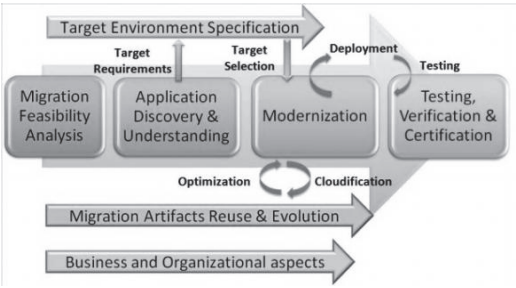


Figure 1: The ARTIST methodology overview.

The ARTIST Methodology consists of three major phases which are explained below. These phases are:

- **Pre-migration:** In this phase, a study on the technical and economic feasibility will be conducted as a prelude to perform modernization of the legacy system.
- **Migration:** This phase will perform the migration process itself, by using reverse as well as forward engineering techniques to deploy the legacy system in the cloud.
- **Post-migration:** In this phase the modernized application will be deployed on the target environment and checked if both technical and business objectives established in the pre-migration phase have been achieved. Moreover, a certification model will be created in order to increase customer confidence in the SaaS system.

2.1.1 Methodology Process Tool

The ARTIST methodology is a detailed (ARTIST WP6 2 2013), but also generic, methodology that covers all migration tasks and processes. In order to practically support this methodology, we incorporate a central component in the overall architecture: the **ARTIST Methodology Process Tool (MPT)**.

The objective of the ARTIST MPT is to allow the customization and instantiation of the ARTIST Migration Methodology for end-users based on the Migration Assessment results of the particular migration project. The MPT, exploiting the results processed and obtained during the assessment, defines a customized modernisation process, tailored to the concrete legacy application needs. The tool shows the customized process in detail, its tasks broken down step-by-step, including hooks to invoke the tools required to accomplish each task acting as an umbrella for ARTIST tools.

During the first year of the ARTIST Project, we worked on the requirements analysis to effectively design and implement MPT. Besides the requirements analysis, we also investigated the possible implementation approaches. The identified approaches are:

- Eclipse-based
- Web-based
- Spikes Together proprietary solution (SPIKES, 2013)
- Hybrid (Eclipse + Web)

After a first analysis on the pros and cons of each approach, the Hybrid solution seems to be the most adequate one. It combines the smooth integration capabilities of the Eclipse based solution (as most of the ARTIST tools will be Eclipse based) and the adaptability of the Web solution for collaborative development environments. In the next year of the project the final implementation technology will be selected and the first version of the prototype implemented.

2.1.2 Pre-migration Phase and Supporting Tools

The pre-migration takes place even before the migration starts. Software vendors need to check if

what they want to achieve with a specific migration project is actually feasible to them in terms of technology, processes and business (ARTIST D5.1.1, 2013).Once a company has realized that their current product is not sustainable as-it-is any more, the first big challenge faced is therefore the decision of what is more convenient, to migrate or to start from scratch.

ARTIST proposes a pre-migration phase that starts off with the characterization of the legacy application from two points of view: technical and business, and follows up with a technical feasibility analysis and totally intertwined, an economic and business feasibility analysis. The result of this pre-migration phase is a Go / NoGo to the migration.

The first step of the pre-migration is the **Maturity assessment** with the main objective of analysing how mature the application is in terms of technology (i.e. architecture, programming language, database, integration with 3rd party offerings, installation requirements, versioning, etc.) and business (i.e. current business model, existence of SLA, maintenance and upgrades procedures, customer service, etc.) and how the customer wants the application to be in those two axes once the application is migrated. The evaluation of the current situation and the ideal situation allows ARTIST to perform a gap analysis, described in terms of a technical feasibility analysis and the business feasibility analysis.

The Maturity Assessment in ARTIST is supported by the **Maturity Assessment Tool** (MAT). which focuses on two perspectives, business and technical in both situations (initial and final) and provides as a result a picture with the position in a quadrant of the initial and the final situation of the application enriched by a set of high level recommendations and goals to be reached along the migration project (the current prototype supports the maturity analysis, and the positioning of the application).

The second activity to be performed in the pre-migration phase is the **Feasibility Analysis which comprises a Technical Feasibility & Business Feasibility analysis.**

On the one hand the **Technical Feasibility Analysis** aims at supporting ARTIST users on the early technical assessment of the migration of a legacy application to the cloud. No matter how simple the application may be the technical feasibility analysis may require non negligible efforts and concrete expertise to be accomplished. The support for decision making at this early pre-migration stage, can benefit from a detailed breakdown of the migration process into a set of technical tasks, to estimate their required efforts, and to identify other resources needed to accomplish every task, including the selection of the appropriate technical expertise or even the detection of dependencies among tasks or other technical intricacies.

ARTIST supports the technical feasibility analysis through the **Technical Feasibility Tool** (TFT).The aim of the TFT is to estimate the efforts required to migrate a legacy application to a selected target cloud environment, fulfilling some migration goals and requirements and automating the process as much as possible. Our implementation of TFT extends the Cloud Migration Point (Tran, 2011) approach by automating some steps, using techniques such as Model Driven Reverse Engineering (MDRE), Software Metrics or DSL-based heuristics, notably to extract knowledge of the legacy system, propose migration strategies and estimate the component complexity. It provides as a final result the estimated effort required to perform the migration project (based on previous based experiences). The current available prototype supports the extraction off software metrics and the proposition of components and migration strategies.

In parallel to the Technical Feasibility Analysis a **Business Feasibility Analysis** is proposed. The need of the Business Feasibility Analysis stems the observation that, although cloud computing is a software deployment scheme expected to bring a number of advantages (e.g. elastic provisioning, cost savings) to its adopters the same advantages does not apply equally well to all potential users (ARTIST D5.1.1, 2013). Hardware costs savings, as an example, does not equally apply to SME and large enterprises. Moreover there are also potential (often sunken) costs that have to be carefully assessed. The Business Feasibility Analysis in ARTIST aims to provide, not only economic information about how ROI, or payback metrics shall behave in the future, but also which are the main risks to be faced with the migration and the organizational processes affected by the uptake of the new business model.

The tool that performs the Business Feasibility Analysis in ARTIST is the **Business Feasibility Tool** (BFT) which supports decision to estimate costs, benefits and operational changes to be applied within a migration to a cloud deployment scheme. The ARTIST's BFT adopts (i) the Agent-Based modelling (Twomey et al., 2002) paradigm to represent (cloud-based) Business Models (ii) Agent-based Simulation (Tesfatsion et al., 2006) to support

the user to learn how the organizational changes required by the adoption of a cloud deployment scheme may impact an enterprise and (iii) Enterprise Simulation (ES) (Datar et al., 2000) to support "what if" scenario aimed at the evaluation of the impact that additional controls due to the adoption of a cloud deployment scheme may have on the organization.

2.1.3 Migration Phase and Supporting Tools

2.1.3.1 Reverse Engineering

Reverse Engineering is the first step of any migration process, as it consists in performing a prior analysis of the existing system or software that is going to be later migrated (e.g. to the cloud) (Canfora et al., 2011). Within the ARTIST methodology and model-based framework, a global Model Driven Reverse Engineering (MDRE) (Rugaber et al., 2004) approach is being designed and developed to support it.

Model Discovery, cf. the **Model Discovery Toolbox (MDT)** is the initial activity of obtaining "raw" (i.e. low-level) models from the different artefacts composing a given software. This is realized (at least semi-)automatically thanks to software components called *model discoverers*. The main related challenge is to deal with the software artefacts heterogeneity, as they can take many different forms/formats (Java or C# source code, XML documents, databases, etc.). Thus, we have been working on a taxonomy of legacy artefacts that helps better classifying them according to several dimensions (their technical space, internal structure, nature, size, environment, etc). The second objective of this taxonomy is to provide some kind of guidance in the process of deciding on the most suitable way to discover the required initial models.

Model Understanding, cf. the **Model Understanding Toolbox (MUT)** is the activity of processing the initial models, computed from the Model Discovery, in order to identify and build higher-level views on the analyzed software. This is generally performed via (chains of) model transformations, usually refining the models iteratively. There are several issues related to this model understanding context: How to deal with both functional and non-functional properties of a system? How to allow the definition and computation of new views? How to generalize views independently from the nature of the system initially reverse engineered? Intending to answer these open questions, several tracks are being explored. For instance, a view definition DSL is being specified in order to make easier the elaboration of (new) views covering different aspects of a given system. Also, experimentations are being performed on which model transformations are made more generic and so reusable indifferently in the context of several types of systems (e.g. both Java- and C#-based).

These two toolboxes come up with sets of components that can be picked up, used and combined according to the context (i.e. the input system or software, the targeted migration platform, etc.). The final goal is to be able to feed correctly the next steps of the process, and more particularly the beginning of the Forward Engineering phase.

2.1.3.2 Target Platform Selection

One of the most complex activities when migrating to a cloud based solution is the selection of the appropriate cloud platform. Here, several aspects come to play, such as the different services offered by the cloud platforms (in any of their delivery models IaaS, PaaS or SaaS) and the needs from the migrated software to be deployed in an efficient manner (in terms of costs) and fulfilling the required QoS parameters (without compromising the SLAs agreed by the service provider and the final user) in the selected provider.

To ease this process ARTIST proposes (Menychtas et al., 2013) to implement two phases in parallel: one in the application domain and one in the candidate target environment domain.

In the application domain, following the analysis of the application features and the creation of models describing the legacy system using reverse engineering techniques, we examine and profile the performance aspects of each application element and feature (through the profiling tool). In the process, these aspects are linked with specific software solutions exploiting trace analysis and benchmarking, which in sequel are matched to elementary hardware resources (virtualized or physical) such as computation storage and networking.

In the target environment domain, the various offerings, in the IaaS, PaaS and SaaS layers, are described in a way that will facilitate the matching between these offerings and application component requirements. For this, we have developed a unique Cloud Modeling Language (CloudML@artist). The specification of the target environment, including both functional capabilities and detailed performance aspects of common application types, enables effective matchmaking and allows for the selection of the ideal provider for the overall application and/or its specific fragments. Metrics are

also investigated to characterize service offerings based on a combination of performance and cost, to improve the overall deployed application's efficiency.

Models of the candidate cloud targets (infrastructure or platform providers), which are available through the ARTIST repository are matched against the afore-posed requirements, using a model-enabled matchmaking algorithm. The optimal target provider (e.g. best ranked match) could be automatically selected or the selection can be deferred to the end-user, who selects it amongst the found matches.

2.1.3.3 Forward Engineering

The main goal of the **forward engineering phase** is to produce executable code, which is efficiently runnable in cloud environments, from platform independent models (PIMs) which have been produced in the reverse engineering phase. In order to produce the required code from the given models, we use several dedicated steps to cope with the following main challenges. First, the PIMs have to be refined in order to deal with the specifics of a particular cloud environment–this step is called *model cloudification*. For instance, a first basic step when migrating models towards cloud environments is to select appropriate cloud services and a virtualization level. Second, possible *model optimizations* have to be explored in the refinement process to meet the non-functional requirements stated for the migration. Current cloud environments offer a large set of different configuration options. Thus, automation support is urgently required to explore more efficiently and effectively this search space. Third, tailored code for a particular cloud environment has to be generated from the refined models, packaged for, and deployed in the chosen cloud environment involving also the provisioning of the required cloud resources beforehand.

Our approach to support this phase relies on **model transformations as main means for automation**. In particular, we employ different model refinement levels instead of generating code directly out of the PIMs. Using these subsequent refinement steps, we ensure that developers are able to control and guide the model cloudification and optimization for their purposes. We also aim at eliminating tedious recurring tasks by applying the convention-over-configuration principle as well as design exploration techniques when transitioning from PIMs to cloudified platform-specific models (PSMs).

As the Unified Modeling Language (UML) (OMG, 2011) already provides modelling concepts to represent software-, platform-, and infrastructure-related artefacts, especially in version 2.x, e.g., considers UML deployment diagrams, UML is used as host language for specifying PIMs and PSMs in our approach. UML is designed as a platform-independent and general purpose modeling language, but also as an extensible language for considering specific domains and technologies such as cloud computing. The language inherent extension mechanisms of UML are *UML libraries* and *UML Profiles*. On this basis, we developed as one part of the CloudML@ARTIST language a sublanguage called *Cloud Application Modeling Language* (CAML, available as open source project at: http://code.google.com/a/eclipselabs.org/p/caml), which provides cloud-specific modeling concepts and at the same time well established UML modeling concepts. In this respect, we have extended UML by providing the UML-based *Cloud Deployment Modeling Library*. This library allows representing cloud-based deployment models independent from particular cloud providers. To keep the cloud provider specifics separate from the modeling library, additional UML Profiles are utilized. These UML Profiles are essential to allow cloud consumers specifying concrete deployments for a selected cloud provider. In the current first version of CAML, we developed UML Profiles that address functional cloud consumer concerns, such as instance types, storage solutions and service offerings, as well as non-functional ones, such pricing, performance and service levels.

To transition from PIMs to cloudified PSMs, we provide a set of **model transformations** that achieve the refinement of the models based on cloud blueprints, patterns, and best practices. Based on the stated migration goals, the abstract services contained in the PIMs are substituted by the most appropriate specific ones offered by cloud providers. Once a set of services is selected, additional configurations and deployment options are automatically examined in order to improve, e.g., the performance measures or costs of software running in the cloud. Therefore, we reuse **simulation techniques** for the PSMs that are established for the migration validation in the post migration phase (cf. Subsection 3.1.4.2). Based on the results, we foresee dynamic design-space exploration by applying different cloud optimization patterns formalized as model transformations to the models and evaluate their impact on the non-functional properties by iterative simulation runs (Troya et al., 2013).

From the optimized PSMs, application code is

generated using **model-to-text transformation** approaches. Please note that not only the application code is produced, but also supporting code, e.g., deployment scripts, configuration files, etc., are generated. These additional artefacts are needed to automatically transfer the generated applications to pre-configured cloud environments.

We aim for a **highly configurable forward engineering process** built up of reusable transformations (Kusel et al., 2013). By using different model composition techniques to develop transformations chains, we counteract the development of monolithic transformations. By reusing standards, such as UML, and Ecore, we have several possibilities to reuse available transformations for code generation and available metamodels and UML profiles that may be of interest also for the cloud modeling domain such as SoaML and MARTE.

2.1.3.4 Business Model Definition and Organizational Impact

Most of the existing cloud migration approaches (ARTIST DOW, 2012) focus their attention in the analysis and the implementation of the technical migration strategy. But cloudification paradigm affects also the delivery model of the company (from SaaG to SaaS) which implies change also at organizational and business level.

Within ARTIST methodology we propose, in combination with the technical migration activities, a set of tasks to implement the changes required as a result of the migration at business and organizational level.

The activities required **for the re-definition of the business model** are based on Osterwalder's Business Model Generation and adopted for cloud based applications, including Market segment re-definition, cost-structure re-development or customer relationship update among others.

The tasks related to the **organizational changes** include aspects from development process to accountability or providers management processes specially adapted for Cloud based and SaaS provider companies. These ideal processes are based on best practices and standards such as CCRA (ITU-T SG13, 2013).

2.1.4 Post-migration Phase and Supporting Tools

2.1.4.1 Migration Goals

The main objective of the migration goals is to establish in a formal and shareable manner the main **constraints exposed from the application (or its owner) for the migration**. The requirements for the migration that cannot be extracted from the legacy software itself will be established by the migration goals and circulated over the other ARTIST tool to be accessible in all migration phases.

These migration goals are worked out through a questionnaire and finally determined by the end-user and are circulated over other migration phases with the purpose of having information about the non-functional requirements required for the migration (quality metrics required, infrastructure information, etc.). These user specified migration goals will be validated in combination with the functional requirements in the validation phase c.f. subsection 3.1.4.2.

2.1.4.2 Validation of the Migration

The objective of the validation phase is to assess whether the migration was successful or not. A migration is considered to be successful, if the b**ehaviour of the software** has not been affected in an unexpected way during the migration (i.e., preserving functional correctness) and if the **migrated software fulfils the user-specified migration goals**.

Validating these two aspects holds several challenges. For evaluating the functional correctness of the software after the migration, it is necessary to verify whether the business logic of the software after the migration corresponds to the software's business logics before the migration. Especially when an insufficient number of test cases is available for the software, ensuring the correct behaviour of the migrated software with respect to the behaviour before the migration is a challenging task—from a theoretical and a practical perspective. **Concerning the validation of the user-specified migration goals**, we have to evaluate a variety of different types of potentially competing migration goals, ranging from goals, for instance, regarding performance efficiency to goals concerning operating costs. These goals may often not be evaluated by measuring the respective metrics directly in the software deployed in the cloud environment used in production, because, for instance, benchmarking the deployed software in the cloud environment may lead to high costs or the information required for the evaluation may not be available directly due to restrictions and virtualization levels of the cloud environment.

For assessing the functional correctness of the migrated software, we follow three different

approaches. First, we adapt the migration techniques of ARTIST in order to migrate not only the software itself but also potentially existing test cases. If no test cases are available or if their test coverage of is insufficient, we follow a second approach to assess the functional correctness. In particular, we provide techniques to generate test cases automatically from the PIM, corresponding to the overall spirit in ARTIST to apply techniques from model-driven engineering. Therefore, not only structural but also behavioural models are reverse-engineered from the legacy software and represented in the PIM. These behavioural models are then used in a model-based test generator tool developed in the course of ARTIST to derive test inputs until the execution of the software with these test inputs leads to a certain test coverage. For this task, we adapt differential symbolic execution techniques to be applied solely on model-level (Person et al., 2008). Of course, this step has to be guided by the user, who may indicate which parts and to which degree the migrated software shall be tested. If the behavioural models are detailed enough, we may also use them to derive test oracles for the generated test inputs. First experiments show that FUML (OMG, 2012), a recent standard of the OMG providing a concise semantics definition for a subset of UML Classes and Activities, may serve as an adequate modelling language to represent the expected behaviour of the software accurately and independently of any platform or technology. In fact, the semantics of FUML is precise enough to enable the unambiguous simulation of FUML models. Therefore, we developed an extended FUML virtual machine (Mayerhofer et al., 2012), which enables the simulation of FUML models, while providing detailed information about the simulation, such as inputs and outputs of each step and detailed simulation traces. Based on this information, we simulate the models for each test input and obtain test oracles covering the expected outputs, as well as the expected traces. The third approach to assess the functional correctness of the migrated software is based on running the legacy software and the migrated software in parallel, forward each request to both of software versions, and compare their responses.

To provide the means for evaluating whether the migration goals, such as performance efficiency and operation costs, are fulfilled, we have to address the challenge that certain information for computing the verdict about the migration goal may not be available in the cloud environment used for production. Thus, simply benchmarking the deployed software may sometimes be insufficient. Therefore, we provide a dedicated tool that enables model-simulation and analysis techniques to estimate certain property measures that are related to the respective quality characteristics mentioned in the migration goals. These measures are then combined and analyzed to validate whether the defined goals are fulfilled. Our current work shows promising first results indicating that several metrics can be obtained by model-level simulation leveraging the level of details in the PSMs (representing the migrated software on model level) without requiring a translation of the PSMs into dedicated performance models (Berardinelli et al., 2013); (Fleck et al., 2013) which is however the case for existing work in software performance analysis (Balsamo et al., 2004). These techniques come with the additional benefit that they can be performed also when the actual migrated code has not been derived yet and hence may be used to guide the optimization already in early phases of the migration as well as in the forward engineering phase (cf. Subsection 3.1.3.3). However, since the model-level simulations and analyses are only estimations, we further aim at validating the migration goals on code-level as a final step. We run and benchmark the actual migrated software, as far as possible (van Hoorn, 2012) providing dedicated forward-engineering modes that, besides generating the production code, also instrument the generated code, where necessary, to obtain the measures needed to validate the migration goals.

2.1.4.3 Certification Model

One of the shortcomings for the final implantation of SaaS and cloud computing in the software industry is the reluctance of the users to this new service offering.

The purpose of the certification phase in ARTIST is to obtain an independent and impartial judgment of SaaS providers, focusing on businesses, processes and technology aspects (ARTIST D11.4.1, 2013) in order to create consumer confidence in software applications offered as services considering reliability parameters (categories) that will be evaluated.

The certification model proposed by ARTIST, Service based Software Provider (SbSp) focus on organisations that develop and offer software based services using methodologies and business models that are connected to the Future Internet and cloud computing schemes. It is structured into three areas i) The business area aims to analyse the financial

stability and soundness of the business in order to assess the potential for continuity and sustainability, ii) the process area aims to ensure the quality of the process of delivering the IT service to the end customer, and iii) the technology area aims to establish a high level of security and transparency for both customers and providers of dedicated SaaS.

To support the certification process, three questionnaires (corresponding to the three areas) have been developed to be used in the certification method. This method is the procedure through which a third independent part evaluates the practices of a company providing a cloud application to assure by a label that the system fulfils a specified level (Gold, Silver or Bronze). It is a way to ensure that "standard-based" products are implemented: quality products, competitive markets with more choices, commodity pricing, and less opportunity to become "locked in" to a particular vendor.

3 CONCLUSIONS

In this paper we present a novel cloudification approach, supporting software vendors paving the path to a smoother transition to the cloud computing paradigm. The presented ARTIST methodology and framework considers both technical and business aspects of the legacy applications and covers not only the core migration phase but also the pre-migration phase, where based on the assessment of the initial and target situations added to the results of a technical and business feasibility analysis, the various steps and tasks of the methodology are customized, as well as the post-migration phase where the outcomes are validated and certified.

ARTIST proposes a set of tools supporting each methodology phase based on model-driven engineering approaches, allowing when possible the reuse of artefacts. The first version of the ARTIST approach has been applied to the PetStore example as the preliminary test case for validation. Currently, the solution is applied to 4 real-world business scenarios for validation and refinement of the overall approach, improving its impact and standardization on real-world industrial environments.

ACKNOWLEDGEMENTS

This work has been supported by the ARTIST Project and has been partly funded by the European Commission under the Seventh (FP7 - 2007-2013) Framework Programme for Research and Technological Development, grant no. 317859.

REFERENCES

OMG, 2011. *OMG Unified Modeling Language (OMG UML), Superstructure 2.4.1.* Object Management Group, http://www.omg.org/spec/UML.

Kusel, A., Schönböck, J., Wimmer, M., Kappel, G., Retschitzegger, W., Schwinger, W., 2013. *Reuse in Model-to-Model Transformation Languages: Are we there yet?*. SoSym, Springer online first, pp. 1-31.

ARTIST D6.2.1, Menychtas, A., Orue-Echevarria, L., 2013. *D6.2.1 ARTIST Migration Methodology*. http://www.artist-project.eu/sites/default/files/D6.2.1_ARTIST%20Methodology_M12_30092013.pdf

ARTIST D6.3.1, Menychtas, A., 2013. D6.3.1 ARTIST *Methodology Process Framework*. http://www.artist-project.eu/sites/default/files/D6.3.1%20ARTIST%20Methodology%20process%20framework_M12_30092013.pdf

ARTIST D5.1.1, Alonso, J., D5.1.1. *Specification of the Business and Technical Modernization assessment in ARTIST.*

Tran, V.T.K., Lee, K., Fekete, K., Liu, A., Keung, J., 2011. *Size Estimation of Cloud Migration Projects with Cloud Migration Point (CMP)*. Int. Symposium on Empirical Software Engineering and Measurement (ESEM'11), pp 265-274.

Twomey P., Cadman, R., 2002. *Agent-Based Modelling of Customer Behavior in the Telecoms and Media Markets*, Info, Vol. 4(1), pp. 56-63.

Tesfatsion, L., 2006. *Agent-Based Computational Economics: A Constructive Approach to Economic Theory*. Handbook of Computational Economics, Vol. 2, North-Holland/Elsevier.

Datar, M.M., 2000. *Enterprise Simulations: Framework for a strategic application*. Winter Simulation Conference.

Menychtas, A., Santzaridou, C., Kousiouris, G., Varvarigou, T., Gorronogoitia, J., Strauss, O., Senkova, T., Orue-Echevarria, L., Alonso J., Bruneliere, H. Pellens, B., Stuer, P., 2013. *ARTIST Methodology and Framework: A novel approach for the migration of legacy software on the Cloud*. MICAS 2013.

ARTIST DOW, 2013. DOW: *Advanced software-based service provisioning and migration of legacy software*.

ARTIST D11.4.1, Vergara, M., 2013. *ARTIST SbSp Certification Model.*

Berardinelli, L., Langer, P., Mayerhofer, T., 2013. *Combining fUML and Profiles for Non-Functional Analysis Based on Model Execution Traces. Qosa'13, ACM Sigsoft Conference on the Quality of Software Architectures*. ACM, pp. 79-88.

Fleck, M., Berardinelli, L., Langer, P., Mayerhofer, T., Cortellessa, V., 2013. *Resource Contention Analysis of*

Service-Based Systems through fUML-Driven Model Execution. Int'l Workshop Non-functional Properties in Modeling, CEUR, Vol-1074 pp. 6-15.

Mayerhofer, T., Langer, P., Kappel, G., 2012. *A runtime model for fUML*. Int'l Workshop on models@run.time, ACM, pp. 53-58.

Person, Suzette, et al. "*Differential symbolic execution.*". 16th ACM SIGSOFT International Symposium on Foundations of software engineering. ACM, 2008.

Balsamo, Simonetta, et al. "*Model-based performance prediction in software development: A survey.*" Software Engineering, IEEE Transactions on 30.5 (2004): 295-310.

Van Hoorn, André, Jan Waller, and Wilhelm Hasselbring. *"Kieker: A framework for application performance monitoring and dynamic software analysis."* Joint WOSP/SIPEW international conference on Performance Engineering.

ACM, 2012.OMG, 2012. *Semantics of a Foundational Subset for Executable UML Models (FUML)*. Object Management Group http://www.omg.org/spec/FUML

SPIKES, Spikes is member of ARTIST consortium

G. Canfora, M. Di Penta, L. Cerulo, *Achievements and Challenges in Software Reverse Engineering*. ACM 54 (2011) 142-151

S. Rugaber, K. Stirewalt, *Model Driven Reverse Engineering*, IEEE Software 21 (2004) 45-53.

CCRA, *Cloud Computing Reference Architecture* CT-CCA-o-016-Edit-Draft-RA ITU-T SG13/WP6.

Troya, J., Cubo, J., Martín, J.A., Pimentel, E., Vallecillo, A., 2013: *Automated Throughput Optimization of Cloud Services via Model-driven Adaptation*. MODELSWARD, pp. 356-362

ARTIST 2012, www.artist-project.eu

ModaClouds 2012 http://www.modaclouds.eu

PaaSage 2012 http://www.paasage.eu

Towards Bridging the Gap Between Scalability and Elasticity

Nicolas Ferry[1], Gunnar Brataas[2], Alessandro Rossini[1], Franck Chauvel[1] and Arnor Solberg[1]
[1]*Department of Networked Systems and Services, SINTEF, Oslo, Norway*
[2]*Department of Software Engineering, Safety and Security, SINTEF, Trondheim, Norway*
{*firstname.lastname*}*@sintef.no*

Keywords: Scalability, Elasticity, Cloud Computing, Multi-cloud, Self-adaptation, ScaleDL, CloudML.

Abstract: Scalability and elasticity are key capabilities to tackle the variable workload of a service. Cloud elasticity offers opportunities to manage dynamically the underlying resources of a service and improve its scalability. However, managing scalability of cloud-based systems may lead to a management overhead. Self-adaptive systems are a well-known approach to tame this complexity. In this position paper, we propose an approach for the continuous design and management of scalability in multi-cloud systems. Our approach is based on a three-layer architecture and relies on two existing frameworks, namely SCALEDL and CLOUDMF.

1 INTRODUCTION

Scalability and elasticity are key capabilities required to tackle the variable workload of a service. In particular, scalability is the ability of a service to sustain variable workload while fulfilling quality of service (QoS) requirements, possibly by consuming a variable amount of underlying resources. By contrast, elasticity is the ability of a service to rapidly provision and deprovision underlying resources on the fly. Note that both scalability and elasticity are required because one does not guarantee the other.

Cloud computing offers opportunities to improve the scalability of services by enabling the dynamic management of platform and infrastructure resources. Within the scope of the CloudScale[1], MODAClouds[2], and PaaSage[3] projects on cloud computing, specific work has been done to address long-term workload variations (*e.g.,* the workload increases linearly over a year) as well as short-term ones (*e.g.,* the workload has a sudden peak). The CloudScale's Scalability Description Language (SCALEDL) (Brataas et al., 2013) is a language for characterising scalability aspects of cloud-based systems in a provider-independent way. SCALEDL facilitates reasoning about long-term workload evolution in a proactive way. The MODAClouds' and PaaSage's Cloud Modelling Framework (CLOUDMF) (Ferry et al., 2013b; Ferry et al., 2013a) consists of the Cloud Modelling Language (CLOUDML) for specifying the provisioning and deployment of multi-cloud systems at design-time, as well as a models@run-time engine for enacting the provisioning, deployment, and adaptation of these systems at run-time. CLOUDMF facilitates handling short-term workload evolution in a reactive way.

Reasoning about long-term workload evolution proactively and handling short-term workload evolution reactively are typically treated as two independent activities. In order to improve management of workload evolution, these two activities must be combined. This position paper proposes an architecture inspired by self-adaptive systems for self-managing scalability of multi-cloud systems. This architecture consists of three layers: a layer for long-term adaptations that leverages upon SCALEDL, a layer for short-term adaptations that leverages upon CLOUDMF, and an intermediary layer for handling mid-term adaptations that bridges the gap between the other two layers. The proposed approach spurs to a new separation of concerns between mechanisms to handle short-, mid- and long-term scalability, enabling the continuous design and management of scalable cloud-based systems.

The remainder of the paper is organised as follows. Section 2 outlines a motivating example which is used throughout the paper. Section 3 introduces the three-layer architecture for self-managing scalability of multi-cloud systems. Section 4 and 6 outline SCALEDL and CLOUDMF as the foundation for the long-term and short-term adaptations, respectively.

[1]http://www.cloudscale-project.eu
[2]http://www.modaclouds.eu
[3]http://www.paasage.eu

Section 5 presents the new, intermediary layer for handling mid-term adaptations that bridges the gap between SCALEDL and CLOUDMF. Section 7 compares the proposed approach with related works before Section 8 draws some conclusions.

2 MOTIVATING EXAMPLE

SENSAPP[4] is an open-source, service-oriented application for registering sensors, storing their data, and notifying clients when new data are pushed (Mosser et al., 2012).

SENSAPP is currently employed in the CITI-SENSE[5] EU project, which aims at providing a community-based environmental monitoring and information system. In particular, SENSAPP is adopted for collecting environmental monitoring data (air quality, noise level, *etc.*) in public spaces where the workload of SENSAPP may vary significantly throughout a year. In particular, we assume the following predicable variations at various time scales:

Daily The workload has regular peaks at mornings and evenings, when people commute daily.

Weekly The workload has regular peaks on Friday evenings, Sunday nights, and Monday mornings, when people commute for the weekend.

Yearly The workload has gradual increase and decrease before and after Summer and Christmas holiday seasons.

We also assume the following unpredictable variations:

Snow storm and cold wave The workload has an irregular peak due to a snow storm and cold wave, such as the one in Europe on 27 January 2012, which led to the disruption of European air and surface traffic for two weeks.

Volcanic eruption The workload has an irregular peak due to a volcanic eruption, such as the one of the Eyjafjallajkull volcano on 14 April 2010, which led to the stop of all European air traffic and increase of surface traffic for one week.

This scenario is going to be used as a running example throughout the paper in order to illustrate how the proposed approach handles these predictable and unpredictable variations.

[4]http://sensapp.org

[5]http://www.citi-sense.eu

3 ARCHITECTURE

The proposed architecture for self-managing scalability of cloud-based systems is based on the reference three-layer architecture for self-adaptive systems (Kramer and Magee, 2007). This architecture is organised as a three-layer stack, where each layer denotes a particular level of abstraction, complexity, and dynamic, *i.e.,* how fast the layer can react to a variation. Note that these layers retain a certain degree of independence so that each of them can pursue its own dynamic.

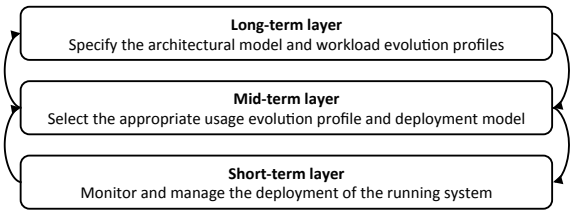


Figure 1: A three-layer architecture for self-managing scalability of multi-cloud systems.

Figure 1 outlines the proposed architecture. From the most abstract to the most concrete (*i.e.,* from the farthest to the closest to the running system), the three layers are described as follows:

Long-term layer. The designer specifies the architectural model of the system as well as usage evolution profiles within different time frames: day (*e.g.,* rush hour), week (*e.g.,* weekend), and year (*e.g.,* holiday season).

Mid-term layer. The mechanisms of this layer select the appropriate usage evolution profile and deployment model based on the current workload and context of the running system.

Short-term layer. The mechanisms of this layer monitor and manage the deployment of the running system.

For each layer, the considered time frames are up to the designer and may vary. However, these have to be organized hierarchically (*e.g.,* short-term: current, mid-term: week, long-term: year).

The independence between the layers enables the continuous evolution of the cloud-based system. The design process at the long-term layer can be done whilst the two other layers continue adapting the running system. This way, the design of the self-adaptive system does not need to be achieved beforehand, but can be done iteratively. Similarly, while the mid-term layer selects a new deployment model, the short-term layer can still manage minor workload evolution. This feature enhances the practicality of the approach

as well as the design and management of the self-adaptive system.

All these layers are built upon model-driven engineering (MDE) techniques and methods. MDE is a branch of software engineering that aims at improving the productivity, quality, and cost-effectiveness of software development by leveraging upon models and model transformations. This approach, which is commonly summarised as "model once, generate anywhere", is particularly relevant when it comes to the provisioning and deployment of applications across multiple clouds, as well as their migration from one cloud to another.

4 LONG-TERM LAYER: ScaleDL

The long-term layer leverages upon SCALEDL, a family of four sub languages to characterise scalability of cloud-based systems (Brataas et al., 2013). In particular, this layer adopts SCALEDL USAGE EVOLUTION, which facilitates the specification of scalability requirements by characterising how the *workload* of a service changes over time. Note that these models are technology- and provider-agnostic so that they can be applied to multi-cloud systems.

As a first task, a designer must specify initial usage evolution profiles (Brataas et al., 2013) for all the operations provided as a service by the application. In the motivating example (see Section 2), the following initial usage evolution profiles of SENSAPP are specified: (*i*) *Browse*, where consumers are looking for some sensors registered in SENSAPP, (*ii*) *Monitor*, where consumers request data from sensors and (*iii*) *Push*, where sensors are pushing data into SENSAPP. These specifications include:

Work. The amount of data processed by the operation; *e.g.,* the work of the Push operation may be 100KB of data.

Load. The number of requests to the operation during a specified time interval; *e.g.,* the load of the Push operation may be an average of one request per second during regular work days, two requests per seconds during regular weekends, and four requests per seconds during snow storms and cold waves.

Quality metric and threshold. The measure of quality of the operation along with the corresponding threshold; *e.g.,* the quality of the Push operation is measured in terms of the average response time and the corresponding threshold is 100 ms.

The initial usage evolution profiles will then evolve over time. SCALEDL USAGE EVOLUTION supports three forms of usage evolution (Brataas et al., 2013): *stable* and *gradual* changes as well as *spikes*. In the motivating example (see Section 2), the usage evolution before and after holiday seasons is modelled using the gradual change usage evolution. Here, both the work and load will have a gradual change, but not necessarily with the same increase or decrease rate; *e.g.,* the amount of data pushed by each sensor increase faster than the number of sensor pushing data. Similarly, the usage evolution of a snow storm and cold wave or volcanic eruption is modelled with a spike. Here, the duration of the spike as well as its impact on work and load may vary.

In addition to the three forms of usage evolution above, it is possible to specify the expected usage evolution for a particular recurring time frame; *e.g.,* daily, weekly, or yearly. In the motivating example (see Section 2), the usage evolution profile of a working day specifies low load during the night, high load during rush hour, and normal load otherwise.

In addition to the usage evolution profiles, the long-term layer is used to specify architectural models. These models follow a service-oriented architecture where applications are defined as an orchestration of reusable services.

5 MID-TERM LAYER

The mid-term layer is responsible for bridging the gap between reasoning about long-term workload evolution proactively and handling short-term workload evolution reactively.

The usage evolution profiles and architectural models from the long-term layer are provided to the mid-term layer, where the responsible mechanisms select a usage evolution profile, predict the future workload, and evaluate if the quality metric and threshold are fulfilled. In particular, the mechanisms of this layer iterate the following process (see Figure 2):

1. The *Usage Evolution Manager* selects a usage evolution profile based on the current workload and context of the running system, *e.g.,* working day profile. This selection relies on event-condition-action rules, *e.g., on Monday (event), if this is a working day (condition), then select the working day profile (action).* In case a designer wants to extend or refine the usage evolution profile for a shorter time frame, she can edit it before initiating the next task.

2. The *Workload Predictor* predicts what the workload will be at the time $t+1$ based on the workload at the current time t.
3. The *Workload Comparator* compares the current workload with the predicted one.
4. The *Architecture Selector* selects the most suitable architectural model from the the set of possible architectural models based on the current and predicted workloads.
5. The *Deployment Selector* selects the most suitable deployment model based on the corresponding architectural model.

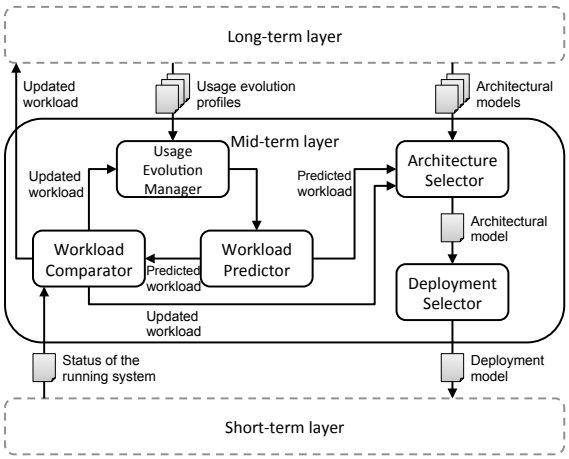


Figure 2: Architecture of the mid-term layer.

An iteration of this process can be triggered in two cases: *(i)* when the Workload Predictor anticipates that the workload is going to change; *(ii)* when the Workload Comparator detects that current workload does not correspond to the predicted one. In the latter case, a notification is raised to the long-term layer, which may alter the workload predictions.

In the motivating example (see Section 2), this process handles the following mid-term adaptations:

- In the event of snow storm and cold wave, the short-term layer reports the handling of the unpredictable workload to the mid-term layer, which analyses it and concludes it is an acceptable deviation from the prediction and hence does not report it to the long-term layer. This is because during winter one can anticipate that snow storms may occur and their average impact.
- In the event of volcanic eruption, the short-term layer reports the handling of the unpredictable workload to the mid-term layer, which analyses it and concludes it is an unacceptable deviation from the prediction and hence reports it to the long-term layer, which in turn alters the prediction. This is because, for instance, the predicted load in the usage evolution profile is two requests per second, while the current load is four. In addition, the load has been twice the predicted also during the two previous hours. Therefore, it may be reasonable to assume that the load will be twice the predicted also during the next hours, so the usage evolution profile for the next hours may be changed accordingly.

In case the duration of the workload is expected to be short, and considering the fact that adapting the deployment of a multi-cloud system may take some minutes, the deployment model may already be able to handle a range of variations of workload. For instance, this may lead to the inclusion of load-balancing mechanisms in the deployment model. These mechanisms are managed by the short-term layer.

6 SHORT-TERM LAYER: CloudMF

The short-term layer leverages upon CLOUDMF, a framework facilitating the management of multi-cloud systems. It consists of: (*i*) CLOUDML, a tool-supported, domain-specific language for specifying the provisioning and deployment of multi-cloud systems at design-time; (*ii*) a models@run-time engine for enacting the provisioning, deployment, and adaptation of these systems at run-time.

The deployment model from the mid-term layer is provided to the short-term layer, where the responsible mechanisms will adapt the running system.

CloudML

The deployments models are specified with CLOUDML. As mentioned, CLOUDML enables modelling the provisioning and deployment of multi-cloud systems. Note that CLOUDML is technology-agnostic, meaning that the multi-cloud systems can be designed and implemented based on arbitrary paradigms and technologies. In particular, CLOUDML allows expressing the following concepts:

Cloud Represents a collection of virtual machines on a particular cloud provider.

Virtual machine Represents a reusable type of virtual machine.

Application component Represents a reusable type of application component to be deployed on a virtual machine, or an external service. This element

can be parameterised by elasticity requirements (*e.g.,* SENSAPP can be instantiated from one to four times).

Port Represents a required or provided interface to a feature of an application component.

Relationship Represents a communication between *ports* of two application components, or the containment of an application component by another.

The interested reader may consult (Rossini et al., 2013) for a more detailed description of CLOUDML.

In the motivating example (see Section 2), SENSAPP is deployed on three different cloud providers. Figure 3 shows a snapshot of the deployment model of SENSAPP to be used when the workload is low: the MongoDB database and the *Dispatcher* of SENSAPP are deployed behind the corresponding load balancers (depicted by LB in the figure) on a private OpenStack cloud, while the *Notifiers* (*i.e.,* the services that notify consumers about new sensor data) are deployed on the Amazon and Flexiant public clouds.

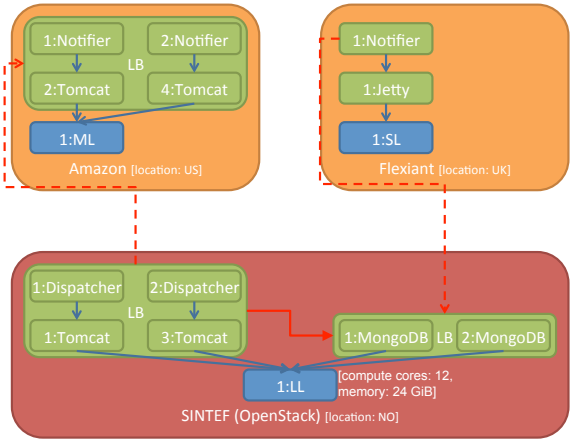


Figure 3: Snapshot of a SENSAPP deployment.

Models@run-time engine

The deployment models are enacted by the CLOUDMF models@run-time engine. Models@run-time is an architectural pattern for dynamic adaptive systems that proposes to leverage models during their execution (Morin et al., 2009). In particular, models@run-time provides an abstract representation of the underlying running system, which facilitates reasoning, simulation, and enactment of adaptation actions. A change in the running system is automatically reflected in the model of the current system. Similarly, a modification of this model can be enacted on the running system on demand.

Figure 4, inspired by (Morin et al., 2009), shows a classical architecture to realise models@run-time. The current model can be consumed by a reasoning system (*e.g.,* the mid-term layer) that produces a target model. The adaptation is then enacted by the adaptation engine and the target model becomes the current model.

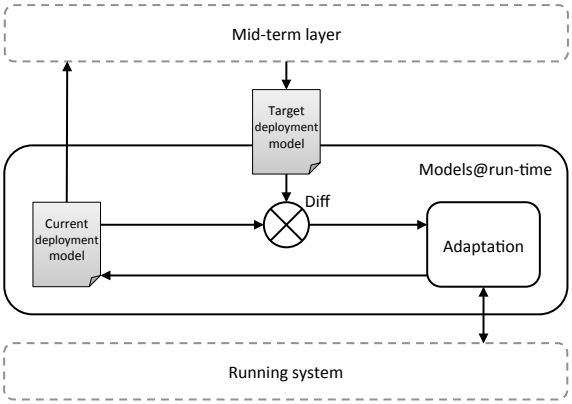


Figure 4: Models@run-time engine.

Adaptations of the model can result in: (*i*) modification of the provisioning and deployment topology or (*ii*) modification of the status of each element of the system. As part of the possible adaptation, a classical solution to handle elasticity requirements is to use load balancers. The models@run-time engine is then responsible for deploying a load balancer and configuring it to manage communications to a group of application components on a defined port.

The monitoring of the running system consists in providing the status of the current deployment together with the quality metrics (see Section 4) so that the current workload and context can be matched against the ones predicted in the long-term layer.

7 RELATED WORK

Elasticity is one of the key features to make cloud-based systems scalable and has been extensively investigated over the past few years. Elastic infrastructure are supported either by provider-specific features such as the Amazon Elastic Load Balancing, or by network level's components such as NGinx. Map/reduce platforms form another level of elasticity provided also as provider-specific services. While these techniques are now mature, they require recurrent management activities through the lifespan of a cloud-based system, as the workload evolves.

The management of cloud-based systems, including their scalability, can be facilitated by various tools. Industrial frameworks such as Cloudify[6], pup-

[6]http://www.cloudifysource.org

pet[7], or Chef[8], as well as research projects such as Reservoir[9] or ARTIST[10], provide capabilities for the automatic deployment and management of cloud systems, including load-balancing mechanisms. However, such frameworks do no support the long-term adaptation of elasticity policies, as does the proposed long-term layer.

Scalability management can also be addressed with performance analysis techniques. Palladio and SimuLizar (Becker et al., 2013), for instance, capture both the system and the scalability logics and produce performance predictions, which can be matched against performance requirements, leading to gradual improvements. However, this is a pure design-time activity which does not leverage run-time information, as does the model@run-time engine in the proposed short-term layer.

The Topology and Orchestration Specification for Cloud Applications (TOSCA) (Palma and Spatzier, 2013) standard is a related specification developed by the OASIS. TOSCA provides a language for specifying the components comprising the topology of cloud applications along with the processes for their orchestration. However, this standard currently lacks a models@run-time representation that enables the continuous evolution of multi-cloud systems.

8 CONCLUSION AND FUTURE WORK

In this position paper, we outlined an approach to self-managing scalability of multi-cloud systems. The proposed solution combines SCALEDL and CLOUDML into a three-layer architecture. In the long-term layer, the designer specifies the architectural model of the system as well as SCALEDL usage evolution profiles. In the mid-term layer, the responsible mechanisms select the appropriate usage evolution profile and deployment model based on the current workload and context of the running system. Finally, in the short-term layer, CLOUDMF monitors and manages the deployment of the running system.

The realisation of this three-layer architecture is an ongoing joint work between the CloudScale, MODAClouds, and PaaSage projects. Future research directions include: finalising the implementation and validation of the proposed approach, and designing a mechanism for collecting past load variations to improve the accuracy of load predictions by means of statistical analysis.

[7]https://puppetlabs.com

[8]http://www.opscode.com/chef

[9]http://www.reservoir-fp7.eu/

[10]http://www.artist-project.eu/

ACKNOWLEDGEMENT

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreements number: 318484 (MODAClouds), 317715 (PaaSage), and 317704 (CloudScale).

REFERENCES

Becker, M., Becker, S., and Meyer, J. (2013). SimuLizar: Design-Time Modeling and Performance Analysis of Self-Adaptive Systems. In *Software Engineering 2013*, volume 213 of *LNI*, pages 71–84. GI.

Brataas, G., Becker, S., Huljenic, D., Kopčak, G., Lehrig, S., Stav, E., and Walderhaug, S. (2013). D1.1 – Design support, initial version. CloudScale deliverable.

Ferry, N., Chauvel, F., Rossini, A., Morin, B., and Solberg, A. (2013a). Managing multi-cloud systems with CloudMF. In *NordiCloud 2013: 2nd Nordic Symposium on Cloud Computing and Internet Technologies*, pages 38–45. ACM.

Ferry, N., Rossini, A., Chauvel, F., Morin, B., and Solberg, A. (2013b). Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems. In *IEEE CLOUD 2013*, pages 887–894. IEEE Computer Society.

Kramer, J. and Magee, J. (2007). Self-Managed Systems: an Architectural Challenge. In *FOSE 2007*, pages 259–268.

Morin, B., Barais, O., Jézéquel, J.-M., Fleurey, F., and Solberg, A. (2009). Models@Run.time to Support Dynamic Adaptation. *IEEE Computer*, 42(10):44–51.

Mosser, S., Fleurey, F., Morin, B., Chauvel, F., Solberg, A., and Goutier, I. (2012). SENSAPP as a Reference Platform to Support Cloud Experiments: From the Internet of Things to the Internet of Services. In *SYNASC 2012*, pages 400–406. IEEE Computer Society.

Palma, D. and Spatzier, T. (2013). Topology and Orchestration Specification for Cloud Applications (TOSCA). Technical report, OASIS.

Rossini, A., Solberg, A., Romero, D., Domaschka, J., Magoutis, K., Schubert, L., Ferry, N., and Kirkham, T. (2013). D2.1.1 – CloudML Guide and Assesment Report. PaaSage deliverable.

Preliminary Design of a Platform-as-a-Service to Provide Security in Cloud

Valentina Casola[1], Alessandra De Benedictis[1], Massimiliano Rak[2] and Umberto Villano[3]
[1]*Università "Federico II" di Napoli, Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione, Napoli, Italy*
[2]*Seconda Università di Napoli, Dipartimento di Ingegneria dell'Informazione, Aversa, Italy*
[3]*Università del Sannio, Dipartimento di Ingegneria, Benevento, Italy*
massimiliano.rak@unina2.it, {casolav, alessandra.debenedictis}@unina.it, villano@unisannio.it

Keywords: Cloud, Cloud Security, SLA, Requirements.

Abstract: Cloud computing is an emerging paradigm, widely adopted in distributed and business computing. Nevertheless, the biggest issue with the large adoption of cloud computing is the perception of loss of security and control over resources that are dynamically acquired in the cloud and that reside on remote providers, and the strong integration of security mechanisms into system architectures. This paper deals with the integration of security features into cloud applications by an as-a-service approach, using Service Level Agreements as a means to clearly define rights and constraints of both customers and providers. The goal is to show the main requirements of a platform dedicated to security and to present the global architecture, in terms of components and their interactions, devoted to negotiate, monitor and enforce the security mechanisms to be applied over existing cloud providers.

1 INTRODUCTION

Cloud computing, based on features as *on-demand self-service* and on the *pay-per-use* business model, is currently the emerging paradigm for distributed computation and has been recently indicated as a strategic asset by the European Commission (Commission, 2011), which is assuming a set of initiatives (e.g. European Cloud Partnership (Commission,)) aimed at supporting its adoption in both private and public bodies.

The biggest issue with the large adoption of cloud computing is related to security (Dekker, 2012; Catteddu, 2011), as the paradigm assumes that all resources are *delegated to the cloud* (i.e., hosted and managed by a remote cloud provider, and can be hardly ever physically located), while typically cloud providers have their own policies for the management of reserved data and privacy. The approach almost universally followed to define guarantees on the provisioning of a service is the introduction of *Service level Agreements* (SLAs). SLAs should specify *Service Level Objectives* (SLOs) dedicated to security, but there is still a lack of standards for the definition of the security terms in a SLA, even if a lot of work is ongoing in this field by dedicated standard groups (as the SLA C-SIG from the European Commission, the CSCC SLA group(CSCC,)) and research projects (see CUMULUS (Pannetrat et al., 2013), A4Cloud (Pearson, 2011), and SPECS (Rak et al., 2013)).

In addition to the above discussed issues, a further problem is linked to the "traditional" design of security systems, which often requires *full control* of the systems to be protected, in contrast with the cloud principles that would require an approach based on *modular* security (Battista et al., 2013). To this end, the FP7 has funded the *SPECS* Project (Rak et al., 2013), which proposes a cloud Platform-as-a-Service (PaaS) dedicated to provide Security-as-a-Service using SLAs. In this paper, we present a preliminary requirement analysis for platforms like SPECS and identify the main involved components. In order to clarify the whole analysis process, we illustrate the simple methodology adopted to derive requirements, which can be successfully applied to the design of any PaaS.

The remainder of this paper is organized as follows. Section 2 illustrates the context of our work, while in Section 3, we present an overview of most relevant PaaS solutions and of the proposed platform. In Section 4 we show, for the most representative functionalities we are interested at offering, the whole extraction process of requirements. Finally, in Section 5 a preliminary platform architecture is presented, and

in Section 6 we draw our conclusions and identify future work directions.

2 MOTIVATION AND CONTEXT

In this Section, we provide some representative scenarios that outline the main open issues related to the management of security requirements in a cloud environment, and help to evidence the main functionalities required from a security-oriented platform. As a first scenario, let us consider a cloud federation whose set-up is explicitly subject to security requirements (i.e. the federated providers must ensure a set of security features). At the state of the art, the evaluation of security attributes related to a cloud provider is carried out *by hand* (for example, based on the provider's existing ISO/IEC 27000-series certifications), as there are no automatic tools to search, among the available cloud offerings, those matching specific security attributes. This is mostly due to the lack of standards to specify the requested/provided security parameters. Recently, some steps have been taken for improving transparency and assurance in the cloud by the Cloud Security Alliance, with its STAR (Security, Trust & Assurance Registry) initiative. However, even if STAR is a powerful means to help users assess the security of cloud providers, there is still a lack of mechanisms to effectively evaluate and compare different offerings, given a specific request.

Similar issues can show up even in the traditional interactions between cloud providers and end-users. Let us consider, for example, a cloud end-user that wishes to acquire a secure cloud storage service to remotely store data with specific data confidentiality requirements. With the currently available approaches, the user has first to manually search for cloud providers with the desired storage features, then check their SLAs in order to verify the kind of confidentiality features offered, study the different available offerings, and finally select the one that fulfills as much as possible its requirements. As for the previous case, it would be desirable to provide the end-users with a single access point to specify their functional and non-functional (security) requirements and with automatic tools to search for and to select the best offering. Moreover, in both scenarios, once a security control has been guaranteed from a provider, the customer should be provided with proper tools to monitor the correct service provisioning while, at the state of art, the customer must accept the provider monitoring services and metrics, and can only create local measurements to obtain further information (e.g., custom local measurements of response time).

3 DEFINING A SECURITY-ORIENTED PAAS

As stated, our main goal is the design and the implementation of a PaaS dedicated to security services, based on an SLA approach. At the state of the art, PaaS solutions are very different from one another, and there is no standard interpretation of how to build them. The most common solution is based on adapting application servers (like Apache Tomcat), in order to decouple application deploying from virtual machines acquired over Infrastructure-as-a-Service solutions. Examples of such solutions are offered by FP7 projects as cloud4SOA (Zeginis et al., 2013), OPTIMIS (OPTIMIS,) or Contrail (in the ConPaaS module (Pierre and Stratan, 2012)). Such PaaS solutions are usually dedicated to a single technology (in the most common case, web applications), which is cloud-enabled in a way completely transparent to application developers. This approach is based on the idea of porting well-known middlewares to the cloud, and of making transparent to middleware users the adoption of the cloud paradigm for lower level resource acquisition. In other words, applications (and their developers) are not aware of the cloud.

An alternative approach is the one proposed by integrated stacks as Google App Engine (GAE) or Microsoft Azure, where developers must adopt dedicated APIs to build cloud-enabled applications. Such platforms are integrated with the offering of the CSP, who manages the full stack of services (SaaS, PaaS and IaaS). Applications are developed using dedicated toolkits: in this case, application developers are aware of the adoption of the cloud paradigm and have a set of dedicated APIs that enable them to exploit the cloud flexibility in their applications. As a drawback, applications are dependent on the platform, and cannot be used over different cloud providers without modifications.

Finally, a further approach has been proposed in solutions as mOSAIC (Petcu et al., 2013) or openshift (RedHat,). These adopt a dedicated *cloudware*, which can be deployed over infrastructure resources and offers APIs similar to the ones offered by GAE or Azure, but independent of the CSP that delivers the (infrastructure-level) resources. Our goal is to start from such solutions and build a PaaS dedicated to offering security (Security-as-a-Service) through an SLA-based approach.

As the first step towards the design of the platform, we analyzed the possible interactions with prospective customers, identifying three interaction models illustrated in details in Section 3.1. After that, in order to define the platform requirements, we

adopted a simple methodology based on the following steps:

- we extracted the required macro-functionalities from some example scenarios described in Section 2 and, for each macro-functionality, we identified the involved actors and roles, and the different usage perspective deriving from the different interaction models defined in the previous phase;
- we described each usage perspective by means of an high-level interaction diagram, pointing out the main actions performed by the platform; from this functionalities we identified a preliminary set of high-level use cases from the previous interaction diagrams;
- we classified the resulting use cases into different service layers and carried out a detailed analysis of each service to identify lower level use cases.

Starting from the above described requirement analysis, we identified a preliminary architecture that subdivides the services offered by the platform in different modules and outlines their interactions. Section 5 will describe such preliminary design.

3.1 Actors, Roles and Interaction Models

The platform is intended for different users and can serve different purposes: both end-users and CSPs can access it in order to (i) specify their cloud security requirements by means of SLAs, (ii) automatically evaluate and compare security features offered by remote CSPs (brokering of security services), (iii) enhance a remote service based on specific security requirements, (iv) manage the full SLA life-cycle (negotiation, monitoring and enforcement), and (v) develop and deploy security services that are *cloud SLA-aware*, implemented as a flexible open-source PaaS.

Such functionalities imply that the involved actors can play different roles depending on the specific functionality they invoke (e.g., developer, administrator, customer ...). Moreover, based on the specific hosting configuration and on the resulting different security and responsibility issues, we can identify three different ways to interact with the platform, which introduce further role classifications. In order to have a clear and shared role definition, we will adopt the high-level stakeholder taxonomy proposed in the Cloud Standards Coordination (CSC) initiative launched by ETSI (ETSI, 2013), which basically considers the roles of **customers**, **providers**, and **partners** for each party (represented by either an individual or an organization). While customers and providers are those asking for a service and offering

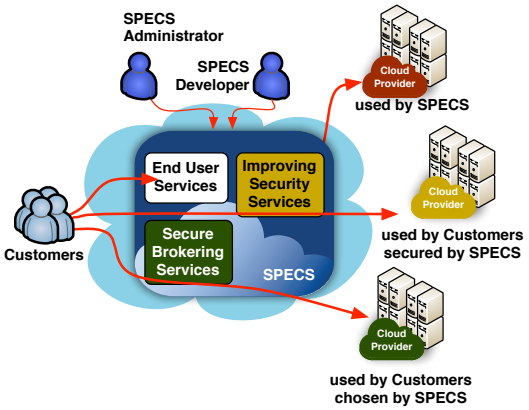


Figure 1: Interaction Model 1: Third Party Security Platform.

the service, respectively, partners are those providing support to the provisioning and/or the consumption of services. To avoid ambiguity, in our discussion we will refer to the end-users as the parties located (as customers) at the beginning of the service supply chain, while the external providers will represent those CSPs located at the end of the supply chain. We do not care about the way these CSPs offer their services, which are in the following considered atomic.

Even if this approach is very general, and covers all the different aspects of cloud service *supply chains*, it has the side effect that in different scenarios the same actor can play different roles, based on the considered perspective, generating confusion in the requirement analysis of a single cloud product that can be deployed in different ways. In particular, the platform can be offered in three different ways as shown in the following interaction models.

The main interaction model, named **Third Party Security Platform** (IM1), is proposed in Figure 1. The platform runs as an independent third-party component, consuming its own resources – acquired from a public or private cloud provider and managed by the administrator – and offering its services to end users. In such model, the platform needs to authenticate its own users and to manage their credentials in a secure way. As for responsibility taking, it depends on the role played by the platform. Indeed, the platform can play both the ETSI roles of CSP and partner: as a CSP, it signs a contract with the end user and takes care of the entire remote service invocation process on his behalf. When it acts as a partner, instead, the user only exploits its brokering capabilities to find the service that bests suits his needs, but then signs a contract with the selected remote service provider to have his requirements granted.

The second interaction model is named **Hosted**

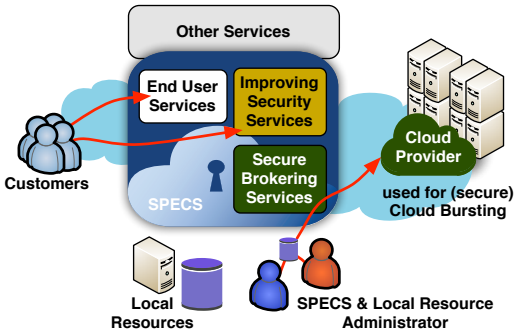


Figure 2: Interaction Model 2: Hosted Platform.

Platform (IM2) and is shown in Figure 2. In such interaction scheme, the administrators are co-located within the Hosting CSP, which internally hosts the platform. A hosting CSP can exploit the platform in order to (i) enhance security features of the services and resources being normally offered to end users, (ii) acquire external CSP services to fulfill users' requests (in this case, secure brokering services can be used also to perform cloud bursting – note that such services are invisible to end users), and offer additional functionalities to users by re-selling the end-uses services implemented by the platform. Differently from the first interaction model, in this case the platform acts only as a CSP, and never as a partner, because it directly provides the hosting CSP with the requested services taking on all responsibilities for them.

The third interaction model is referred to as **User Software** (IM3) and is shown in Figure 3. It assumes that the platform is dedicated to a single end user, who installs and runs it to manage his own activities. In this case, the platform does not offer services to other end users, but it is just adopted by its administrator (the end user who owns it) in order to manage security features over remote services. For this reason, internal services are almost useless (no need for authentication, or for offering something internally dedicated), while brokering and security enhancement are the only exploited functionalities. In this interaction model, the platform acts only as a partner, because the end user adopts its features only to build his own services.

4 PRELIMINARY REQUIREMENT ANALYSIS

In this Section we show the whole process followed to analyze some the main functionalities offered by the platform to derive a preliminary set of requirements. In particular, for the sake of brevity, we focus on one of the macro-functionalities identified in Section 3, that is *Brokering of an IaaS (Service) with granted security parameters*.

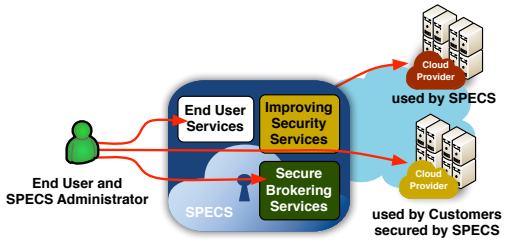


Figure 3: Interaction Model 3: User Software.

Table 1: Brokering usage perspectives.

Interaction Models	Brokering
IM1	(as Partner)IM1-P / (as CSP)IM1-CSP
IM2	(as CSP) IM2-CSP
IM3	(as Partner)IM3-P

With respect to this macro-functionality, we analyzed the behavior of the platform in the three interaction models proposed in the project. In particular, for each interaction model, we considered the possible roles played by the platform (CSP or Partner-P), deriving a set of usage perspectives, shown in Table 1. As illustrated in Section 3.1, not all the combinations are allowed: for example, in the interaction model 2, the platform acts only as a CSP and never as a Partner, because it directly provides the Hosting CSP with the requested services taking on all responsibilities for them. In the following, we are going to detail a couple of usage perspectives, and report high level interaction diagrams from which to derive a preliminary set of use cases.

In Figure 4, we report the high level interaction diagram for the usage perspective named *IM1-CSP*: the aim is to identify the basic steps related to the execu-

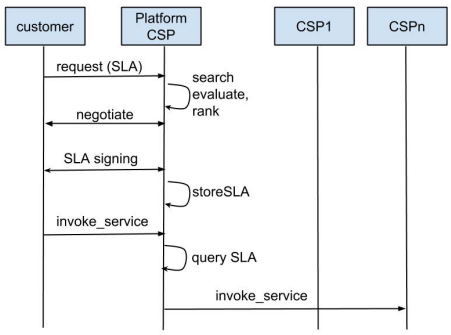


Figure 4: The platform acts as a broker in IM1.

tion of the brokering macro-functionality in the case when the platform takes on all responsibilities on the *quality* (i.e. level of assured security) of the invoked services. As depicted, the involved actors are the Customer, the Platform-CSP and one or more External CSPs. We assume that the Platform-CSP manages a repository of existing CSPs and related SLAs and a customer submits to the Platform-CSP a SLA request with a list of security requirements. The basic flow of events can be described as follows:

1. a customer submits to the Platform CSP his security requirements in terms of a SLA,
2. the Platform CSP compares the SLA's security requirements against those specified in its CSP SLA repository and possibly applies a ranking algorithm,
3. the Platform CSP negotiates with the customer the target CSP to invoke based on its SLA,
4. the Platform CSP and the customer sign the SLA, and the Platform CSP stores the signed SLA in a signed SLA repository,
5. the customer invokes one of the negotiated services on the Platform CSP, which forwards the invocation request to the proper access point and retrieves possible results, to return them back to the customer.

With respect to the other interaction models, the brokering function is basically carried out in a similar way, except that no SLA is signed between the customer and the platform whenever the latter acts as a Partner.

From the analysis of this scenario, we were able to derive a first set of functionalities, which can be logically split into three service layers: the *application service layer* includes all services related to the implementation of the business logic of the application built on top of the platform (in this case, the application that offers a broker of secure services). The *core service layer* includes all services related to negotiation of SLAs and search, evaluation and ranking of providers based on their SLAs. Finally, the *Platform service layer* includes all basic enabling services (e.g.set-up, activation, deployment of services on the cloud infrastructure) and services related to SLA lifecycle management.

5 PRELIMINARY ARCHITECTURE DESIGN

A preliminary architecture design is shown in Figure 5: as illustrated in the previous Section, the services offered through the platform can be classified into three service layers: the Application layer, the Core Services layer and the Platform layer. The applications built on top of the platform belong to the application layer and are implemented by using the services offered by both Core and Platform layers. The main concept behind the platform is that every functionality is offered as a service and all applications are managed through SLAs. For this reason, the interactions among an application hosted by the platform and its customers are subject to negotiation and controlled via enforcement and monitoring functionalities. According to this view, the Core Services layer is composed by the following modules:

- Negotiation Module: manages all the services dedicate to guide customers and providers to reach the agreement on SLAs.
- Enforcement Module: manages all the services dedicated to apply security mechanisms and correctly configure and use services, on the basis of the agreed SLAs.
- Monitoring Module: manages all the services dedicated to collect and evaluate the state of the services acquired.

At the Platform layer, as discussed in the previous Section, we locate both low-level (enabling) services, to support the correct execution of higher-level services in the cloud environment and offer basic functionalities (repository management, SLA management), and services for interoperability, which implement the actual communication among services located at the core layer. Indeed, architecture modules must be usable independently one of the others, provided interfaces among modules will be as simple as possible and all modules will support the SLA-based philosophy. This implies that the communication among services of the Core layer will be possible only through the Platform layer services, which will also support the interoperability among modules. From the architecture point of view, the Platform layer will be then composed of two sub-layers:

- Enabling Platform: enables the execution of components (configuration, starting, stopping, packaging) on-top of a cloud-like (IaaS-like) infrastructure and offers other generic support services(component configuration, communication, storage).
- SLA Platform: offers services to enable the management of SLA life cycle, including the management of SLA repositories and the communication among services dedicated to different phases of SLA life cycle, acting as the SLA-based glue

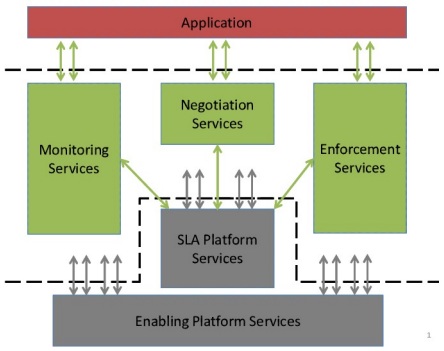


Figure 5: Architecture.

for the security services we are interested to offer over the platform.

The layers also put in evidence that SPECS is able to provide both Software-as-a-Service (SaaS) and PaaS service models: as a SaaS, it offers a deployed Platform, hosting a set of applications, which provides application-specific services to its own customers; as a PaaS, it offers a deployed Platform that enables the deployment of new applications. As an example, from one side the customers can use the platform to invoke (use) an Improved Security Service, consisting in a service either locally developed in the platform or obtained by improving an existing service offered by a remote CSP, which has specific security features. From the other side, the customers can use the platform to develop, for example, an Improved Security Service, based on basic enabling services (those offered by the Enabling Platform and the SLA Platform).

6 CONCLUSIONS

In this paper we presented the preliminary analysis of a *cloudware* dedicated to create a PaaS that offers security services through an SLA approach. We proposed a methodology to design such kind of platforms, and sketched the preliminary results of the requirement analysis, starting from some of the usage scenarios identified in the context of the SPECS FP7 project. We argue that the proposed methodology is of general interest for the design of *cloudwares* and dedicated PaaS security solutions. The requirement analysis presented here can be reused independently of the technology which will can be involved, offering a high-level view over the provision of security as-a-service using SLAs. Moreover, we proposed a preliminary design of the Platform, based on the considerations and the classification of services made in our preliminary analysis. We plan to upgrade this requirement analysis in the future, collecting feedbacks from cloud stakeholders to refine the requirements and the preliminary design, and to start the prototyping phase of the platform.

REFERENCES

Battista, E., Casola, V., Mazzocca, N., Ficco, M., and Rak, M. (2013). Developing secure cloud applications:a case study.

Catteddu, D. (2011). Security & resilience in governmental clouds. Technical report.

Commission, E. European cloud partnership. http://ec.europa.eu/digital-agenda/en/european-cloud-partnership.

Commission, E. (2011). Unleashing the potential of cloud computing in europe. Technical report.

CSCC. The cscc practical guide to cloud service level agreements. Technical report.

Dekker, M. (2012). Critical cloud computing a ciip perspective on cloud computing services. Technical report.

ETSI (2013). Cloud standards coordination. Technical report.

OPTIMIS. The optimis project web site. http://www.optimis-project.eu/.

Pannetrat, A., Hogben, G., Katopodis, S., Spanoudakis, G., and Cazorla, C. (2013). D2.1: Security-aware sla specification language and cloud security dependency model. technical report, certification infrastructure for multi-layer cloud services (cumulus).

Pearson, S. (2011). Toward accountability in the cloud. *Internet Computing, IEEE*, 15(4):64–69.

Petcu, D., Martino, B. D., S., V., Rak, M., Mhr, T., Lopez, G. E., Brito, F., Cossu, R., Stopar, M., Sperka, S., and Stankovski, V. (2013). Experiences in building a mosaic of clouds. *JOURNAL OF CLOUD COMPUTING*, 2:–.

Pierre, G. and Stratan, C. (2012). Conpaas: a platform for hosting elastic cloud applications. *IEEE Internet Computing*, 16(5):88–92.

Rak, M., Suri, N., Luna, J., Petcu, D., Casola, V., and Villano, U. (2013). Security as a service using an sla-based approach via specs. In IEEE, editor, *Proceedings of IEEE CloudCom Conference 2013*.

RedHat. Openshift web site. https://www.openshift.com/.

Zeginis, D., D'Andria, F., Bocconi, S., Cruz, J. G., Martin, O. C., Gouvas, P., Ledakis, G., and Tarabanis, K. A. (2013). A user-centric multi-paas application management solution for hybrid multi-cloud scenarios. *Scalable Computing: Practice and Experience*, 14(1).

EU Project SeaClouds*

Adaptive Management of Service-based Applications Across Multiple Clouds

Antonio Brogi[1], José Carrasco[2], Javier Cubo[2], Francesco D'Andria[3], Ahmad Ibrahim[1], Ernesto Pimentel[2] and Jacopo Soldani[1]

[1]*Department of Computer Science, University of Pisa, Pisa, Italy*

[2]*Department of Computer Science, University of Malaga, Malaga, Spain*

[3]*ATOS, Barcelona, Spain*

Keywords: Multi-Cloud Deployment, Monitoring, Dynamic Reconfiguration, Service Orchestration.

Abstract: The adaptive management of complex applications deployed across multiple heterogeneous PaaS platforms is one of the problems that have emerged with the cloud revolution. The ongoing EU research project SeaClouds aims at providing seamless adaptive multi-cloud management of complex applications by supporting the distribution, monitoring and migration of application modules over multiple heterogeneous PaaS platforms. In this paper we present context, motivations and objectives of SeaClouds, its relation with other cloud initiatives, and its initial architecture.

1 INTRODUCTION

Current cloud technologies suffer from a lack of standardization, with different providers offering similar resources in a different manner (M. Armbrust et al., 2010). As a result, cloud developers are often locked in a specific platform environment because it is practically unfeasible for them, due to high complexity and cost, to move their applications from one platform to another (N. Loutas et al, 2011). To take full benefit of the flexibility provided by the cloud, modules of a complex application could be deployed on multiple clouds depending on their characteristics. Thus, in a scenario where a complex application is distributed across different cloud service providers, a solution is needed to manage and orchestrate the distribution of modules in a sound and adaptive way (Parameswaran and Chaddha, 2012).

Consider an application consisting of three modules (A, B, C), each having its own requirements (R_A, R_B, R_C). Suppose that four PaaS platforms are available (α, β, γ, δ), each providing different offers (O_α, O_β, O_γ, O_δ) and based on different technologies. The application developer can deploy the application either on a single PaaS or over multiple PaaS platforms. In the latter case, then she has to figure out the best offer for each module (e.g., $R_A : \{O_\gamma\}$, $R_B : \{O_\beta, O_\gamma\}$, $R_C : \{O_\delta\}$) and perform the deployment accordingly (e.g., $A \rightarrow \gamma$, $B \rightarrow \beta$, $C \rightarrow \delta$). Afterwards, she has to monitor both the single modules and the whole application. In case some requirements are violated (e.g., R_B is no longer satisfied by O_β), she has to manually reconfigure the deployment (e.g., $A \rightarrow \gamma$, $B \rightarrow \gamma$, $C \rightarrow \delta$). This implies that to manage multi-cloud applications, the developer must have knowledge of the different employed PaaS platforms and has to continuously monitor the application. This is a costly and cumbersome process and it would be much better to have some framework doing all the work automatically.



Figure 1: SeaClouds logo.

In this paper, we discuss the ongoing project SeaClouds (Seamless adaptive multi-cloud management of service-based applications) which focuses on the problem of deploying and managing complex multi-component application over heterogeneous clouds in an efficient and adaptive way. SeaClouds works towards giving organizations the capability of "Agility After Deployment" (Figure 1) for cloud-based appli-

*Work supported by EU-FP7-ICT-610531 SeaClouds project (start date: October 2013, end date: March 2016).

cations. Its approach is based on the concept of service orchestration and it is designed to fulfill functional and non-functional properties over the whole application. Applications will be dynamically reconfigured by changing the orchestration of the services when the monitoring detects that such properties are not respected. So, SeaClouds main objective is *the development of a novel platform which performs a seamless adaptive multi-cloud management of service-based applications*. More specifically, the objectives of SeaClouds are:

O_1) Orchestration and adaptation of services distributed over different cloud providers,

O_2) Monitoring and run-time reconfiguration of services distributed over multiple heterogeneous cloud providers,

O_3) Providing unified application management of services distributed over different cloud providers, and

O_4) Compliance with major standards for cloud interoperability.

The rest of the paper is organized as follows. Section 2 will position SeaClouds with respect to other cloud initiatives and it will describe its main challenges. Section 3 will present the SeaClouds approach along with its initial architecture. Finally, Section 4 will draw some concluding remarks.

2 SeaClouds IN CONTEXT

This section describes how SeaClouds, with the purpose of achieving its main goal, will advance the state of the art with respect to O_1, O_2 and O_3, and will contribute to obtain O_4. Figure 2 illustrates how SeaClouds intends to relate to the previously mentioned initiatives and standards.

Orchestration and Adaptation in the Cloud. O_1 will be addressed by developing adaptive orchestrators of cloud-based application modules. Orchestrators are widely used in the service-oriented computing paradigm (A. Brogi, J. Camara, C. Canal, J, Cubo and E. Pimentel, 2007; C. Canal and Salaun, 2008; J. Cámara, J.A. Martín, G. Salaün, J. Cubo, M. Ouederni, C. Canal and E. Pimentel, 2009; H. Nezhad, G.Y. Xu and B. Benatallah, 2010; J. Cubo and E. Pimentel, 2011), mainly focusing on behavioural and context-aware adaptation of services, by coordinating the interactions between different services. Several approaches exist that target formal adaptation of orchestrated services, but, to the best of our knowledge, only few (middleware-based) attempts have been proposed to face challenges such as heterogeneity of cloud platforms and migration to different cloud providers (e.g., (Guillén et al., 2013)).

Challenges in Orchestration and Adaptation for the Cloud. SeaClouds will address the following challenges in order to extend service-oriented approaches to the cloud: (i) Adaptation contracts need to take into account cloud provider characteristics and Service Level Agreements (SLA), (ii) Violations of Quality of Service (QoS) properties need to be monitored across different cloud platforms, and (iii) Dynamic architecture reconfiguration might involve migrating some components of the application to other cloud providers at runtime. The latter two challenges are discussed further in the following sections.

Monitoring of Multi-cloud Services. The EU FP7 Cloud4SOA project (www.cloud4soa.eu) provides an open source interoperable framework for application developers and PaaS providers. Cloud4SOA aims at supporting developers in deploying and monitoring their application with the ultimate objective of reducing the risk of vendor lock-in. The monitoring is based on unified metrics, but Cloud4SOA monitors each application separately and it is not able to aggregate monitoring results of multi-component applications.

Several Commercial and Open Source Initiatives Target Monitoring of Cloud Applications. Often these initiatives address only particular platforms, for example Appsecute (www.appsecute.com) monitors only (open-source) CloudFoundry-based platforms. More platform-independent technologies are available for the IaaS level, since the latter has undergone a stronger harmonization effort. Deltacloud (deltacloud.apache.org) encapsulates the native API cloud provider to enable management of resources in different IaaS platforms, such as Amazon EC2. Rightscale (www.rightscale.com) supports monitoring several public (e.g., Amazon Web Services, Google Cloud Platform, Rackspace, Windows Azure) and private IaaS clouds (e.g., CloudStack, Eucalyptus, OpenStack). Truly platform-independent monitoring solutions exist, the most known being NewRelic (www.newrelic.com). NewRelic achieves platform-independency by requiring each provider to implement a monitoring component and to integrate it in the offered cloud platform. On the one hand, this approach yields the best results from a monitoring point of view. On the other hand, it forces providers to invest quite some resources in order to implement the monitoring.

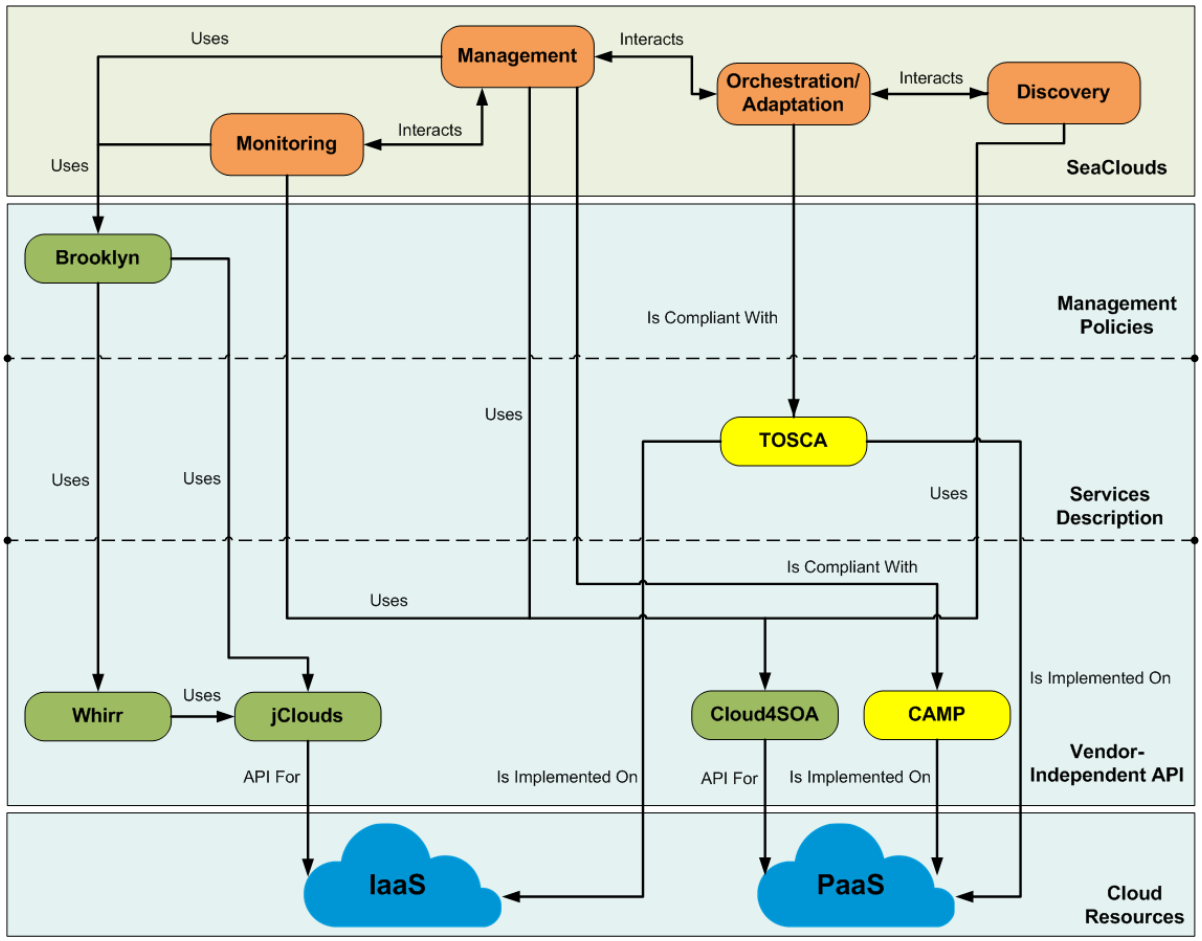


Figure 2: Positioning SeaClouds.

Challenges in Monitoring of Services on Multiple Clouds. In order to address O_2, SeaClouds monitoring will use and enhance existing monitoring functionalities for the PaaS and IaaS levels: (i) With respect to the IaaS level, SeaClouds will simply reuse what is available (e.g., Deltacloud), and (ii) With respect to the PaaS level, SeaClouds aims at augmenting the set of metrics currently available from Cloud4SOA (response time and up-time). For both the IaaS as the PaaS level, SeaClouds aims at coordinating, monitoring and aggregating monitoring information at the single service level to fulfill the purposes of whole orchestration. Thus, SeaClouds aims at: (i) monitoring each application component, and (ii) combining and aggregating the above mentioned data to highlight possible problems and their impact.

Unified Management of Multi-cloud Applications. Brooklyn (www.brooklyn.io) is an open source, policy-driven control plane for distributed applications delivered by CloudSoft (a member of the SeaClouds consortium). It enables single-click deployment of applications across machines, locations and clouds. Then it continuously optimizes running applications to ensure ongoing compliance with policies. Brooklyn uses two open source tools to operate on cloud resources, Apache Whirr (whirr.apache.org) and jClouds (www.jclouds.org), that support several IaaS providers. The already mentioned Cloud4SOA project also offers deployment and lifecycle management functionality using a harmonized API layer to encapsulate the providers' APIs. Furthermore, the previously mentioned initiatives (RightScale, Deltacloud for the IaaS level, and Appsecute, CloudFoundry for the PaaS level) all offer management functionality, with the same limitations as discussed while explaining monitoring.

Challenges in Unified Application Management of Services Distributed over Different Cloud Providers. SeaClouds will use existing management functionalities. Specifically, (i) SeaClouds discovery functionality may use and extend existing matchmaking functionalities to match application requirements with

PaaS offerings, and (ii) SeaClouds management will use a REST harmonized API for the deployment, management and monitoring of simple cloud-based applications across different and heterogeneous cloud PaaS offerings. SeaClouds intends to use Brooklyn's policy-driven functionality to integrate support for IaaS providers. Moreover, Brooklyn's approach to policy modeling and enforcing can provide guidance for SeaClouds orchestration/adaptation and management functionality. On the other hand, Brooklyn only targets the IaaS level and provides no support for orchestration. Beyond what Brooklyn provides, SeaClouds will therefore extend policy-driven functionality to the PaaS level and also add support for adaptation and orchestration. Due to a common partner in both initiatives (CloudSoft), Brooklyn can also benefit from integrating SeaClouds functionalities, especially regarding the integration of adaptation techniques in supported policies.

Standards for Cloud Interoperability. CAMP (Cloud Application Management for Platforms) (OASIS, 2012a) aims at defining harmonized APIs, models, mechanisms and protocols for the self-service management (provisioning, monitoring and control) of applications in a PaaS, independently of the cloud provider. However, CAMP is only a protocol specification, so it needs to be implemented by parties adopting the protocol.

The OASIS TOSCA (Topology and Orchestration Specification for Cloud Applications) (OASIS, 2012b) aims at enabling the interoperable description of application and infrastructure cloud services, the relationships between parts of the service, and the operational behaviour of these services, independently from the cloud provider. By increasing service and application portability in a vendor-neutral ecosystem, TOSCA aims at enabling portable deployment to any compliant cloud, smoother migration of existing applications to the cloud, as well as dynamic, multi-cloud provider applications.

Challenges in Standards for Cloud Interoperability. SeaClouds intends to actively contribute to the standardization effort of CAMP (OASIS, 2012a) both by exploiting CAMP-compliant interfaces provided by PaaS providers, and by contributing review proposals that will possibly emerge while specifying properties of SeaClouds orchestrations, adaptations and monitoring. SeaClouds will exploit the TOSCA specification to drive the design of the model for specifying cloud service orchestrations. In doing so, SeaClouds might actively contribute to the standardization effort of TOSCA, by contributing review proposals that will emerge while trying to devise TOSCA-compliant instances of the SeaClouds service orchestration model. On the other hand, SeaClouds will also focus on developing functionalities that are deliberately out of the scope of TOSCA (OASIS, 2012b) to solve issues about policies for the dynamic management of service orchestrations. Although the currently available implementations of TOSCA and CAMP do not yet support the management of complex applications over multiple clouds, SeaClouds will work towards building such management on top of them.

3 THE SeaClouds APPROACH

SeaClouds aims at homogenizing the management of different cloud providers and at supporting sound and scalable orchestrations of services across them. Moreover, systems developed with SeaClouds will inherently support the evolution of their constituent services, so as to easily and dynamically cope up with needed changes. The development, monitoring and reconfiguration via SeaClouds will include a unified management service, where services can be deployed, replicated, and administered by means of standard and harmonized APIs.

In order to obtain its main goals (viz., the development of the platform and the definition, implementation and validation of the use cases), SeaClouds will build a foundation to allow **"Agility After Deployment"** by providing necessary tools and a framework for *Modelling*, *Planning* and *Controlling* cloud applications.

Figure 3 illustrates SeaClouds initial architecture. The **Planner** module will implement SeaClouds planning policy to orchestrate the multi-cloud deployment of application modules. The Planner will take the input data provided by SeaClouds users, who will specify the application modules to be deployed, as well as desired QoS properties of the whole application and/or QoS and technology requirements for individual application modules. The Planner will determine the best offer for each module (by using the **Discovery API**) and will determine how to orchestrate the modules across the selected cloud platforms (viz., *orchestration specification*).

The **Controller** module will implement the multi-cloud deployment of application modules and the monitoring policy. In particular, the **Multi-Cloud Deployer** component will input the *orchestration specification* generated by the Planner, and deploy (by exploiting the **Multi-Cloud Deployment API**) the application modules on the specified clouds. The **Monitor** component will be in charge of monitoring (by exploiting the **Monitoring API**) that application

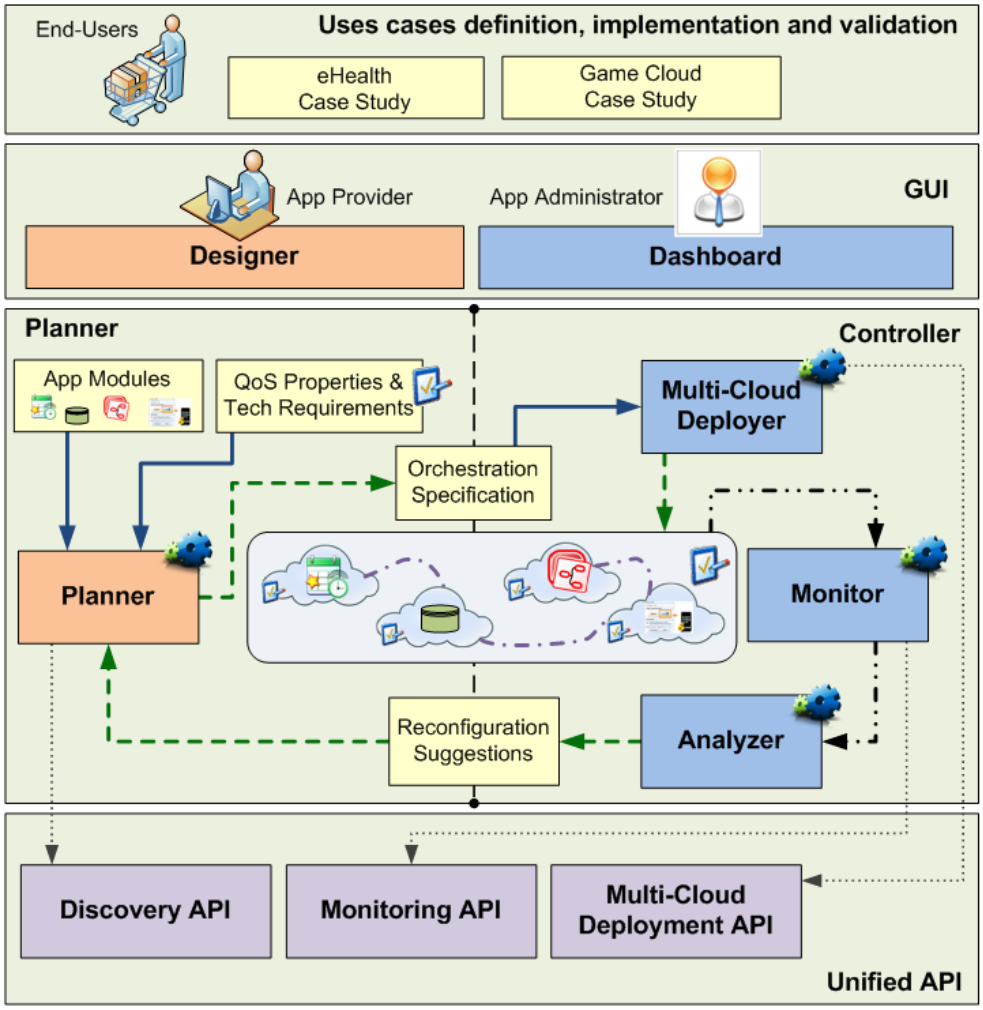


Figure 3: SeaClouds initial architecture.

(module) requirements are not violated. If the monitoring component detects (QoS or technological) requirement violations, then it will trigger the **Analyzer** component. The latter will generate (if needed) *reconfiguration suggestions* and pass them to the Planner in order to trigger the generation of a new orchestration plan. Users will exploit the Graphical User Interface (**GUI**) module to interact with SeaClouds. The GUI module will contain a **Designer** component to allow users to provide SeaClouds input data (viz., application modules, QoS and technological requirements). The GUI will also contain a **Dashboard** component to let users easily manage and visualize the state of the multi-cloud deployed applications.

4 CONCLUSIONS AND PERSPECTIVES

In this paper, we have presented context, motivations and objectives of the SeaClouds project (www.seaclouds-project.eu). The project aims at providing an open source framework to address the problem of deploying, managing and reconfiguring complex applications over multiple and heterogeneous clouds. The SeaClouds approach works towards achieving "Agility After Deployment" by tackling the problem from the service orchestration perspective. A complex application, which consists of modules and (technological and QoS) requirements, is provided as input to the SeaClouds planner. The latter generates an orchestration by assigning (groups of) modules to different PaaS platforms. The orchestration is then deployed and monitored according to standard metrics. If requirements are violated, then the Sea-

Clouds analyzer will generate reconfiguration information, which leverages the creation of a different orchestration of the application.

The expected results of the SeaClouds project can be summarized as follows:

- *Orchestration of Cloud Services*. High-level orchestration specifications will be used to describe the topology and behavior of the application distributed across different platforms.
- *Monitoring Process*. SeaClouds will monitor QoS requirements to determine whether/which modules should be migrated.
- *Traceability of QoS Violations*. Monitoring will have to keep track of possible requirement violations and their causes. This is needed for the generation of reconfiguration suggestions.
- *Reconfiguration Mechanism*. SeaClouds will feature flexible reconfiguration by supporting the migration of application modules across different PaaS platforms.
- *Unified Management API for Cloud Platforms*. SeaClouds will facilitate the access and the administration of both public and private cloud providers by providing multi-cloud management tools as well as offering cloud providers and consumers a REST-based approach to cloud-based application management.
- *Dashboard for Administration of Services Distributed across Multiple PaaS Platforms*. SeaClouds will provide a graphical interface to visualize and manage the current configuration of the deployed services.
- *Contribute to the Development of Cloud Standards*. SeaClouds will provide feedback to CAMP by proposing new functionalities and implementing a CAMP-compliant interface. It will also contribute to the standardization effort of TOSCA, by employing TOSCA as an orchestration modeling language.

REFERENCES

A. Brogi, J. Camara, C. Canal, J, Cubo and E. Pimentel (2007). Dynamic contextual adaptation. In *Proc. of Fifth International Workshop on the Foundations of Coordination Languages and Software Architectures 2006 (FOCLASA'06), vol. 175, no. 2*, pages 81–95. Elsevier.

C. Canal, P. P. and Salaun, G. (2008). Model-Based Adaptation of Behavioural Mismatching Components. *IEEE Transactions on Software Engineering*, 34(4):546–563.

Guillén, J., Miranda, J., Murillo, J. M., and Canal, C. (2013). A service-oriented framework for developing cross cloud migratable software. *Journal of Systems and Software*, 86(9):2294–2308.

H. Nezhad, G.Y. Xu and B. Benatallah (2010). Protocol-aware matching of web service interfaces for adapter development. In *Proc. of 19th International Conference on World Wide Web 2010 (WWW'10)*, pages 731–740. ACM.

J. Cámara, J.A. Martín, G. Salaün, J. Cubo, M. Ouederni, C. Canal and E. Pimentel (2009). Itaca: An integrated toolbox for the automatic composition and adaptation of web services. In *Proc. of International Conference on Software Engineering 2009 (ICSE'09)*, pages 627–630. IEEE Computer Society Press.

J. Cubo and E. Pimentel (2011). Damasco: A framework for the automatic composition of component-based and service-oriented architectures. In *I. Crnkovic, V. Gruhn, M. Book (editors), European Conference on Software Architecture 2011 (ECSA'11), LNCS 6903*, pages 388–404. Springer-Verlag.

M. Armbrust et al. (2010). A view of cloud computing. *Commun. ACM*, (4).

N. Loutas et al (2011). D1.1 Requirements Analysis Report. Cloud4SOA Project Deliverable, http://www.cloud4soa.eu/.

OASIS (2012a). CAMP 1.0 (Cloud Application Management for Platforms), Version 1.0. http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.html/.

OASIS (2012b). TOSCA 1.0 (Topology and Orchestration Specification for Cloud Applications), Version 1.0. http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.pdf.

Parameswaran, A. and Chaddha, A. (2012). Cloud Interoperability and Standardization. *SETLabs Briefings - Infosys*, 7(7):19–26.

AUTHOR INDEX

Abdelbaki, N. 671
Abran, A. 375
Afanasiev, A. 411
Alfieri, G. 299
Alonso, J. 737
Alvarez, P. 551
Amme, W. 53
Andelfinger, U. 395
Andreolini, M. 341
Andrikopoulos, V. 5, 276, 308
April, A. 375
Aral, A. 527
Armendáriz-Iñigo, J. 329
Arrieta, L. 714, 737
Avila, S. 257
Aydin, M. 604
Azaiez, M. 135
Azraoui, M. 489
Ba, C. 268
Balboni, A. 341
Balraj, K. 714
Belalem, G. 118
Bellakhal, R. 315
Benedictis, A. 752
Benfenatki, H. 95
Benghabrit, W. 489
Benharkat, N. 95
Benmerar, T. 211
Benslimane, S. 170
Benyi, A. 519
Bernsmed, K. 489
Binz, T. 499, 559
Bitsaki, M. 308
Blanca, M. 387
Bonacquisto, P. 38
Brandic, I. 691, 722
Brataas, G. 746
Breitenbücher, U. 499, 559
Breton, V. 199
Brogi, A. 758
Brunelière, H. 737
Bui, T. 199
Caminero, B. 129
Carrasco, J. 758
Carrión, C. 129
Casola, V. 299, 752
Celesti, A. 584
Cerbo, F. 551
Cervera, S. 513
Chainbi, W. 135, 315
Chang, D. 179
Chauvel, F. 746
Chehab, A. 467
Cherubini, D. 610, 616
Chihi, H. 135
Christmann, C. 88
Ciuffoletti, A. 143
Clemente-Castelló, F. 513
Costa, U. 268
Crăciun, C. 598
Cubo, J. 758
Cucinotta, T. 610, 616
Cui, X. 15
Dai, T. 221
D'Andria, F. 758
David, S. 451
Diaz, R. 679
Djemame, K. 257
Domaschka, J. 569
Dombi, J. 519
Dustdar, S. 221
Ecuyer, A. 551
El-Gazzar, R. 78
Elkhiyaoui, K. 489
Emeakaroha, V. 476
Evangelinou, A. 714
Evenou, P. 451
Falkner, J. 88
Farokhi, S. 722
Fatema, K. 476
Fazio, M. 584
Fehér, P. 543
Fernandez, H. 352
Ferrari, M. 268
Ferré, R. 268
Ferry, N. 746
Finance, B. 361
Fowley, F. 60
Franke, M. 652
Fuchs, A. 242

AUTHOR INDEX (CONT.)

Gabbrielli, M. 425
Galante, N. 714
Ghédira, K. 315
Ghodous, P. 95
Giallorenzo, S. 425
Giammatteo, G. 714
Gillam, L. 70
Grall, H. 489
Gu, J. 403
Guidi, C. 425
Gürgens, S. 242
Hans, R. 158
Haque, R. 361
Healy, P. 476
Heinze, T. 53
Heinzlreiter, P. 387
Hohenstein, U. 417
Horch, A. 88
Horn, G. 569
Hribernik, K. 652
Ibrahim, A. 758
Itani, W. 467
Jacob, J. 604
Jaeger, M. 417
Jansen, M. 645
Jeffery, K. 569
Jiménez-Ruano, V. 329
Jrad, F. 186, 691, 722
Jul, E. 616
Junior, F. 151
Kachouane, M. 211
Kantere, V. 439
Karagiannis, G. 633, 639
Karastoyanova, D. 276, 308
Karimzadeh, M. 633, 639
Karlsson, J. 387
Kayssi, A. 467
Kella, A. 118
Kertesz, A. 519
Kett, H. 88
Kevani, E. 714
Khalfi, M. 170
Klein, K. 652
Kõ, A. 543
Kopaneli, A. 714
Kopp, O. 287, 499, 559
Kousiouris, G. 622, 714
Kreb, R. 433
Kudoh, T. 103
Kyriazis, D. 714
Lampe, U. 158, 193, 578
Langer, P. 737
Leymann, F. 5, 287, 499, 559
Li, F. 221
Lockner, N. 230
Loesch, M. 433
Lugones, D. 610
Lynn, T. 476
Manieri, A. 164
Mayo, R. 513
Medernach, E. 199
Melhem, R. 15
Menychtas, A. 714, 737
Mills, B. 15
Minor, M. 323
Miranda, J. 645
Mitterecker, A. 387
Modica, G. 38
Mont, M. 27
Morrison, J. 476
Moser, S. 53
Munir, K. 27
Muñoz, V. 679
Murillo, J. 645
Musicante, M. 268
Nakada, H. 103
Navarro, J. 329
Nguyen, D. 361
Nguyen, H. 199
Nikolaou, C. 308
Nishida, S. 457
Normand, N. 451
O'Loughlin, J. 70
Oberle, K. 610
Oliveira, A. 489
Önen, M. 489
Orue-Echevarria, L. 714, 737
Östberg, P. 230
Oulebsir-Boumghar, F. 211
Ovatman, T. 527, 535

AUTHOR INDEX (CONT.)

Pahl, C. 60, 60
Panarello, A. 584
Parrein, B. 451
Patni, A. 590
Pauly, M. 158
Peralta, V. 268
Perkins, J. 387
Pertin, D. 451
Petcu, D. 598
Petrakis, E. 112
Petralia, G. 38
Pierre, G. 352
Pietri, M. 341
Pimentel, E. 758
Pino, L. 242
Preventis, A. 112
Proaño, J. 129
Psycharaki, A. 308
Puliafito, A. 584
Qanbari, S. 221
Quintana-Ortí, E. 513
Radchenko, G. 445
Rahmouni, H. 27
Rak, M. 299, 752
Ramo, A. 679
Ranea, J. 387
Reinhardt, M. 395
Robert, S. 268
Rocha, T. 151
Rossini, A. 746
Royer, J. 489
Ruehl, S. 395
Rybina, K. 590
Sadovykh, A. 703
Sáez, S. 5, 276, 308
Saidah, A. 671
Sancho-Asensio, A. 329
Satria, T. 633, 639
Savchenko, D. 445
Schill, A. 590
Schubert, L. 569
Schuller, D. 193
Schulte-Zurhausen, E. 323
Schütte, A. 395
Seçinti, C. 535
Sellami, M. 489
Shinkawa, Y. 457
Siebenhaar, M. 193
Silva, C. 95
Silva, M. 703
Solberg, A. 746
Soldani, J. 758
Solomonides, T. 27
Sotiriadis, S. 112
Spanoudakis, G. 242
Stampoltas, C. 714
Steinmetz, R. 158, 193, 578
Stratan, C. 352
Strauch, S. 5
Stravoskoufos, K. 112
Streit, A. 186, 691, 722
Stuer, P. 714
Sturm, T. 186
Sukhoroslov, O. 411
Szabó, Z. 543
Taher, Y. 361
Takano, R. 103
Takefusa, A. 103
Tao, J. 691
Thoben, K. 652
Tomarchio, O. 38
Torreño, Ó. 387
Tosi, S. 341
Trabelsi, S. 551
Trelles, O. 387
Tsaregorodtsev, A. 679
Ulbricht, S. 53
Valtulina, L. 639
Varvarigou, T. 714
Verclas, S. 395
Vernet, D. 329
Villalpando, L. 375
Villano, U. 752
Villari, M. 584
Wagner, S. 287
Wehle, H. 53
Weisbecker, A. 88
Weiß, A. 276
Wellsandt, S. 652
Wenge, O. 578

AUTHOR INDEX (CONT.)

Wettinger, J. 559
Wieland, M. 499
Williams, K. 205
Wimmer, M. 737
Wright, P. 164
Yanagita, S. 103
Yang, Y. 179
Yokoyama, S. 661
Yoshioka, N. 661
Zaballos, A. 329
Zhang, C. 403
Zhang, H. 179
Zhang, L. 60, 60
Zhang, W. 403
Zimmermann, M. 559
Znati, T. 15

Proceedings of CLOSER 2014
4th International Conference on Cloud Computing and Services Science
ISBN: 978-989-758-019-2 | www.closer.scitevents.org



Copyright 2014 SCITEPRESS
Science and Technology Publications
All Rights Reserved

INSTICC IS MEMBER OF:





LOGISTICS PARTNER:



PROCEEDINGS WILL BE SUBMITTED FOR INDEXATION BY:









