

Allocating and Scheduling Measurement Applications

Raquel Rugani Lage
Orange Labs - Télécom SudParis
Châtillon, France - Évry, France
raquel.ruganilage@orange.com

Anthony Lambert
Orange Labs
Châtillon, France
anthony.lambert@orange.com

Abstract—Improvements in containerization and orchestration have facilitated the management of distributed measurement platforms, helping rationalize networks and services monitoring, a vital but demanding activity. Nonetheless, orchestration in this domain, still requires optimization, which we address in this paper. More precisely, we invest our efforts in the optimization of allocating and scheduling network and service measurement applications, deployed in a pool of physical machines in the network. First, we formulate the problem into an Integer Linear Programming model and prove it to be NP-Hard. Then, we propose an approximate resolution based on the Ant Colony System (ACS) metaheuristic. Finally, we introduce an open-source distributed measurement platform implementing our algorithm. Results show that the ACS algorithm provides efficient orchestration solutions that outperform those provided by a Branch and Bound algorithm and a First Fit Decreasing algorithm.

Index Terms—Monitoring, Dynamic reassignment and scheduling problem, Integer Linear Programming, Ant Colony System

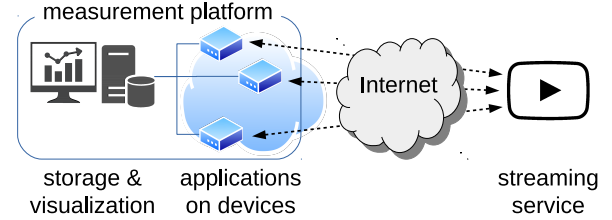


Figure 1: Applications testing a streaming service and retrieving measurements for storage, visualization or alerting.

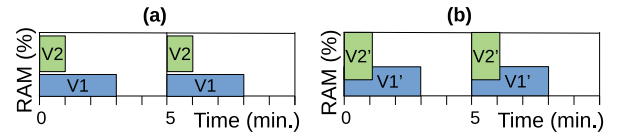


Figure 2: RAM usage: (a) no shortage / (b) shortage.

I. INTRODUCTION

Distributed measurement platforms are on the rise, from dedicated infrastructures set up by network operators or content providers toward monitoring quality and security of networks and services [1], [2], to public multipurpose large-scale platforms such as RIPE Atlas [3], used for various network tomography studies [4]–[8]. As illustrated on Fig. 1, a measurement platform usually consists in a set of devices on which reside applications that periodically test some network features (e.g., routing, traffic) or services (e.g., content-delivery, web quality). By doing so they generate time series of measurements intended to reflect the behavior of networks and services monitored, which are sent to servers for storage, dashboarding or alerting. An application can be deployed on a dedicated device (alone) or collocated with others. In this case, when many applications run on the same physical equipment, one has to make sure each of them has enough resources (especially RAM, CPU, bandwidth) when executing to ensure accuracy of the monitoring system. For instance, Fig. 2 illustrates the RAM consumption of two collocated applications in two situations. Every five minutes, V1 (resp. V1') runs during three minutes; and V2 (resp. V2'), during one minute. In doing so, they both consume a certain amount of RAM on the physical machine they are deployed.

Although on case (a) there is enough RAM to satisfy their needs, on case (b), applications compete for RAM usage, which can lead them to not execute properly and induce perturbations. Recent developments in containerization [9] and orchestration [10] have made it easier to deploy applications on physical machines, allowing users to precisely manage their execution and limit their allocated resources through quotas. However, despite these improvements, orchestrating measurement applications still remain a difficult task. In particular, it comes down to satisfying new demands for allocating and scheduling applications on a set of physical devices, on which a set of applications is already running. Consequently, one of the issues distributed measurement platforms must address to guarantee their accuracy, lies in the optimization of dynamically allocating resources and scheduling tests.

To this end, we: (i) formulate the problem as an Integer Linear Programming (ILP) model and prove it to be NP-Hard, (ii) provide an approximate resolution based on the Ant Colony System (ACS) metaheuristic, (iii) present an open-source distributed measurement platform that implements our algorithm and will be released publicly. The rest of this paper is organized as follows: Section II gives a formal definition of the problem. Section III provides an overview of related work. Section IV presents our ILP model, whose complexity is evaluated in Section V. Section VI details our ACS-

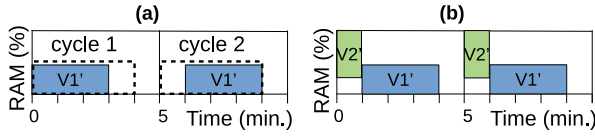


Figure 3: Start time delay: (a) example / (b) advantage.

based algorithm; and Section VII, our distributed measurement platform. Section VIII discusses the results of computational experiments. Finally, Section IX concludes this paper.

II. PROBLEM DEFINITION

Let $\mathcal{R}$ represent the set of different resources (e.g., CPU, RAM, bandwidth). Suppose time $\mathcal{T}$ is divided into a set of discrete time intervals (e.g., minutes, hours) indexed by $t = 1, \dots, |\mathcal{T}|$, and $t^0 = 0$ represents the time slot at the moment of a new deployment of applications into the system. Consider a pool of physical machines $\mathcal{M}$ ¹ that are characterized by an amount C_{mrt} of available resources $r \in \mathcal{R}$ during each time interval $t \in \mathcal{T}$, and a set of containerized applications $\mathcal{V}$ that are already deployed on them. Each application $v \in \mathcal{V}$ is characterized by an amount C_{vr} of resources $r \in \mathcal{R}$ needed to execute a test for a duration h_v , and migration-related overhead resources (i.e., resource consumption when performing a container migration): C_{vr}^- (resp. C_{vr}^+) when v is migrating away from (resp. into) a machine. Each one is periodically activated during a specified time interval $[Init_v, End_v] \in \mathcal{T}$, with a period p_v . This defines cycles during which it executes once. Additionally, an acceptable delay d_v of tests' start time can be provided: it allows the start of a measurement to be postponed during a cycle if needed. Whenever an application must execute with a strict periodicity, a start time delay of zero can be applied to enforce this behavior. In this way, each test $k \in \mathcal{K}(v)$ of an application $v \in \mathcal{V}$ has a start time at the earliest $\hat{t}_{vk} = Init_v + (k-1) \times p_v$, and a finish time at the latest $f_{vk} = \arg \min \{ \hat{t}_{vk} + d_v + h_v, End_v \}$. Where the number of cycles (or number of tests) for each application, can be defined by $|\mathcal{K}(v)| = \lfloor ((End_v - Init_v) / p_v) \rfloor$. Fig. 3 (a) illustrates the RAM consumption of an application over time. It runs for three minutes every five minutes between the defined time interval of $[0, 10]$. This defines two cycles. An acceptable delay of one minute is allowed. The first test is performed without delay, whereas the second one starts within the allowed delay of a minute. The advantage of this delay is illustrated on Fig. 3 (b), contrary to Fig. 2 (b). Notably it can enable fixing some resources competition cases.

Requests for deploying new applications arrive from multiple users in real-time and resources planning decisions must be made as soon as they arrive without prior knowledge of future requests. It is only at the time of arrival of a new request that information of the applications are revealed to the system. Once a request is submitted, we want to deploy and schedule the new applications into the requested machines. We officially

¹ $\mathcal{M}$ can be reduced to a subset of all machines, eventually one, to enforce location of applications when needed.

define this problem as a dynamic reassignment and scheduling problem, such that: by the time a new request arrives, there is an initial set of measurement applications that are already running on the physical machines, and the problem constitutes in finding a new assignment and schedule for applications' tests that respect resource and periodicity constraints at all times. This new assignment can be characterized as a series of migrations to change the initial positioning state to the desired one. Migration also allows relocation of a containerized application to fix an overloaded physical machine. However, it does not occur without costs in terms of resources that must also be taken into account. Consequently, while reassigning and scheduling applications we minimize the absolute number of migrations over time and the sum of tests' start time delays.

III. RELATED WORK

Distributed measurement infrastructures have been an active area of research for decades, from early works of Paxson et al. on NIMI's architecture [11], to the rise of large-scale measurement platforms, such as RIPE Atlas [3] or CAIDA Ark [12] (We also refer the reader to [13], for an in-depth survey on distributed measurement platforms which is out of scope for this paper.). If many efforts have been devoted to design such platforms [14]–[16], some studies have also been questioning their ability to provide accurate measurements without undesired side effects, such as interference [17], as well as proposing solutions to avoid them. For instance, Sanchez et al. [18] have presented a technique to coordinate measurements to prevent undesired load on networks or services through probes originating from crowdsourced measurement platforms. Gangam et al. [19] have proposed heuristics to schedule measurements in order to avoid interference while minimizing total completion time of measurement tasks. In the same vein, [20] deals with resource contention by providing a clique-based algorithm that takes into account the scheduling of on-demand measurement applications. Similarly, [21] have presented a conflict-free and MLA-compliant algorithm that handles on-demand measurement requests with different priorities while minimizing cycle time and maximizing user satisfaction ratio. Calyam et al. [22] have solved the problem of scheduling offline and on-demand measurement requests, by proposing a heuristic algorithm that optimizes time efficiency.

Collectively, these studies outline the need for preventing simultaneous scheduling on applications that would interfere with each other (anti-affinity measurement applications) as to ensure the accuracy of the measurement system and not overload computation resources. However, our study is different from these studies in an important way. We take advantage on container-based technology and we explore the possibility of migrating applications into different physical equipment that they are allowed to run. In this way, we propose an innovative allocation and scheduling policy that (i) respects applications' periodicity and delay, (ii) takes into account affinity and anti-affinity applications and (iii) can re-adapt the system through container migration over a (iv) fine-grained time horizon. We propose a multi-objective optimization ILP

Table I: Notations

Indices and sets:

$v \in \mathcal{V}$	Set of applications	$r \in \mathcal{R}$	Set of resources
$m \in \mathcal{M}$	Set of machines	$t \in \mathcal{T}$	Set of discrete time slots
$k \in \mathcal{K}(v)$	Set of tests of application $v \in \mathcal{V}$		

Constants:

C_{mrt}	Machine m capacity of resource r during time t
C_{vr}	Amount of resource r needed to process application v
C_{vr}^-	Amount of resource r needed to migrate v out of a machine
C_{vr}^+	Amount of resource r needed to migrate v into a machine
$\hat{t}_{vk}$	Start time at the earliest of test k of application v
f_{vk}	Finish time at the latest of test k of application v
h_v	Execution time of each test of application v
x_{vm0}	Initial application mapping to physical machines
t^0	Time slot at the moment of a new deployment
ω_1	Weight of migration costs in objective function
ω_2	Weight of delays costs in objective function
U_{mrt}	Load of machine m for resource r during time t

Variables:

x_{vmt}	binary, $x_{vmt} = 1$, if v is mapped to machine m during time t
z_{vmt}^-	binary, $z_{vmt}^- = 1$, if v is migrated out of machine m during t
z_{vmt}^+	binary, $z_{vmt}^+ = 1$, if v is migrated into machine m during t
y_{vmt}	binary, $y_{vmt} = 1$, if v is running a test in machine m during t
y_{vkt}	binary, $y_{vkt} = 1$, if v is running test k during time slot t
a_{vkt}	binary, $a_{vkt} = 1$, if v runs test k at time slots greater than t
b_{vkt}	binary, $b_{vkt} = 1$, if v runs test k at time slots smaller than t
δ_{vk}	positive integer, actual start time delay of test k of application v

model, which we prove to be NP-Hard and implement an ACS algorithm to solve it. We take interest into optimizing computation resources while ensuring the accuracy of the measurement system. We present an open-source distributed measurement platform that implements our algorithm and will be released publicly.

IV. INTEGER LINEAR PROGRAMMING MODEL

Let $\mathcal{V}$ denote the set of all applications that were already in the network plus the new ones that we want to deploy in the system. Assignments are represented by binary variable x_{vmt} that equals 1 if application v is mapped to machine m during time slot t . At t^0 , initial application mapping to machines is defined by x_{vm0} . Binary variables z_{vmt}^- and z_{vmt}^+ represent container migrations: z_{vmt}^- (resp. z_{vmt}^+) is equal to 1 if application v is migrated away from (resp. into) machine m during time slot t . To define if an application v is running or not, we use the binary variables y_{vmt} that equals 1 if v is running a test in machine m during time slot t , and y_{vkt} that is equal to 1 if v is running test k during time slot t . Auxiliary binary variables a_{vkt} and b_{vkt} help estimate y_{vkt} : a_{vkt} equals 1 (resp. b_{vkt}), if application v runs test k at time slots that are greater (resp. smaller) than t . Table I summarizes notations. The action of deploying and scheduling applications into a preferred set of machines, can be, therefore, expressed as the following ILP formulation:

$$\min \sum_{v \in \mathcal{V}} \left(\sum_{m \in \mathcal{M}, t \in \mathcal{T}} (z_{vmt}^- + z_{vmt}^+) \times \omega_1 + \sum_{k \in \mathcal{K}(v)} \delta_{vk} \times \omega_2 \right) \quad (1)$$

subject to

$$\sum_{m \in \mathcal{M}} x_{vmt} = 1 \quad \forall v \in \mathcal{V}, t \in \mathcal{T}, \quad (2)$$

$$x_{vm(t-1)} - x_{vmt} \leq z_{vmt}^- \quad \forall v \in \mathcal{V}, m \in \mathcal{M}, t \in \mathcal{T}, \quad (3a)$$

$$x_{vmt} - x_{vm(t-1)} \leq z_{vmt}^+ \quad \forall v \in \mathcal{V}, m \in \mathcal{M}, t \in \mathcal{T}, \quad (3b)$$

$$\sum_{v \in \mathcal{V}} (C_{vr} \times y_{vmt} + C_{vr}^- \times z_{vmt}^- + C_{vr}^+ \times z_{vmt}^+) \leq C_{mrt} \quad \forall m \in \mathcal{M}, r \in \mathcal{R}, t \in \mathcal{T}, \quad (4)$$

$$\sum_{t \in \mathcal{T}} y_{vkt} = h_v \quad \forall v \in \mathcal{V}, k \in \mathcal{K}(v), \quad (5a)$$

$$\sum_{t < f_{vk}} y_{vkt} = h_v \quad \forall v \in \mathcal{V}, k \in \mathcal{K}(v), \quad (5b)$$

$$a_{vkt} \geq a_{vk(t+1)} \quad \forall v \in \mathcal{V}, k \in \mathcal{K}(v), t \in \mathcal{T} \setminus \{|\mathcal{T}|\}, \quad (6a)$$

$$b_{vkt} \geq b_{vk(t-1)} \quad \forall v \in \mathcal{V}, k \in \mathcal{K}(v), t \in \mathcal{T} \setminus \{1\}, \quad (6b)$$

$$a_{vkt} + b_{vkt} + y_{vkt} = 1 \quad \forall v \in \mathcal{V}, k \in \mathcal{K}(v), t \in \mathcal{T}, \quad (6c)$$

$$\delta_{vk} = \sum_{t \in \mathcal{T}} a_{vkt} - \hat{t}_{vk} + 1 \quad \forall v \in \mathcal{V}, k \in \mathcal{K}(v), \quad (7)$$

$$y_{vmt} + z_{vmt}^- \leq 1 \quad \forall v \in \mathcal{V}, m \in \mathcal{M}, t \in \mathcal{T}, \quad (8a)$$

$$y_{vmt} + z_{vmt}^+ \leq 1 \quad \forall v \in \mathcal{V}, m \in \mathcal{M}, t \in \mathcal{T}, \quad (8b)$$

$$x_{vmt} \geq y_{vmt} \quad \forall v \in \mathcal{V}, m \in \mathcal{M}, t \in \mathcal{T}, \quad (9a)$$

$$\sum_{k \in \mathcal{K}(v)} y_{vkt} \geq y_{vmt} \quad \forall v \in \mathcal{V}, m \in \mathcal{M}, t \in \mathcal{T}, \quad (9b)$$

$$\sum_{k \in \mathcal{K}(v)} y_{vkt} + x_{vmt} \leq y_{vmt} + 1 \quad \forall v \in \mathcal{V}, m \in \mathcal{M}, t \in \mathcal{T}, \quad (9c)$$

$$x_{vmt}, z_{vmt}^-, z_{vmt}^+, y_{vmt} \in \{0, 1\} \quad \forall v \in \mathcal{V}, m \in \mathcal{M}, t \in \mathcal{T}, \quad (10a)$$

$$y_{vkt}, a_{vkt}, b_{vkt} \in \{0, 1\} \quad \forall v \in \mathcal{V}, k \in \mathcal{K}(v), t \in \mathcal{T}, \quad (10b)$$

$$\delta_{vk} \in \mathbb{Z}^+ \quad \forall v \in \mathcal{V}, k \in \mathcal{K}(v) \quad (10c)$$

The objective function (OF) (1) minimizes the weighted sum of the number of application migrations plus the total delay associated with applications' tests start time. At each time slot, we have a reassignment problem that respects allocation (2), (3a), (3b) and capacity (4) constraints. At the same time we have a scheduling problem that makes sure each test will run for a duration that equals its execution time (5a), while respecting its periodicity, and allowing no interruptions once it has started running (5b)–(6c), (8a), (8b). Equation (7) states the absolute delay δ_{vk} for each test $k \in \mathcal{K}(v) \forall v \in \mathcal{V}$. Constraints (9a)–(9c) establish the link between decision variables x_{vmt} , y_{vkt} , and y_{vmt} : (9a) defines that an application v has to be allocated in a machine m during time slot t in order for v to be running in it during this interval of time; (9b) states

that a task k of application v has to be running during t in order for v to be running in a machine m during this time; and (9c) establishes that an application v will be considered to be running in the machine m during time slot t when it is allocated to this machine and it has a test k that is running during t . Finally, integrality constraints (10a)–(10c) establish the domain of the variables. This optimization problem is proved to be NP-Hard by reduction from the Bin-Packing Problem (Section V). In Section VIII we evaluate it using instances of our open-source distributed measurement platform deployed in an Internet Service Provider's (ISP) network.

A. Rule-based constraints

In addition to basic formulation presented above, additional rule-based constraints can be used as an extension of our problem. Let $\mathcal{S} = \{S_1, S_i, \dots, S_{|\mathcal{S}|}\}$, such that $S_i \subset \mathcal{V}$ represents a subset of affinity or anti-affinity measurement applications. New binary variables are defined: $y_{s_i,mt}$ equals 1 if $\exists v \in S_i | y_{vmt} = 1, m \in \mathcal{M}, t \in \mathcal{T}$, and $x_{s_i,mt}$ equals 1 if $\exists v \in S_i | x_{vmt} = 1, m \in \mathcal{M}, t \in \mathcal{T}$. The following equations are then created to establish the relation between y_{vmt} and $y_{s_i,mt}$: (11a) ensures that $y_{s_i,mt}$ will be equal to 1 if there is at least one application $v \in S_i$ that is running in machine m during time slot t ; and (11b) guarantees that $y_{s_i,mt}$ will be valued to 0 if there is no application $v \in S_i$ running in machine m during time slot t .

$$\sum_{v \in S_i} y_{vmt} \leq |S_i| \times y_{s_i,mt} \quad \forall S_i \in \mathcal{S}, m \in \mathcal{M}, t \in \mathcal{T}, \quad (11a)$$

$$\sum_{v \in S_i} y_{vmt} \geq y_{s_i,mt} \quad \forall S_i \in \mathcal{S}, m \in \mathcal{M}, t \in \mathcal{T} \quad (11b)$$

Relation between variables $x_{s_i,mt}$ and x_{vmt} is defined by the the following constraints, where (12a) and (12b) are analogous to (11a) and (11b) respectively.

$$\sum_{v \in S_i} x_{vmt} \leq |S_i| \times x_{s_i,mt} \quad \forall S_i \in \mathcal{S}, m \in \mathcal{M}, t \in \mathcal{T}, \quad (12a)$$

$$\sum_{v \in S_i} x_{vmt} \geq x_{s_i,mt} \quad \forall S_i \in \mathcal{S}, m \in \mathcal{M}, t \in \mathcal{T} \quad (12b)$$

The following rule-based constraints ensure that affinity or anti-affinity applications are allocated in different machines (13a) (resp. in the same machine (13b)), or that they run alternately (13c) (resp. run together (13d)). This could be important, for instance, for preventing monitoring failure in the case of physical flaw, or guarantee monitoring exactitude by making sure that there is no interference between their tests.

$$\sum_{v \in S_i} x_{vmt} \leq 1 \quad S_i \in \mathcal{S}, \forall m \in \mathcal{M}, t \in \mathcal{T}, \quad (13a)$$

$$\sum_{v \in S_i} x_{vmt} \geq 2 \times x_{s_i,mt} \quad S_i \in \mathcal{S}, \forall m \in \mathcal{M}, t \in \mathcal{T}, \quad (13b)$$

$$\sum_{v \in S_i} y_{vmt} \leq 1 \quad S_i \in \mathcal{S}, \forall m \in \mathcal{M}, t \in \mathcal{T}, \quad (13c)$$

$$\sum_{v \in S_i} y_{vmt} \geq 2 \times y_{s_i,mt} \quad S_i \in \mathcal{S}, \forall m \in \mathcal{M}, t \in \mathcal{T}, \quad (13d)$$

These constraints are to be added directly to the previous formulation when needed.

V. COMPUTATIONAL COMPLEXITY

Complexity theory classifies decision problems according to their difficulty. NP-hardness defines the set of problems that are hard in the sense that they cannot be solved using a polynomial time deterministic algorithm (unless $P=NP$), and which ones are the hardest among these. If an optimization problem is easy/difficult, its decision problem is also easy/difficult [23].² In the following we associate a decision problem to our optimization problem, and then a polynomial reduction is performed in order to prove it to be NP-Hard:

INSTANCE: Finite set $\mathcal{R}$ of resources. Finite set $\mathcal{T}$ of time slots $\in \mathbb{Z}^{*+}$. Finite set $\mathcal{V}$ of containers. For each $v \in \mathcal{V}$, a set of tasks $\mathcal{K}(v)$ to be executed with multiple resource requirements $C_{vr}, C_{vr}^-, C_{vr}^+ \in \mathbb{Z}^+ \forall r \in \mathcal{R}$, and with length $h_v \in \mathbb{Z}^+$. For each task $k \in \mathcal{K}(v) : v \in \mathcal{V}$, a release time $\hat{t}_{vk} \in \mathbb{Z}^{*+}$, and a deadline $f_{vk} \in \mathbb{Z}^{*+}$. Finite set $\mathcal{M}$ of machines of given capacities: $C_{mrt}, m \in \mathcal{M}, r \in \mathcal{R}, t \in \mathcal{T}$. An initial container mapping to physical machines represented by $|\mathcal{M}|$ disjoint subsets $\mathcal{V}_1^0, \dots, \mathcal{V}_{|\mathcal{M}|}^0$. Two positive integers δ and L .

QUESTION: Is there a container mapping at each time slot and a schedule for all tasks from all containers that satisfies release time constraints, meets all deadlines and obeys resource constraints, such that the overall container migration is L or less and the overall tardiness is δ or less? i.e.:

Are there $|\mathcal{M}|$ disjoint subsets $\mathcal{V}_1^t, \dots, \mathcal{V}_{|\mathcal{M}|}^t \forall t \in \mathcal{T}$,

where $\bigcup_{m \in \mathcal{M}} \mathcal{V}_m^t = \mathcal{V} \quad \forall t \in \mathcal{T}$,

and a function $\sigma : (\mathcal{K}(v), \mathcal{V}) \rightarrow \mathcal{T} \quad (k, v) \mapsto \sigma(k, v)$

with $\sigma(k, v) \geq \hat{t}_{vk}$ and $\sigma(k, v) + h_v \leq f_{vk}$,

such that for all $t \in \mathcal{T}, m \in \mathcal{M}$, if $\mathcal{Z}_m^{-t}$ is the set

of all $v \in \mathcal{V}$ where $\mathcal{Z}_m^{-t} = \mathcal{V}_m^{(t-1)} - \mathcal{V}_m^t$

and that for all $t \in \mathcal{T}, m \in \mathcal{M}$, if $\mathcal{Z}_m^{+t}$ is the set

of all $v \in \mathcal{V}$ where $\mathcal{Z}_m^{+t} = \mathcal{V}_m^t - \mathcal{V}_m^{(t-1)}$

and that for all $t \in \mathcal{T}$, if $Y(t)$ is the set of all $v \in \mathcal{V}$

for which $\sigma(k, v) \leq t < \sigma(k, v) + h_v$,

then we have that

$$\forall [m \in \mathcal{M}, r \in \mathcal{R}, t \in \mathcal{T}] \Rightarrow \sum_{v \in Y(t)} C_{vr} +$$

$$\sum_{v \in \mathcal{Z}_m^{-t}} C_{vr}^- + \sum_{v \in \mathcal{Z}_m^{+t}} C_{vr}^+ \leq C_{mrt}$$

and such that $\sum_{t \in \mathcal{T}} \sum_{m \in \mathcal{M}} (n(\mathcal{Z}_m^{-t}) + n(\mathcal{Z}_m^{+t})) \leq L$ ³

and $\sum_{v \in \mathcal{V}} \sum_{k \in \mathcal{K}(v)} \sigma(k, v) - \hat{t}_{vk} \leq \delta$

This problem is NP-hard by reduction from the Bin-Packing Problem (BPP), which was proven to be strongly NP-Complete by reduction from 3-Partition by Garey and Johnson 1979 [23]. The decision problem associated to the BPP is:

²For further details on the complexity theory, we suggest consulting [23].

³ $n(A)$ represents the number of elements of the finite set A .

BPP INSTANCE: Finite set $\mathcal{U}$ of items, for each $u \in \mathcal{U}$ a size $s(u) \in \mathbb{Z}^+$. A positive integer bin capacity b and a positive integer B .

QUESTION: Is there a partition of $\mathcal{U}$ into disjoint subsets $\mathcal{U}_1, \dots, \mathcal{U}_B$ such that the sum of the sizes of the items in each $\mathcal{U}_i$ is b or less?

Proof. Any instance I of BPP can be polynomially transformed to an equivalent instance I' of our problem by setting:

$$\begin{aligned} |\mathcal{R}| &= 1; & C_{vr} &= s(u) \quad \forall v = u, r \in \mathcal{R}; \\ \mathcal{V} &= \mathcal{U}; & C_{mrt} &= b \quad \forall m \in \mathcal{M}, r \in \mathcal{R}, t \in \mathcal{T}; \\ |\mathcal{M}| &= B; & \hat{t}_{vk} &= 0 \quad \forall v \in \mathcal{V}, k \in \mathcal{K}(v); \\ |\mathcal{K}(v)| &= 1 \quad \forall v \in \mathcal{V}; & f_{vk} &= |\mathcal{T}| \quad \forall v \in \mathcal{V}, k \in \mathcal{K}(v); \\ L &= 0; & h_v &= |\mathcal{T}| \quad \forall v \in \mathcal{V}; & \delta &= 0. \end{aligned}$$

Which implies that $\mathcal{U}_i = \mathcal{V}_{tm} \quad t \in \mathcal{T}, \forall m = i$ in any "yes" instance. $\square$

The implication of this proof is that if $P \neq NP$, then there is no polynomial time deterministic algorithm for solving our optimization problem. Thus, due to the time complexity of solving an NP-Hard problem, a resolution by exhaustive search becomes dreadfully large, even for small size instances, and it is often hardly suitable for practical use. Therefore, we have to content ourselves with finding an approximate solution, which will not be guaranteed to be optimal.

VI. ANT COLONY SYSTEM

Ant Colony Optimization (ACO) takes inspiration from the natural food discovery behaviour of real ants. Throughout their path between food source and nest, ants deposit a chemical substance called pheromone. Other ants perceive the presence of pheromone and tend to follow paths where its concentration is higher. Initially, when pheromone level is not intense, each ant chooses randomly a path that goes to the food source. However, after some time, smaller paths to the food source will present higher concentration of pheromone as a result of a higher number of ants passing through and, after some time the colony tends to converge towards the shortest path [24]. After the initial proof-of-concept application to the traveling salesman problem [25], the ACO algorithm and its variations have been successfully applied to many NP-Hard problems such as routing, assignment, scheduling, and subset problems [26], [27]. Thus, we adapt a variation of the ACO (the ACS [24]) to our problem in order to quickly find high-quality solutions – since solving it optimally with the Branch-and-Bound algorithm takes unreasonable time (these results will be addressed in the following Section).

Our ACS-based algorithm is shown in Algorithm 1. In each iteration (lines 2-14), each artificial ant receives all applications $v \in \mathcal{V}$ in random order⁴ (line 5) and designates to each of their tests $k \in \mathcal{K}_v : \hat{t}_{vk} \geq t^o$ a *component* machine-start time $m, \hat{t} : m \in \mathcal{M}, \hat{t} \in \mathcal{T}, \hat{t}_{vk} \leq \hat{t} \leq \hat{t}_{vk} + d_v$ (line

⁴The choice of distributing ants on applications randomly is not by chance. As explained in [24], all ACO algorithms for assignment problems have to take two decisions into account during the construction of a solution. The first is choosing what item will be assigned next, while the second refers to where the item will be allocated. We associate pheromone trail with the second type of decision, while making the first one a random decision.

Algorithm 1 ACS-based algorithm

```

1: Initialize parameters
2: while stopping criterion not satisfied do
3:   Comment: at this level each loop is called an iteration
4:   for all ants  $a = \{1, 2, \dots, Ants\}$  do
5:     Position ant  $a$  on a random application  $v \in \mathcal{V}$ 
6:     for all  $k \in \mathcal{K}_v : \hat{t}_{vk} \geq t^o$  do
7:       Ant  $a$  applies state transition rule
8:       Ant  $a$  applies solution update rule
9:       Ant  $a$  applies local pheromone update rule
10:    end for
11:  end for
12:  Comment: here all ants have built a solution
13:  Global pheromone update rule is applied
14: end while
15: Return best solution

```

7). This is achieved by using a *state transition rule* (Section VI-A), which describes the desirability for an ant to choose a component for an application's test. This rule is based on the current *pheromone* concentration on this designation, and on a *heuristic information* (Section VI-B). Every time an ant chooses a component for an application's test, it updates its solution accordingly (line 8) (Section VI-C), and applies a *local pheromone update* (line 9) (Section VI-D). Finally, at the end of an iteration, after each ant has constructed a solution, the *global pheromone of the system is updated* (line 13). The algorithm iterates until a certain criterion is satisfied (e.g., OF value, CPU time), and returns the best solution found (line 15).

A. State transition rule

According this rule, detailed in (14), an ant will choose a machine m and a start time $\hat{t} = \hat{t}_{vk} + \delta_{vk}$ to allocate and run an application's test represented by $vk : v \in \mathcal{V}, k \in \mathcal{K}_v$:

$$m, \hat{t} = \begin{cases} \arg \max_{m, \hat{t} \in N_{vk}} \{ [\tau_{vk, m, \hat{t}}]^\alpha \times [H_{vk, m, \hat{t}}]^\beta \}, & \text{if } q \leq q_o \text{ (exploitation)} \\ \text{biased exploration (17), otherwise} & \end{cases} \quad (14)$$

where, q is a random variable uniformly distributed over $[0, 1]$; and q_o , a parameter that determines the relative importance of exploitation versus exploration⁵ for this rule. $\tau_{vk, m, \hat{t}}$ denotes the pheromone based desirability of allocating application v into machine m and starting its test k at time $\hat{t}$. Heuristic information is represented by $H_{vk, m, \hat{t}}$, which guides ants toward choosing more promising components $m, \hat{t}$ for test vk . Two parameters $\alpha, \beta \geq 0$ are used to either emphasize more the pheromone or the heuristic information. Moreover, N_{vk} defines the set of admissible (that does not violate any constraints) components $m, \hat{t}$ for allocating and running an application's test vk . Therefore, in the construction of

⁵Exploitation stresses the ability to thoroughly examine particular research areas; while exploration, the ability to discover promising research areas.

the set N_{vk} , we make sure that capacity and migration constraints are not being violated during test vk execution ($[\hat{t}, \hat{t}_{vk} + d_v + h_v]$) and during the period of time before it starts running ($[\hat{t}_{v(k-1)} + \delta_{v(k-1)} + h_v, \hat{t}]$), in which a migration could take place. N_{vk} is defined in (15):

$$N_{vk} = \left\{ \begin{array}{l} m, \\ \hat{t} \end{array} \left| \begin{array}{l} (x_{vm(\hat{t}_{vk}-1)} = 1 \wedge N_{vkm} \cap \emptyset \rightarrow \hat{t} \in N_{vkm}) \\ \wedge \\ (x_{vm(\hat{t}_{vk}-1)} = 0 \wedge N_{vkm} \cap \emptyset \rightarrow \hat{t} \in N_{vkm} \\ \wedge \exists m', t_{mig} (m' \in \mathcal{M} \wedge x_{vm'(\hat{t}_{vk}-1)} = 1 \wedge \\ \hat{t}_{v(k-1)} + \delta_{v(k-1)} + h_v \leq t_{mig} < \hat{t}_{vk} \\ \wedge y_{vm't} = 0 \wedge \forall r \in \mathcal{R} \Rightarrow [U_{m'rt_{mig}} + C_{vr}^- \\ \leq C_{m'r} \wedge U_{mrt_{mig}} + C_{vr}^+ \leq C_{mr}]) \end{array} \right. \right. \quad (15)$$

where U_{mrt} denotes the load of machine m for each resource $r \in \mathcal{R}$ during time slot $t \in \mathcal{T}$ and N_{vkm} defines the set of all time slots that are eligible as the start time of test vk if it is assigned to machine m . N_{vkm} is defined as follows:

$$N_{vkm} = \left\{ \hat{t} \left| \begin{array}{l} \hat{t} \in \mathcal{T} \wedge \hat{t}_{vk} \leq \hat{t} \leq \hat{t}_{vk} + d_v \\ \wedge \forall [r \in \mathcal{R} \wedge t \in \mathcal{T} : \hat{t} \leq t < \hat{t} + h_v] \\ \Rightarrow [U_{mrt} + C_{vr} \leq C_{mr}] \end{array} \right. \right. \quad (16)$$

Finally, the *biased exploration* is defined as the probability to assign $m, \hat{t} \in N_{vk}$ to an application's test vk :

$$P_{vk,m,\hat{t}} = \left\{ \begin{array}{l} \frac{[\tau_{vk,m,\hat{t}}]^\alpha \times [H_{vk,m,\hat{t}}]^\beta}{\sum_{m', \hat{t} \in N_{vk}} [\tau_{vk,m',\hat{t}}]^\alpha \times [H_{vk,m',\hat{t}}]^\beta}, \text{ if } m, \hat{t} \in N_{vk} \\ 0, \text{ otherwise} \end{array} \right. \quad (17)$$

B. Heuristic information

Heuristic information helps minimizing the OF value by encouraging $x_{vm\hat{t}_{vk}} = x_{vm(\hat{t}_{vk}-1)}$, and $\delta_{vk} = 0$. This is accomplished by setting the value of $H_{vk,m,\hat{t}}$ as the inverse of the additional cost that would be produced in the OF if application v is allocated to machine m to run test k at time $\hat{t} = \hat{t}_{vk} + \delta_{vk}$, as stated in (18):

$$H_{vk,m,\hat{t}} = \left\{ \begin{array}{l} 2, \text{ if } x_{vm(\hat{t}_{vk}-1)} = 1 \wedge \hat{t} = \hat{t}_{vk} \\ \frac{1}{(\hat{t} - \hat{t}_{vk}) \times \omega 2}, \text{ if } x_{vm(\hat{t}_{vk}-1)} = 1 \wedge \hat{t} \neq \hat{t}_{vk} \\ \frac{1}{2 \times \omega 1 + (\hat{t} - \hat{t}_{vk}) \times \omega 2}, \text{ otherwise} \end{array} \right. \quad (18)$$

C. Solution update

During the construction of a solution, after each ant has applied the *state transition rule*, and therefore assigned a machine m and a start time $\hat{t}$ to an application's test vk , it will update variables according to its assignment:

$$\begin{aligned} \delta_{vk} &= \hat{t} - \hat{t}_{vk}; & y_{vmt} &= 1 \quad \forall \hat{t} \leq t < \hat{t} + h_v; \\ x_{vmt} &= 1 \quad \forall \hat{t}_{vk} \leq t < \hat{t}_{v(k+1)}; \\ U_{mrt} &= U_{mrt} + C_{vr} \quad \forall r \in \mathcal{R}, \hat{t} \leq t < \hat{t} + h_v. \end{aligned}$$

If $x_{vm(\hat{t}_{vk}-1)} = 0$ the application has migrated from a machine $m' : x_{vm'(\hat{t}_{vk}-1)} = 1$ to the new one. By

construction of the set N_{vk} , there exists a time slot t_{mig} of when this migration took place without violating any constraints. In our algorithm, if it is possible, we chose to migrate the container in a time interval t_{mig} where $U_{m'rt_{mig}}, U_{mrt_{mig}} > 0 : r \in \mathcal{R}$, and as close as possible to the end of the application's cycle, i.e., t_{mig} will be valued as highest as possible to $\hat{t}_{v(k+1)}$. Variables associated to this solution are then updated:

$$\begin{aligned} U_{mrt_{mig}} &= U_{mrt_{mig}} + C_{vr}^+ \quad \forall r \in \mathcal{R}; & z_{vmt_{mig}}^+ &= 1; \\ U_{m'rt_{mig}} &= U_{m'rt_{mig}} + C_{vr}^- \quad \forall r \in \mathcal{R}; & z_{vm't_{mig}}^- &= 1; \\ x_{vmt} &= 1 \quad \forall t_{mig} \leq t < \hat{t}_{vk}; & x_{vm't} &= 0 \quad \forall t_{mig} \leq t < \hat{t}_{vk}. \end{aligned}$$

D. Pheromone update

In the first iteration of our ACS-based algorithm, ants are given all applications in random order, and we initialize the *initial pheromone* value as (19), where μ is a value bigger than the upper bound of the OF.

$$\tau_0 = \frac{1}{\mu} \quad (19)$$

During the construction of a solution, after each ant has chosen a machine m and a start time $\hat{t}$ for an application's test vk , it will apply a *local pheromone update* that will change the amount of pheromone on this assignment as defined in (20):

$$\tau_{vk,m,\hat{t}} = (1 - \rho) \times \tau_{vk,m,\hat{t}} + \rho \times \tau_0 \quad (20)$$

where $0 < \rho \leq 1$ is a constant that is used to simulate pheromone evaporation. Higher values for ρ lead to increased evaporation rate, meaning less impact of past solutions.

At the end of each iteration, we keep the best solution found in *best*, i.e., the one with the smallest associated OF value (of_{best}). We need to strengthen this promising solution by increasing its pheromone value. This is achieved by applying a *global pheromone update* rule as follows:

$$\tau_{vk,m,\hat{t}} = \left\{ \begin{array}{l} (1 - \rho) \times \tau_{vk,m,\hat{t}} + \rho \times \frac{1}{of_{best}}, \\ \text{if assignment } (vk - m\hat{t}) \in \text{best} \end{array} \right. \quad (21)$$

Note that when $of_{best} = 0$ the ratio in (21) cannot occur. This is not a problem for the algorithm since finding $of_{best} = 0$ is equivalent to finding the best possible solution.

Furthermore, similarly to Max-Min Ant System [24], pheromone values on each assignment are limited to $\tau_{max} : \tau_{max} = 2 \times \tau_0$ and a $\tau_{min} : \tau_{min} = \tau_0 \times (1 - \rho)^{max_evap}$. Where τ_{min} equals an approximate value after having the associated assignment submitted to a predefined maximum number (*max_evap*) of pheromone local update. This allows a better exploration of the solutions in the algorithm as it can avoid early stagnation of the search [26].

VII. NETWORK MAAS (MONITORING AS A SERVICE)

We implement our ACS-based algorithm as a component of an open-source distributed measurement platform, which we call the Network MaaS (Monitoring as a Service) and intend to release publicly so it can be used and extended [28]. The Network MaaS enables on-demand deployment and management of containerized measurement applications on a pool of

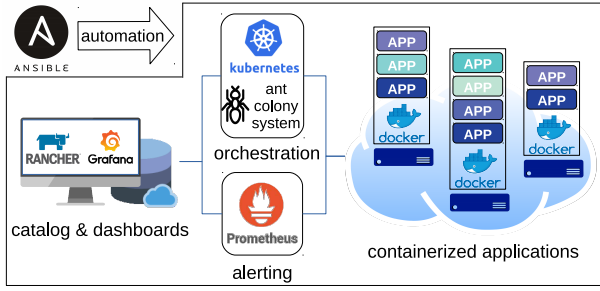


Figure 4: Network MaaS overview.

Table II: Network MaaS Catalog

<i>App.</i>	<i>Description</i>	h_v	p_v	$Init_v$	End_v
App1	gets web cache servers ip addr.	1'	30'	8h00	19h00
App2	gets web cache servers ip addr.	1'	30'	0h00	24h00
App3	gets streaming cache servers ip addr.	3'	120'	8h00	19h00
App4	measures web page download time	5'	120'	8h00	19h00
App5	measures streaming startup delay	5'	120'	8h00	19h00
App6	downloads BGP tables	30'	24h	0h00	24h00
App7	downloads BGP updates	1'	5'	0h00	24h00

machines spread across the network to be monitored. This is made possible via an online graphical interface, allowing one to: visualize and manage the pool of machines as well as applications already deployed; choose applications from a catalog and deploy them on-demand; visualize tests results and alerts raised. As sketched on Fig. 4, the whole platform is automatically deployed by Ansible [29], which installs and configures its components. Docker [9] enables to embed applications into containers, while Kubernetes [10] orchestrates their lifecycle. Our ACS-based component handles users requests for application deployments. It gets the current state of resources on machine, via Kubernetes, computes new resource allocations and test schedules as well as potential migrations, and enforces them via Kubernetes. Prometheus [30] runs as a database and stores time series measurements sent by the applications, while Grafana [31] enables to visualize them through graphs and diagrams. It is coupled with Alert-Manager [30] that sends notifications whenever a measurement reaches pre-configured thresholds. The catalog is powered by Rancher [32], which offers a local storage for developers to publish their own applications in a repository fashion. First applications to be released with the Network MaaS are briefly presented on Table II (Complete description is out of scope of this article.). For each application, its test description, duration (h_v), period (p_v), start time ($Init_v$) and end time (End_v) are given. Main benefits from the Network MaaS are its open-source design, the ability to rapidly integrate applications to the platform thanks to the catalog system and its container-based architecture, and, of course, its ACS-based component that enables proper resource allocation and tests schedule of measurement applications.

VIII. EVALUATION

Our evaluation is based on the use case of a Network MaaS instance deployed in an ISP's network. This system

Table III: Applications' Features

<i>App.</i>	$C_v(CPU)$	$C_v(RAM)$	$C_v(bandwidth)$	<i>Probability</i>
App1	5%	5%	5%	20%
App2	5%	5%	5%	20%
App3	5%	10%	100%	8%
App4	5%	5%	100%	30%
App5	5%	10%	100%	8%
App6	10%	15%	5%	4%
App7	5%	5%	5%	10%

consists of physical devices installed in different locations in the network, where each of them have the Applications' resource consumption (modeled as a percentage of total available resources on a machine), execution duration, period and interval of tests are described in Tables II and III. Particularly, it can be noted that 100% of the bandwidth is reserved for App3, App4 and App5 to be sure that they execute alone so that there is no test interference. For reproducibility matters, we generate a set of instances based on this catalog to perform our experiments. The number of applications to be deployed and scheduled in the network ranges from 50 to 250. They are generated with an estimated probability of being chosen from the catalog based upon typical users' choices (Table III). Additionally, each application requires 1% of each resource to migrate into or out of a machine, and takes at most one minute to do so. Their acceptable delay in minutes is $d_v = \arg \min\{p_v - h_v, 25\}$. The time horizon considered varies from 3 hours to 24 hours, with discrete time intervals of one minute, i.e., $|\mathcal{T}| = \{180, 480, 720, 1440\}$. The number of machines varies from 5 to 25, and at time t^0 they all have 100% of all resources (CPU, RAM and bandwidth).

We present, in the following, experimental results of our optimization problem. All experiments were run in a machine with the following characteristics: Intel(R) Xeon(R) CPU (E5-2650 v2 - 8C/16T - 2.60 GHz) and 252 GB RAM.

A. Experimental results

We compare our ACS algorithm with solutions from the Branch-and-Bound (BB) algorithm of IBM CPLEX and from the First Fit Decreasing (FFD) algorithm [33]. The BB algorithm gives us a ratio of how far from the optimal are the approximate resolutions (gap). The FFD is a well known and largely used classical heuristic of the bin-packing problem. We adapt it such that: applications are sorted into the decreasing order of the scalar value of their $|\mathcal{R}|$ -dimensional resource requirement vector (C_{vr}). For that we use the L1-norm. Then, in this order, each of their tests is assigned to the first machine $m \in \mathcal{M}$ and start time $\hat{t} : \hat{t} \in \mathcal{T}, \hat{t}_{vk} \leq \hat{t} \leq \hat{t}_{vk} + d_v$ where no constraints are violated. We define the migration costs' weight ω_1 high enough, compared to delay costs' weight ω_2 , so that it would prevent a migration from happening unless it is absolutely necessary: $\omega_1 = \frac{|\mathcal{T}|}{2}$ and $\omega_2 = 1$. Since an acceptable start time delay is given, we prefer to have high delays than container migrations. Given the high complexity of solving our optimization problem (Section V), a limit of 120 hours (real time) was established to the exact resolution. More

Table IV: Experimental Results. (OF values are presented in the form of *(total number of migrations + sum of delays)*)

Instance	OF	CPU time (hrs)
V50_M5_T180	4 + 306	38
V50_M5_T480	0 + 569	2493

(a) Results using Branch-and-Bound – relative gap tolerance of 100%

Instance	OF (Best Average Worst)			CPU time (s)
V50_M5_T180	0 + 233	0 + 245	0 + 251	180
V50_M5_T480	0 + 480	0 + 502	0 + 529	180
V50_M5_T720	0 + 766	0 + 815	0 + 859	180
V50_M5_T1440	0 + 1689	0 + 1845	0 + 2122	180
V100_M10_T180	0+530	0 + 573	0 + 622	180
V100_M10_T480	0+1078	0.2+1162	2 + 1180	180
V100_M10_T720	0+1846	1.2+2060	4 + 1963	180
V100_M10_T1440	0+4608	5 + 4167	12+4314	180
V250_M25_T180	6 + 1602	10+1760	16+1874	180
V250_M25_T480	14+2927	22+3029	34+3015	180
V250_M25_T720	22+5219	33+5449	58+5395	180
V250_M25_T1440	40+10814	57+11212	84+11243	180

(b) Results using ACS-based algorithm over 10 runs.

Instance	OF	CPU time (s)
V50_M5_T180	30 + 640	0.02
V50_M5_T480	10 + 1019	0.05
V50_M5_T720	30 + 1918	0.08
V50_M5_T1440	30 + 3835	0.16
V100_M10_T180	60 + 820	0.08
V100_M10_T480	20 + 1738	0.21
V100_M10_T720	60 + 2396	0.3
V100_M10_T1440	60 + 4750	0.64
V250_M25_T180	150 + 3614	0.57
V250_M25_T480	50 + 6028	1.61
V250_M25_T720	150 + 9750	2.5
V250_M25_T1440	150 + 18804	5.23

(c) Results using First Fit Decreasing.

than that, we define the relative mixed integer programming gap tolerance parameter of CPLEX to equal 1.0 to instruct CPLEX to stop as soon as it has found a feasible integer solution, i.e., a solution with relative gap tolerance of a 100%.

Results are given in Table IV. We study the solution quality in terms of number of migrations and tests' start delays produced by the proposed allocating and scheduling resolutions. For each instance, the best OF value found is given in bold (when comparing all three approaches of resolution). As expected, data in Table IVa show that CPLEX's BB algorithm takes unreasonable time. It finds the first integer solution in approximately 38 CPU hours for the smallest instance, and in 2493 CPU hours for the second smallest one. All other generated instances could not have been solved under the given time limit. Table IVb presents results for our ACS algorithm. For each instance, the best, average and worst values of OF over 10 runs are reported. ACS parameters values are set as:

$$\alpha = 2; \quad \rho = 0.7; \quad Ants = 30; \quad \tau_0 = \frac{1}{|\mathcal{T}| \times |\mathcal{V}|}$$

$$\beta = 2; \quad q_o = 0.7; \quad max_evap = 30;$$

The values of these parameters are chosen through empirical observation. Our goal is to allow better exploitation of the solutions while avoiding local stagnation by using a high pheromone evaporation value. The stopping criterion consid-

ered is either an OF value that equals zero or a CPU time limit of 3 minutes. The presented results corroborate our proposed algorithm considering that we are able to obtain high-quality solutions under 3 minutes. Solutions found by our ACS-based algorithm outperformed the BB and FFD ones. Table IVc presents results using the FFD algorithm. Although this algorithm is the fastest among the others, it finds solutions that are sub-optimal comparing to our ACS-based algorithm.

A few issues arise regarding the optimal performance and efficiency of our ACS algorithm. Although it finds the best solutions, it takes much more time than the FFD algorithm. In future investigations we intend to do some studies on the convergence of our algorithm over time. Additionally, mostly regarding the larger instances, different solutions with a higher variation of the OF values was found for the same instance. This can be explained by the fact that by design the ACS metaheuristic is sensitive to the chosen parameters. They define the behavior of each ant, and therefore how fast it will converge to near optimal solutions of a given problem instance. This phenomenon has been already pointed out in the literature [34]–[36]. Further research should analyse the impact of the values of these parameters on solutions and determine how to make an optimal setting depending on the instance considered. Furthermore, because we are dealing with an NP-Hard problem, a resolution by exhaustive search (BB algorithm) becomes dreadfully large. Since we could not get the optimum OF value from our exact resolutions we cannot say how close to the optimum our algorithm is. Hence, we could only compare our solutions to those of the FFD. Therefore, in the future, we would like to use some decomposition techniques in order to solve it optimally.

IX. CONCLUSION

The rise of distributed measurement platforms leads to new challenges. One of them lies in the optimization of dynamically allocating resources and scheduling tests. To face this issue, we have proposed an innovative solution to the orchestration problem of measurement applications. For that we have defined a Linear Programming (LP) model and evaluated its complexity by proving it to be NP-Hard. We have proposed an approximate resolution based on the Ant Colony System (ACS) metaheuristic, and presented an open-source distributed measurement platform that implements it. The ACS-based algorithm has resulted in an efficient strategy to enhance our model's scalability by finding high-quality solutions in reasonable time. Results showed it outperformed a Branch-and-Bound and a First Fit Decreasing algorithm. Despite these promising results, some questions remain. ACS metaheuristic is known to be sensitive to its various parameters. Further research will, therefore, analyze their impact on performance and how to tune them adequately, along with a study of its convergence over time. Additionally, future work will investigate some decomposition techniques for solving our LP model optimally, therefore giving an account on the ACS's performance when compared to the optimal solution.

REFERENCES

- [1] “Samknows,” 2019. [Online]. Available: <https://samknows.com/>
- [2] “ip-label,” 2019. [Online]. Available: <https://www.ip-label.fr/>
- [3] “RIPE NCC ripe atlas,” 2019. [Online]. Available: <https://atlas.ripe.net>
- [4] C. Anderson, P. Winter *et al.*, “Global network interference detection over the {RIPE} atlas network,” in *4th {USENIX} Workshop on Free and Open Communications on the Internet ({FOCI} 14)*, 2014.
- [5] G. Da Lozzo, G. Di Battista, and C. Squarcella, “Visual discovery of the correlation between bgp routing and round-trip delay active measurements,” *Computing*, vol. 96, no. 1, pp. 67–77, 2014.
- [6] T. Yakimov, J. van der Ham, and B. van Kampen, “Detecting routing anomalies with ripe atlas,” 2014.
- [7] A. Faggiani, E. Gregori, A. Improta, L. Lenzini, V. Luconi, and L. Sani, “A study on traceroute potentiality in revealing the internet as-level topology,” in *2014 IFIP Networking Conference*. IEEE, 2014, pp. 1–9.
- [8] S. Roy and N. Feamster, “Characterizing correlated latency anomalies in broadband access networks,” in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 525–526.
- [9] “Docker,” 2019. [Online]. Available: <https://www.docker.com/>
- [10] “Kubernetes,” 2019. [Online]. Available: <https://kubernetes.io/>
- [11] V. Paxson, J. Mahdavi, A. Adams, and M. Mathis, “An architecture for large scale internet measurement,” *IEEE Communications Magazine*, vol. 36, no. 8, pp. 48–54, 1998.
- [12] K. Claffy, Y. Hyun, K. Keys, M. Fomenkov, and D. Krioukov, “Internet mapping: from art to science,” in *2009 Cybersecurity Applications & Technology Conference for Homeland Security*. IEEE, 2009, pp. 205–211.
- [13] V. Bajpai and J. Schönwälder, “A survey on internet performance measurement platforms and related standardization efforts,” *IEEE Communications Surveys & Tutorials*, vol. 17, no. 3, pp. 1313–1341, 2015.
- [14] S. Kulkarni, E. Bachmakov, E. Gallagher, V. Gehlot, A. Dammann, P. Rokowski, and J. Brzozowski, “A scalable architecture for performance measurement in broadband networks,” in *2015 IEEE Conference on Standards for Communications and Networking (CSCN)*. IEEE, 2015, pp. 174–180.
- [15] Y. I. Wijata, D. Niehaus, and V. S. Frost, “A scalable agent-based network measurement infrastructure,” *IEEE Communications Magazine*, vol. 38, no. 9, pp. 174–183, 2000.
- [16] M. A. Sánchez, J. S. Otto, Z. S. Bischof, D. R. Choffnes, F. E. Bustamante, B. Krishnamurthy, and W. Willinger, “Dasu: Pushing experiments to the internet’s edge,” in *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, 2013, pp. 487–499.
- [17] T. Holterbach, C. Pelsser, R. Bush, and L. Vanbever, “Quantifying interference between measurements on the ripe atlas platform,” in *Proceedings of the 2015 Internet Measurement Conference*. ACM, 2015, pp. 437–443.
- [18] M. A. Sánchez, F. E. Bustamante, B. Krishnamurthy, and W. Willinger, “Experiment coordination for large-scale measurement platforms,” in *Proceedings of the 2015 ACM SIGCOMM Workshop on Crowdsourcing and Crowdsharing of Big (Internet) Data*. ACM, 2015, pp. 21–26.
- [19] S. Gangam and S. Fahmy, “Mitigating interference in a network measurement service,” in *Proceedings of the Nineteenth International Workshop on Quality of Service*. IEEE Press, 2011, p. 37.
- [20] Z. Qin, R. Rojas-Cessa, and N. Ansari, “Descending-order clique-based task scheduling for active measurements,” in *2007 Workshop on High Performance Switching and Routing*. IEEE, 2007, pp. 1–6.
- [21] P. Calyam, L. Kumarasamy, and F. Ozguner, “Semantic scheduling of active measurements for meeting network monitoring objectives,” in *2010 International Conference on Network and Service Management*. IEEE, 2010, pp. 435–438.
- [22] P. Calyam, C.-G. Lee, P. K. Arava, and D. Krymskiy, “Enhanced edf scheduling algorithms for orchestrating network-wide active measurements,” in *26th IEEE International Real-Time Systems Symposium (RTSS’05)*. IEEE, 2005, pp. 10–pp.
- [23] M. R. Garey and D. S. Johnson, “Computers and intractability,” vol. 29, 2002.
- [24] M. Dorigo and T. Stützle, “Ant colony optimization,” 2004.
- [25] M. Dorigo, V. Maniezzo, A. Colomi *et al.*, “Ant system: optimization by a colony of cooperating agents,” *IEEE Transactions on Systems, man, and cybernetics, Part B: Cybernetics*, vol. 26, no. 1, pp. 29–41, 1996.
- [26] M. Dorigo, G. D. Caro, and L. M. Gambardella, “Ant algorithms for discrete optimization,” *Artificial life*, vol. 5, no. 2, pp. 137–172, 1999.
- [27] M. Dorigo and C. Blum, “Ant colony optimization theory: A survey,” *Theoretical computer science*, vol. 344, no. 2-3, pp. 243–278, 2005.
- [28] “GitHub NMaaS open source by orange,” 2019. [Online]. Available: <https://github.com/Orange-OpenSource/NMaaS>
- [29] “Ansible,” 2019. [Online]. Available: <https://www.ansible.com/>
- [30] “Prometheus,” 2019. [Online]. Available: <https://prometheus.io/>
- [31] “Grafana,” 2019. [Online]. Available: <https://grafana.com/>
- [32] “Rancher,” 2019. [Online]. Available: <https://rancher.com/>
- [33] G. Dósa, “The tight bound of first fit decreasing bin-packing algorithm is $\text{ffd}(i) \leq 11/9 \text{opt}(i) + 6/9$,” in *International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*. Springer, 2007, pp. 1–11.
- [34] P. Li and H. Zhu, “Parameter selection for ant colony algorithm based on bacterial foraging algorithm,” *Mathematical Problems in Engineering*, vol. 2016, 2016.
- [35] D. Gaertner and K. Clark, “On optimal parameters for ant colony optimization algorithms,” vol. 1, 01 2005, pp. 83–89.
- [36] D. Gomez-Cabrero and D. Ranasinghe, “Fine-tuning the ant colony system algorithm through particle swarm optimization,” *arXiv preprint arXiv:1803.08353*, 2018.