

A Cloud-Agnostic Framework to Enable Cost-Aware Scheduling of Applications in a Multi-Cloud Environment

Fan Jiang¹, Kyle Ferriter², and Claris Castillo³

¹Department of Computer Science, University of North Carolina at Chapel Hill

²Department of Computer Science, North Carolina State University

³Renaissance Computing Institute (RENCI), University of North Carolina at Chapel Hill

Abstract—We have witnessed a surge in both the big data applications being hosted by an assortment of cloud vendors, and in the astronomical amount of data they produce and consume on a daily basis. Traditional cluster computing frameworks can hardly cope with the unprecedented data volume and the geo-distributed, cross-cloud data distribution due to their limited scalability and adaptability across the heterogeneous clouds. Moreover, running data-intensive applications across clouds at will is extremely cost-inefficient and likely to incur outrageous expenses. Hence, we introduce our cloud-agnostic system PIVOT with the novel *cost-aware* scheduling algorithm, which enables data-intensive applications to run and scale across clouds instantly in a cost-efficient manner. We evaluate our system and scheduling algorithm extensively with the Alibaba production cluster trace, as well as real-world big data applications on a 100-node deployment across 11 regions (31 availability zones) on AWS and GCP. The experimental results show that PIVOT achieves up to 90.8% saving in expense for VM subscription and 99.2% for egress network traffic compared to the state-of-the-art baselines. Notably, the *cost-aware* scheduling also achieves over 4x speedup in data transfers for data-intensive applications.

I. INTRODUCTION

As the demand for cloud computing continues to grow, a number of cloud-based platforms have been developed to address the increasing needs for data analysis and processing in the scientific and commercial sectors. As a result, valuable datasets are stored in the cloud across various geographical regions and cloud platforms. For instance, recently the National Institute of Health (NIH) established the Commons Cloud Pilot project¹ to broaden access to high-valued biomedical datasets across Amazon Web Services (AWS) and Google Cloud Platform (GCP). Similarly, the National Oceanic and Atmospheric Administration (NOAA) provide access to real-time and archival weather radar data in AWS for weather data analysis and ². In spite of these advances, scientists face steep challenges to be able to use these valuable datasets in ways that will enable new research collaborations.

The distribution of data across cloud providers imposes three major challenges for data-driven analysis and applications: 1) The heterogeneity in resources, networking, APIs and runtime among clouds prevents applications from scaling out across clouds, leveraging their unique capabilities and offerings (e.g., Google Genomics³) and executing close to data. In

other words, a user either runs an application on one cloud or the other but not across; 2) Without the ability to scale across clouds in a seamless fashion, the applications trigger cross-region and cross-cloud data movements, which can hinder application performance due to low network throughput over the wide-area network (WAN). That is, to analyze combined data hosted in two clouds, data must be moved from one cloud into the other; 3) Since commercial clouds monetize egress network traffic, *i.e.*, traffic resulting from moving data outside a cloud region, cross-cloud data analysis can incur prohibitive monetary cost for large datasets.

Therefore, minimizing the *financial* burden that hosting data and running computation in the cloud imposes on the user is critical to promote the adoption of cloud computing in the scientific community. In our experiments we show the implications of making compute placement decisions that are oblivious to this cost. Application efficiency is another aspect that deters scientists from migrating their data-intensive applications into cloud environments. Workflow engines such as Toil [34] and Galaxy⁴ are commonly used in the scientific community for data analysis. These solutions, albeit providing predictable performance in controlled campus infrastructure, are not designed to perform well on distributed cloud environments; their scheduling capabilities are oblivious to both the network infrastructure that connects compute and storage resources as well as data locality.

To address the aforementioned challenges we propose PIVOT, a cloud-agnostic framework that creates an abstraction layer over compute and storage resources distributed across cloud providers to create the illusion of one very large computer, thus hiding the complexity and heterogeneity of individual providers, services, and offerings. These resources are presented to users through a unified API by which data processing applications such as workflows can be executed and scaled across resources independently of where data is located.

To achieve this, PIVOT decouples the abstraction and management of data and compute, and builds on advanced middleware mechanisms that orchestrate how these resources are utilized *jointly* to provide applications with acceptable performance across multiple clouds, while taking into account financial cost on behalf of the users. For resource and application orchestration we develop a new scheduling algorithm that

¹NIH Data Commons Pilot: <https://commonfund.nih.gov/commons>

²NOAA Open Data: <https://registry.opendata.aws/noaa-nexrad/>

³Google Genomics: <https://cloud.google.com/genomics/>

⁴Galaxy: <https://usegalaxy.org/>

factors egress network traffic cost into scenarios where data is distributed across multiple clouds and/or cloud regions. The fine-grained resource model for cloud topology in combination with this *cost-aware* scheduling algorithm allows PIVOT to place computation close to the data –in a cloud-agnostic manner– in order to minimize data movement and egress network traffic cost. We deployed PIVOT across AWS and GCP and demonstrated its effectiveness through experiments with Alibaba cluster trace⁵ as well as big data applications. Our results show that PIVOT can minimize egress network traffic cost (up to 99.2%) and improve network throughput over a factor of 4x as compared to using traditional cost- and data-locality-oblivious scheduling strategies.

The key contributions of our work are manifold: 1) We have pioneered a *cloud-agnostic* framework and architecture that creates the illusion of one single large computer to users thus hiding the heterogeneity and complexity of cloud platforms. 2) We have developed middleware that orchestrates these resources in a unified manner and enables the deployment of applications across multiple clouds without imposing additional burden on the user. 3) We have developed a novel scheduling algorithm aware of cost and data locality to discerns between cloud regions and providers to place applications in proximity to the data, thus minimizing data movement, improving performance, and reducing financial cost. 4) We performed an empirical and simulation-based evaluation of data-intensive applications in cross-cloud environments respecting cost and throughput. To the best of our knowledge this is a first of a kind analysis on cross-cloud environments. 5) We have developed an open-source implementation of this framework that is available to the community to deploy their own software stack.

The rest of the paper is organized as follows: We introduce the architecture and key components of PIVOT in Section II. The cost-aware scheduling algorithm is detailed in Section III and extensively evaluated in Section IV; we review relevant related research works in Section IV; we conclude the paper and propose future works in Section VII.

II. PIVOT

In PIVOT, we seek to support cross-cloud, cross-region execution of data-intensive applications while hiding the complexity of the underlying heterogeneous systems and respecting cost and performance requirements of the application. We achieve this through the introduction of a *cloud-agnostic framework* that abstracts virtual infrastructure provided by IaaS across clouds, and a versatile *two-level scheduling* approach that minimizes cost by optimizing for data locality. In this section, we describe the architectural components of PIVOT in detail.

A. PIVOT Architecture

The PIVOT architecture (Fig. 1) is designed with *cloud agnosticism* at its heart. At the bottom, we build a *cross-cloud virtual infrastructure* that enables seamless network

connectivity among geo-distributed regions and clouds. On top of it, we build the *abstraction layer* that abstracts computing and storage resources and transforms them into standard units uniformly consumable by applications. Finally, a *scheduling layer* controls task placement while considering both system-wide goals and application-level scheduling strategies.

The *cross-cloud virtual infrastructure* unifies resources provisioned in different regions and clouds in a standard fashion, creating an illusion of a single pool and orchestrating all aspects of virtual resource management and communication across cloud regions. This layer is built on top of core cloud IaaS services commonly provided by major cloud vendors including VM instances, persistent storage, virtual private cloud (VPC) and virtual private networking (VPN). In every cloud region, two types of hosts are provisioned – *computing hosts* for computation and *storage hosts* for data persistence. The hosts are interconnected through a *supernet* spanning over all the regions and clouds. We construct the *supernet* with VPCs and VPN tunnels; there is one VPC in every region peered up with other regions using *VPC peering*; regions in different clouds are connected via VPN tunnels. Every VPC is allocated with a unique, contiguous block of private IP addresses to ensure that every host in the network is uniquely addressed. The *supernet* greatly simplifies both the communication among hosts by connecting them within the same network, and the routing among application tasks.

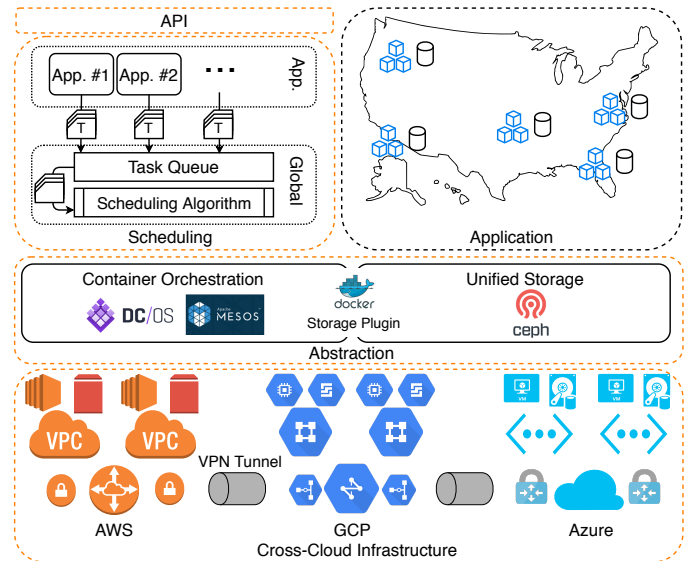


Fig. 1: PIVOT Architecture

The *abstraction layer* ensures that applications can run, scale, and access data among all the regions and clouds seamlessly. It consists of two components – *container orchestration* (e.g., Docker⁶) for application deployment, configuration and provisioning, and *unified storage* for high accessibility to data. Containerization allows us to encapsulate application logic in a self-contained manner and makes applications portable

⁵Alibaba Cluster Trace: <https://github.com/alibaba/clusterdata>

⁶Docker: <https://www.docker.com/>

among heterogeneous environments. For high accessibility to data, we deploy a unified distributed storage system across all clouds; this approach makes data and persistent storage accessible by applications anywhere in the system. Every task in an application is executed within a *container* on the *computing host*, and ingests/persists data from/to one or many persistent *volumes* located on the *storage hosts* if needed. Additionally, we have developed a *Docker volume plugin*⁷ that allows *volumes* to be mounted as POSIX-compliant file systems within *containers* for easy data access. Moreover, by leveraging the *node affinity* feature available in common container orchestration [7] [12] and distributed storage [3] systems, the *abstraction layer* is able to leverage *placement hints* provided by the upper-layer components for custom placement of *containers* and *volumes* on specific hosts, availability zones (AZs), regions or clouds. This capability is critical for the *scheduling layer* to manipulate the placement of tasks and associated datasets, e.g., pinning a deep learning application to a GPU-enabled host. In our prototype implementation, we adopt DC/OS [6] and Ceph [2] for container orchestration and distributed data storage, respectively. However, it is worth noting that PIVOT's architecture is generic and thus container orchestration and distributed storage implementations can be substituted by alternative technologies such as Kubernetes [11] and GlusterFS [9], accordingly.

B. Two-level scheduling framework

To accommodate the diverse scheduling goals of applications, we devise a modular, and extensible *two-level scheduling framework* that allows for custom scheduling algorithms thus following separation-of-concerns design principle. The framework consists of a single *global scheduler* and an extensible array of *application schedulers*.

The *application scheduler* is responsible for task scheduling within an application. Traditionally, a *application scheduler* is built-in and provisioned with the application; it schedules tasks so to achieve application-level goals. For instance, for workflows the default *application scheduler* implements the classic dependency resolution algorithm that resolves dependencies among tasks. In PIVOT, a custom scheduler is developed by implementing and installing the abstract *application scheduler* as a *framework module*. Newly installed schedulers can be extended and reused by other applications with every application running its own instance of the *scheduler*. The scheduler operates in a phased fashion; it monitors the state of tasks and queries the system-wide resource availability periodically throughout the lifetime of the application. Scheduling decisions at the application-level are communicated to the global-scheduler for system-wide scheduling considerations. The *global scheduler* has a built-in *task queue* where tasks submitted by the *application schedulers* are queued for scheduling and final dispatching. The *global scheduler* also runs periodically, invoking the global scheduling algorithm to

schedule tasks in the *task queue* and then clearing the queue at each phase. The scheduling algorithm makes placement and resource allocation decisions based on the task specifications and the resources available in the system. The scheduled *containerized* tasks are sent to the container orchestrator in the *abstraction layer* for provisioning, configuration, and execution. When the minimum requirements of a task cannot be fulfilled due to issues like resource shortage, it is put back into the *task queue* for the next scheduling iteration and the *global scheduler* increments a counter of scheduling failures for the task. Once the counter has reached a configurable threshold, the *global scheduler* notifies the corresponding *application scheduler* to handle the scheduling failure.

Our proposed two-level design of the scheduling framework enhances the versatility of the *scheduling layer* by enabling off-loading of application-specific scheduling logic to the upper-level scheduler.

III. COST-AWARE SCHEDULING ALGORITHM

We are concerned with lowering the financial burden resulting from high egress expenses incurred by moving data in/out of cloud regions while improving application efficiency. To strike a balance between the factors we propose a *cost-aware scheduling heuristic* which aims at **saving cloud expenditures** for geo-distributed applications running on PIVOT while **improving data transfer efficiency** in a *best-effort* manner. We implement our algorithm as a *global scheduler* in PIVOT.

A. Problem definition

Below we introduce a formal definition of the problem. We consider a workload model wherein an application consists of potentially multiple tasks, and these tasks can have dependencies that drive application-level scheduling decisions.

Tasks T are placed and executed on geo-distributed virtual machine (VM) hosts H provisioned across different regions and clouds. Note that herein we use VM and host interchangeably. Each task $\tau \in T$ has a resource demand d_τ and each host $\eta \in H$ is associated with R_η amount of resources available. The vectors d_τ and R_η are 4-dimensional, representing the resource demand and capacity of CPUs, RAM, disk and GPUs, respectively. Here we use K to represent the resource dimensionalities. Following the IaaS business model in the cloud, every running host η incurs a VM subscription cost of i_η dollars per hour, and we assume VMs are turned off timely when idle. We use a bit array $u_{|H|}$ to indicate whether host η is in use. Additionally, moving data between hosts η and η' results in egress cost $e_{\eta,\eta'}$ per gigabyte. Lastly, we use a binary matrix $P_{|T| \times |H|}$ to represent the placement of the tasks on the hosts. The problem is formulated as below.

The objective is to minimize the total expense for VM subscription (1a) and egress network traffic (1b). The constraint (1c) is the capacity constraint that limits the total resource demand of tasks placed on a host to the resource capacity of the host in any dimension. The constraint (1d) ensures that a task can only be placed on one host at any point of time. The constraints (1e) and (1f) indicate that U

⁷CephFS Docker volume plugin: <https://github.com/dcvan24/cephfs-docker-volume-plugin>

and P are binary, respectively. We recognize the problem as a variant of the **multi-dimensional vector bin packing problem (MDVPP)** [19] [24], which is proven NP-hard [24]. Hence, we propose a **cost-aware heuristic algorithm** in this paper to tackle the problem.

$$\min \quad \alpha \cdot \sum_{\eta \in H} i_{\eta} \cdot u_{\eta} \quad (1a)$$

$$+ \beta \cdot \sum_{\tau \in T} \sum_{\tau' \in T} \sum_{\eta \in H} \sum_{\eta' \in H} (P_{\tau\eta} + P_{\tau'\eta'}) \cdot e_{\eta\eta'} \quad (1b)$$

$$\text{s.t.} \quad \sum_{\tau \in T} d_{\tau k} \cdot P_{\tau\eta} \leq R_{\eta k}, \quad \forall \eta \in H, k \in K, \quad (1c)$$

$$\sum_{\eta \in H} P_{\tau\eta} = 1, \quad \forall \tau \in T, \quad (1d)$$

$$u_{\eta} \in \{0, 1\} \quad \forall \eta \in H, \quad (1e)$$

$$P_{\tau\eta} \in \{0, 1\} \quad \forall \tau \in T, \forall \eta \in H \quad (1f)$$

B. Algorithm design

As identified in Section III-A, cloud expenditures are mainly comprised of expenses for *VM subscription* and *egress network traffic*, caused primarily by host **resource underutilization** and excessive **egress data transfers**. Furthermore, since network bandwidth is typically constrained among regions and clouds, egress data transfers tend to exhibit low efficiency and high delay. Ideally, consolidation of tasks into the least number of hosts and egress data transfers can both avoid unnecessary expenses and improve efficiency.

We realize that the problem of task consolidation is analogous to the known **vector bin packing (VBP)** problem, in which the tasks and hosts are *items* and *bins*, respectively. Greedy approximation algorithms such as first-fit (FF) [29], best-fit (BF) [16], and their variants, are proven effective in approximating the optimum for the problem. We refer to the family of FF and BF algorithms as VBP algorithms. We refer the reader to [22] [29] for an in-depth analysis of VBP algorithms.

However, whereas the bins in the classic VBP problem are presumably homogeneous, hosts in PIVOT are distinguished by **egress cost**, **network bandwidth** and **data locality**. More specifically, it is preferable for a task to be placed on a host with its input data for data locality, which we refer to as the *anchor*. If the *anchor* is unavailable, another host is selected based on a function of its egress cost and available network bandwidth to the *anchor*. Intuitively, hosts with lower egress cost and higher bandwidth are preferred for potentially better cost saving and data transfer efficiency.

Following this analysis, we design our *cost-aware scheduling algorithm* as follows (Fig. 2): 1) the initial VBP problem is divided into sub-problems based on the data locality requirements of the tasks; 2) an *anchor* is selected for every sub-problem, while tasks and hosts are sorted *w.r.t.* the selected *anchor*; 3) the classic FF algorithm is used to pack tasks into

the prioritized hosts. In the rest of this section, we describe each step of the algorithm in detail.

```

Func schedule( $T, H$ ):
1   $G \leftarrow \text{groupTasks}(T)$ 
2  for  $g \in G$  do
3     $g \leftarrow \text{sortTasks}(g)$ 
4     $\text{hosts} \leftarrow \text{sortHosts}(H, \theta)$ 
5     $\text{FirstFit}(g, \text{hosts})$ 
6  return  $\text{tasks}$ 

```

Fig. 2: Cost-aware scheduling

C. Task grouping and data locality inference

The first step is to divide the problem based on data locality requirements. In effect, tasks are grouped by their *anchors*. Note that although PIVOT allows users to provide fine-grained data placement information, such information is seldom available and has to be inferred to retain data locality. We adopt a **data locality inference** approach to deduce data locality requirements from implicit observations in task specifications. We leverage two pieces of information – the *application locality* and *dependencies* encoded in application specifications. Intuitively, we assume tasks of the same application are likely to share data among each other due to the inter-operations and communications. Furthermore, co-dependent tasks, *e.g.*, workflows, tend to ingest data from tasks upstream since they may use the output data of the precedent tasks as input data. Hence, it is intuitive to co-locate tasks with their peer (application-locality) or precedent (dependencies) tasks for data locality. As we show later in Section IV this measure has a significant impact on data-intensive applications such as workflows.

```

Func groupTasks( $T$ ):
1   $G \leftarrow \{\}, \theta \leftarrow \emptyset$ 
2  for  $\tau \in T$  do
3    if  $\tau.\text{dataPlacement} \neq \emptyset$  then
4       $\theta \leftarrow \tau.\text{dataPlacement}$ 
5    else if  $\tau.\text{dependencies} \neq \emptyset$  then
6       $\theta \leftarrow \text{Host } \eta \text{ where most of } \tau\text{'s predecessors are placed}$ 
7    else if  $\tau.\text{application has the anchor set}$  then
8       $\theta \leftarrow \text{Anchor of } \tau.\text{application}$ 
9    else
10      $\theta \leftarrow \text{Random host}$ 
11     Set  $\theta$  as the anchor of  $\tau.\text{application}$ 
12    $g_{\theta} \leftarrow g_{\theta} \cup \{\tau\}, G \leftarrow G \cup \{g_{\theta}\}$ 
13 return  $G$ 

```

Fig. 3: Task grouping

Fig. 3 illustrates the steps of *task grouping*. First, tasks with explicit information about placement of input data use the location of input data as the *anchor* (Line 4); tasks with dependencies select the host wherein most predecessors are placed as the *anchor*. Following, tasks are grouped by their *anchor* (Line 9). Tasks without an *anchor* are grouped based on the application they belong to and each such group is assigned a random *anchor* (Line 7-8) to ensure the peer tasks are co-located to meet application-locality requirements.

D. Task ordering

For each group formed in the previous step, the algorithm sorts the tasks in decreasing order of their resource demand [22] which consequently results in *lower VM subscription cost* for the user.

To sort the tasks, we model the resource demand of each task *geometrically* as an *Euclidean vector* and calculate its L_2 -norm ($\|d_\tau\|_2$) as the *size* of the task. We note that this is considered one of the most effective geometric approaches in the realm of VBP problems [29].

E. Host ordering

The host ordering is critical since it captures important distinctions among hosts that have measurable impact on the quality of the scheduling decisions. We introduce a *host scoring function* as below to facilitate the host ordering quantitatively.

$$Score(\eta, \theta) = \frac{\|R_\eta\|_2 \cdot (b_{\eta\theta} + b_{\theta\eta})}{C_{\eta\theta} + C_{\theta\eta} + \epsilon}$$

We factor in bidirectional network bandwidth (b) and egress cost (C) between the host η and the *anchor* (θ) – the score is positively correlated with the network bandwidth but negatively with the egress cost. Likewise, we model the resource capacity on host η (R_η) as a 4-dimensional Euclidean vector and use its length as the host *capacity*, which is also positively correlated with the host score. The hosts are sorted in the decreasing order of the host score given by the function. Effectively, the algorithm prioritizes hosts with larger *capacity*, more abundant bandwidth but less egress cost from/to the *anchor* for task placement.

In practice, we estimate the network bandwidth regularly using *Assolo* [25] – a non-intrusive bandwidth estimation tool. It imposes limited impact on running applications and greatly saves the egress cost for bandwidth measurements as compared to most intrusive alternatives (e.g., *iperf*⁸).

F. Vector bin packing

To pack tasks into the sorted hosts we use the classic *FF algorithm*. By design, the algorithm attempts to place each task onto the first host it can fit in (first-fit), and only consider the next host (open a new *bin*) when no fit is found. With the sorted host array, tasks are packed onto high-score hosts first and proceed in decreasing order of the host score.

IV. EVALUATION

To evaluate the effectiveness and feasibility of our approach in PIVOT, we deploy PIVOT across AWS and GCP. We drive the experiments with simulations on Alibaba production cluster trace and real-world big data applications to investigate the system and proposed algorithm in depth. For performance metrics, we focus on the cost savings in VM subscription and egress network traffic, but also observe the efficiency of application executions and data transfers.

⁸iperf3: <http://software.es.net/iperf/>

A. Experiment setup

1) *Alibaba cluster trace and simulation*: The Alibaba cluster trace provides an 8-day collection of batch job trace in their data centers in 2018. Each job consists of a number of tasks with data dependencies among them specified in the dataset. In this evaluation, we randomly sample a total of 35,000 batch jobs (3,181,620 tasks) from the trace dataset. Each job is run as an application of PIVOT respecting the data dependencies among the tasks. Each task produces and transfers 10–400MB data. The data size is proportional to the RAM demand of each task. We simulate the PIVOT deployment on 600 VMs evenly distributed among 31 AZs of the 11 North America regions⁹ on AWS and GCP. Each VM is configured with 16 CPUs and 128GB RAM (*r5.4xlarge* and alike instance/machine type). The network is constructed based on a two-week network trace collection among regions and clouds in AWS and GCP using *iperf3*. The simulator is written in Python using *Simpy* [14] and available on Github [13].

2) *Big data applications and real deployment*: We also replicate the PIVOT deployment in the real world to evaluate the system and algorithm in depth with big data applications. Different from the simulation, each VM is configured with 4 Skylake CPUs, 8GB RAM and 150GB disk space (*c5.xlarge* and alike). In this deployment, there are 100 VMs evenly distributed across the 31 AZs.

We introduce two featured, real-world use cases - 1) the Hail [10] genomic analysis and 2) the TOPMed alignment workflow [5]. Both workloads exhibit high-level of parallelism, computation intensity and data dependencies thus stressing the challenges encountered in the cloud-distributed environments we consider in this work. Hail is an open-source genomic analytical tool running on Spark [1] to enable large-scale genomic analysis; the TOPMed alignment workflow is an example of workflows encoded in Common Workflow Language (CWL) [15] and publicly available at Dockstore [8]. The Hail workload are executed as regular Spark applications atop containerized Spark clusters scheduled and run by PIVOT, in which data processing tasks are executed and the intermediate data is exchanged among distributed, containerized *workers*. The TOPMed workflow used for this experiment consists of parallel data processing tasks with dependencies among each other. The number of parallel tasks varies between 10 – 70. The topology of the dependency graph is a MapReduce structure starting with splitting the input, two intermediate phases processing the chunks, and a final aggregation phase. The high level of parallelism and data dependencies lends itself well to analysis of distributed scheduling algorithms. We have ported both Hail cluster and CWL workflows as applications runnable on PIVOT to serve the biomedical community and enable them to take advantage of the cross-cloud scalability.

3) *Baseline*: In our evaluation we compare our cost-aware to the following baseline algorithms.

⁹us-east-1, us-east-2, us-west-1, us-west-2, ca-central-1 on AWS, and us-east1, us-east4, us-west1, us-west2, us-centrall, northamerica-northeast1 on GCP

- *Opportunistic* is a common scheduling strategy that assigns tasks to hosts with sufficient resources *opportunistically* for high resource utilization in overall as adopted in [26] [18]. In our implementation, the scheduler assigns tasks randomly to the hosts where they can fit in.
- VBP consists of the FF and BF family of algorithms.
- Mesos [26] uses a *resource offer* mechanism that achieves high data locality and scalability within the data center. It is a sophisticated adaptation of the *Opportunistic* algorithm. In the evaluation, we compared our algorithm to Mesos with real applications.

B. Results

1) *Simulation on Alibaba cluster trace*: Fig. 4 compares our algorithm to the baseline algorithms across several aspects. As shown in the figure, the *Opportunistic* strategy performs the worst in cost efficiency since it does neither task packing nor is cost aware. This follows intuition since this scheduling strategy tends to spread out tasks randomly across regions, thus fragmenting resource utilization on the hosts and leading to high cost in host subscription and egress cost. In contrast, the *cost-aware* strategy saves up to 90.8% and 99.2% of the host subscription and egress cost respectively as compared to the *Opportunistic* algorithm. Although saving comparably 92.1% in host subscription cost due to the effective consolidation of tasks into fewer hosts, the VBP only reduces 15.9% of the egress cost. This is mainly because VBP is oblivious to data locality and the egress cost model and blindly scales out applications across regions and clouds, thus causing excessive egress cost. In comparison, by grouping tasks and sorting hosts respecting data locality, *cost-aware* is able to schedule tasks in proximity to their input data and *radially* scales out applications centered around their *anchor*. Effectively, the algorithm favors cost-efficient hosts when placing tasks and therefore achieves the greatest cost saving. As the result of the high level of data locality achieved by the algorithm, *cost-aware* incurs the least application runtime on average.

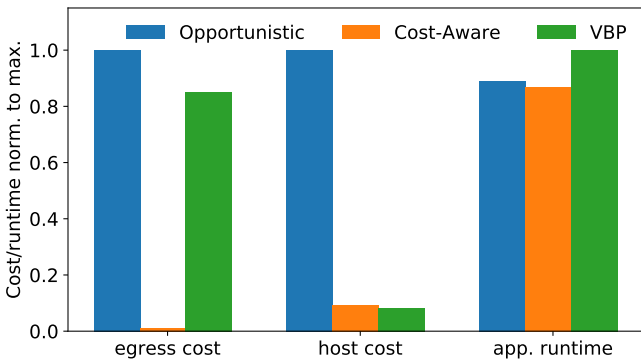


Fig. 4: Comparison of egress cost, host subscription cost, average number of hosts used and application runtime. *cost-aware* saves up to 90.8% and 99.2% of the cost for host subscription and egress traffic.

Interestingly, we observe *cost-aware* also achieves the least average application runtime as compared to the baselines.

In further analysis, we find that the runtime improvement is mostly due to the reduced data transfer time. As illustrated in Fig. 5, *cost-aware* saves up to 82.8% of the data transfer time per task. Notably, VBP performs the worst due to the significant delay caused by network congestion, which represents 85% of the data transfer time. Moreover, we find that the congestion exacerbates when the hosts are upgraded with more CPUs and RAM. This is because the increase of network bandwidth can rarely keep up with the increase of CPUs and RAM during VM upgrade in the cloud. Consequentially, more concurrently running tasks tend to compete for the limited bandwidth and aggravate the network congestion. Hence, we argue that adopting naive bin-packing strategy can be detrimental to application scheduling in the distributed cloud environment. On the other hand, the result also highlights the importance of the awareness of network bandwidth and data locality for the scheduling algorithm to make wise scheduling decisions in the distributed cloud environment. More specifically, without the awareness of network bandwidth and data locality, VBP can place tasks onto hosts with bandwidth-limited network paths to their *anchor* and increase the data transmission delay. To worsen the situation, the task packing further stresses the constrained network path and thus creates *hot spots*, where excessive network traffic overwhelms the congested path. As a consequence, VBP greatly slows down the application execution as shown in Fig. 4. In contrast, since the bandwidth factor outweighs other factors in the host scoring function when the network bandwidth being oversubscribed, the *cost-aware* algorithm is able to avoid the congestion dynamically and trade reasonable resource fragmentation for load balancing in the long run.

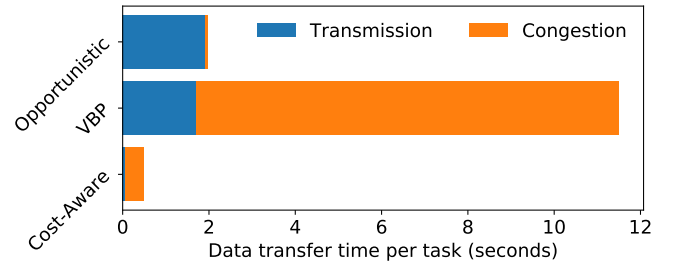


Fig. 5: Average data transfer time per task. *cost-aware* saves up to 82.8% of data transfer time due to strategic selection of fast network path and avoidance of network congestion.

Fig. 6 shows the financial cost as a function of the number of applications running in PIVOT. As observed, *cost-aware* incurs trivial egress cost – it incurs at most \$13.62 as compared to >\$1,000 incurred by the baselines. More importantly, the cost increases at the lowest rate with the increasing system load. The results demonstrate that *cost-aware* is able to limit the footprints of applications within the fewest regions and scales out the applications *strategically* to minimize the egress cost. Furthermore, as the number of applications increases, all the evaluated algorithms inevitably scale out the applications as reflected in the increasing cost for host

subscription. However, `cost-aware` exhibits the comparable efficiency in task packing to VBP – it incurs only 16.3% additional host subscription cost on average for improved load balancing and congestion avoidance as shown in Fig. 5.

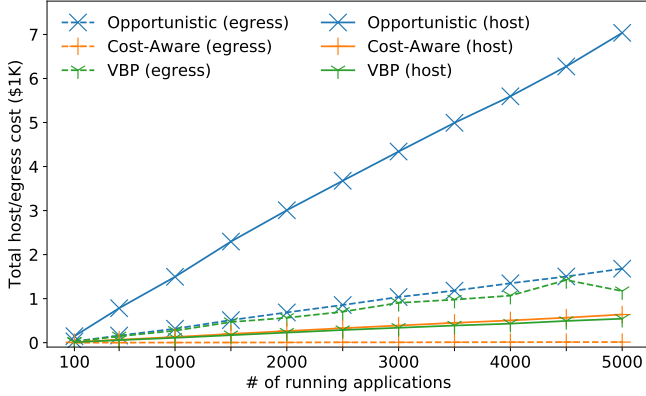


Fig. 6: Variation of cloud expenses with increasing number of running applications. Despite the lowest egress and host subscription cost, `cost-aware` achieves mildest cost increase as applications scale up.

2) *The Hail cluster*: A Spark cluster consists of a set of *worker processes* (workers) that execute data processing jobs. In PIVOT, Hail *workers* are considered long-running tasks in our workload model. The management of the Hail workload is managed by the Spark framework and therefore is completely *opaque* to PIVOT. Therefore, our `cost-aware` algorithm only takes into account information about the application locality, *i.e.*, all Hail tasks (workers) are grouped, when making scheduling decisions. To compare the effectiveness of our algorithm against VBP and Mesos, we launch Hail clusters with 10 – 100 *workers*; following we discuss the results of this experiment.

Fig. 7a illustrates the placement distribution of *workers* for Hail clusters of varying sizes. As observed, the `cost-aware` algorithm co-locates the majority of the *workers* within the same region and cloud (50.8% and 90.3%, respectively).

Additionally, in Fig. 8 we show a cost-throughput trade off analysis for all three scheduling algorithms. Notice that the origin of the graph represents the optimum point at which data transfers can be completed instantly without any monetary cost, and every point represents the average data transfer throughput and egress cost for a Hail cluster instance. As observed, most scheduling decisions under the `cost-aware` algorithm are close to the optimum while those under Mesos and VBP are concentrated on the top right corner of the graph. This follows intuition since by co-locating *workers* the `cost-aware` algorithm improves data locality (as reflected in throughput) and reduces egress cost. This is in contrast to the low throughput resulting from the sparse distribution of *workers* when using Mesos and VBP. Note the outlier placement decisions landing at the far top right corner under the `cost-aware` scheduler. These outliers reveal the limitation of application-locality-based scheduling as application-locality does not imply data-locality in a distributed system (an

assertion valid in centralized environments, *e.g.*, single node). We argue that considering the limited information provided by the application to the scheduler, `cost-aware` strikes a good trade-off between cost and data locality improvement.

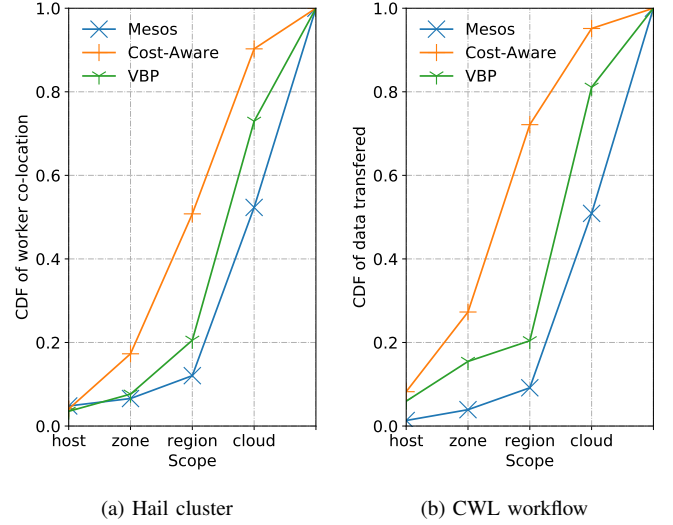


Fig. 7: CDF of worker co-locations/data transferred for Hail clusters/CWL workflows in different scopes, respectively. `cost-aware` effectively co-locates the *workers*/tasks for improved data locality and less egress cost.

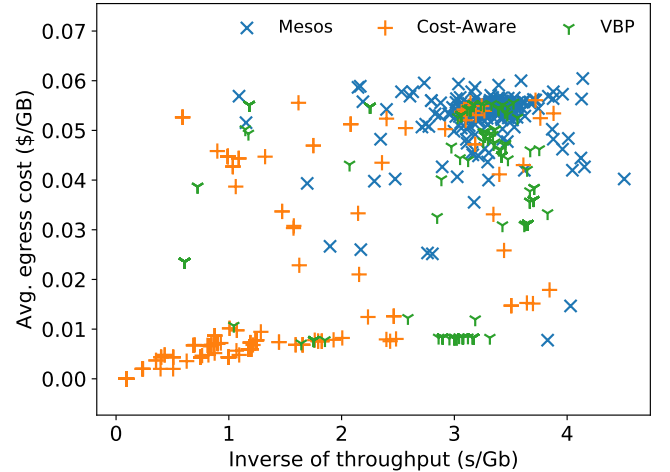


Fig. 8: Footprints of the Hail cluster deployments in the *cost-throughput* space. Most deployments by `cost-aware` are clustered in proximity to the optimum, while those by the baselines mostly distributed at the far end.

3) *CWL workflows*: CWL workflows are representative of the applications considered in our earlier simulation analysis. More specifically, data dependencies among workflow jobs are defined in the CWL workflow specification [4] – described as input and output files for each step of the workflow. Thus, the scheduling algorithm can take advantage of this additional information to infer data locality accurately when making placement decisions.

In Fig. 7b we notice that the locality of interdependent tasks which read their input data from a preceding task increases

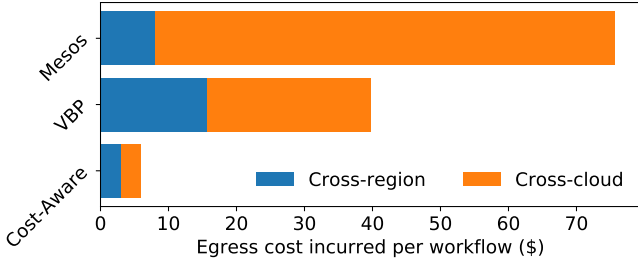


Fig. 9: Average egress cost incurred by running the CWL workflows. *cost-aware* saves up to \$2,092 (92.2%) in total for 30 workflow runs.

significantly under the *cost-aware* scheduler. A more detailed analysis of our results shows that once the anchor tasks are placed, the scheduling strategy makes a best effort to balance locality and performance as reflected by the *clustering* of descendent tasks.

Recall that cross-region cost is significant, therefore we are concerned with keeping data transfers within cloud regions. Note that in Fig. 7b, 9.1%, 20.4%, and 72.1% of data transfers were kept within a region using the *Mesos*, *VBP*, and *cost-aware* schedulers, respectively. The steep CDF slope of the *cost-aware* strategy demonstrates the effectiveness of our algorithm; only 4.9% of transfers span clouds compared to 18.9% and 49.0% for *VBP* and *Mesos*, respectively. Turning to Fig. 9 we see a drastic reduction in egress charges, and a shift from charges mostly being from cross-cloud in *Mesos* (89.2%) and *VBP* (60.2%), to charges being mostly from cross-region within a cloud in *cost-aware* (53.4%).

V. RELATED WORK

VM and container packing. VM packing is a well-studied problem [16] [17] [29] that aims at packing VMs into the least number of physical machines. The problem is generalized as a MDVPP problem and a family of FF- [29] and BF-based [17] heuristics have been developed and thoroughly studied. [29] provides a comprehensive summary of heuristics for the problem. More recently, the packing approaches are extended to applications such as data streaming [30] for optimal operator placement in the cluster. With the emergence of containers and their adoption in the cloud, there is a rising interest in packing containers into the least number of VMs to minimize VM subscription cost, which is analogous with VM packing. Stratus [20] has developed a cost-aware container scheduler that minimizes the VM instance hours and the associated cost in the cloud. These works partially share the same objective with ours and provide intuitions for our algorithm. However, we extend them into a cross-cloud context and investigate the impact of egress cost within and across clouds, which is unique in this context. Our evaluation also demonstrates that it is sub-optimal to directly apply existing VBP solutions in the cross-cloud context due to the absence of cost awareness.

Geo-distributed data locality. Recent works recognize the increasing importance of data locality in the context of geo-

distributed computing and data analytics [28] [21] [27] [31]. Awan [28] develops a locality-aware scheduler for geo-distributed data-intensive applications leveraging the explicit task runtime and data placement. DRASH [21] and Cachalot [27] dynamically replicate and cache commonly used data across geo-distributed resources to enhance data locality.

Cloud agnosticism. Only a few research works explore the cloud-agnostic and multi-cloud solutions for running distributed applications, and most of them were done prior to the widespread adoption of container technologies, which drastically change the landscape of cloud agnosticism. [23] classifies a stack of cloud solutions for easing the heterogeneity and enhancing inter-operability among clouds. [32] and [33] develop multi-cloud storage systems with bandwidth and cost efficiency in mind. PIVOT addresses all aspects of resource abstraction in a holistic manner.

VI. FUTURE WORK

In the future, we plan to leverage spot/preemptive VM instances in the cloud to further reduce the VM subscription cost. We are also interested in adoption of distributed caching in PIVOT in order to improve locality of commonly used data and save additional egress cost.

We also plan to integrate Kubernetes into PIVOT for resource abstraction considering its versatility and maturity in container orchestration and the immense support offered by its open-source community behind. With the node affinity feature [12] enabled in the latest version of Kubernetes (in beta phase), PIVOT is able to leverage this feature directly to enforce container placement in the analogous manner as it works with Mesos. That said, the cloud agnosticism and cost awareness demonstrated in this paper can be achieved with trivial effort with Kubernetes.

VII. CONCLUSION

We have introduced PIVOT as a holistic cloud-agnostic system, which enables seamless execution and scaling of data-intensive applications across geo-distributed regions and clouds. More importantly, we have developed an innovative cost-aware scheduling algorithm that effectively reduces up to 90.8% and 99.2% of VM subscription and egress cost for applications running across clouds, respectively. Particularly, with the data locality inference approach adopted in the algorithm, the algorithm effectively enhances the data locality and achieves over 4x speedup in data transfers.

VIII. ACKNOWLEDGEMENT

This work is supported by NSF SciDAS (ACI #1659300) and NIH Data Commons Pilot (OT3 ODO25464-01). We thank Dr. Paul Avillach and his team at Harvard university for providing the bioinformatic use case for Hail. We also thank Chris Ball at RTI International for his help with the genomic workflows.

REFERENCES

- [1] Apache Spark. <https://spark.apache.org/>.
- [2] Ceph. <https://ceph.com/>.
- [3] Ceph CRUSH Maps. <http://docs.ceph.com/docs/mimic/rados/operations/crush-map/>.
- [4] CWL Workflow. <https://www.commonwl.org/v1.0/Workflow.html>.
- [5] DataBiosphere/topmed-workflows. <https://github.com/DataBiosphere/topmed-workflows>.
- [6] DC/OS. <https://dcos.io/>.
- [7] DC/OS Marathon Placement Constraints. <https://docs.mesosphere.com/1.12/deploying-services/marathon-constraints/>.
- [8] Dockstore. <https://dockstore.org/>.
- [9] GlusterFS. <https://docs.gluster.org/en/latest/>.
- [10] Hail. <https://github.com/hail-is/hail>.
- [11] Kubernetes. <https://kubernetes.io/>.
- [12] Kubernetes Node Affinity. <https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity>.
- [13] PIVOT Scheduling Simulator. <https://github.com/heliumdatacommons/pivot-scheduling>.
- [14] Simpy. <https://simpy.readthedocs.io/en/latest/index.html>.
- [15] Peter Amstutz, Michael R. Crusoe, Neboja Tijani, Brad Chapman, John Chilton, Michael Heuer, Andrey Kartashov, Dan Leehr, Herv Mnager, Maya Nedeljkovich, Matt Scales, Stian Soiland-Reyes, and Luka Stojanovic. Common Workflow Language, v1.0. 7 2016.
- [16] Anton Beloglazov and Rajkumar Buyya. Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers. In *MGC@ Middleware*, page 4, 2010.
- [17] Anton Beloglazov and Rajkumar Buyya. 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, 2012.
- [18] Eric Boutin, Jaliya Ekanayake, Wei Lin, Bing Shi, Jingren Zhou, Zhengping Qian, Ming Wu, and Lidong Zhou. Apollo: Scalable and coordinated scheduling for cloud-scale computing. In *OSDI*, volume 14, pages 285–300, 2014.
- [19] Chandra Chekuri and Sanjeev Khanna. On multi-dimensional packing problems. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 185–194. Society for Industrial and Applied Mathematics, 1999.
- [20] Andrew Chung, Jun Woo Park, and Gregory R Ganger. Stratus: cost-aware container scheduling in the public cloud. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 121–134. ACM, 2018.
- [21] M. W. Convolbo, J. Chou, S. Lu, and Y. C. Chung. DRASH: A Data Replication-Aware Scheduler in Geo-Distributed Data Centers. In *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 302–309, December 2016.
- [22] György Dósa. The tight bound of first fit decreasing bin-packing algorithm is $\text{ffd} (i) \leq 11/9 \text{opt} (i) + 6/9$. In *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, pages 1–11. Springer, 2007.
- [23] Nicolas Ferry, Alessandro Rossini, Franck Chauvel, Brice Morin, and Arnor Solberg. Towards Model-Driven Provisioning, Deployment, Monitoring, and Adaptation of Multi-cloud Systems. pages 887–894, June 2013.
- [24] Hans Frenk, János Csirik, Martine Labbé, and Shuzhong Zhang. On the multidimensional vector bin packing. *University of Szeged. Acta Cybernetica*, pages 361–369, 1990.
- [25] Emanuele Goldoni, Giuseppe Rossi, and Alberto Torelli. Assolo, a new method for available bandwidth estimation. In *Internet Monitoring and Protection, 2009. ICIMP'09. Fourth International Conference on*, pages 130–136. IEEE, 2009.
- [26] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D Joseph, Randy H Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, volume 11, pages 22–22, 2011.
- [27] Fan Jiang, Claris Castillo, and Stan Ahalt. Cachalot: A network-aware, cooperative cache network for geo-distributed, data-intensive applications. In *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018.
- [28] Albert Jonathan, Abhishek Chandra, and Jon Weissman. Awan: Locality-aware resource manager for geo-distributed data-intensive applications. In *2016 IEEE International Conference on Cloud Engineering (IC2E)*, pages 32–41. IEEE, 2016.
- [29] Rina Panigrahy, Kunal Talwar, Lincoln Uyeda, and Udi Wieder. Heuristics for vector bin packing. *research. microsoft. com*, 2011.
- [30] Boyang Peng, Mohammad Hosseini, Zhihao Hong, Reza Farivar, and Roy Campbell. R-storm: Resource-aware scheduling in storm. In *Proceedings of the 16th Annual Middleware Conference*, pages 149–161. ACM, 2015.
- [31] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. Low Latency Geo-distributed Data Analytics. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM '15*, pages 421–434, New York, NY, USA, 2015. ACM.
- [32] Y. Singh, F. Kandah, and Weiyi Zhang. A secured cost-effective multi-cloud storage in cloud computing. In *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 619–624, April 2011.
- [33] Emil Stefanov and Elaine Shi. Multi-cloud Oblivious Storage. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, CCS '13*, pages 247–258, New York, NY, USA, 2013. ACM.
- [34] John Vivian, Arjun Arkal Rao, Frank Austin Nothhaft, Christopher Ketchum, Joel Armstrong, Adam Novak, Jacob Pfeil, Jake Narkizian, Alden D Deran, Audrey Musselman-Brown, et al. Toil enables reproducible, open source, big biomedical data analyses. *Nature biotechnology*, 35(4):314, 2017.