Future Generation Computer Systems 79 (2018) 739–750

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms

Maria A. Rodriguez *, Rajkumar Buyya

Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Australia

HIGHLIGHTS

- A dynamic and scalable algorithm to schedule multiple workflows is presented.
- The algorithm is designed for multi-tenant Workflow as a Service platforms.
- It aims to minimize the total cost of leased resources while meeting the individual deadline of workflows.
- The use of containers is proposed to address resource usage inefficiencies.

ARTICLE INFO

Article history:
Received 31 January 2017
Received in revised form
10 April 2017
Accepted 7 May 2017
Available online 10 May 2017

Keywords:
Cloud computing
Cost minimization
Deadline
Workflow as a service
Resource provisioning
Scheduling

ABSTRACT

With the advent of cloud computing and the availability of data collected from increasingly powerful scientific instruments, workflows have become a prevailing mean to achieve significant scientific advances at an increased pace. Emerging Workflow as a Service (WaaS) platforms offer scientists a simple, easily accessible, and cost-effective way of deploying their applications in the cloud at anytime and from anywhere. They are multi-tenant frameworks and are designed to manage the execution of a continuous workload of heterogeneous workflows. To achieve this, they leverage the compute, storage, and network resources offered by Infrastructure as a Service (IaaS) providers. Hence, at any given point in time, a WaaS platform should be capable of efficiently scheduling an arbitrarily large number of workflows with different characteristics and quality of service requirements. As a result, we propose a resource provisioning and scheduling strategy designed specifically for WaaS environments. The algorithm is scalable and dynamic to adapt to changes in the environment and workload. It leverages containers to address resource utilization inefficiencies and aims to minimize the overall cost of leasing the infrastructure resources while meeting the deadline constraint of each individual workflow. To the best of our knowledge, this is the first approach that explicitly addresses VM sharing in the context of WaaS by modeling the use of containers in the resource provisioning and scheduling heuristics. Our simulation results demonstrate its responsiveness to environmental uncertainties, its ability to meet deadlines, and its cost-efficiency when compared to a state-of-the-art algorithm.

1. Introduction

Workflows are defined by a set of computational tasks with dependencies between them and are a commonly used application model in computational science. They enable the analysis of data in a structured and distributed manner and have been successfully used to make significant scientific advances in various fields such as biology, physics, medicine, and astronomy [1]. Their importance is highlighted in todays big data era as they offer an efficient way of processing and extracting knowledge from the data produced by increasingly powerful tools such as telescopes, particle accelerators, and gravitational wave detectors. Hence, it is common for scientific workflows to be large-scale data and compute intensive applications that are deployed on distributed environments in order to produce results in a reasonable amount of time.

The emergence of cloud computing has brought with it several advantages for the deployment of scientific workflows. In particular, Infrastructure as a Service (IaaS) clouds allow Workflow Management Systems (WMSs) to access a virtually infinite pool of resources that can be acquired, configured, and used as needed and are charged on a pay-per-use basis. IaaS providers offer virtualized

* Corresponding author.
E-mail addresses: marodriguez@unimelb.edu.au.com (M.A. Rodriguez), rbuyya@unimelb.edu.au.com (R. Buyya).

http://dx.doi.org/10.1016/j.future.2017.05.009
0167-739X/

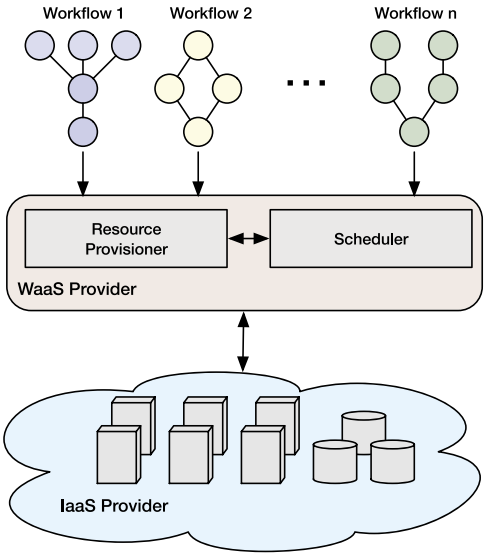

Fig. 1. High-level overview of a workflow as a service application scenario.

compute resources called Virtual Machines (VMs) for lease. They have a predefined CPU, memory, storage, and bandwidth capacity and different resource bundles (i.e., VM types) are available at varying prices. They can be elastically acquired and released and are generally charged per time frame, or billing period. While VMs deliver the compute power, IaaS clouds also offer storage and networking services, providing the necessary infrastructure for the execution of workflow applications.

Scheduling algorithms tailored for scientific workflows are crucial in taking advantage of the benefits offered by clouds and they have been widely studied in recent years. To achieve this, they need not only to focus on the task to resource mapping but also on deciding the number and type of resources to use throughout the execution of the workflow (i.e., resource provisioning). The majority of existing approaches focus on generating resource provisioning and scheduling plans for a single instance of a workflow. They assume application and resource models in which a single user submits a single workflow for execution to a WMS. The WMS is then responsible for provisioning the required resources and mapping tasks to them so that the workflow execution is completed within the Quality of Service (QoS) constraints. While this is a valid model, as the adoption of cloud computing becomes more widespread among the scientific community, new application models are emerging.

In particular, Workflow as a Service (WaaS) is an emerging concept in which the execution of workflows is offered as a service to scientists. WaaS can be classified as an offering either at the Platform as a Service or Software as a Service layers as providers make use of compute, storage, and network resources offered by IaaS vendors to fulfill requests sent to a multi-tenant WMS. Workflows submitted to such WMS belong to different users and are not necessarily related to each other; they may vary in structure, size, input data, application, and QoS requirements among other features. As a result, schedulers should be able to process a workload of workflows with different configurations that are continuously arriving for execution (without assuming that the number and type of workflows are known in advance). A high-level overview of a WaaS platform is depicted in Fig. 1. Frameworks realizing this service model are beginning to appear in the literature. For example, Filgueira et al. [2] present a data-intensive workflow as a service model that enables the easy composition and deployment of stream-based workflow applications on cloud platforms using containers. Similarly, Skyport [3] is an execution environment capable of managing the execution of multiple workflows in clouds by leveraging Docker containers to address software deployment problems and resource utilization inefficiencies. Other examples include the middleware described by Esteves and Veiga. [4] and the architecture presented by Wang et al. [5].

Scientific workflows are generally composed of tasks of different types. In practical terms, all tasks of the same type run the same software program; that is, they perform the same set of computations potentially on different data sets. This means that different task types require different software components for their execution. Virtualization allows for the execution environment of these tasks to be easily customized. For instance, hardware-level virtualization can be used in such way that the operating system, software packages, and directory structures, among others, can all be tailored for a specific task and stored as a VM image. This image can then be easily used to deploy VMs capable of executing the task or tasks they were designed for. This is the model considered by the majority of existing workflow scheduling algorithms for clouds. They focus on efficiently leasing and releasing VMs with specific characteristics in order to fulfill a set of QoS requirements and in general assume that all VMs can be deployed using a single VM image that contains all of the software required to execute any workflow task. This assumption is realistic and reasonable when considering the scheduling of a single workflow but not when scheduling multiple workflows from different users.

The main reason for this is the impracticality of tailoring a single VM image to support the execution of different tasks from different workflows (e.g., consider the size of the image and the incompatibility between software components required by different tasks from different workflows). WaaS platforms can adopt different approaches to circumvent this issue. An option is to execute each individual workflow on its own set of dedicated VMs or a set of related workflows on their own dedicated VMs, however, this may result in an inefficient use of resources and higher costs. Another option that addresses this issue is to combine the use of VMs and containers, which are a form of operating system-level virtualization. Containers allow applications to be packaged and configured by providing a virtual environment that has its own CPU, memory, block I/O, and network space. By allowing each task or workflow to have a corresponding container image, a VM can be reused to run tasks belonging to different workflows by launching the corresponding container when a task is due to start its execution on that resource. In this way, resource utilization is maximized by reducing the wastage of idle time slots on leased VMs.

In addition to having a well-defined VM sharing model, algorithms tailored for WaaS platforms should be dynamic as they have no knowledge on the arriving workflows. They should also be scalable and capable of making decisions quickly as the number of tasks that need to be processed at any given point in time may be very large. Another important factor that should be taken into consideration is the efficient auto-scaling and management of VMs in order to increase their utilization as a cost controlling mechanism while still being able to satisfy the QoS requirements of individual workflows. This will potentially result in lower costs for users and higher profit for providers. Finally, algorithms should also address common challenges derived from the resource model offered by clouds such as the abundance and heterogeneity of resources, uncertainties derived from performance variation, VM provisioning delays, and billing period pricing models.

In response to these requirements, we propose EPSM, an Elastic resource Provisioning and Scheduling algorithm for Multiple workflows designed for WaaS platforms. It considers containers to address resource utilization inefficiencies and aims to minimize the overall cost of leasing resources while meeting the independent deadline constraint of workflows continuously arriving for execution. Although there are some existing algorithms designed to schedule multiple workflows, they either explicitly or implicitly assume that each workflow, or workflow type in some cases

(i.e., same workflow but different number of tasks), has its own designated resources. To the best of our knowledge, this is the first approach that explicitly addresses VM sharing in the context of WaaS by modeling the use of containers in the resource provisioning and scheduling heuristics. Furthermore, the algorithm is dynamic and scalable and our simulation results demonstrate its responsiveness to environmental uncertainties, its ability to meet deadlines, and its cost-efficiency when compared to a state-of-the-art algorithm.

2. Related work

The majority of algorithms in the literature focus on optimizing the execution of a single workflow with its own QoS requirements. Hence, the resources are used exclusively for the execution of a single application belonging to a single user. Most of these algorithms have as objectives minimizing the total execution cost while meeting a deadline constraint. Examples include IC-PCP and IC-PCPD2 [6], EIPR [7], TB [8], the approach proposed by Dziok et al. [9], and CCA [10]. To achieve this, they have policies in place to elastically acquire and release resources and focus mostly on reusing idle time slots of provisioned VMs when possible in order to maximize resource utilization and save cost. However, the deadline constraint requirements and the dependencies between tasks mean that unused time slots cannot be fully eliminated.

Some scientific applications [11–13] are composed of interrelated workflow instances known as ensembles. These workflows are grouped together because their combined execution produces a desired output [14]. There are several scheduling algorithms designed for this type of applications in the literature [14–17]. They differ from the solution presented in this paper in three ways. Firstly, the QoS requirements are not defined for each workflow instance, but rather for the entire ensemble. Hence, algorithms are generally concerned with the amount of work (number of executed workflows) completed and tend to include this in the scheduling objectives. Secondly, the number of workflow instances is generally known in advance and the scheduling strategies may use this when planning the execution of tasks. Finally, the workflows in an ensemble are of the same type, meaning they have a similar structure but differ in size and input data.

Yu and Shi [18] considered an application model similar to the one addressed in this paper and proposed a scheduling algorithm for multiple workflow applications submitted at different times by different users. However, their solution is tailored for cluster environments and as a result does not consider the cost of leasing the infrastructure and it assumes a fixed number of resources that are readily available. Also, the algorithm's objective is to minimize the makespan (i.e., the total execution time) of each workflow instead of meeting a deadline constraint. Despite these differences, we consider their work relevant as the authors not only identify the need for such type or algorithms, even in cluster environments, but also identify some key issues and characteristics that need to be considered regardless of the distributed platform such as the need for the algorithm to be dynamic and the importance of considering the overall resource utilization from a system management perspective.

Several works consider the scheduling of multiple workflows in clouds. An example is the work by Jiang et al. [19], however, the application model they consider is different to the one addressed in this paper as they assume the number and type of workflows are known in advance and that all of the workflows are submitted for execution at the same time. Also, their algorithm does not consider cost nor deadlines as its only objective is to improve the utilization of resources. Another algorithm is proposed by Stavrinides and Karatza [20], it is based on a list scheduling heuristic and their application model is similar to ours but differs in the fact that they consider the quality of the data produced by each workflow as part of the QoS requirements. Similarly, Xu et al. [21] propose a strategy with different objectives as they consider a budget constraint as part of the QoS. Finally, Chen et al. [22] present an algorithm that has the same application model and scheduling objectives as our proposed solution. However, their solution as well as the aforementioned solutions, differ from our proposal in a key aspect. They assume a finite set of heterogeneous VMs that is available throughout the entire lifetime of the system and hence do not consider the resource provisioning problem under elastic and abundant resources.

On the contrary, Dyna [23] is a scheduling algorithm designed for cloud workflow service providers with auto-scaling features to dynamically allocate and deallocate VMs based on the current status of tasks. It works by selecting VM types for each workflow task based on an A star search so that the cost is minimized. However, it differs from our approach in that it offers probabilistic deadline guarantees and considers VMs priced under two different models: static (e.g., Amazon on-demand instances) and dynamic (e.g., Amazon spot instances). Also, although the VM sharing policy is not explicitly stated in the paper, it can be inferred that a model in which VMs are shared only between workflows of the same type is used. This based on the resource model considered by the authors and the evaluation which is performed with a workload consisting of the same type of workflow but with different number of tasks.

SCS [24] and WPPDS [25] are also capable of scheduling a workload of workflows in clouds and have an auto-scaling mechanism. SCS makes an initial resource provisioning plan based on a global optimization heuristic and then refines it at runtime to respond to delays that were unaccounted for. However, the refinement of the provisioning plan is done by running the global optimization algorithm for the remaining tasks every time a task is scheduled. This introduces a high computational overhead and hinders its scalability in terms of the number of tasks in the workflow. WPPDS on the other hand considers a budget for the entire workload and individual workflow deadlines and its objective is to finish as many high-priority workflows as possible with the given budget.

Wang et al. [5] proposed an architecture for a WaaS system along with four heuristic-based scheduling algorithms: static, dynamic, adaptive, and greedy. They differ from our proposal in that they only allow VMs to be shared between tasks of the same workflow but not between tasks belonging to different workflows. Also, the objectives of their proposed algorithms are to minimize makespan and cost and they do not consider VM provisioning delays or data transfer times. It is worth noticing that out of the surveyed algorithms, this is the only work that explicitly defines a VM sharing policy. All of the other algorithms either naively assume that any task can be deployed on any of the available VMs or fail to adequately define their application model.

Finally, Skyport [3] and Asterism DIaaS [2] are other examples of WaaS frameworks. They are relevant to this work in that they identify the benefits and need of using containers but focus on how they can be used to package workflow tasks and the convenience of deploying them on already-provisioned VMs so that these are capable of executing any tasks from any workflow. They do no focus on the resource provisioning and scheduling problems addressed in this paper. Esteves and Veiga [4] also define a prototypical middleware framework that embodies the vision of a WaaS system and address issues such as workflow description and WMS integration, cost model, and resource allocation. Their work focuses on workflows for continuous and incremental processing of data, that is, workflows in which the end of the execution of a task does not immediately trigger its successor tasks, instead, they are only triggered when the task has generated output data with sufficient impact in relation to the terminal task of the workflow.

Although the authors emphasize on the design of a WaaS platform, their proposed scheduling algorithm focuses on a single workflow.

3. Application and resource models

This work is designed to schedule a continuous workload of scientific workflows submitted by users to a WaaS provider. The workflows may have different characteristics such as application type, number of tasks, input data, and deadline. The WaaS provider leases resources from a public IaaS vendor to fulfill the users' requests and its goal is to minimize the total cost of renting infrastructure resources while meeting the deadline constraint of each of the submitted workflow applications.

Workflows are modeled as Directed Acyclic Graphs (DAGs); that is, graphs with directed edges and no cycles or conditional dependencies. At any given point in time, there is a set $W = \{w_1, w_2, \ldots, w_n\}$ of workflows that need to be scheduled. Formally, a workflow w is composed of a set of tasks $T = \{t_1, t_2, \ldots, t_n\}$ and a set of edges E. An edge $e_{ij} = (t_i, t_j)$ exists if there is a data dependency between t_i and t_j, case in which t_i is said to be the parent task of t_j and t_j the child task of t_i. Based on this, a child task cannot run until all of its parent tasks have completed their execution and its input data is available in the corresponding compute resource. Finally, each workflow is associated with a deadline δ_w, defined as a soft time limit for the execution of the workflow and a container c_w that contains all the libraries and software required to execute any of the workflow tasks.

We consider a model in which tasks execute within containers which in turn are deployed on VMs. A container can be deployed on a VM at any time with a provisioning delay $prov_c$. This delay corresponds to the time it takes to download the container image from a global storage system such as Amazon S3 and initialize it on the VM. We consider a model where only one container can be deployed on a VM at a given point in time and as a result, we assume containers inherit the same CPU and bandwidth capacity of the host VM. We assume task executions can be triggered by external schedulers by sending custom signals to the container by using commands such as Docker exec. As a result, multiple tasks can be executed sequentially on one container without the need of redeployment.

We assume a pay-as-you go model where VMs are leased on-demand and are charged per billing period τ. We acknowledge that any partial utilization results in the VM usage being rounded up to the nearest billing period. We consider a single cloud provider and a single data center or availability zone. In this way, network delays are reduced and intermediate data transfer fees eliminated. Finally, we impose no limit on the number of VMs that can be leased from the provider.

The IaaS provider offers a range of VM types $VMT = \{vmt_1, vmt_2, \ldots, vmt_n\}$ with different prices and configurations.VM types are defined in terms of their cost per billing period c_{vmt}, CPU processing capacity p_{vmt}, and bandwidth capacity b_{vmt}. An average measure of their provisioning $prov_{vmt}$ delay is also included as part of their definition. The execution time, E_t^{vmt}, of every task on every VM type is available to the scheduler. Different performance estimation methods can be used to obtain this value, in our approach we calculate it based on an estimate of the size of the task (I_t, in millions of instructions) and the CPU capacity of the VM type (in million of instructions per second) as shown in Eq. (1). Note that our solution acknowledges that this value is simply an estimate and does not rely on it being one hundred percent accurate to achieve its objectives.

$$E_t^{vmt} = I_t / p_{vmt}. \tag{1}$$

Based on the characterization of workflow tasks performed by Juve at al. [26] and the VM types offered by Amazon EC2, we assume that all VM types have sufficient memory to execute any of the workflows' tasks. However, as a future work we will consider extending the algorithm to include heuristics that ensure tasks are assigned to VMs with sufficient memory to execute them. Additionally, we assume VMs have a single core for scheduling purposes and hence are only capable of processing one task at a time.

We define the sharing of data between tasks to take place via a global storage system such as Amazon S3. In this way, tasks store their output in the global storage and retrieve their inputs from the same. We assume a global storage system with sufficient storage capacity and reading and writing rates of GS_r and GS_w respectively. The time it takes to transfer and write d output data from a VM of type vmt into the storage is defined as

$$N_{d,vmt}^{out} = (d/b_{vmt}) + (d/GS_w). \tag{2}$$

Similarly, the time it takes to transfer and read a task's input data from the storage to a VM of type vmt is defined as

$$N_{d,vmt}^{in} = (d/b_{vmt}) + (d/GS_r). \tag{3}$$

We acknowledge that characteristics such as virtualization, multi-tenancy, and the heterogeneity of non-virtualized hardware in clouds result in variability in the performance of resources [27–31]. In particular, we assume a variation in the performance of network resources and VM CPUs with their maximum achievable performance being based on the bandwidth and CPU capacity advertised by the provider. Ultimately, this results in a degradation of data transfer times and task execution times. We do not assume there is an additional degradation in performance due to the use of containerized environments [32,33].

As shown in Eq. (4), the total processing time PT_t^{vmt} of task t on a VM of type vmt is calculated as the sum of the task's execution time and the time it takes to read the required n_{in} input files from the storage and write n_{out} output files to it. Notice that there is no need to read an input file whenever it is already available in the VM where the task will execute. This occurs when parent and child tasks run on the same resource.

$$P_t^{vmt} = E_t^{vmt} + \left(\sum_{i=1}^{n_{in}} N_{d_i,vmt}^{in}\right) + \left(\sum_{i=1}^{n_{out}} N_{d_i,vmt}^{out}\right). \tag{4}$$

The cost of using a resource r_{vmt} of type vmt for $lease_r$ time units is defined as

$$C_{r_{vmt}} = \lceil (prov_{vmt} + lease_r)/\tau \rceil * c_{vmt}. \tag{5}$$

Finally, we assume data transfers in and out of the global storage system are free of charge, as is the case for products like Amazon S3, Google Cloud Storage and Rackspace Block Storage. As for the actual data storage, most cloud providers charge based on the amount of data being stored. We do not include this cost in the total cost calculation of neither our implementation nor the implementation of the algorithms used for comparison in the experiments. The reason for this is to be able to compare our approach with others designed to transfer files in a peer-to-peer fashion. Furthermore, regardless of the algorithm, the amount of stored data for a given workflow is most likely the same in every case or it is similar enough that it does not result in a difference in cost.

4. The EPSM algorithm

We propose EPSM, a dynamic heuristic-based algorithm that makes resource provisioning and scheduling decisions to satisfy the deadline of individual workflows while minimizing the cost of leasing VMs. Its simplicity was a main design goal to facilitate its implementation in real-world WaaS frameworks and to ensure

its scalability with respect to the number of workflows and tasks. Overall, the algorithm maintains a pool of resources which is scaled in and out based on the current requirements of tasks that are ready for execution. Its main goal is to efficiently utilize these resources as a cost-controlling mechanism without compromising the deadline of workflows. A high-level overview of a scheduling scenario is depicted in Fig. 2 and the detailed strategy is as follows.

When a workflow is submitted to the scheduler, it is preprocessed and a sub-deadline is assigned to each task. This sub-deadline will guide the decisions made at runtime when mapping each task onto either an existing or a newly provisioned resource. The first step of the deadline distribution strategy is to calculate the earliest finish time for each workflow task defined as $eft_t = \max_{p \in t.parents}\{eft_p\} + PT_t^{vmt}$, where *vmt* corresponds to the VM type with the least amount of CPU capacity. For simplicity, from here on, we will refer to this VM type as the slowest type and to the VM type with the most amount of CPU capacity as the fastest one. In this way, the task runtime estimated using *vmt* leads to the largest value (slowest runtime) but potentially the lowest cost (assuming the price is proportional to the CPU capacity).

Afterwards, the makespan of the workflow, defined as $m_w = \max_{t \in T}\{eft_t\}$, is calculated. If this value exceeds the workflow's deadline, then the earliest finish time of tasks is recalculated using the next fastest VM type until the makespan is smaller than or equal to the workflow deadline. We assume the deadline is always sufficient and hence do not consider cases in which the fastest available VM type still leads to a makespan that is larger than the deadline. An option for WaaS providers in this case would be to reject the execution of the workflow or renegotiate the QoS requirements with users.

After obtaining a suitable makespan, the amount of spare time available defined as the difference between the deadline and the makespan (i.e., $\delta_w - m_w$) is calculated. This spare time is then distributed to individual tasks in a way that is proportional to their runtime, that is, tasks with longer runtimes get assigned a larger portion of the spare time compared to tasks with smaller runtimes. Finally, each task is assigned a deadline $\delta_t = t.start + PT_t^{vmt} + t.spare$ where $t.start$ is the task's start time and $t.spare$ is the spare time assigned to the task.

Once a DAG is preprocessed the scheduling of its tasks can begin, this process is illustrated in Algorithm 1. Its main goal is to avoid leasing new VMs when possible and instead reuse existing ones. In this way, the impact of VM provisioning delays in terms of cost and uncertainty are reduced and the resources are better utilized. This ultimately leads to a smaller number of VMs used and less billing periods consumed.

At first, all the entry tasks (those that have no parent tasks) in the workflow become ready for execution and are placed in a scheduling queue. As the execution of the workflow progresses and tasks are completed, child tasks that are ready to run (i.e., those for which all parent tasks have completed their execution) are released onto the queue. As a result, at any given point in time, this queue contains all the tasks from all the workflows submitted to the platform that are ready to be scheduled.

Every scheduling cycle, which occurs every $sched_{int}$, each task in the queue is processed in the following way. The first step is to find an idle VM that can finish the task on time with minimum cost and schedule the task on it. When estimating the time taken to execute a task on an existing VM, not only is PT_t^{vmt} taken into consideration, but also $prov_c$ in cases in which the container required to execute the task needs to be deployed on the VM. As for the cost, if the task can finish before the VM's next billing cycle, then the cost is estimated as zero, otherwise, it is calculated based on Eq. (6), where $remaining_r$ corresponds to the time remaining in the idle VM until its next billing cycle,

$$C_{r_{vmt}}^t = \lceil (P_t^{vmt} - remaining_r)/\tau \rceil * c_{vmt}. \quad (6)$$

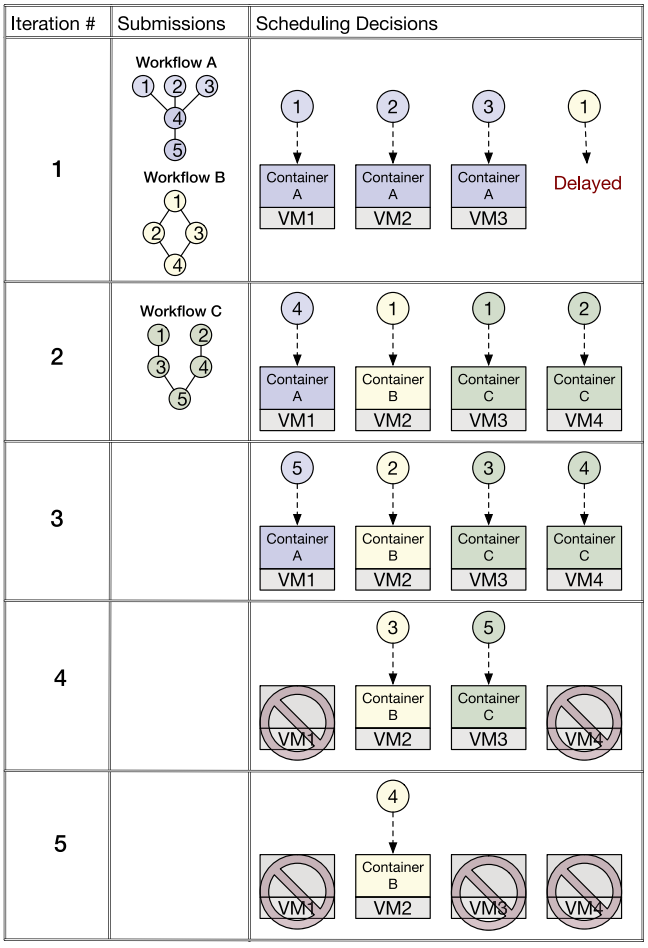

Fig. 2. Sample scheduling and resource provisioning scenario using EPSM.

The idle VM is first looked for in the set VM_{idle}^{input} which is composed of all currently idle VMs that contain all or part of the task's input data. In this way, child and parent tasks are always encouraged to run on the same VM. The reasons for this are to reduce the use of the data center networks as they are well-known bottlenecks and sources of uncertainty, to reduce the processing time of tasks as the input data does not need to be transferred from the global storage system, and as a result to reduce cost by incurring in less billing periods. Additionally, by considering container provisioning delays when estimating the runtimes and cost of tasks on leased VMs, an idle VM with the corresponding container deployed on it will always be favored as long as it does not lead to a violation of the deadline.

If no suitable VM is found in VM_{idle}^{input}, then the algorithm tries to reuse one from the set $VM_{idle}^{container}$ containing all the idle VMs in which the container associated to the task's workflow is currently deployed. In this way the container provisioning delay $prov_c$ is eliminated. If the set does not contain a VM that can finish the task on time, then the algorithm looks for any remaining existing idle VM that can satisfy the deadline with lowest cost.

If an existing VM is found, then the task is immediately scheduled on it. If not, then as a last resort to reuse a leased VM, the task is delayed to be scheduled in a subsequent scheduling cycle, but only if that does not lead to a potential deadline violation. Hence, the decision to delay a task is made based on the task's runtime on the slowest VM type and the amount of time remaining to complete the task on time. Specifically, if scheduling the task on the next cycle on a VM of the slowest available type still leads to the task finishing by its deadline, then the task is delayed so

Algorithm 1 Scheduling

```
1: procedure SCHEDULEQUEUEDTASKS(q)
2:     sort q by ascending deadline (EDF)
3:     while Q is not empty do
4:         t = q.peek
5:         dag = t.dag
6:         container = dag.container
7:         vm = null
8:         VM_idle = set of all idle VMs
9:         VM_idle^input = set of vm ∈ VM_idle that have t's input data
10:        vm = vm ∈ VM_idle^input that can finish t before t.deadline with minimum
           cost
11:        if vm == null then
12:            VM_idle = VM_idle \ VM_idle^input
13:            VM_idle^container = set of all vm ∈ VM_idle that have container deployed
14:            vm = vm ∈ VM_idle^container that can finish t before t.deadline with
               minimum cost
15:            if vm == null then
16:                VM_idle = VM_idle \ VM_idle^container
17:                vm = vm ∈ VM_idle that can finish t before t.deadline with
                   minimum cost
18:                if vm == null then
19:                    vmt = cheapest VM type
20:                    runtime = P_t^vmt
21:                    remaining = t.deadline − currentTime
22:                    spare = remaining − runtime − schedInterval
23:                    if spare <= 0 then
24:                        vmt = cheapest VM type that can finish t on time
25:                        vm = provisionVM(vmt)
26:                    end if
27:                end if
28:            end if
29:        end if
30:        if vm != null then
31:            if vm.container! = container then
32:                deployContainer(vm, container)
33:            end if
34:            q.poll
35:            scheduleTask(t, vm)
36:        end if
37:    end while
38: end procedure
```

Algorithm 2 Resource Provisioning

```
1: procedure MANAGERESOURCES
2:     VM^idle = all leased VMs that are currently idle
3:     for each vm_idle in VM^idle do
4:         t_r = time remaining until next billing period
5:         t_d = Deprovisioningdelayestimate
6:         if (t_r − t_d ≥ 0) AND
7:                         (t_r − t_d ≤ PROV_POLLING_TIME) then
8:             terminate vm_idle
9:         end if
10:    end for
11: end procedure
```

that it can be potentially scheduled on an existing VM on a later cycle.

If the task cannot be delayed, then the VM type that can finish the task with the least amount of money within its deadline is chosen. If there is no VM type that can finish the task on time, then the VM type that can finish the task the fastest is selected. When estimating the runtime and cost of tasks on different VM types, our approach considers the task's processing time, the VM provisioning delay, and the container initialization delay. A VM of the selected type is then provisioned, the corresponding container deployed, and the task scheduled on it.

To better adapt to unexpected delays and environmental uncertainties, every time a task finishes either earlier or later than expected, the deadline of the remaining workflow tasks is updated. In this way, if a task finishes earlier, child tasks will have more time to run and hence they can either be assigned to a cheaper VM or delayed to be scheduled in subsequent cycles. If a task finishes later than expected, adjusting the deadline of the remaining tasks may prevent the deadline from being missed.

Regarding the resource provisioning strategy, as already stated, VMs are only provisioned when tasks cannot be delayed any further. As for the deprovisioning strategy, leased VMs are monitored every $prov_{int}$ and any VMs that are idle and approaching their next billing cycle are shut down. An overview of this strategy is depicted in Algorithm 2. It is worthwhile mentioning that both scheduling and provisioning intervals ($sched_{int}$ and $prov_{int}$) are configurable parameters that can be provided as input to the algorithm and their sizes lead to a trade-off between performance in terms of cost and makespan and processing time spent on scheduling and provisioning operations.

5. Performance evaluation

The performance of our proposal was evaluated using five well-known workflows from different scientific areas. The Montage application from the astronomy field is used to generate custom mosaics of the sky based on a set of input images. Most of its tasks are characterized by being I/O intensive while not requiring much CPU processing capacity. The Ligo workflow from the astrophysics domain is used to detect gravitational waves. It is composed mostly of CPU intensive tasks with high memory requirements. SIPHT is used in bioinformatics to automate the search for sRNA encoding-genes. Most of the tasks in this workflow have high CPU and low I/O utilization. Also in the bioinformatics domain, the Epigenomics workflow is a CPU intensive application that automates the execution of various genome-sequencing operations. Finally, CyberShake is used to characterize earthquake hazards by generating synthetic seismograms and can be classified as a data intensive workflow with large memory and CPU requirements. Samples of these workflows are depicted in Fig. 3 and their full description and characterization is presented by Juve et al. [26].

The evaluation was performed with different workloads containing a combination of all the aforementioned workflows of five different sizes: extra-small (30 tasks), small (50 tasks), medium (100 tasks), and large (1000 tasks). Each workload is composed of a different number of workflows ranging from 1000 to 4000 and different arrival rates which were modeled following a Poisson distribution. The workflows were generated using the WorkflowGenerator [34] tool provided by the Pegasus group which uses traces of real executions to generate synthetic workflows. For the experiments presented here, we used the runtime generated for each task as the size of the task in Millions of Instructions (MI).

Each workflow in a workload was assigned a deadline. To do this, first the minimum and maximum makespan values were determined for each combination of workflow type and size. The minimum makespan was estimated by simulating the execution of each task on a VM of the fastest type. The maximum time was defined as the execution time resulting from running all tasks sequentially on a single VM of the slowest type. A deadline between these minimum and maximum values was randomly chosen for each workflow based on a uniform distribution. For each workload, the number of workflows of each type and size was selected at random based on a uniform distribution.

We extended CloudSim [35] to support the execution of workflows and containers. An IaaS provider offering a single data center and four types of VMs was modeled. The VM type configurations used are shown in Table 1. Their CPU capacity and price are a simplified version of the compute optimized (c4) instance types offered by Amazon EC2 in which the EC2 Compute Units (which provide a measure of the integer processing power of an instance) and price have a linear relationship. A VM billing period of one hour was modeled and for all VM types, the

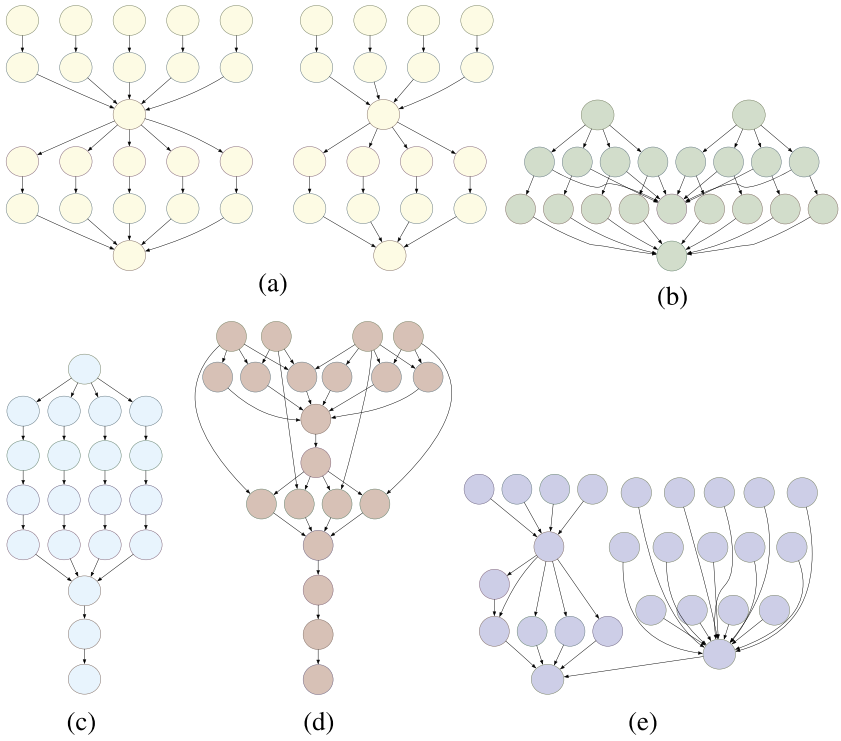

Fig. 3. Sample structure of five scientific workflows used in the evaluation of the algorithms. (a) LIGO. (b) Epigenomics. (c) Montage. (d) CyberShake. (e) SIPHT.

Table 1
Types of VMs used in the evaluation.

Name	CPU capacity (MIPS)	Price per hour
Small	2	$1
Medium	4	$2
Large	8	$4
Extra-large	16	$8

Table 2
Workloads used in the evaluation of the algorithms' performance.

Workload name	# of workflows	# of tasks
Small	1000	307 303
Medium	2000	626 600
Large	4000	1 193 422

provisioning delay was set to 100 s based on the study by Mao et al. [36]. Based on an average container image size of 600 MB, a bandwidth of 500 Mbps, and an initialization delay of 0.4 s [37], the container provisioning delay was set to 10 s. CPU performance variation was modeled after the findings by Schad et al. [27]. The performance of a VM was degraded by at most 24% based on a normal distribution with a 12% mean and a 10% standard deviation. Based on the same study, the bandwidth available for each data transfer within the data center was subject to a degradation of at most 19% based on a normal distribution with a mean of 9.5% and a standard deviation of 5%.

5.1. Algorithm performance

We compare EPSM with Dyna [23], an algorithm developed for a similar application scenario. Both solutions differ however, in two aspects. Dyna was developed to offer probabilistic deadline guarantees and to use not only statically priced VMs (e.g., Amazon on-demand instances), but also VMs priced dynamically (e.g., Amazon spot instances). With simple modifications, we adapted Dyna to consider non-probabilistic deadlines and use statically priced VMs exclusively. Dyna performs an A star search to generate an instance configuration plan for every task associating it to a VM type. At runtime, this configuration plan, in addition to instance consolidation and reuse techniques are used to schedule the tasks. The authors of Dyna do not specify a VM sharing policy between workflows and make no use of containers but rather deploy tasks directly on VMs. We implemented two versions, one in which any task can be deployed on any VM without the notion of containers (referred to as Dyna), and one in which VMs can only be reused between workflows of the same type (referred to as Dyna-WS).

To demonstrate the benefits of using containers and sharing VMs, we implemented two additional versions of our algorithm, EPSM-NC and EPSM-NCWS. EPSM-NC ignores the use of containers and naively assumes VMs can be used to run any task, it is comparable to Dyna. EPSM-NCWS also ignores the use of containers but assumes that VMs can be reused between tasks belonging to workflows of the same type (e.g., between a Ligo workflow with 50 tasks and a Ligo workflow with 1000 tasks), it is similar to Dyna-WS. Finally, the scheduling and provisioning intervals for all versions of EPSM were set to 10 s and 1 s respectively.

The goal of this set of experiments is to evaluate the performance of the algorithms in terms of cost and ability to meet deadlines. We evaluate EPSM and Dyna as well as their variants under three different workloads composed of 1000, 2000, and 4000 workflows as shown in Table 2. The arrival rate for the three workloads was set to 60 workflows per minute.

Fig. 4(a), (b) and (c) depict the cost obtained for each of the algorithms and the small, medium, and large workloads respectively. EPSM-NC obtains the lower cost in all of the three scenarios, closely followed by EPSM. The slight difference between the costs is due to the container provisioning delay considered by EPSM since no tasks can be executed during the time in which the container is being initialized. This demonstrates that with adequate policies that avoid incurring in this delay when possible, the additional cost of using containers is marginal.

As expected, the cost obtained by EPSM-NCWS is higher than that obtained for EPSM for the three workloads. The reason for this is that, by using containers and being able to reuse any VM for any

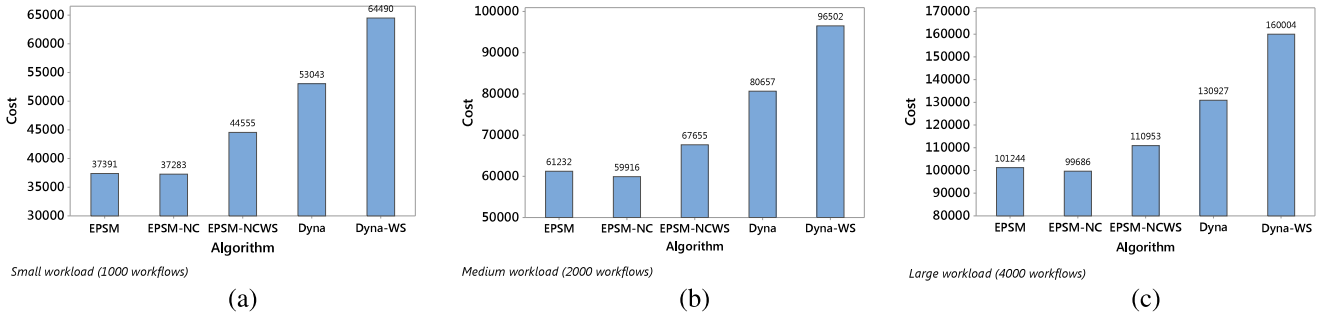

Fig. 4. Cost of executing three different workloads. (a) Small workload with 1000 workflows. (b) Medium workload with 2000 workflows. (c) Large workload with 4000 workflows.

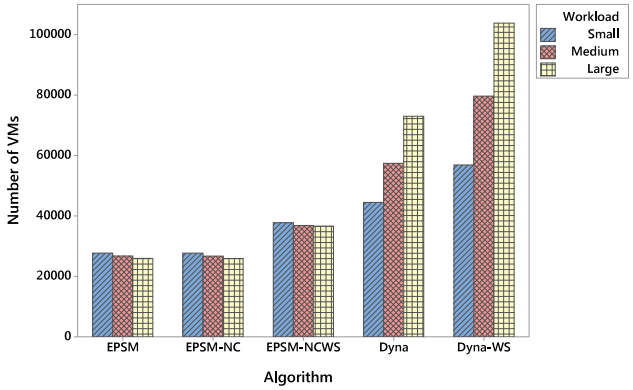

Fig. 5. Total number of VMs leased throughout the execution of the small, medium, and large workloads.

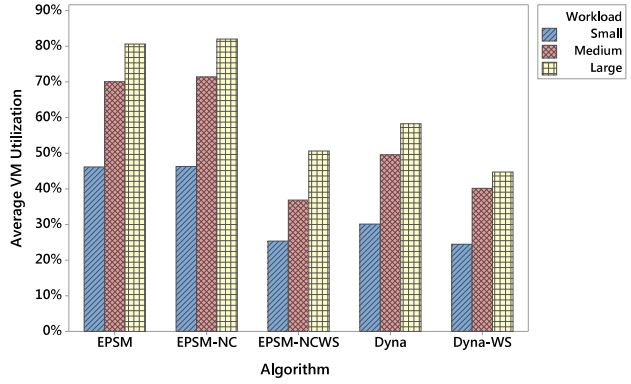

Fig. 6. Average VM utilization during the execution of the small, medium, and large workloads.

task, EPSM is able to better utilize the leased VMs. EPSM-NCWS on the other hand, although it does not incur in additional costs resulting from using containers, is restricted to deploy tasks only on those VMs assigned to workflows of the same type. This further illustrates the benefits of using containerized environments to reduce cost in WaaS environments. The same reasoning applies for Dyna and Dyna-WS.

When compared to Dyna, EPSM achieves considerably lower costs for all the workloads even despite the fact that Dyna is not impacted by container provisioning delays. As shown in Fig. 6, this is due to EPSM achieving a much better utilization of the leased VMs by reducing the number of idle time slots. The results are similar for EPSM-NCWS and Dyna-WS.

To better understand the performance and behavior of the algorithms, we evaluated them in terms of the total number of VMs leased throughout the execution of the workload and their average utilization. We define a VM's utilization as the percentage

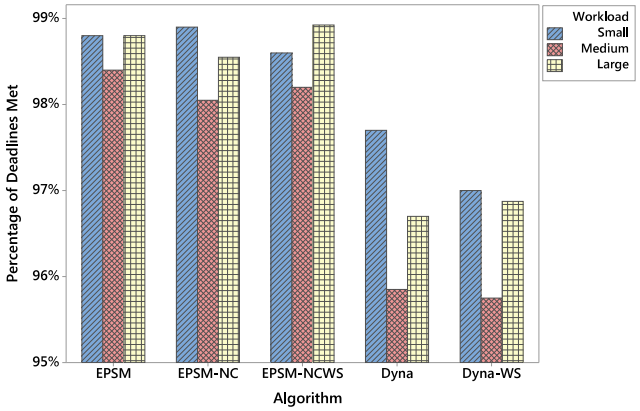

Fig. 7. Percentage of deadlines met for three different workloads: small (1000 workflows), medium (2000 workflows), and large (4000 workflows).

of time that it was busy executing tasks during its lease time. The results for the three workloads are presented in Figs. 5 and 6. EPSM and all of its variants lease a considerably smaller number of VMs when compared to Dyna and Dyna-WS and utilize them better, ultimately leading to the cost differences observed between the algorithms. As expected, the results obtained for EPSM and EPSM-NC are very similar and the utilization of the VMs is lower for EPSM-WS when compared to EPSM. Interestingly, in the case of EPSM, the number of VMs remains almost the same for the three workloads but the utilization increases as the number of workflows in the workload increases. The utilization for the small workload is the lowest for all of the algorithms, this is explained by the high arrival rate of the workflows and their deadlines, the algorithms are left with no choice but lease new VMs in order to fulfill the deadline requirements of tasks. However, as the number of workflows increase, there are more opportunities to reuse idle time slots as there are more tasks with different deadlines that can be scheduled on them.

The percentage of workflows that finished within their deadline for each algorithm and workload are depicted in Fig. 7. All of the evaluated algorithms have a satisfactory performance in this area with all of the percentages being over 95%. This is one of the main advantages of dynamic algorithms as they are able to recover from unexpected delays due to performance degradation or other sources of uncertainties. The slight variations in performance between the different variants of EPSM may be attributed to two factors. The first one is the fact that the VM sharing policy has an impact on the number of VMs leased, their type, and how they are reused. The second one is the statistical fluctuation resulting from the stochastic nature of the simulated performance degradation. As for the difference in performance between the different sized workloads, this can be explained by the nature of the workflows (type, size, and deadline) in the workloads as

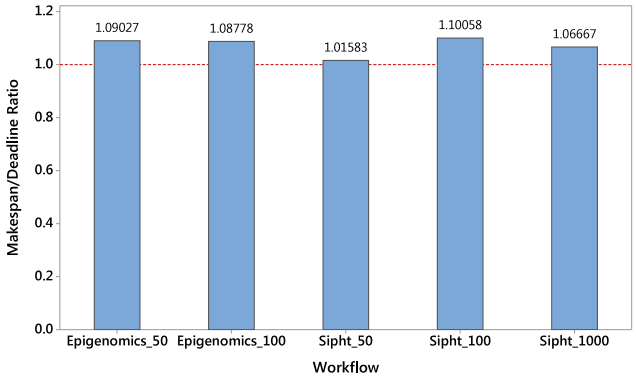

Fig. 8. Average makespan to deadline ratio of workflow executions that completed after their corresponding deadline for the small workload.

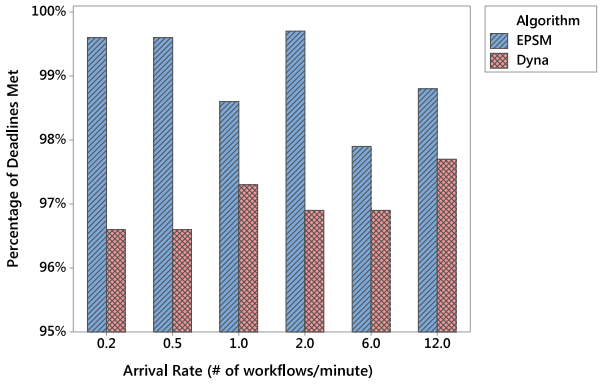

Fig. 9. Percentage of deadlines met for six workloads of 1000 workflows each and different arrival rates.

well as their submission time. Overall, EPSM and all its variants outperform Dyna and Dyna-WS.

To analyze the makespan obtained in those cases in which the deadline was missed, we plotted the average makespan to deadline ratio for each of the workflow execution instances in which the deadline was missed. The results are shown in Fig. 8, where a ratio value greater than one indicates a makespan larger than the deadline. The missed deadlines correspond to two types of workflows, Epigenomics and SIPHT, and the ratio values are below 1.1 in every case. This means that the difference between the makespan and the deadline was marginal. Furthermore, after analyzing the workflow instances for which Dyna failed to meet the deadlines, we found that both algorithms failed to meet the constraint for the same type of workflows the same number of times. This may indicate that in those specific cases, the deadline might have been too strict for the algorithms to be able to finish on time. Another factor contributing to this is the fact that Epigenomics and SIPHT are both CPU-bound workflows [26] and hence are more negatively impacted by VM CPU performance degradation.

EPSM was also evaluated under different workflow arrival rates. For this purpose, we conducted experiments with five workloads each composed of 1000 workflows and five different arrival rates. The results are shown in Figs. 9–12. Firstly, the percentage of deadlines met is over 98% in every case, demonstrating the ability of the algorithm to adapt and fulfill the time constraints under different situations. The utilization and number of VMs greatly depend on the workload as they are affected by the opportunities found by EPSM to reuse already-leased VMs. The number of VMs remains almost constant for the arrival rates of 0.5 through to 6 workflows per minutes but the utilization consistently increases. This means the algorithm successfully identifies opportunities to better utilize

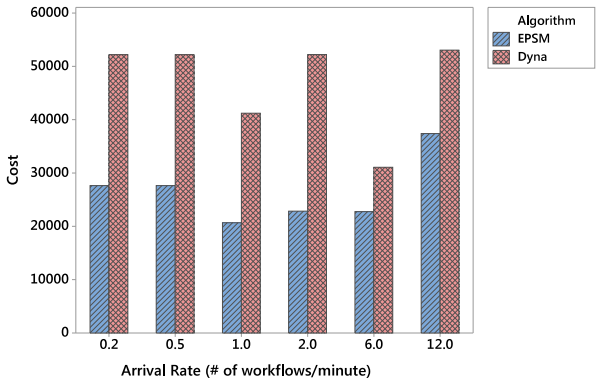

Fig. 10. Cost of executing six workloads of 1000 workflows each and different arrival rates.

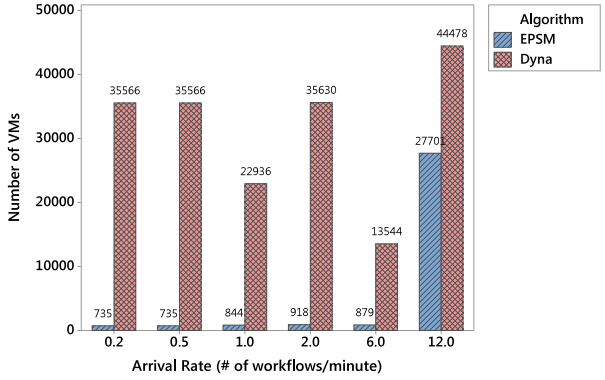

Fig. 11. Total number of VMs leased throughout the execution of six workloads of 1000 workflows each and different arrival rates.

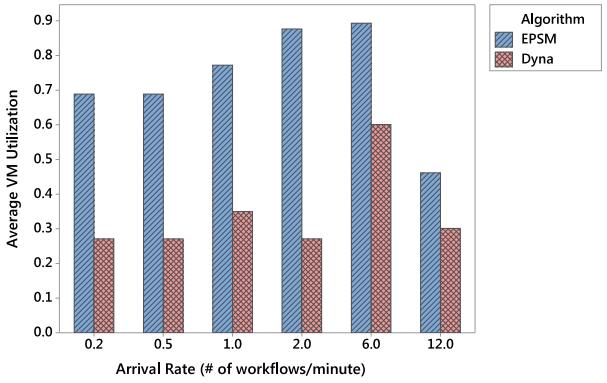

Fig. 12. Average VM utilization during the execution of six workloads of 1000 workflows each and different arrival rates.

idle time slots. For the arrival rate of 12 workflows per minute however, there are more tasks at any given point in time in the queue and a considerably larger number of VMs need to be leased in order to execute them on time, leading to a lower utilization rate and higher cost. We include the results obtained for Dyna as a point of comparison and in every case EPSM outperforms it.

5.2. Provisioning delays sensitivity

Due to the large number of tasks in the scheduling queue at any given point in time, frequent VM provisioning operations may be performed in order to fulfill the approaching deadlines of tasks. Therefore, it is important to evaluate the ability of EPSM to finish the execution of the submitted workflows with a makespan no greater than the given deadline under different

Table 3
Evaluation of EPSM and Dyna with two workloads of 1000 workflows under various VM provisioning delays.

Arrival rate	VM provisioning delay (s)	Algorithm	# of deadlines missed	Cost	# of VMs	Average VM utilization
60 workflows/min	0	EPSM	4	$26 855	1 127	0.763
		Dyna	24	$44 895	37 312	0.369
	50	EPSM	13	$28 782	16 085	0.652
		Dyna	26	$46 143	38 143	0.356
	100	EPSM	12	$37 391	27 701	0.461
		Dyna	23	$53 043	44 478	0.301
	150	EPSM	12	$49 370	40 039	0.327
		Dyna	24	$51 839	42 732	0.308
	200	EPSM	11	$60 683	51 045	0.257
		Dyna	28	$62 652	53 561	0.246
	250	EPSM	13	$78 203	62 622	0.207
		Dyna	31	$77 353	68 364	0.193
6 workflows/min	0	EPSM	12	$22 980	1 110	0.879
		Dyna	27	$30 416	13 366	0.618
	50	EPSM	13	$22 956	1 018	0.885
		Dyna	25	$30 786	13 459	0.607
	100	EPSM	21	$22 769	879	0.893
		Dyna	31	$31 077	13 544	0.600
	150	EPSM	10	$22 934	1 588	0.901
		Dyna	34	$31 683	13 830	0.582
	200	EPSM	13	$24 862	4 592	0.818
		Dyna	33	$32 379	14 257	0.562
	250	EPSM	16	$25 070	4 744	0.815
		Dyna	29	$33 046	14 552	0.542

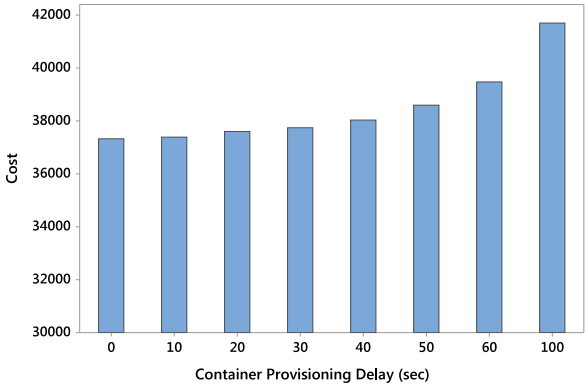

Fig. 13. Cost of executing the small workload (1000 workflows, 60 workflows/minute) under different container provisioning delays.

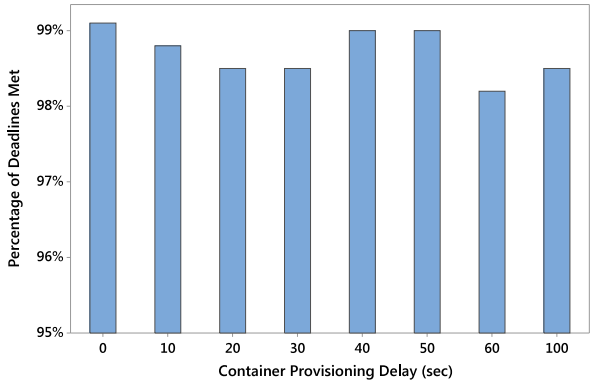

Fig. 14. Percentage of deadlines met for the small workload (1000 workflows, 60 workflows/minute) under different container provisioning delays.

VM provisioning delays. As a result, we evaluated EPSM under six different VM provisioning delays ranging from 0 to 250 s and two workloads with 1000 workflows and different arrival rates. The results obtained for EPSM and Dyna are summarized in Table 3.

For EPSM, the percentage of deadlines met is over 97% in every case, with the number of deadlines missed ranging from 4 to 21. For the arrival rate of 60 workflows per minute, there is a consistent decrease in the VM utilization and a consistent increase in cost and number of VMs as the provisioning delay increases. A possible explanation is that larger delays lead to lower utilization rates as the time period in which the VM is available to run tasks is reduced. Another reason is the fact that as VMs take longer to provision, the time remaining to run tasks in the queue becomes smaller, forcing EPSM to lease new VMs in order to fulfill the deadline constraints. As a result, the increased number of VMs and lower utilization rates lead to higher costs. The behavior of the algorithm under different delays will also greatly depend on the workload and tasks in the execution queue. Longer provisioning times delay the execution of tasks, which in turn result in subsequent tasks having stricter deadlines. Depending on the type and number of tasks in the queue as well as their deadlines and dependencies with previously executed tasks, idle slots may not be reused and faster VM types may have to be provisioned to avoid deadline violations. To demonstrate this, the results obtained for a workload with an arrival rate of 6 workflows per minute are also shown in Table 3. In this case, the number of VMs does not necessarily increase every time the delay increases, the differences in cost are less significant and the average VM utilization is much higher. It is worthwhile noticing as well that the number of VMs and utilization are not the only factors to impact the cost, but also they type of VMs and how long they are leased for.

The results obtained for Dyna were included as a reference point and in every case except one, EPSM outperforms Dyna on every metric. For the 60 workflow/minute arrival rate and provisioning delay of 250 s, Dyna achieves a lower cost than EPSM. However, the cost difference is marginal (1%) and Dyna misses more than double the number of deadlines than EPSM. This may be a result of EPSM's decision to lease more powerful VMs in order to finish tasks on-time.

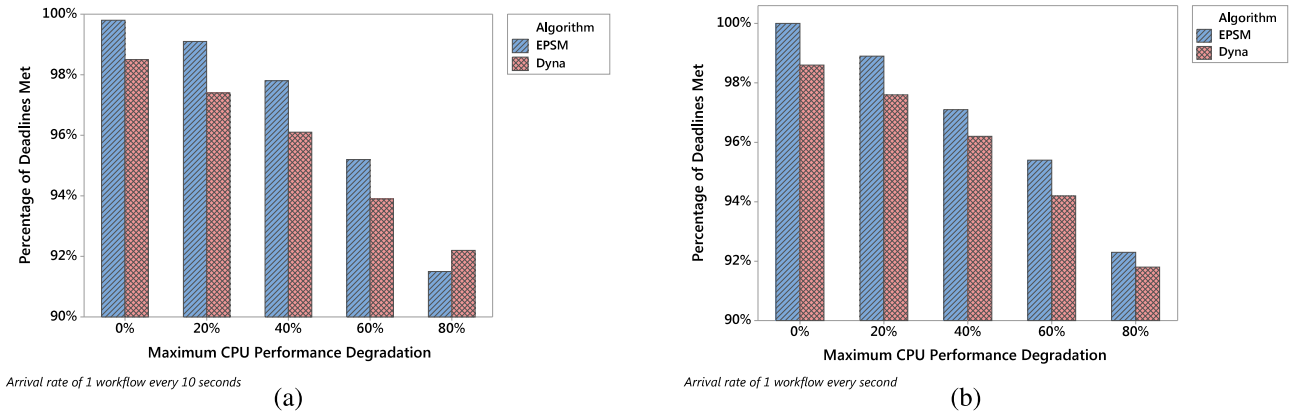

Fig. 15. Percentage of deadlines met for a workload of 1000 workflows with two different arrival rates and varying maximum performance degradation percentages. (a) Arrival rate of 1 workflow every 10 s. (b) Arrival rate of 1 workflow every second.

Regarding the container provisioning delays, we evaluated the performance of EPSM with values ranging from 0 to 100 s. The costs and percentages of deadlines met are depicted in Figs. 13 and 14 respectively. As expected, the larger the delay, the higher the cost as a larger portion of a VM's lease period is spent initializing containers. However, by acknowledging container provisioning delays when estimating runtimes and selecting idle VMs to schedule tasks, EPSM tries to reduce such an increase in cost. This is demonstrated by the obtained results of an increase in cost of approximately 10% between a delay of 0 s to a delay of 100 s. As for the percentage of deadlines met, the algorithm is capable of adapting to increasing delays by maintaining the percentage of deadlines met over 98% in every case. The marginal difference of less than 1% between the performance EPSM under different delays can be attributed to statistical variation and to the fact that different delays lead to different types and number of VMs being leased depending on the duration and submission order of the tasks.

5.3. Performance degradation sensitivity

Recognizing performance variability is important for schedulers so that they can recover from unexpected delays and fulfill the QoS requirements. The sensitivity of the algorithm to VM CPU performance variation was studied by analyzing the percentages of deadlines met, cost, number of VMs, and their average utilization under different degradation values. This degradation was modeled using a normal distribution with a variance of 1% and different average and maximum values. The average values were defined as half of the maximum CPU performance degradation which ranged from 0% to 80%.

The results obtained for two workloads of 1000 workflows are depicted in Fig. 15. For both algorithms and workloads, the percentage of deadlines met decreases as the degradation percentage increases, however, even with a maximum performance degradation of 80%, the percentage of deadlines met remains over 92%. It is not possible to completely eliminate the negative impact of performance degradation when scheduling workflows as even if algorithms are dynamic, they still rely on an estimate of a task's runtime to make decisions. In this way, even a single task taking longer than expected may cause the deadline to become insufficient either because the delay was significant enough, because it was the last or exit workflow task, or because its delay affected its child tasks causing a domino effect. In particular, we identify two characteristics of EPSM that affect its adaptability. The first one is the deadline distribution strategy, which although repeated throughout the execution of the workflow, is based on estimates of task runtimes. The second one is the decision to delay tasks in order to favor leased VMs, which enables the algorithm to better minimize the cost of workflows but affects its responsiveness to changes in the environment. These results demonstrate however, that despite this, EPSM is still successful in achieving its deadline goal in the vast majority of cases.

6. Conclusions and future work

WaaS platforms are emerging with the vision of providing scientists with the ability to deploy their applications for execution in the cloud in a simple and cost-effective manner. They have the potential to revolutionize the way in which scientific workflows are processed by offering a utility-like service that can be accessed on-demand by anyone and from anywhere. An important aspect, as is for any multi-tenant cloud-based framework, is to efficiently manage the execution of workflows belonging to different users and with different QoS requirements. This involves having a scalable scheduling algorithm in place capable of making decisions for large numbers of tasks dynamically as well as a resource provisioning strategy capable of handling the abundance of heterogeneous and elastic cloud resources. As a result, we proposed EPSM, a dynamic algorithm designed to schedule multiple workflows in WaaS environments. Its performance is analyzed in detail and compared to Dyna, demonstrating not only that EPSM is capable of producing higher-quality schedules but also the benefits of sharing resources between multiple workflows in terms of cost which can be achieved in practice by using containers.

This work is a first step towards scheduling in WaaS platforms using containers. There are various aspects that can be considered to improve EPSM and are left as future work. An example is considering the case in which container images are cached on a VM's local storage; this would reduce the use of the data center's network and have a positive impact on cost and makespan metrics. For this purpose, the amount of available storage should be included as part of the definition of VM and policies to decide the number of cached images, their lifetime, and the tradeoff between storing input/output files versus images should be considered. Another future direction is to explore the deployment of multiple simultaneous containers on a single VM in order to execute multiple tasks in parallel. Investigating the effects of sharing resources among multiple workflows and using containers on the consumption of energy is also left as future work. Finally, it would be of interest for WaaS platforms in general to collect and make use of workflow execution data to better estimate the runtime of tasks, to address security and privacy issues that arise from their multi-tenant nature, to develop failure recovery strategies at various levels of the framework, and to study different pricing models that WaaS providers could adopt to charge their customers.

References

[1] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, J. Myers, Examining the challenges of scientific workflows, IEEE Comput. 40 (12) (2007) 26–34.

[2] R. Filgueira, R.F. da Silva, A. Krause, E. Deelman, M. Atkinson, Asterism: Pegasus and dispel4py hybrid workflows for data-intensive science, in: Proceedings of the International Workshop on Data-Intensive Computing in the Cloud, IEEE Press, 2016, pp. 1–8.

[3] W. Gerlach, W. Tang, K. Keegan, T. Harrison, A. Wilke, J. Bischof, M. D'Souza, S. Devoid, D. Murphy-Olson, N. Desai, et al., Skyport: container-based execution environment management for multi-cloud scientific workflows, in: Proceedings of the International Workshop on Data-Intensive Computing in the Clouds, IEEE Press, 2014, pp. 25–32.

[4] S. Esteves, L. Veiga, Waas: Workflow-as-a-service for the cloud with scheduling of continuous and data-intensive workflows, Comput. J. 59 (3) (2016) 371–383.

[5] J. Wang, P. Korambath, I. Altintas, J. Davis, D. Crawl, Workflow as a service in the cloud: architecture and scheduling algorithms, Procedia Comput. Sci. 29 (2014) 546–556.

[6] S. Abrishami, M. Naghibzadeh, D.H. Epema, Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds, Future Gener. Comput. Syst. 29 (1) (2013) 158–169.

[7] R.N. Calheiros, R. Buyya, Meeting deadlines of scientific workflows in public clouds with tasks replication, IEEE Trans. Parallel Distrib. Syst. 25 (7) (2014) 1787–1796.

[8] S. Liu, K. Ren, K. Deng, J. Song, A task backfill based scientific workflow scheduling strategy on cloud platform, in: Proceedings of the International Conference on Information Science and Technology, IEEE, 2016, pp. 105–110.

[9] T. Dziok, K. Figiela, M. Malawski, Adaptive multi-level workflow scheduling with uncertain task estimates, in: Parallel Processing and Applied Mathematics, Springer, 2016, pp. 90–100.

[10] A. Deldari, M. Naghibzadeh, S. Abrishami, Cca: a deadline-constrained workflow scheduling algorithm for multicore resources on the cloud, J. Supercomput. (2016) 1–26.

[11] P. Maechling, E. Deelman, L. Zhao, R. Graves, G. Mehta, N. Gupta, J. Mehringer, C. Kesselman, S. Callaghan, D. Okaya, et al., SCEC CyberShake workflowsautomating probabilistic seismic hazard analysis calculations, in: Workflows for E-Science, Springer, 2007, pp. 143–163.

[12] E. Deelman, G. Singh, M. Livny, B. Berriman, J. Good, The cost of doing science on the cloud: the montage example, in: Proceedings of the ACM/IEEE Conference on Supercomputing, IEEE Press, 2008, p. 50.

[13] J.-S. Vöckler, G. Juve, E. Deelman, M. Rynge, B. Berriman, Experiences using cloud computing for a scientific workflow application, in: Proceedings of the International Workshop on Scientific Cloud Computing, ACM, 2011, pp. 15–24.

[14] M. Malawski, G. Juve, E. Deelman, J. Nabrzyski, Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds, Future Gener. Comput. Syst. 48 (2015) 1–18.

[15] I. Pietri, M. Malawski, G. Juve, E. Deelman, J. Nabrzyski, R. Sakellariou, Energy-constrained provisioning for scientific workflow ensembles, in: Proceedings of the International Conference on Cloud and Green Computing, IEEE, 2013, pp. 34–41.

[16] Q. Jiang, Y.C. Lee, A.Y. Zomaya, Executing large scale scientific workflow ensembles in public clouds, in: Proceedings of the International Conference on Parallel Processing, IEEE, 2015, pp. 520–529.

[17] P. Bryk, M. Malawski, G. Juve, E. Deelman, Storage-aware algorithms for scheduling of workflow ensembles in clouds, J. Grid Comput. 14 (2) (2016) 359–378.

[18] Z. Yu, W. Shi, A planner-guided scheduling strategy for multiple workflow applications, in: Proceedings of the International Conference on Parallel Processing-Workshops, IEEE, 2008, pp. 1–8.

[19] H.-J. Jiang, K.-C. Huang, H.-Y. Chang, D.-S. Gu, P.-J. Shih, Scheduling concurrent workflows in hpc cloud through exploiting schedule gaps, in: Proceedings of the International Conference on Algorithms and Architectures for Parallel Processing, Springer, 2011, pp. 282–293.

[20] G.L. Stavrinides, H.D. Karatza, A cost-effective and qos-aware approach to scheduling real-time workflow applications in paas and saas clouds, in: Proceedings of the International Conference on Future Internet of Things and Cloud, IEEE, 2015, pp. 231–239.

[21] M. Xu, L. Cui, H. Wang, Y. Bi, A multiple qos constrained scheduling strategy of multiple workflows for cloud computing, in: Proceedings of the IEEE International Symposium on Parallel and Distributed Processing with Applications, IEEE, 2009, pp. 629–634.

[22] W. Chen, Y.C. Lee, A. Fekete, A.Y. Zomaya, Adaptive multiple-workflow scheduling with task rearrangement, J. Supercomput. 71 (4) (2015) 1297–1317.

[23] A.C. Zhou, B. He, C. Liu, Monetary cost optimizations for hosting workflow-as-a-service in iaas clouds, IEEE Trans. Cloud Comput. 4 (1) (2016) 34–48.

[24] M. Mao, M. Humphrey, Auto-scaling to minimize cost and meet application deadlines in cloud workflows, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, ACM, 2011, p. 49.

[25] J. Shi, J. Luo, F. Dong, J. Zhang, A budget and deadline aware scientific workflow resource provisioning and scheduling mechanism for cloud, in: Proceedings of the IEEE International Conference on Computer Supported Cooperative Work in Design, IEEE, 2014, pp. 672–677.

[26] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, K. Vahi, Characterizing and profiling scientific workflows, Future Gener. Comput. Syst. 29 (3) (2013) 682–692.

[27] J. Schad, J. Dittrich, J.-A. Quiané-Ruiz, Runtime measurements in the cloud: observing, analyzing, and reducing variance, Proc. VLDB Endow. 3 (1–2) (2010) 460–471.

[28] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, D. Epema, A performance analysis of EC2 cloud computing services for scientific computing, in: Cloud Computing, Springer, 2010, pp. 115–131.

[29] A. Gupta, D. Milojicic, Evaluation of hpc applications on cloud, in: Open Cirrus Summit (OCS), 2011, pp. 22–26. http://dx.doi.org/10.1109/OCS.2011.10.

[30] A. Iosup, S. Ostermann, M.N. Yigitbasi, R. Prodan, T. Fahringer, D. Epema, Performance analysis of cloud computing services for many-tasks scientific computing, IEEE Trans. Parallel Distrib. Syst. 22 (6) (2011) 931–945. http://dx.doi.org/10.1109/TPDS.2011.66.

[31] K.R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H.J. Wasserman, N.J. Wright, Performance analysis of high performance computing applications on the amazon web services cloud, in: Proceedings of the IEEE Second International Conference on Cloud Computing Technology and Science, (CloudCom), IEEE, 2010, pp. 159–168.

[32] P. Di Tommaso, E. Palumbo, M. Chatzou, P. Prieto, M.L. Heuer, C. Notredame, The impact of docker containers on the performance of genomic pipelines, PeerJ 3 (2015) e1273.

[33] W. Felter, A. Ferreira, R. Rajamony, J. Rubio, An updated performance comparison of virtual machines and linux containers, in: International Symposium on Performance Analysis of Systems and Software, (ISPASS), IEEE, 2015, pp. 171–172.

[34] R.F. da Silva, W. Chen, G. Juve, K. Vahi, E. Deelman, Community resources for enabling research in distributed scientific workflows, in: Proceedings of the IEEE International Conference on E-Science, Vol. 1, (e-Science), IEEE, 2014, pp. 177–184.

[35] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A. De Rose, R. Buyya, Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Softw. - Pract. Exp. 41 (1) (2011) 23–50.

[36] M. Mao, M. Humphrey, A performance study on the VM startup time in the cloud, in: Proceedings of the IEEE International Conference on Cloud Computing, IEEE, 2012, pp. 423–430.

[37] S.F. Piraghaj, A.V. Dastjerdi, R.N. Calheiros, R. Buyya, Containercloudsim: An environment for modeling and simulation of containers in cloud data centers, Softw. - Pract. Exp. 47 (4) (2017) 505–521.

Maria A. Rodriguez is a Post-Doctoral Research Fellow in the Cloud Computing and Distributed Systems (CLOUDS) Laboratory in the Department of Computing Information Systems, The University of Melbourne, Australia. Her research interests include resource management and scheduling in clouds and scientific computing.

Rajkumar Buyya is a Professor of Computer Science and Software Engineering and Director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne, Australia. He is also the founding CEO of Manjrasoft, a spin-off company of the University, commercializing its innovations in Cloud Computing. He has authored over 400 publications and four textbooks. He is one of the highly cited authors in computer science and software engineering worldwide.