Workflow Scheduling with Amazon EC2 Spot Instances: Building Reliable Compute Environments

Altino M. Sampaio and Jorge G. Barbosa

Abstract—**Amazon Elastic Compute Cloud (EC2) gives access to resources in the form of instances. EC2 Spot Instances (SIs) offer spare compute capacity at steep discounts compared to reliable and fixed price on-demand instances. However, SIs are unreliable since they can be reclaimed by EC2 at any given time, with a two-minute interruption notice. In this paper, we propose a container migration-based solution to build reliable compute environments on top of unreliable EC2 instances. Our solution leverages recent findings on performance and behavior characteristics of EC2 SIs. We compare the performance of our algorithm to that of state-of-the-art algorithms, by running a real-life workflow application constrained by user-defined deadline and budget parameters. The results show that our solution is able to build reliable virtual compute environments on top of EC2 on-demand-, spot block, and SI purchasing models, and successfully conclude submitted workflow applications with budget and deadline constraints, for a worse-case scenario.**

Index Terms—**Amazon EC2, cloud computing, reliability, migration of containers.**

I. Introduction

Cloud computing is a relatively new distributed-computing paradigm that offers users easy access to resources from anywhere and at anytime, and in a pay-as-you-go manner [1], [2]. Cloud computing exploits virtualization to provision computational resources in the form of Virtual Machine (VM) instances [3]. Cloud provider Amazon EC2 offers four ways to pay for compute instances, namely[1] : 1) On-demand; 2) Spot Instances (SIs); 3) Reserved instances; and 4) Dedicated hosts. Reserved instances and dedicated hosts are the two most expensive alternatives and imply the payment of a yearly fee (of hundreds to thousands of dollars). On-demand instances allow users to pay for compute capacity by hours without long-term commitments, but the hourly fee is a bit higher than that of options 2 and 3. On the other hand, SIs allow users to bid for idle resource capacity for up to 90% off the on-demand price. The trade-off is that if the current spot price is greater than the user's bid price or if EC2 lacks resources for on-demand or reserved instances, SIs will be reclaimed with a two minutes notification. Spot block are a new model of instances that are promised to run continuously for a finite duration (1 to 6 hours). Pricing is based on the requested duration and the available capacity, which is typically 30% to 45% less than on-demand instances. Despite the cheapest alternative, SIs provide no guarantee with respect to termination time. Recent observations on the reliability of SIs show that interruption rate can be as much as 34% in some EC2 datacenters, with a total of 8.58% of SIs being interrupted within 20 minutes [4]. Taking into account that EC2 in particular, and Cloud in general, has been progressively adopted to run a wide range of applications encompassing various domains, such as science, and that much of these applications may run for hours [5], [6], providing reliability to compute environments built on top of EC2 SIs becomes critical.

This paper considers the problem of scheduling scientific workflow applications on on-demand, spot block, and SIs under budget and deadline constraints. Schedules of workflow tasks on SIs may result in significant monetary cost reduction although at the expenses of compute availability. As such, one needs for an effective solution to mitigate resource failures (i.e., SIs unreliability) and fulfill user requests regarding to monetary cost and time. Our solution proposes a reliability-aware scheduling algorithm that uses containers [7] on top of EC2 instances to build virtual compute environments. Unreliability of underlying EC2 instances is masked by migrating containers from one just-reclaimed SI to another available EC2 instance. To the best of our knowledge, this is the first attempt to leverage migration of containers to deal with EC2 SIs interruptions and provide reliable compute environments to end users. The rest of the paper is organized as follows. Section II discusses related work. Section III introduces a proposal overview, by describing the system, workflow applications, scheduling algorithm, and feasibility of migration of containers. Section IV presents the tested scenario by characterizing the simulator, the workloads, the algorithms used for comparison, the SI interruptions, and performance metrics. Section V presents and discusses the obtained results. Conclusions are presented in Section VI.

II. Related Work

Scientific applications and experiments are commonly expressed as workflows. A workflow application is typically described by a graph consisting of a certain number of tasks (or nodes), with various computation and data needs, and a set of edges that represent control and data dependencies [8]. In particular, many of workflow applications fall in the category of Directed Acyclic Graph (DAG) [9], where the nodes represent the temporal relations between the tasks. Fig. 1 shows the structure of the Montage workflow, a typical

Manuscript received December 13, 2018; revised November 24, 2019.

Altino M. Sampaio is with the CIICESI, Escola Superior de Tecnologia e Gestão, Instituto Politécnico do Porto, Felgueiras, Portugal (e-mail: ams@estg.ipp.pt).

Jorge G. Barbosa is with LIACC, Departamento de Engenharia Informática, Faculdade de Engenharia, Universidade do Porto, Porto, Portugal (e-mail: jbarbosa@fe.up.pt).

[1]https://aws.amazon.com/ec2/pricing/

scientific workflow application that delivers science-grade mosaics of the sky to the community composed of both professional and amateur astronomers [10].

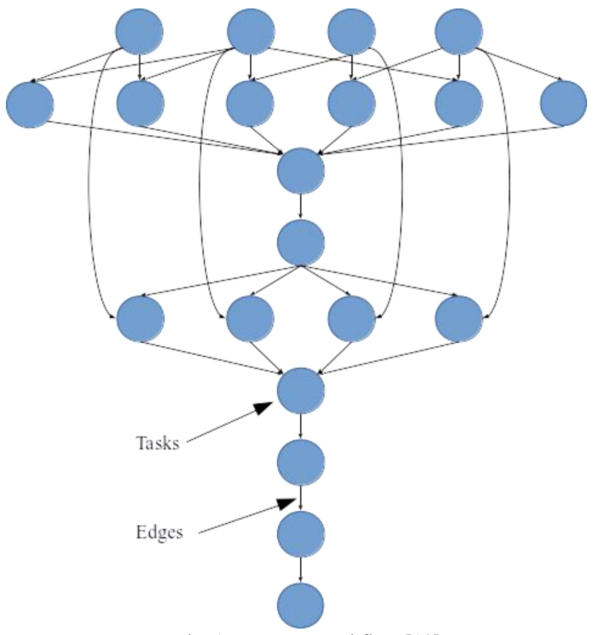

Fig. 1. Montage workflow [10].

Workflow scheduling in Cloud computing systems is a well-known NP-complete problem [11], which refers to the process of spatial and temporal mapping of workflow tasks onto resources in order to satisfy some or multiple performance criterion. Comparing to older distributed computing models, Cloud computing addresses many technical and economic advantages. From the scientists point of view, cloud computing provides many resources as they need, when they need them and for as long as they need them. Additionally, since Clouds apply the utility model, scientists pay only for what they use. In spite of this, Clouds such as EC2 have been the target of relevant research over the last years towards the execution of scientific applications. Deepak *et al.* [12] proposed an adaptive, just-in-time scheduling algorithm to schedule scientific workflows on spot and on-demand instances. The aim is to minimize execution time and monetary costs. Eventual interruption of SIs is mitigated by means of replication of tasks. Authors show the effectiveness and robustness of their solution by means of extensive sets of simulations. Replication of tasks is a fault tolerance technique that applies redundancy. Sampaio and Barbosa [13] recently showed that the technique leads to increase of monetary costs since it implies the use of additional resources. Also Long *et al.* [14] addressed the problem of SIs unreliability caused by the fluctuations of the bidding prices (i.e., SI may be terminated at any time when the bidding price is lower than the spot price). Aiming at minimizing the total renting monetary cost under deadline constrains, authors propose to construct schedules for workflows with both non-preemptive and preemptive tasks on spot block and on-demand instances. A scheduling algorithm is proposed to determine the block time for SIs and to improve the task-to-instance mapping. Sets of experiments showed promising results regarding the reduction of average monetary costs, compared to scheduling strategies with only on-demand instances. Unfortunately, authors consider only EC2 spot block instances, and ignore the competitiveness of SIs in terms of monetary costs. Zhou *et al.* [15] developed a probabilistic framework named Dyna for the scheduling of scientific workflows. The objective is to minimize the monetary cost while satisfying workflows deadline guarantees. Authors also presented a hybrid instance configuration refinement mechanism of spot and on-demand instances for price dynamics. To analyze the proposal, Dyna framework was deployed on Amazon EC2 and experiments were carried out. Authors concluded that their solution was able to execute workflows at lower monetary costs compared to state-of-the-art approaches, while meeting users' requirements. Nonetheless, the solution lacks SIs unreliability due to out-of-bid and resources scarcity events, which causes premature termination of running tasks. Unlike the solutions reported above, in this paper we propose to leverage migration of containers to build reliable compute environments on top of on-demand, spot block, and unreliable SIs. The aim is to run scientific workflow applications subject to budget and deadline constraints.

III. System Overview

This section provides a formal description of the proposed solution to build reliable compute environments on top of EC2 on-demand, spot block, and unreliable SIs.

A. Cloud Service Provider

We envision a typical Cloud service provider (CSP) such as that of Amazon EC2. The considered CSP is represented in Fig. 2. In this scenario, sets of resources are available to users in the form of VM instances that are provisioned and charged per time unit. As defined by EC2, the price of instances is stated according to the type (i.e., power) of VMs. Thus, the higher cost is assigned to the most powerful VM (i.e., faster and/or higher number of processors), while less powerful VMs are cheaper. Three renting model instances are considered, namely: 1) on-demand; 2) spot block; and 3) SIs. While on-demand model is the more expensive alternative, SIs are the cheaper. SIs are considered unreliable since they can be reclaimed by EC2 with a two-minute interruption notification. Tasks running on SIs are abruptly terminated in the moment of instance interruption. Containers are deployed on top of VM instances. A container has fully access to resources of relative VM. The set of containers form the user's virtual cluster executing environment. The CSP operating model is as follows: a) a user submits a set of workflow applications and defines deadline and budget constraints for each one; then, b) the CSP handles the request by providing the necessary VMs to run the tasks under the specified constraints. Workflow deadlines activate right after submission and the CSP has no prior knowledge of when workflow applications arrive. In order to deal with users' requests, CSP implements four modules, namely: a) scheduler; b) type and payment model selector; c) container controller; and d) interrupt event handler. When a workflow is submitted, the CSP passes it to the scheduler, which starts by decomposing it into a set of tasks. Then, the scheduler selects the type (i.e., t2.small, etc.) and payment model (i.e., on-demand, etc.) instance to run each task and instructs the type and payment model module to handle the VM requests.

After VM deployment, the container controller module launches the containers and starts the tasks. A container encapsulates the task execution environment and is the unit of migration in the system. Each container runs a single task and can be multiplexed among various tasks over time as one task finishes executing and another one gets ready to start. The interrupt event handler monitors interruption notifications.

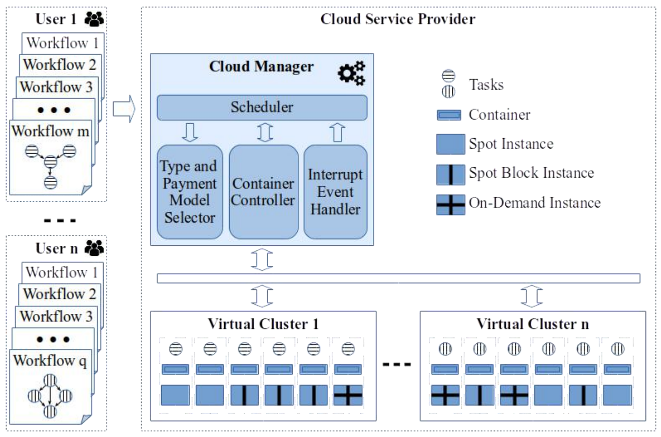

Fig. 2. Cloud service provider.

B. Enhancing Computing Reliability

The solution proposed in this paper leverages containers to build reliable compute environments on top of unreliable EC2 SIs. Containers in Linux are a virtualization technique that provides isolation of processes. Linux containers are implemented primarily via cgroups and namespaces to feature resource management, isolation, and security. While cgroups can be used to resize and limit the resources of a container, and to terminate all processes inside of it, namespaces provide isolation between processes that are not part of the same container [16]. Nowadays, several Linux based container management tools exist, such as LXC, OpenVZ, and Docker [17].

A container is much more lightweight than a VM because the isolation of processes is implemented at the operating system level [7]. As a result, containers exhibit similar performance to that of native execution, a fact that extends to the case of containers running inside VMs[2] . Moreover, since a container is much more lightweight than a VM, migration of containers is faster to accomplish than that of migration of VMs. Ma, Yi, and Li [18] showed that live migration of containers running various real applications can be accomplished within few tens of seconds at most, a value that can be reduced to less than 5 seconds if one uses distributed storage systems. Furthermore, deployment and booting of containers is concluded within a couple of seconds. These container operating properties are essential and constitute an opportunity to enhance reliability of compute environments, because the time needed to complete the migration of a container is much less than the two-minute SI interruption notice.

C. Workflow Application Model

Users submit scientific workflow applications that are expressed as DAGs. A DAG can be modeled by a tuple $G = < T, E >$, where T is the set of n tasks of the workflow application, such that $T = \{t_1, t_2, \ldots, t_n\}$. The set of edges E represent the data dependencies among tasks, where each dependency indicates that a child task cannot be executed before all its parent tasks finish successfully and transfer the required child input data. For the sake of simplicity, we assume that all workflow data is stored in a shared cloud storage system (e.g., Amazon S3), and the intermediate data transfer times are known or can be estimated. The data transfer times between the shared storage and the containers are equal for different containers so that task placement decisions do not impact the runtime of the tasks. The runtime estimates and the CPU computational needs for the workflow tasks are known. Due to heterogeneity of VM instance types, a task may present different execution times. Only workflows for which all tasks are finished within deadline and budget constraints are considered complete.

D. Scheduling Algorithm

ALGORITHM I: MISER SCHEDULING ALGORITHM

```
1  **Procedure** MISER
2    **for** $t_{i,j} \in$ ReadyToExecuteList **do**
3      determine $ranku(t_i, j)$                          ⊳ (1)
4    $VM_{free} \leftarrow$ free resources
5    $c \leftarrow 0$
6    **while** ReadyToExecuteList.size > 0 **&&** $VM_{free} > 0$ **do**
7      $j \leftarrow$ next workflow **mod** $c{+}{+}$                 ⊳ round-robin
8      $t_{i,j} \leftarrow$ task with highest $ranku(t_i, j)$
9      $VM_{accept} \leftarrow$ instance model policy $\cap$ $VM_{free}$
10     $VM_{accept} \leftarrow CP \cap VM_{accept}$                     ⊳ (2)
11     $VM_{accept} \leftarrow$ DP $\cap$ $VM_{accept}$                     ⊳ (3)
12     **for** $VM_K \in VM_{accept}$ **do**
13       determine $Q(t_i, VM_K)$                          ⊳ (7)
14     $VM_{sel} \leftarrow$ instance $VM_K$ with highest $Q$
15     assign $t_{i,j}$ to $VM_{sel}$
16     update $\Delta_{cost}(j)$
17     $VM_{free} \leftarrow VM_{accept} - VM_{sel}$
18     remove $t_{i,j}$ from ReadyToExecuteList
```

Algorithm I presents MISER, the best-effort multi-workflow deadline- and budget-constrained scheduler algorithm proposed in this paper. MISER works in two phases, namely: a) task selection; and b) VM instance selection. In the first phase (line 2 and 3 of Algorithm I), a priority is assigned to each ready-to-execute task, $ranku(t_i,j)$, based on its critical path [19], as specified by (1), where $\overline{ET(t_i)}$ is the average execution time of task t_i, belonging to workflow j, on available EC2 instance types (e.g. t2.small, t2.medium, t2.xlarge, t2.2xlarge) with different performance and prices, $\overline{c_{i,s}}$ is the average communication time between tasks t_i and its successor t_s, which can be calculated as the average network bandwidth and latency among instance pairs.

$$rank_u(t_i, j) = \overline{ET(t_i)} + \max_{\forall t_s \in succ(t_i)} \left(\overline{c_{i,s}} + rank_u\left(t_s, j\right) \right). \quad (1)$$

Then, a round-robin-based strategy is applied to select the ready-to-execute task with the highest priority from each

[2]https://blogs.vmware.com/performance/2014/10/docker-containers-performance-vmware-vsphere.html

workflow (lines 7 and 8 of Algorithm I). The rationale is to avoid starvation of workflow applications, and guarantee that all of them participate in the current scheduling round. A scheduling round ends when there are no more free resources to allocate tasks or when all the ready-to-execute tasks are finally scheduled. On phase two, a VM instance model (e.g., on-demand, spot block, and unreliable SI) and type (e.g., t2.small, etc.) is selected to run the task. First, the instance pricing model is chosen based on the scheduling reason (line 9 of Algorithm I). Three policies apply, namely: a) a new task t_i can be scheduled on any instance model; b) a task forced to migrate from just-reclaimed SI can be allocated into any already running instance model or into new on-demand or spot block instance models. The rationale behind b) scheduling policy is SIs are interrupted mainly due to scarcity of capacity in the datacenter [4]. So, if one SI is being reclaimed by EC2 it is likely that more SIs follow the same destiny. Second, the VM instance type is selected based on monetary cost and deadline policies (lines 10 and 11 of Algorithm I). VM instance candidates that do not comply with monetary cost and deadline policies are out of the candidate set VMaccept. Equation 2 shows the monetary cost policy $CP(t_i, j)$, where $Cost_{min}(t_i, j)$ is the minimum execution cost of task t_i of workflow j among all instance candidates, and $\Delta_{Cost}(j)$ is the spare budget which is determined as the difference between remaining budget at scheduling round, and the cheapest monetary cost assignment for unscheduled tasks belonging to workflow j.

$$CP(t_i, j) = Cost_{\min}(t_i, j) + \Delta_{Cost}(j) . \tag{2}$$

The deadline policy $DP(t_i, j)$ is shown by (3), where a sub-deadline is assigned to task t_i, which is determined recursively by traversing the task graph upwards, starting from the exit task. It depends on the minimum execution time of task t_s, successor of t_i, $ET_{min}(t_s)$, among available instances, on the average communication time $\overline{c_{i,s}}$, on the maximum migration time $MT_{max}(t_s)$ for all successor tasks of t_i, and on the migration time for task t_i, $MT_{max}(t_i)$. $\Theta(t_s)$ is a Bernoulli variable, which is one if successor task t_s can be allocated on SI, and is zero otherwise. The rationale is that sub-deadline considers the eventual need of migrating successor tasks allocated to unreliable SIs. The same rationale applies to the eventual necessity to migrate task t_i. Just like the well-known HEFT algorithm [20], the rationale for the minimum execution time $ET_{min}(t_s)$ relates to the possibility of reducing execution costs, since expanding the sub-deadline assigned to task t_i allows the scheduler to exploit idle slots between two already scheduled tasks on a VM instance, as long as precedence constraints are preserved.

$$DP(t_i, j) = \min_{\forall t_s \in succ(t_i)} \begin{pmatrix} DP(t_s, j) - \overline{c_{i,s}} - ET_{\min}(t_s) \\ - MT_{\max}(t_s) \times \Theta(t_s) \end{pmatrix}, \quad \text{where } \Theta(t_s) = \begin{cases} 1, \text{if } t_s \text{ can be scheduled on SIs} \\ 0, \text{otherwise} \end{cases} . \tag{3}$$

The next step consists of testing each $VM_k \in VM_{accept}$ in order to obtain VM_{sel} as the most appropriated VM instance to allocate task t_i (lines 12 to 15 of Algorithm I). To assist in such decision making, three relative quantities are defined, namely time quality TQ, monetary cost quality CQ, and reliability quality RQ. These three quality parameters are normalized in order to make each one to fall into [0, 1] interval. TQ is determined as shown by (4), where $FT(t_i, j, VM_k)$ represents the finish time for task t_i running on VM_k instance, and $FT_{max}(t_i, j)$ and $FT_{min}(t_i, j)$ are the maximum and minimum execution times determined among all available instance types and pricing models, respectively.

$$TQ(t_i, j, VM_k) = 1 - \frac{FT(t_i, j, VM_k) - FT_{\min}(t_i, j)}{FT_{\max}(t_i, j) - FT_{\min}(t_i, j)} . \tag{4}$$

CQ is calculated as shown by (5), where $Cost(t_i, j, VM_k)$ is the monetary cost for executing task t_i on VM_k instance, and $Cost_{max}(t_i, j)$ and $Cost_{min}(t_i, j)$ are the maximum and minimum monetary costs determined among all available instance types and pricing models, respectively.

$$CQ(t_i, j, VM_k) = 1 - \frac{Cost(t_i, j, VM_k) - Cost_{\min}(t_i, j)}{Cost_{\max}(t_i, j) - Cost_{\min}(t_i, j)} . \tag{5}$$

RQ is computed as shown by (6), where χ is: a) zero for on-demand or spot block instances; or b) the cumulative distribution function CDF_{logn} (μ is the mean and σ is the standard deviation). $\chi_{max}(t_i, j)$ and $\chi_{min}(t_i, j)$ are the maximum and minimum values of χ among all available instances, respectively. CDF_{logn} tends to the unity as $FT(t_i, j, VM_k)$ moves towards SI aging. The rationale is to favor selection of fresh SIs over long- running ones, since probability of a SI be reclaimed increases with aging. A log-normal probability distribution was selected because the curve fitted well with the data provided by recent studies on performance and behavior characterization of Amazon EC2 SIs [4]. Parameters μ and σ can be adjusted online based on historical data.

$$RQ(t_i, j, VM_k) = 1 - \frac{\chi(t_i, j, VM_k) - \chi_{\min}(t_i, j)}{\chi_{\max}(t_i, j) - \chi_{\min}(t_i, j)}, \quad \text{where } \chi(t_i, j, VM_k) = \begin{cases} CDF_{\log n}\left(FT(t_i, j), \mu, \sigma\right), \text{if } VM_k \text{ is SI} \\ 0, \text{otherwise} \end{cases} . \tag{6}$$

Finally, utility function Q is determined as shown by (7), which combines TQ, CQ, and RQ.

$$Q(t_i, j, VM_k) = TQ(t_i, j, VM_k) + CQ(t_i, j, VM_k) + RQ(t_i, j, VM_k) . \tag{7}$$

Q is used to obtain the best balance between time, monetary cost, and reliability. MISER selects the VM instance model and type VM_{sel} that provides the highest value of Q.

In order to deal with interruptions of SIs, MISER is combined with a monitoring mechanism (i.e., interrupt event handler module in CSP) that is responsible for handling EC2 notification events regarding SI reclaims. Therefore, MISER algorithm will run when: a) there are ready-to-execute tasks; and b) an interruption notification event is received from EC2. The objective is to make sure that all tasks allocated into now-reclaimed SIs can be re-scheduled and finished within their deadlines.

IV. Evaluation and Simulation Scenario

This section describes the characteristics of the evaluation scenario in terms of SIs reliability, instance prices, workloads, metrics, scheduling algorithms for performance comparison, and simulation setup.

A. Description of SIs Reliability

Pham, Ristov, and Fahringer have recently conducted an extensive set of experiments aiming at characterizing SIs reliability on three different EC2 datacenters [4]. The evaluation resulted in a variety of findings. These findings revealed a spot request fulfillment rate of more than 99.2% in most of the datacenters considered. Once an SI is provisioned and deployed, it can be interrupted by EC2 after no-capacity or low-bid-price status messages. Characterizing SI interruptions, authors reported that some datacenters had similar interruption rates of approximately 12.5%, while others resulted in 34.0% interruptions. Regarding the running time of deployed SIs with interruption, half of all interrupted SIs run at least 3 hours, a value that can decrease to less than 1.5 hours in datacenters. Additional examination concerning how long SIs run until interruption indicates in worse datacenters 95.2% of SIs run at least the first 20 minutes. Based on these findings, we generated a set of reliability periods which were then assigned to SIs launched by the CSP. Also, spot requests are fulfilled in no more than 4 seconds. The generation of the reliability set considered the region with the worse performance and behavior to deeply test the effectiveness of our solution.

B. Description of Workloads

The Pegasus project made available a set of realistic workflows from diverse scientific applications. These workflows are available in DAG in XML (DAX) format, under different sizes (i.e., number of tasks). A DAX file characterizes in detail the structure, data and computational requirements for a specific workflow. This study leverages the Montage workflow structure from the Pegasus project, a general engine for computing mosaics of input images in the Flexible Image Transport System (FITS) format. Since it is a realistic workflow, the Montage has been used in current research [9], [19], [21]. The memory used by a task was randomly assigned implying different migration times ranging in {8, 10, 12} seconds. Since most of Montage tasks are small, we multiplied by 25: a) the time necessary to execute each task in the workflow; and b) the size of files transferred among tasks necessary. Based on this, we generated a total of 55 workflows, summing 1375 tasks, randomly submitted by 10 different users. Users arrive to the system at random time instants, ranging in {10, 30, 50}% of the minimum execution time of last submitted workflow application. The same applies for submission of workflow applications once the user arrives to the system.

C. EC2 Instance Types

Amazon EC2 offers extensive variety of VM instance types, each providing different amount of resources in terms of CPU, GPU, and memory. In this paper, we have considered the VM instance types shown in Table I. The price of each instance was taken from Amazon EC2 website at the time of writing this paper. Billing characteristics are as follows: 1) if a SI is interrupted in the first hour, the user is not charged for that usage; 2) if the SI is interrupted in any subsequent hour, the user is charged for usage to the nearest second; 3) if SI interruption is triggered by its user then it will be charged to the nearest second. The price of a spot block instance depends on the specified duration (1 to 6 hours). In this regarding, the monetary cost of spot blocks for 2, 3, 4, and 5 hours was defined by extrapolating the known prices for 1 and 6 hours. VM instances are interrupted the user as soon as all their workflows are successfully finished or unable to execute by the deadline.

TABLE I: EC2 VM Instance Price ($)

VM Instance	**On-demand (1h)**	**SI (1h)**	**Spot block (1h)**	**Spot block (6h)**
t2.small	0.0230	0.0069	0.0130	0.0160
t2.medium	0.0464	0.0139	0.0260	0.0320
t2.xlarge	0.1856	0.0557	0.1020	0.1300
t2.2xlarge	0.3712	0.1114	0.2040	0.2600

D. Metrics Characterization

Three metrics were defined to measure the performance of the proposed solution. The first metric is the completion rate of tasks (CRT), which is determined as the ratio of the number of successfully finished tasks to the number of submitted tasks. The second metric is the completion rate of workflows (CRW), and is calculated as the ratio of the number of successfully accomplished workflows to the number of submitted workflows. The last metric is the scheduled rate of tasks (SRT), which is computed as the ratio of the number of scheduled tasks to the total of tasks in the system.

A task that does not fulfill the budget and deadline constraints defined for its relative workflow is accounted as unfinished. A workflow is considered successfully finished if all its tasks are finished within their deadline and budget constraints.

E. Alternative State-of-the-Art Scheduling Algorithm

In order to better assess MISER scheduling algorithm proposed in this paper, we have also implemented MW DBS, a multi-workflow deadline- and budget- constrained scheduling algorithm, recently proposed by Arabnejad and Barbosa [19]. MW-DBS was designed to schedule multiple workflows on heterogeneous resources. The algorithm works in two steps: first, a ready task from each workflow is selected and a priority based on individual deadline is assigned; second, a suitable resource to execute the current task that satisfies budget and deadline constraints of the workflow is determined. MW-DBS assigns tasks only on free resources at scheduling round, unlike MISER which is able to plan allocations on idle slots between two already scheduled tasks.

F. Simulator Setup

To finally evaluate the effectiveness of our solution, we used the discrete-event Cloud simulator introduced in [2]. Discrete-event simulation guarantees the repeatability and reproducibility of large-scale experiments, for a wide range of application configurations in a reasonable amount of time. The simulator implements two main entities: the Cloud manager and the Scheduler. The Cloud manager starts and terminates clusters of VM instances to serve users' requests. The Cloud manager also launches and manages containers on top of VMs according to Scheduler instructions. It is also its duty to manage the execution of individual tasks on top of containers. The Scheduler creates task-to-instance resource mappings by choosing the type and renting model of VM instances. The simulator reads workflow description files in DAX format, from the Pegasus project [9]. For each workflow, budget and deadline constraints where computed by (8) and (9), respectively. Parameters $min_{cost}(j)$ and $max_{cost}(j)$ account for the absolute highest and lowest possible monetary costs for running the workflow application *j*, and are determined by summing the maximum and the minimum execution costs for all tasks in that workflow.

$$Budget(j) = \min_{cost}(j) + B \times \left(\max_{cost}(j) - \min_{cost}(j)\right). \quad (8)$$

$$Deadline(j) = \min_{time}(j) + D \times \left(\max_{time}(j) - \min_{time}(j)\right). \quad (9)$$

Parameters $min_{time}(j)$ and $max_{time}(j)$ represent the absolute highest and lowest possible execution times for the workflow application *j*, and are determined based on the lowest and highest possible makespans for available instance types. Therefore, it is considered the processing time for the critical path and the average communication time between tasks and their critical parents. Sets of experiments were carried out by varying B in {0.25, 0.75} and D in {1.0, 2.0}. The rationale is to analyze the impact of different budgets and deadlines on the performance of the scheduling algorithms. VM instances which are part of the user compute environment are terminated as soon its workflows are finished.

V. Results and Analysis

For each budget B and deadline D defined, each scheduling algorithm was executed in three different ways, regarding the used instance models to allocate a task. For example, MISER(OBS) means that MISER algorithm considered cumulatively on-demand (O), spot block (B), and SIs (S) to schedule a task. Unlike MISER, MW-DBS is agnostic regarding reliability of SIs.

Fig. 3 shows the results for the CRT metric. The proposed algorithm, MISER, outperforms MW-DBS for all the defined budget and deadline constraints, reaching 100% of tasks successfully concluded even when tasks are allocated in SIs (i.e., the MISER(OBS) case). In turn, for low and high budgets (i.e., B = 0.25 and B = 0.75, respectively) and low deadline (i.e., D = 1.0), both MW-DBS(OBS) and MW-DBS(OB) are able to successfully execute more than 99.3% of tasks. For the same low deadline, the performance of MW-DBS(O) is more than 89% and less than 92% for low and high budgets, respectively. Now, considering the high deadline (i.e. D = 1), MW-DBS(O) behaves similarly to low deadline case, which suggests that MW-DBS schedules in more expensive instance models benefit from higher budgets, and is less sensible to the deadline. On the other hand, MW-DBS(OBS) and MW-DBS(OB) alternatives suffer performance degradation with high deadline (i.e., D = 2). We noticed that for high deadline, both MW-DBS(OBS) and MW-DBS(OB) produce schedules in which tasks run longer, which implies that additional VM instances need to be launched to execute incoming workflows (a new workflow is submitted before the last one finishes executing), incurring extra monetary costs and eventual scarcity of resources assigned to each user (MW-DBS only allocates tasks on instances that are idle at the scheduling round). Additionally, since tasks run longer for high deadline, the probability of SIs to be interrupted by EC2 augments for MW-DBS(OBS), with consequent impact on CRT. This performance behavior is not observed for MW-DBS(O), in which the number of instances to run the tasks remains almost the same, disregarding the deadline.

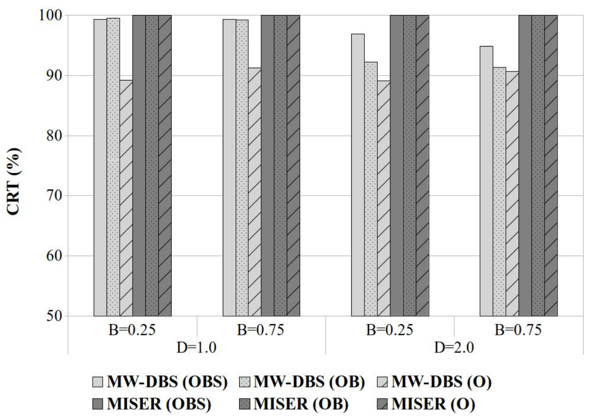

Fig. 3. Completion rate of tasks (B and D parameters define the budget and deadline for a workflow, according to (8) and (9)).

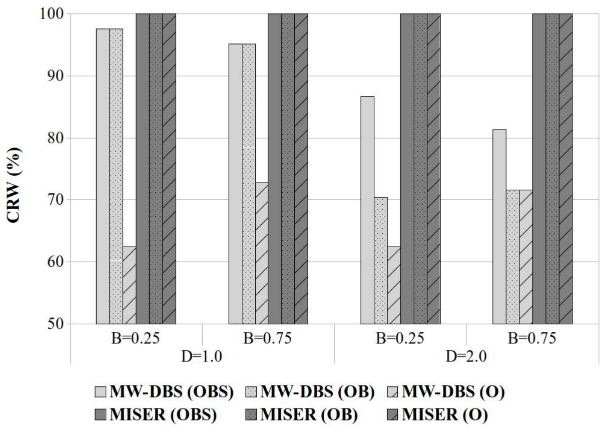

Fig. 4. Completion rate of workflows (B and D parameters define the budget and deadline for a workflow, according to (8) and (9)).

Fig. 4 illustrates the results for CRW metric. For every considered combination of instance models (i.e., OBS, OB, and O), MISER outperforms MW-DBS, accomplishing 100% of workflows. Regarding MW DBS, its behavior is similar to that of CRT, as budget and deadline evolves. In fact, high deadline allows the scheduler to allocate tasks on cheaper, and less powerful, VM instances, leading to tasks running longer and to less available resources to allocate incoming tasks.

Fig. 5 shows the results obtained for SRT. It can be observed that MW-DBS(O) is the case that presents less quality in finding a valid schedule map for each workflow application in the scenario. The quality of their schedules lightly improves with deadline. On the other hand, as deadline evolves from low to high, the performance of MW-DBS(OBS) and MW-DBS(OB) in finding valid schedule maps degrades. The loss of quality is related to longer task runtimes, less available VM instances for incoming workflows, and eventually extra monetary costs.

As a final remark, we can observe in all figures that MISER is able to build reliable compute environments on top of unreliable EC2 SIs, to successfully run workflows applications under budget and deadline constraints.

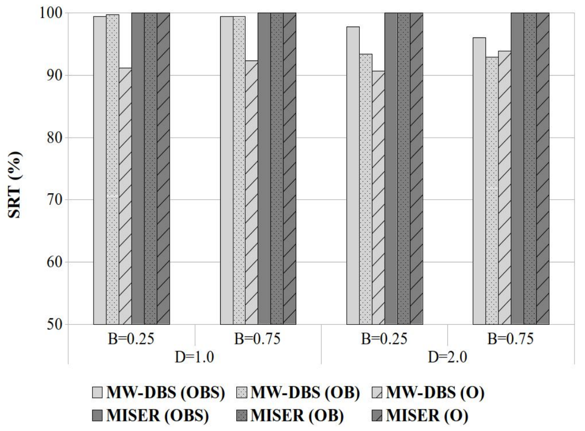

Fig. 5. Scheduled rate of tasks (B and D parameters define the budget and deadline for a workflow, according to (8) and (9).

VI. Conclusions and Future Work

Amazon EC2 spot instances represent a cheaper alternative for users comparing to traditional solutions based on on-demand purchasing model. However, the trade-off is that SIs may be reclaimed by EC2 at anytime, with a two-minute interrupt notification. In such cases, the state of running tasks is lost. In order to tackle this issue, we propose the construction of virtual compute environments on top of EC2 instances based on containers. This strategy, combined with MISER, a SI reliability-aware multi-workflow scheduling algorithm, that cumulatively considers budget and budget constraints, improves the reliability of users' compute environments. In order to assess the performance of the proposed solution, a set of experiments was carried out with MISER and a state-of-the-art scheduling algorithm. Results show the effectiveness of proposed solution in building reliable compute environments on top of unreliable Amazon EC2 instances.

For future work we are considering the optimization of monetary cost or deadline parameters, according to user expectations.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Altino M. Sampaio conducted the research, implementing the simulation scenario and conducting the experiments; Altino M. Sampaio and Jorge G. Barbosa analyzed the data; Altino M. Sampaio and Jorge G. Barbosa wrote the paper and approved the final version.

References

[1] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica, "Above the clouds: A berkeley view of cloud computing," Rep. UCB/EECS, Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, vol. 28, no. 13, 2009.

[2] A. M. Sampaio and J. G. Barbosa, "Towards high-available and energy-efficient virtual computing environments in the cloud," *Future Generation Computer Systems*, Elsevier, vol. 40, pp. 30-43, 2014.

[3] G. Vallee, T. Naughton, C. Engelmann, H. Ong, and S. L. Scott, "System-level virtualization for high performance computing," in *Proc. 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, 2008, pp. 636-643.

[4] T. P. Pham, S. Ristov, and T. Fahringer, "Performance and behavior characterization of Amazon EC2 Spot Instances," in *Proc. IEEE 11th International Conference on Cloud Computing (CLOUD*), 2018, pp. 73-81.

[5] G. Juve, and E. Deelman, "Scientific workflows in the cloud," G*rids, Clouds and Virtualization*, Springer, pp. 71-91, 2011.

[6] S. Schulte, C. Janiesch, S. Venugopal, I. Weber, and P. Hoenisch, "Elastic business process management: State of the art and open challenges for BPM in the cloud," *Future Generation Computer Systems*, vol. 46, pp. 36-50, 2015.

[7] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux Journal, Belltown Media*, vol. 2014, no. 239, pp. 2, 2014.

[8] F. Wu, Q. Wu, and Y. Tan, "Workflow scheduling in cloud: A survey," *The Journal of Supercomputing*, vol. 71, no. 9, pp. 3373-3418, 2015.

[9] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-Science: An overview of workflow system features and capabilities," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 528-540, 2009.

[10] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M. H. Su, and K. Vahi, "Characterization of scientific workflows," in *Proc. Third Workshop on Workflows in Support of Large-Scale Science,* pp. 1-10, 2008.

[11] M. R. Garey, and D. S. Johnson, "Computers and intractability: A guide to the theory of npcompleteness (series of books in the mathematical sciences), ed," *Computers and Intractability*, vol. 340, 1979.

[12] D. Poola, K. Ramamohanarao, and R. Buyya, "Enhancing reliability of workflow execution using task replication and spot instances," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 10, no. 4, pp. 30, 2016.

[13] A. M. Sampaio, and J. G. Barbosa, "A comparative cost analysis of fault-tolerance mechanisms for availability on the cloud," *Sustainable Computing: Informatics and Systems*, vol. 19, pp. 315-323., 2017.

[14] L. Chen, X. Li, and R. Ruiz, "Idle block based methods for cloud workflow scheduling with preemptive and non-preemptive tasks," *Future Generation Computer Systems*, vol. 89, pp. 659-669, 2018.

[15] A. C. Zhou, B. He, and C. Liu, "Monetary cost optimizations for hosting workflow-as-a-service in IaaS clouds," *IEEE Transactions on Cloud Computing*, vol. 4, no. 1, pp. 34-48, 2016.

[16] Z. Kozhirbayev, and R. O. Sinnott, "A performance comparison of container-based technologies for the cloud," *Future Generation Computer Systems*, vol. 68, pp. 175-182, 2017.

[17] R. Peinl, F. Holzschuher, and F. Pfitzer, "Docker cluster management for the cloud-survey results and own solution," *Journal of Grid Computing*, vol. 14, no. 2, pp. 265-282, 2016.

[18] L. Ma, S. Yi, and Q. Li, "Efficient service handoff across edge servers via docker container migration," in *Proc. the Second ACM/IEEE Symposium on Edge Computing*, 2017, p. 11.

[19] H. Arabnejad and J. G. Barbosa, "Maximizing the completion rate of concurrent scientific applications under time and budget constraints," *Journal of Computational Science*, vol. 23, pp. 120-129, 2017.

[20] H. Topcuoglu, S. Hariri, and M. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260-274, 2002.

[21] A. M. Sampaio, and J. G. Barbosa, "A study on cloud cost efficiency by exploiting idle billing period fractions," in *Proc. 2016 IEEE International Conference on Cloud Engineering Workshop* (IC2EW), 2016, pp. 138-143.

Altino M. Sampaio received his B.Sc. degree in electrical and computer engineering from the Faculty of Engineering of the University of Porto (FEUP), Portugal, in 2002, and the Ph.D. degree in informatics engineering from FEUP, in 2015. Since 2008 he serves as an assistant professor in several computer science courses at Instituto Politécnico do Porto, Porto, Portugal. His research interests are cloud and distributed computing systems, especially resource scheduling, considering fault tolerance and energy optimization.

Jorge G. Barbosa received the B.Sc. degree in electrical and computer engineering from the Faculty of Engineering of the University of Porto (FEUP), Portugal, the M.Sc. degree in digital systems from the University of Manchester Institute of Science and Technology, England, in 1993, and the Ph.D. degree in electrical and computer engineering from FEUP, Portugal, in 2001. Since 2001 he is an assistant professor at FEUP. His research interests are related to parallel and distributed computing, heterogeneous computing, scheduling in heterogeneous environments, cloud computing and biomedical engineering.