algorithms

MDPI

Article

Constructing Reliable Computing Environments on Top of Amazon EC2 Spot Instances †

Altino M. Sampaio 1,* and Jorge G. Barbosa 2

1 CIICESI, Escola Superior de Tecnologia e Gestão, Instituto Politécnico do Porto, 4610-15 Felgueiras6, Portugal
2 LIACC, Departamento de Engenharia Informática, Faculdade de Engenharia, Universidade do Porto, 4200-465 Porto, Portugal; jbarbosa@fe.up.pt
* Correspondence: ams@estg.ipp.pt

† The 2019 International Conference on High Performance Computing & Simulation (HPCS 2019), Dublin, Ireland, 15–19 July 2019.

Received: 10 July 2020; Accepted: 29 July 2020; Published: 31 July 2020

check for updates

Abstract: Cloud provider Amazon Elastic Compute Cloud (EC2) gives access to resources in the form of virtual servers, also known as instances. EC2 spot instances (SIs) offer spare computational capacity at steep discounts compared to reliable and fixed price on-demand instances. The drawback, however, is that the delay in acquiring spots can be incredible high. Moreover, SIs may not always be available as they can be reclaimed by EC2 at any given time, with a two-minute interruption notice. In this paper, we propose a multi-workflow scheduling algorithm, allied with a container migration-based mechanism, to dynamically construct and readjust virtual clusters on top of non-reserved EC2 pricing model instances. Our solution leverages recent findings on performance and behavior characteristics of EC2 spots. We conducted simulations by submitting real-life workflow applications, constrained by user-defined deadline and budget quality of service (QoS) parameters. The results indicate that our solution improves the rate of completed tasks by almost 20%, and the rate of completed workflows by at least 30%, compared with other state-of-the-art algorithms, for a worse-case scenario.

Keywords: cloud computing; amazon EC2 (Elastic Compute Cloud); spot instances; reliability; scheduling; workflow applications

1. Introduction

Cloud computing is a resource provisioning and sharing paradigm, providing better use of distributed resources, while offering dynamic, flexible infrastructures and QoS. Main advantages to users include easy access to resources from anywhere and at anytime. As cloud computing follows a utility model of consumption, users consume computing power based on their expected needs and are charged for what they use [1,2]. The cloud computing paradigm exploits virtualization to provision computational resources in the form of virtual machine (VM) instances [3]. Computational capacity in the cloud, such as the service provided by Amazon EC2, has been progressively adopted to run a wide range of applications encompassing various domains, including science, engineering, consumer, and business [4–8]. Many of these applications are commonly modeled as workflows. Typically, a workflow is described by a graph that consists of a set of nodes (or vertices) and a set of edges, where nodes represent computational tasks or data transfers, and edges represent control and data dependencies [9]. In particular, many of workflow applications fall into the category of directed acyclic graphs (DAGs) [10], where the edges represent the temporal relations between the tasks. The use of cloud computing to execute workflow applications is advantageous because users do not have to concern themselves with managing and maintaining their own hardware and software infrastructure, or with service placement and availability issues.

Amazon EC2, a pioneer in cloud services and one of the world's largest players in cloud computing, provides a wide selection of instance types, comprising diverse combinations of CPU, memory, storage, and networking capacity, and four ways to pay for those computing instances, namely: (1) on-demand; (2) spot instances (SIs); (3) reserved instances; saving plans; and (4) dedicated hosts (the most expensive option) which grant users with EC2 instance capacity on dedicated physical servers [11–13]. Saving plans use a flexible pricing model that offers low prices on EC2 in exchange for commitments to a consistent amount of usage for a 1 or 3 year term. Reserved instances imply the payment of a yearly fee (of hundreds to thousands of dollars) and buy users the ability to launch reserved instances whenever they want to and receive a significant discount for the resource renting, which decreases the average monetary cost. This option is attractive when utilization can be planned in advance, leading to the low resource utilization rate otherwise. On the other hand, on-demand instances allow users to pay for computational capacity by the hour or second (minimum of 60 s) with no long-term commitments, but the hourly fee is a bit higher. In order to improve Amazon EC2 data center's utilization, SIs allow users to bid for idle resource capacity for up to 90% of the on-demand price. Users can launch SIs in two different forms: fleet (the default form) or block. In either case, users simply submit a SI request and specify the amount of resources they need, along with the maximum price per hour that they are willing to pay. When SIs are rented as a block, users must further specify the number of hours they want their instances to run, which can range from one to six hours. Their price is based on the requested duration and is typically 30% to 45% less than on-demand instances (i.e., spot blocks and SIs are priced separately). Despite SIs corresponding to the cheapest purchasing model, two relevant trade-offs need to be considered: (1) they provide no guarantee regarding both launch and termination-time; and (2) in the event of lack of EC2 resources for on-demand or reserved instances, SIs rented in the form of a fleet will be reclaimed with a two minutes of a notification. This second trade-off does not affect the availability of spot blocks, since they run for the specified number of hours (1–6 h). Recent observations based on experimental studies for EC2 SIs [14] show that less than 76% of spot requests can be fulfilled, and as for these fulfilled requests, only about 81% of spot requests have been fulfilled within 4 s, for a worse-case scenario.

Scheduling workflow applications on clouds has the potential to immensely reduce monetary cost and time (e.g., two of the most relevant user concerns nowadays). Workflow scheduling is a well-known NP-complete problem [15] and refers to the process of spatial and temporal mapping of workflow tasks onto resources in order to satisfy some or multiple performance criteria. In the specific case of Amazon EC2, the scheduling of workflow tasks onto SIs to further reduce the monetary costs needs to be addressed carefully because of their unreliable behavior. Furthermore, since complex applications such as workflows may consist of thousands of tasks, unreliable resources hinder the execution progress of the tasks. This paper considers the problem of scheduling scientific workflows using on-demand, spot block, and SIs options under budget and deadline constraints. Schedules of workflow tasks on SIs may result in significant monetary cost reduction, although at the expense of computational availability. This challenge emphasizes the need for an effective budget and deadline constraint workflow scheduling strategy that is able to mitigate resource failures, caused by the unreliable nature of SIs and the uncertainty of spot request fulfillment, and satisfy user requests regarding to monetary cost and time. To the best of our knowledge, this is the first attempt to leverage container [16] migration to deal with the specific case of EC2 SI interruption and delay in acquiring SIs to provide reliable computing environments to end users.

To summarize, the major contributions of this paper are as follows:

1. A review of multi-criterion scheduling algorithms for EC2 resource provisioning that considers the diversity of instance types and pricing models;
2. The development of a best-effort multi-workflow deadline and budget-constrained scheduling algorithm to execute scientific workflows on Amazon EC2 infrastructures;

3. The development of a dynamic scheduling strategy to provide reliable virtual computing environments on top of Amazon EC2 on-demand, spot block, and unreliable SIs, which reactively detects reclaimed SIs and unfulfilled spot requests, and to perform resource reconfiguration to maintain required QoS in terms of deadline and budget;
4. An extensive evaluation of the dynamic strategy and the scheduling algorithm with applications that follow real-life scientific workflow characteristics constrained by user-defined deadline and budget QoS parameters, and with SIs behaving and performing accordingly to the results of recent experimental evaluations of Amazon EC2 SIs.

The scheduling algorithm proposed in this paper is an improved version of our previous work [17]. Additionally, we introduce an extended evaluation using an additional state-of-the-art scheduling algorithm and further detailed results are provided. The migration time of containers was extended to reflect a more realistic scenario. The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 introduces a proposal overview, by characterizing the system, workflow applications, scheduling algorithm, and feasibility of using container migration. Section 4 presents the test scenario by describing the simulator configuration parameters, the workloads, the algorithms used for comparison, the SI interruptions, and the performance metrics. Section 5 discusses the obtained results. Conclusions are presented in Section 6.

2. Related Work

Scientific applications and experiments are usually composed of a certain number of tasks, with various computational and data needs. Such experiments were initially addressed with dedicated high-performance computing (HPC) infrastructures, and new capabilities (e.g., selection of most appropriate set of machines meeting the applications requirements) have been lately introduced with grid computing [18]. In turn, clouds offer many technical and economic advantages over other distributed computing models. By using cloud-based services, scientists have easy access to large distributed infrastructures, with the ability to scale up and down the computing resources according to the application's needs, available in a pay-as-you-go manner. From the scientists point of view, cloud computing provides many resources they need, when they need them, and for as long as they need them. In this regard, commercial clouds such as Amazon EC2 have been the target of relevant research over the last few years towards the execution of scientific applications. Much of the research conducted has focused particularly on developing effective fault-tolerant and robust workflow scheduling algorithms that use spot and on-demand resources to schedule workflow tasks in an effort to minimize the execution cost of the workflow and at the same time satisfy the deadline constraint. This is the case of Deepak et al. [13], who have proposed an adaptive, just-in-time scheduling algorithm for scientific workflows. The presented scheduling algorithm, essential critical path task replication (ECPTR), uses both spot and on-demand instances and consolidates resources to further minimize execution time and monetary cost. Fault tolerance regarding eventual SI interruption is achieved by means of replication of tasks. Based on extensive simulations, the authors have shown the effectiveness and robustness of proposed heuristics in scheduling scientific workflow tasks with minimal makespan and monetary cost. Replication of tasks is a fault tolerance technique based on redundancy, but previous studies [19] have shown that it increases monetary costs due to the additional consumption of resources. Long et al. [12] have addressed related problems, namely: (1) SIs' unreliability caused by the fluctuations of the bidding prices (i.e., a SI may be terminated at any time when the bidding price is lower than the spot price); and (2) the appropriate number of spot and on-demand resources in terms of users' requirements. Aiming at minimizing the total renting cost under deadline constraints, the authors propose to construct schedules for workflows with both non-preemptive and preemptive tasks on spot block and on-demand instances. The idle time three step block-based algorithm (ITB) determines the block time for SIs and improves the task-to-instance mapping. Based on sets of experiments, the authors demonstrate the effectiveness of the proposed algorithm in reducing monetary cost on average, compared to scheduling strategies with only on-demand instances. Unfortunately, Amazon EC2 spot block instances represent a more expensive alternative than SIs, which gives us room to exploit SIs rented in the form of a fleet in order to decrease monetary costs with tasks execution. Additionally, today's SIs' unreliability is caused mainly by scarcity of EC2 resources, since spot prices tend to be relatively stable now. Zhou et al. [20] have developed a probabilistic framework named Dyna for the scheduling of scientific workflows aimed at minimizing the monetary cost while satisfying their probabilistic deadline guarantees. The authors also presented a hybrid instance configuration refinement mechanism of spot and on-demand instances for price dynamics. The Dyna framework was deployed on Amazon EC2 and experiments were carried out. The authors concluded that their solution is able to achieve lower monetary cost than the state-of-the-art approaches while accurately meeting users' probabilistic requirements. Nonetheless, the solution lacks SI unreliability due to EC2 resource scarcity events which cause premature termination of running tasks. In an effort to efficiently utilize both spot and on-demand resources, Lee and Irwin [21] have proposed SpotCheck, a derivative cloud computing platform to transparently manage the risks associated with using spot servers for users. The objective is to intelligently use a mix of spot and on-demand servers to provide high availability guarantees that approach those of on-demand servers at a low cost that is near that of spot servers. The mechanisms used are nested VMs with live bounded-time migration to eliminate the risk of losing VM state. The authors claim that SpotCheck is able to achieve a cost that is nearly five times less than that of the equivalent on-demand servers, with nearly five times the availability, and little performance degradation. Since nested VMs introduce a higher performance overhead than containers do [22,23], in a later project called HotSpot, Shastri and Irwin [24] proposed using containers inside EC2 instances to deal with revocation risk as spot prices change. An evaluation of the prototype, using job traces from a production Google cluster allocated to on-demand and spot VMs (with and without fault-tolerance) in EC2, showed that the strategy was able to lower costs and reduce the revocation risk without degrading performance. Unfortunately, both works did not consider user-defined budget and deadline constraints to execute applications. Moreover, only on-demand and SIs were taken into account, and the eventual high delay in acquiring SIs was omitted. Several other techniques encompassing task migration [25,26], duplication [26], and checkpointing [25,27,28] have been proposed. Mishra et al. [29] have published an article extensively surveying on improving the fault tolerance level of spot instances in Amazon EC2. Much of the research was devoted to tolerating the out-of-bid events, neglecting the problems of delay and failure in acquiring spot instances. Furthermore, in early 2018 Amazon EC2 changed the spot pricing algorithm, and since then spot instances have given predictable prices that adjust slowly over days and weeks, depending less on bidding strategies, which makes their acquisition more uncertain. Regarding the checkpoint-restart mechanism, it is worth noting that it introduces a significant performance overhead in terms of the task execution time.

Further research addressed the problem of workflow scheduling on heterogeneous instances (i.e., not necessarily focusing on Amazon EC2 cloud computing, but on distributed systems in general), in which workflows had individual deadlines and budgets, which are two conflicting QoS parameters. Such was the case in the multi-workflow deadline-budget scheduling (MW-DBS) algorithm proposed by Arabnejad and Barbosa [30]. The objective of the proposal is to schedule multiple workflows that can arrive in the system at any time and satisfy individual workflow requirements in terms of monetary costs and deadline. The purpose of MW-DBS is not to perform optimization but to guarantee that the deadline and budget defined for each job are not exceeded. MW-DBS algorithm works in two phases, namely, task selection and processor/instance selection. In the first phase, a priority is assigned to each task, based on its critical path. The critical path in a workflow application is defined as the longest execution path from the start activity to the end activity [31]. From each workflow application, a single ready-to-execute task with the highest priority is selected and added into a ready-to-execute pool of tasks. To determine which task should be selected for scheduling among the ready-to-execute pool of tasks, a priority is assigned to each task, which is inverse to its deadline and proportional to the ratio of the number of scheduled tasks to the total number of tasks in the workflow application. The task with the highest priority is selected for scheduling. In the second phase, the algorithm selects a processor to run the task, based on the combination of the two QoS factors, time and cost, in order to obtain the best balance between time and cost minimum values. Additionally, Chen et al. [32] suggested a scheduling algorithm to allocate tasks from different workflows. The algorithm adequately exploits the idle time slots on resources to improve the cost and resource efficiency, while guaranteeing the deadlines of workflows. For each task of each workflow, the latest start and finish times are calculated to determine when each task must be executed in order to guarantee workflows finish before their deadlines. The task with the smaller latest start time is allocated first to a VM.

Unlike the above work, we propose constructing reliable computational environments on top of on-demand, spot block, and unreliable SIs, to execute scientific workflows and respect their budget and deadline constraints.

3. Reliable Computing Environments

This section provides a formal description of the proposal to build migration-based reliable virtual compute environments on top of on-demand, spot block, and unreliable SIs.

3.1. System Overview

We assume a typical cloud service provider, Amazon EC2, where a set of resources is available to users via VM instances that may be provisioned and charged per time unit. Typically, the price is defined in such a way that VM instances with the most powerful processors have the highest cost and those with least powerful processors are cheaper. VM instances are charged according to one of the following models: on-demand, spot block, and SIs. The last option represents the cheapest alternative, although the instance can be interrupted with two minutes of warning, causing the early termination of the running task.

Figure 1 shows the proposed system for building reliable virtual cluster execution environments. In this scenario, users submit their workflows to the cloud service provider (CSP) and specify the budget limit and deadline based on the workflow's longest task. The CSP reserves the necessary resources, in the form of VM instances, from the cloud infrastructure to execute workflow applications with budget and deadline constraints. Instances differ in type (t2.small, etc.) and pricing model (e.g., on-demand, spot block, and spot instances). Workflow deadlines become activated right after submission. The CSP implements four main modules: (a) scheduler module; (b) payment model selector module; (c) container controller module; and (d) interrupt event handler module. The scheduler module has no knowledge of when workflow applications arrive. It decomposes users' workflow applications into a set of tasks and runs the resource allocation algorithm that creates task-to-instance mappings. This module interacts with all the other three, receiving monitoring information and managing the virtual cluster execution environments. The payment model selector gives instructions on behalf of scheduler about the instance model to choose to execute a specific task. It is the responsibility of the container controller module to launch and manage containers on top of VM instances over which tasks will execute. A container has full access to its VM resources. The set of containers will form the user's virtual cluster executing environment. The interrupt event handler module is in charge of monitoring spot interruption events and of informing the scheduler module to find another instance to which the container executing the task will be migrated. It is also the interrupt event handler module's duty to monitor the spot request fulfillments. The objective is to mitigate the unreliable nature of SIs, and control spot request fulfillments. 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 is ready to start.

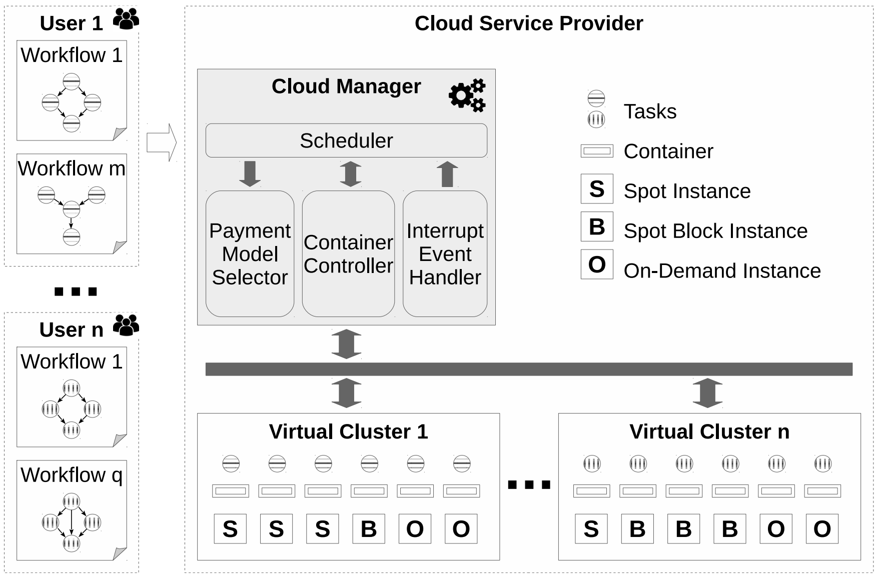


Figure 1. Cloud workflow scheduling onto reliable computing environments.

3.2. *Achieving Computational Reliability*

This paper proposes containers in Linux to construct reliable virtual computing environments. Container-based virtualization implements isolation of processes at the operating system level, thereby avoiding the overhead typically imposed by hypervisor-based virtualization [16,33,34]. Each container has its own process and network space. Linux containers are implemented primarily via cgroups and namespaces in recent Linux kernels. While the cgroups mechanism allows resource manipulation (e.g., limits and priorities) for groups of processes, namespaces provides a private, restricted view towards certain system resources within a container (i.e., a form of sandboxing) [35]. Several container-based solutions exist nowadays, such as LXC [36], Docker [16], OpenVZ, and CoreOs Rocket [34]. As containers on a host share the underlying operating system kernel, container migration is much more lightweight than a VM migration. Previous works have shown the effectiveness of said tiering approach (i.e., running containers within VM instances) in terms of performance [37], and as a first solution with which to implement fault-tolerance mechanism to run the applications on spot VMs [24]. Moreover, in [38–40] it is shown that migration of containers running various real applications is accomplished within a few tens of seconds at most. These results are confirmed by Shastri and Irwin in their work on handling VMs spot price fluctuations on EC2 [24]. More specifically, authors refer to the following key aspects regarding migration of containers: (1) the time to transfer a container's memory state and restore it as a function of its memory footprint; (2) memory-to-memory stop-and-copy transfers (i.e., copies the source container's memory state to the memory of the destination container without saving it to stable storage) are near linear in the amount of data transferred; (3) memory-to-memory transfer enables migrations of up to 32 GB in ∼30 s using EC2's 10 Gbps interfaces; (4) the time needed to transfer the container's disk and network is ∼28 s, in average. In turn, a couple of seconds is sufficient to deploy and boot up a container, which remains essentially constant with respect to the image size [41]. Since the application downtime is a function of the time to disconnect and reconnect the container's disk and network interfaces, and physically transfer the container's memory state, the average time to migrate a container ranges from 30 to 90 s [24], depending on the size of the memory state. Therefore, the time needed to complete the migration of a container is much less than the two-minute SI interruption notice.

3.3. *Workflow Application Model*

The solution proposed in this work focuses on building reliable virtual cluster computing environments to execute scientific workflows that can be modeled as DAGs. A DAG is a directed graph with no cycles wherein the nodes in the graph represent the computational tasks and the edges represent the temporal relations between the tasks. 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. It is assumed that all workflow data are stored in a shared cloud storage system (e.g., Amazon S3), and the intermediate data transfer times are known or can be estimated. It is also assumed that data transfer times between the shared storage and the containers are equal for different containers so that task placement decisions do not impact the runtimes of the tasks. The runtime estimates and the CPU computational needs for the workflow tasks are known. Due to heterogeneity of available VM instance types, each task may have a different execution time on each container. Only workflows for which all tasks are completed on time and on budget are considered complete.

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

3.4. Scheduling Algorithm

To elaborate task-to-instance mappings, this work proposes MISER, a dynamic best-effort multi-workflow deadline and budget-constrained scheduling algorithm. It considers simultaneously budget and deadline constraints defined by users for concurrent workflow scheduling in heterogeneous computing systems. Algorithm 1 shows the details of MISER. The algorithm works in two main phases, namely, the task selection phase and VM instance selection phase. In the first phase (lines 2, 3, and 4), a priority, $rank_u$, is assigned to each task in a ready list, $ReadyList$. This ready list contains all the tasks whose state matches one of the following conditions: (1) a task is not running and is ready to start executing; (2) a task is running on a reclaimed SI; or (3) a task is waiting for an instance whose request was not fulfilled. The priority assigned to a task t_i, belonging to workflow j, is based on its critical path, as specified by Equation (1), where $\overline{ET(t_i)}$ is the average execution time of the task on available instance types offered by the cloud provider (e.g., t2.small, t2.medium, t2.xlarge, t2.2xlarge for Amazon EC2) with different performances and prices; $\overline{c_{i,s}}$ is the average communication time between tasks t_i and its successor t_s which is determined based on the average network bandwidth and latency among instance pairs.

Then, a round-robin-based strategy is applied to the ready list to select a single task with the highest priority from each workflow (lines 7, 8, and 9). For a scheduling round, the round-robin loop ends when there are no more free resources to schedule tasks or the ready list is empty (i.e., all the tasks are finally scheduled). This procedure intends to avoids a high number of ready tasks causing some workflow applications to not participate in the current scheduling round. In phase two, MISER chooses the VM instance pricing model (e.g., on-demand, spot block, and unreliable SI) and type (t2.small, etc.) that better balance deadline and monetary cost QoS parameters, and provide higher reliability (lines 10, 11, and 12). Three policies apply to instance pricing model selection according to the scheduling reason: (a) a new task t_i can be scheduled to any instance model; (b) a task forced to migrate due to SI reliability issues can be scheduled to any already running instance or to a new on-demand instance; and (c) a task waiting for spot request fulfillment can be re-rescheduled to an on-demand instance. The rationale behind instance selection of policies (b) and (c) is that *no-capacity* is the main reason for interruption of SIs and it has a higher impact on the fulfillment rate [14]. The instance type influences only the execution time and monetary cost. Only instances that comply with budget and deadline constraints will integrate the final set of VM candidates, VM_{accept}. The budget constraint is implemented by cost policy $CP(t_i, j)$ as defined by Equation (2), and is determined based on two factors: (a) $Cost_{min}(t_i, j)$, the minimum execution cost of task t_i, belonging to workflow j, among available instance candidates; and (b) $\Delta_{Cost}(j)$, the spare budget which is determined as the difference between remaining budget at a scheduling round, and the cheapest monetary cost assignment for unscheduled tasks belonging to workflow j.

Algorithm 1 MISER scheduling algorithm.

```
1:  procedure MISER ($ReadyList$)
2:      for $t_{i,j} \in ReadyList$ do
3:          $rank_u(t_i, j)$                      ▷ A priority is assigned to each task based on Equation (1)
4:      end for
5:      $VM_{free} \leftarrow$ free resources
6:      $c \leftarrow 0$                          ▷ $c$ is necessary for round-robin strategy in line 8
7:      while $ReadyList.size > 0$ & $VM_{free} > 0$ do
8:          $j \leftarrow$ next workflow $MOD\ \ c$++            ▷ Round-robin to select the next workflow
9:          $t_i \leftarrow$ task of workflow $j$ with highest $rank_u$     ▷ Gets task with highest priority
10:         $VM_{accept} \leftarrow$ instance model policy $\cap$ $VM_{free}$   ▷ Choose instances assuring sched. policy
11:         $VM_{accept} \leftarrow CP \cap VM_{accept}$        ▷ Instances must follow cost policy in Equation (2)
12:         $VM_{accept} \leftarrow DP \cap VM_{accept}$      ▷ Instances must follow deadline policy in Equation (3)
13:         for $VM_k \in VM_{accept}$ do
14:             find quality measure $Q(t_i, VM_k)$       ▷ For each instance determine Q in Equation (7)
15:         end for
16:         $VM_{sel} \leftarrow$ instance $VM_k$ with highest $Q$
17:         assign task $t_i$ to $VM_{sel}$
18:         update $\Delta_{Cost}(j)$
19:         $VM_{free} \leftarrow VM_{free} - VM_{sel}$
20:         remove task $t_i$ from $ReadyList$
21:     end while
22: end procedure
```

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

In turn, the deadline constraint is implemented by deadline policy $DP(t_i, j)$. Based on the workflow application deadline, $DP(t_i, j)$ assigns a sub-deadline to task t_i, which is computed recursively by traversing the task graph upwards, starting from the exit task. Equation (3) shows that the sub-deadline depends on the $ET_{min}(t_s)$, the minimum execution time of task t_s, the successor of t_i, among available instances. $DP(t_s, j)$ is the sub-deadline of t_s, $\overline{c_{i,s}}$ is the average communication time between tasks t_i and its successor t_s, and $MT_{max}(t_i)$ is the maximum migration time for t_i among available instances. The Bernoulli parameter $\Theta(t_i)$ ensures migration time accounting for whether SIs are allowed by the system scheduling policy. The workflow application deadline defines the sub-deadline of the exit task. The rationale behind the minimum execution time 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. Actually, this is the policy applied by the well known HEFT scheduling algorithm [42].

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

The next step consists of selecting VM_{sel}, that is, the best VM instance to run t_i from the set of acceptable VM candidates VM_{accept} (lines 13, 14, and 15). Three relative quantities are defined, namely, time quality T_Q, monetary cost quality C_Q, and reliability quality R_Q for task t_i on each acceptable instance $VM_k \in VM_{accept}$. Equations (4)–(6) define T_Q, C_Q, and R_Q, respectively. In order to make the value of each relative quantity fall into the $[0, 1]$ interval, T_Q, C_Q are normalized. Since R_Q is calculated as the difference between the unity and a cumulative distribution function, the result naturally falls within $[0,1]$ interval.

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

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. T_Q reaches the unity for a VM instance providing the minimum execution time.

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

where $Cost(t_i, j, VM_k)$ is the monetary cost of 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. C_Q reaches unity for the cheapest VM instance. Instance pricing models present different reliability. While on-demand instances remain active until they are terminated by the client, spot block instances will not be terminated within a specified duration of 1–6 h. On the contrary, SIs can be reclaimed by EC2 at any time. Reliability quality equals the unity for on-demand and spot block instances. For SIs, reliability quality is probabilistically determined based on the cumulative distribution function. Equation (6) shows the reliability quality $R_Q(t_i, j, VM_k)$ for an instance VM_k, where $\chi(t_i, j, VM_k)$ 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). CDF_{logn} tends to the unity as $FT(t_i, j, VM_k)$ distantiates from SI requesting time. The rationale is to favor selection of fresh SIs over long-running ones, in order to probabilistically avoid failure and to balance monetary cost and reliability since SIs are cheaper yet less reliable. In this work, we used the log-normal probability distribution because the curve fitted well with the data provided by recent studies on performance and behavior characterization of Amazon EC2 SIs [14]. In a real usage scenario, μ and σ can be adjusted online based on historical data.

$$R_Q(t_i, j, VM_k) = 1 - \chi(t_i, j, VM_k), \\ \text{where } \chi(t_i, j, VM_k) = \begin{cases} CDF_{logn}(FT(t_i, j, VM_k), \mu, \sigma), & \text{if } VM_k \text{ is SI} \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

The multi-objective utility function Q, defined by Equation (7), combines the three objectives T_Q, C_Q, and R_Q. By definition, utility functions not only smoothly express the degree of preference under different values for each objective, but also normalize the ranges of all the objectives into the same interval $[0,1]$. Therefore, Q is used to obtain the best balance between time, monetary cost, and reliability.

$$Q(t_i, j, VM_k) = T_Q(t_i, j, VM_k) + C_Q(t_i, j, VM_k) + R_Q(t_i, j, VM_k) \tag{7}$$

The algorithm selects the instance model and type VM_{sel} that provides the highest quality measure Q to run task t_i (lines 13, 14, and 15).

MISER is combined with a monitoring mechanism that is responsible for checking the waiting time to fulfill spot requests (i.e., characteristic that affects spot block and SIs), and for handling notification events regarding SI interruptions. As stated by Algorithm 2, MISER algorithm (i.e., Algorithm 1) will be invoked in a new scheduling iteration when an event occurs. The events that may occur are: (a) there are ready-to-execute tasks (lines 4, 5, and 6); (b) an interruption notification event is received from EC2 (lines 7, 8, and 9); (c) Equation (8) is satisfied, i.e., the spot request is waiting for fulfillment and the difference between the task sub-deadline $DP(t_i, j)$ and the finish time $FT_{\gamma_1}(t_i, j, VM_{sel})$ updated at instant γ_1 becomes smaller than the difference between updated $FT_{\gamma_1}(t_i, j, VM_{sel})$ and the expected finish time $FT_{\gamma_0}(t_i, j, VM_{sel})$ (i.e., the time the task would finish if the spot request at instant γ_0 had been fulfilled immediately) plus the monitoring window size w (lines 10, 11 and 12).

Algorithm 2 Cloud manager.

```
1: procedure RESOURCEMANAGER
2:     $event \leftarrow NULL$
3:     while $true$ do
4:         if $event =$ ready to execute then
5:             $MISER$(tasks ready to execute)
6:         end if
7:         if $event =$ ec2 interruption then
8:             $MISER$(tasks running on reclaimed SIs)
9:         end if
10:        if $event =$ request delay threshold then          ▷ Equation (8)
11:            $MISER$(tasks waiting for instance)
12:        end if
13:        $event \leftarrow$ wait for event
14:    end while
15: end procedure
```

$$DP(t_i, j) - FT_{\gamma_1}(t_i, j, VM_{sel}) < FT_{\gamma_1}(t_i, j, VM_{sel}) - FT_{\gamma_0}(t_i, j, VM_{sel}) + w \tag{8}$$

This last item handles the case in which a spot request is not fulfilled. The objective is to ensure that a task previously allocated to a new spot instance and whose request has not been fulfilled can be re-scheduled and executed by its deadline. The monitoring system verifies spot requests' fulfillment with a $w = 10$ s monitoring window size. Then, the algorithm waits for an event in line 13.

4. Evaluation Scenario

This section describes the evaluation scenario, regarding workflows and spot instances' interruption characteristics, scheduling algorithms tested, and metrics.

4.1. Workloads Description

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. In this study, we used real data by leveraging the Montage, CyberShake, and Ligo [43] workflows from the Pegasus project. As realistic workflows, the Montage, CyberShake, and Ligo applications have been used in current research [30,44–46]. The memory used by a task was randomly assigned to $\{1, 2, 4, 8\}$ GB. Taking into account the results reported with great detail in [24], which considered the latency associated with migration operations within Amazon EC2 (e.g., time to disconnect and reconnect the container's disk and network interfaces, and to physically transfer the container's memory state) and the time needed to accomplish the direct memory-to-memory network transfer of containers memory state (near linear in the amount of data transferred), we defined the respective migration time as follows: $\{1\,\text{GB} => 60\,\text{s}, 2\,\text{GB} => 61\,\text{s}, 4\,\text{GB} => 62\,\text{s}, 8\,\text{GB} => 65\,\text{s}\}$. Migration of instances within the same physical server is assumed to finish in a few seconds. Based on this, we generated a total of 190 workflow applications, randomly submitted by 25 different users. Users arrived to the system at random time instants, ranging in $\{10, 30, 50\}\%$ of the minimum execution time of last submitted workflow application. The same applied for submissions of workflow applications once the user arrives to the system.

4.2. SI Interruption Description

In a recent study, Pham et al. [14] conducted a comprehensive experimental evaluation of Amazon EC2 SIs to characterize their performance and behavior in three different regions (each in a different continent): Frankfurt (F), North Virginia (NV), and São Paulo (SP). A broad study of 3840 EC2 spot requests based on a model for the life cycle of a SI resulted in a variety of findings. These findings revealed spot request fulfillment rates of 99.2%, 75.0%, and 99.8%, in F, SP, and NV respectively. Regarding the waiting time of fulfilled spot requests, the study showed that 90.5%, 80.1%, and 89.5% of spot requests were fulfilled within four seconds in F, SP, and NV respectively. In most other cases, users had to wait for more than 60 s. Additionally, F fulfilled 100% spot requests in less than 1.38 h, whereas NV required 3.96 h to fulfill all spot requests. In turn, SP had the highest rate of long waiting times (more than 60 s). Once an SI is provisioned and deployed, it can be interrupted by EC2 after no-capacity (revocations have not been directly correlated with spot prices since early 2018, when EC2 changed how the price is set such that it now only reflects long-term changes in supply/demand and not the current price). Characterizing SI interruptions, authors reported that F and NV had similar interruption rates of approximately 12.5%, while SP resulted in 34.0% interruptions. Regarding the running time of deployed SIs with interruption, half of all interrupted SIs ran at least three hours in F, less than two hours in NV, and less than 1.5 h in SP. Additional examination concerning how long SIs run until interruption indicates that 93.5% and 91.3% of SIs run at least 30 min in F and NV, respectively. In SP, 95.2% of SIs run at least the first 20 min. Based on these findings, we generated a set of reliability periods which were then assigned to SIs launched by the CSP. We also generated a set of waiting times for fulfilled spot requests. In generation of the two sets, we considered the region with the worst performance and behavior (i.e., SP) to deeply test the effectiveness of our solution.

4.3. Specification of Instances

The underlying VM instances to allocate the containers which in turn will process the tasks are described in Tables 1 and 2. The t2 product family is a low-cost, general purpose instance type that provides a baseline level of CPU performance with the ability to burst above the baseline based on a credit refill system. According to Amazon EC2 documentation [47], a CPU credit is equal to one vCPU running at 100% CPU utilization for one minute. What happens is that t2 instances accrue CPU credits when they are idle, and consume CPU credits when they are active. As a result, an instance consuming more CPU than its baseline performance level might run out of the CPU credit balance and therefore be limited in the amount of CPU it can use. To overcome this limitation, Amazon EC2 allows t2 instances to sustain high CPU performance for as long as a workload needs it at a flat additional fee of five cents per vCPU-hour. The policy applies to the entire t2 product family, regardless of the instance type and pricing model. The price for each instance was taken from Amazon EC2 instance pricing website at the time of writing this paper (i.e., May 2020). If a spot instance is interrupted in the first instance hour the user is not charged for that usage. If the spot instance is interrupted in any subsequent hour, the user is charged for usage to the nearest second. However, if spot interruption is triggered by user then he/she is charged to the nearest second. The price of a spot block instance depends on the specified duration. Thus, the prices for spot blocks for 2, 3, 4, and 5 h were defined by extrapolating the known prices for 1 and 6 h. A spot block instance is terminated automatically at the end of the time block. It should be noted that unlike spot blocks and unreliable SIs, for which provision can take from a few seconds to tens of minutes to conclude, on-demand instances are provisioned almost immediately.

Table 1. On-demand (O) and spot instance (S) characterization price per hour.

VM Instance	O (USD/hour)	S (USD/hour)
t2.small	0.0230	0.0069
t2.medium	0.0464	0.0139
t2.xlarge	0.1856	0.0557
t2.2xlarge	0.3712	0.1114

Table 2. Spot block instance (B) characterization price per block duration.

VM Instance	B 1 h (USD/hour)	B 6 h (USD/hour)
t2.small	0.0130	0.0160
t2.medium	0.0260	0.0320
t2.xlarge	0.1020	0.1300
t2.2xlarge	0.2040	0.2600

4.4. Algorithms Considered for Comparison

We implemented two other algorithms to compare their performances with that of our algorithm, namely, ECPTR [13] and MW-DBS [30]. ECPTR heuristic schedules tasks onto free slots. If no free slots are found, then a running instance that accomplishes the task deadline is selected. When neither free slots nor running instances are found, the heuristic launches a new instance. The instance model (i.e., on-demand versus SI) is selected based on slack time. If there is no slack time, the algorithm opts by on-demand instances, choosing SIs otherwise. As stated by its authors, the algorithm selects an instance whose price to performance ratio is the lowest. A tasks is replicated (i.e., it uses additional resources) to provide fault tolerance when the deadline is short (i.e., there is no slack time). The allocation of resources to a task replica is similar to the allocation scenarios presented before for original task. The priority in task scheduling is based on their critical path. In turn, MW-DBS, a multi-workflow deadline and budget-constrained state-of-the-art scheduling algorithm works in two phases, namely, task selection and processor/instance selection. In the task selection phase, a ready task from each workflow is selected and a priority based on individual deadline is assigned. In the processor/instance selection phase, a suitable resource/instance to execute the current task that satisfies budget and deadline constraints of the workflow to which the task belongs is determined. The algorithm tries to obtain the best balance between time and monetary cost minimum values. Since MW-DBS was built to work with reliable instances, its implementation was changed to be able to schedule tasks in on-demand, spot block, and spot instances.

4.5. Performance Metrics

In order to analyze the proposed solution, we defined four metrics to score the performance of the scheduling algorithm and container-based migration strategy to build reliable virtual cluster execution environments. The first metric is shown by Equation (9) and denotes the task efficiency E_T, which is computed as the ratio of the number of successfully finished tasks T_F to the total number of tasks submitted T_S.

$$E_T = \frac{T_F}{T_S}, \tag{9}$$

The second metric, workflow efficiency E_W, is presented in Equation (10). It is determined as the ratio of successfully completed workflows W_F to the number of submitted workflows W_S.

$$E_W = \frac{W_F}{W_S}, \tag{10}$$

The third metric, planning efficiency E_P, is expressed by Equation (11), and it is calculated as the ratio of number of tasks T_R successfully scheduled to the total number of tasks submitted T_S. E_P is determined for the whole system.

$$E_P = \frac{T_R}{T_S}, \tag{11}$$

The fourth metric E_C intends to measure the performance of the scheduling algorithm in terms of both monetary costs and average task runtime. E_C is expressed by Equation (12). For a particular experiment i, E_C is determined in two steps: (1) it first calculates $C(x_i)$, which is the ratio of the task efficiency $E_T(x_i)$ to the product of average task runtime $AVG\_TR(x_i)$ and total monetary cost $MCost(x_i)$ with rented time slots found necessary to execute the tasks belonging to each workflow; (2) and then, this result is normalized with reference to the maximum result obtained, $max\{C(x_j) : j = 1, \ldots, n\}$, considering all the n experiments, in order to express the result in the range $[0, 1]$. E_C increases as: (1) the number of successfully finished tasks increases; (2) the average task execution time decreases; and (3) the monetary costs in executing tasks decreases.

$$\begin{aligned} E_C &= \frac{C(x_i)}{max\{C(x_j) : j = 1, \ldots, n\}}, \\ \text{where } C(x_i) &= \frac{E_T(x_i)}{AVG\_TR(x_i) \times MCost(x_i)} \end{aligned} \tag{12}$$

The rationale behind E_C is to measure how well the scheduling algorithm performs in providing a time and cost-effective service to end user. Both costs and runtimes are considered since none of the tested scheduling algorithms deploys the means to choose which parameter dominates the other (i.e., if runtime is more important than cost, or vice versa). Apart from the ratio of successfully finished tasks $E_T(x_i)$, the quality of the service provided also depends on the $AVG\_TR(x_i) \times MCost(x_i)$, which expresses the concept of service cost perceived by end users. By means of an example, the service cost tends to increase as the average task runtime and/or monetary costs related to instances renting are augmented.

All the metrics fall into the $[0, 1]$ interval, and the performance of the scheduling algorithm increases as the metrics approach the unity.

4.6. Simulation Setup

We simulated the cloud computing infrastructure described in Figure 1 by using the discrete-event cloud simulator introduced and validated in [2]. Discrete-event simulation allows us to ensure the repeatability and reproducibility of large-scale experiments, for a wide range of application configurations in a reasonable amount of time. The simulator consisted of two main entities: the cloud manager and the scheduler. The cloud manager started and terminated clusters of VM instances to serve users' requests, and also launched and managed containers on top of VMs according to scheduler instructions. It was also its duty to manage the execution of individual tasks on top of containers. The scheduler was responsible for scheduling tasks and selecting the type and purchasing model of VM instances. The simulator reas workflow description files in DAX format from the Pegasus project [48]. Budget and deadline limits for each workflow application were defined based on Equations (13) and (14), respectively. Sets of experiments were carried out by varying $B \in \{0.25, 0.50\}$ and $D \in \{1.0, 1.5\}$. The rationale was to analyze the impact of different budget and deadline QoS parameters on the performance of the scheduling algorithms.

$$Budget = min_{cost} + B \times (max_{cost} - min_{cost}) \tag{13}$$

where max_{cost} and min_{cost} represent the absolute highest and lowest possible costs for executing the workflow application, respectively. These two parameters are calculated by summing the maximum and the minimum execution costs for each task, respectively. Communication cost imposes overhead on deadline only, since data transfer within cloud provider usually does not incur costs.

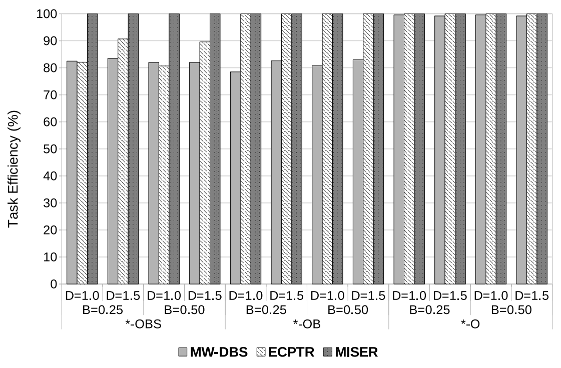
$$Deadline = min_{time} + D \times (max_{time} - min_{time}) \tag{14}$$

where max_{time} and min_{time} correspond to the highest and lowest execution time, respectively. They were determined based on the highest and lowest possible makespans for the universe of instance types under study, which were the processing time for the critical path and the average communication time between tasks and their critical parents. VM instances forming the virtual cluster were terminated when all user's workflows were finished.

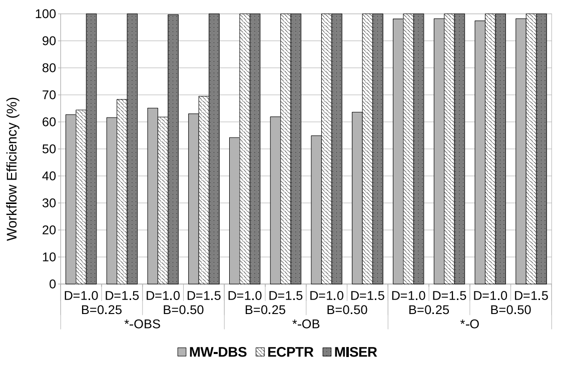
5. Results and Analysis

This section presents the results and analysis of the performance of the proposed solution. A set of 5370 workflow tasks were submitted, contained in 190 workflows, and distributed by 25 users. In the figures, B and D are the parameters introduced by Equations (13) and (14), respectively, to create several budget and deadline constraints. In turn, *-OBS, *-OB, and *-O mean that algorithms used that instance pricing models to schedule tasks (i.e., O—on-demand, B—spot block, S—SIs). For example, MW-DBS-OB means that the MW-DBS algorithm considered cumulatively on-demand and spot block instances to schedule the tasks. Unlike MISER and ECPTR, MW-DBS is agnostic regarding reliability of SIs. ECPTR-O and ECPTR-OB report the same results on every figure because the algorithm is confined to allocate tasks in on-demand and SIs instances only.

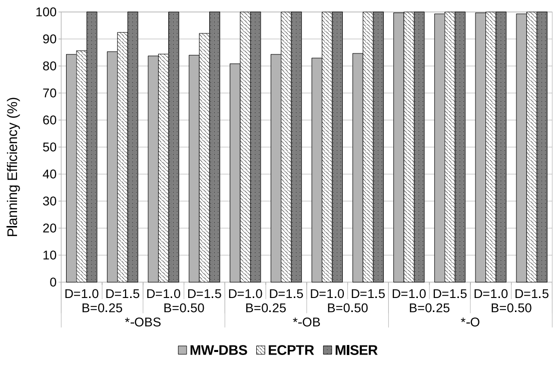
Figure 2a presents the task efficiency E_T. For the case of renting only on-demand instances (i.e., *-O), our algorithm was able to successfully finish 100% of the tasks, independently of the deadline and budget values. ECPTR was able to perform similarly to MISER, for every deadline and budget constraints. On the other hand, MW-DBS completed between 99.2% and 99.6% of tasks, for all considered combinations of deadline and budget constraints. However, the difference in performance among MISER, ECPTR, and MW-DBS was noticeable as both spot block and SIs started being considered for scheduling. In the case of virtual clusters on top of on-demand and spot block instances (i.e., *-OB), MISER finished more than 99.9% of tasks for a combination of high budget and low deadline (i.e., $B = 0.50$ and $D = 1.0$), reaching 100% of completed tasks for all the other combinations of budget and deadline constraints. As stated before, ECPTR-O and ECPTR-OB reported equal performance, since ECPTR was not designed by its authors to consider the management of spot block instances. On the other hand, MW-DBS-OB dropped its performance to values between 78.5% ($D = 1.0$ and $B = 0.25$) and 83.0% ($D = 1.5$ and $B = 0.50$) for the rate of tasks completed successfully. In this case, the performance of the algorithm marginally improved as the budget augmented, an improvement that was most noticeable as the deadline increased. Despite being more immune to deadline and budget changes, similar performance was achieved by MW-DBS when SIs were considered for scheduling. In this case, the performance varied between ∼82% ($D = 1.0$, $B = 0.50$) and ∼83.5% ($D = 1.5$, $B = 0.25$). In this case, MISER-OB outperformed MW-DBS-OB by around 20%. ECPTR-OBS combination evidenced inferior performance with respect to ECPTR-O*. In detail, when ECPTR considered the scheduling of tasks onto SIs, its performance decreased to around 81% for low deadlines. For high deadlines, the rate of completed tasks dropped to around 90%. Therefore, ECPTR-OBS performed better for higher deadlines, whilst the change in budget parameter had residual/no impact on the algorithm's performance. Comparing ECPTR-OBS and MW-DBS-OBS performances, the first clearly performs better than the former for high deadlines (i.e., $D = 1.5$), and similarly for low deadlines (i.e., $D = 1.0$). In opposition, MISER finished all tasks, irrespective of budget and deadline, outperforming MW-DBS-OBS and ECPTR-OBS by as much as almost 20%.



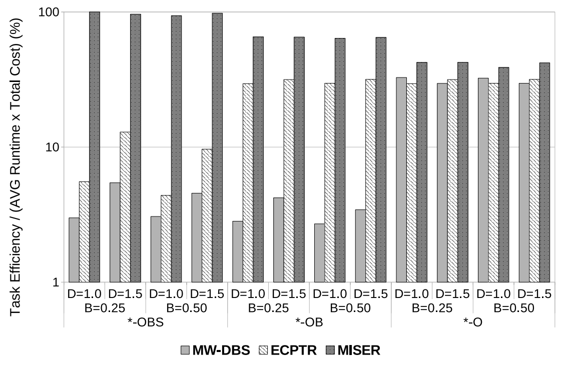
(**a**)



(**b**)



(**c**)



(**d**)

Figure 2. The performance of the scheduling algorithm and container-based migration strategy in terms of: (**a**) Task efficiency E_T. (**b**) Workflow efficiency E_W. (**c**) Planning efficiency E_P. (**d**) E_C, the ratio of the completion rate of tasks (or task efficiency) to the product of average task runtime and monetary costs (higher is better).

Figure 2b shows the workflow efficiency E_W. The performance of MW-DBS degrades as spot block and SIs start to take part in the scheduling process. The same applies for ECPTR with respect to SIs instances. By scheduling only in on-demand instances, ECPTR achieved 100% for completion rate of workflows. However, if ECPTR schedules tasks on SI resources, its performance drops and changes between ∼62% and ∼70%, as the deadline evolves from $D = 1.0$ to $D = 1.5$. Just like the case of task completion rate, the variation in budget parameter has residual/no impact on the algorithm performance. MW-DBS-O is able to successfully complete more than 97.5% of workflows, decreasing its performance as spot resources become part of the schedules. In truth, the performance of MW-DBS-OB slightly improves as the deadline increases, varying between 54.2% ($D = 1.0$ and $B = 0.25$) and 63.6% ($D = 1.5$ and $B = 0.50$). As for the *-OBS resources and MW-DBS combination, the strategy completed between ∼62% and ∼65% of workflows. Similarly to the case of task efficiency, ECPTR performs better than MW-DBS for high deadlines (i.e., $D = 1.5$). MISER showed itself to be able to maintain the completion rate of workflows at 100% for almost all cases, even when tasks were allocated on spot resources.

The planning efficiency is shown in Figure 2c. Here, the objective was to analyze the success rate of finding a valid schedule map for each workflow task in the scenario considering deadline and budget constraints. MISER proved to be stable in finding task-to-instance mappings, successfully scheduling 100% of the tasks for all considered budget and deadline constraints, and for all possible *-OBS, *-OB, and *-O configurations. What is more, MISER-OBS outperformed MW-DBS-OBS and ECPTR-OBS by as much as 16.3% and 15.6%, respectively. The ECPTR-O* performances fell from 100% of planning efficiency to 84.4% for *-OBS ($D = 1.0$ and $B = 0.50$). Here, the same pattern was observed: ECPTR performance (a) improved with increasing deadline; and (b) beat the performance of MW-DBS for high deadlines. In turn, adopting a scheduler that does not consider the nature of different instance purchasing models, as is the case of MW-DBS, leads to a planning success rate that can be as low as 83.7%, 80.8%, or 99.3 for *-OBS, *-OB, and *-O configurations, respectively.

Figure 2d characterizes the performances of the scheduling algorithms in terms of completion rate of tasks, monetary costs in executing those tasks, and average task execution time. The idea is to understand how good the algorithm is at balancing all the three factors. Comparing the results obtained for all the scheduling algorithms, the best results were achieved by the MISER and *-OBS configuration. In fact, MISER was able to utilize on-demand, spot blocks, and SIs more efficiently in order to strongly reduce the average execution time, and cumulatively slightly decrease the monetary costs with rented instances while achieving a completion rate of tasks of 100%. More precisely, the average execution time of tasks and rate of completed tasks are the two factors that most contribute to the good results obtained, compared with the factor monetary costs. The second best results were obtained for MISER and *-OB configuration. Spot block instances are cheaper than on-demand instances, which allows the scheduler to chose powerful instance types and reduce the average task execution time, while not increasing the monetary costs with resources. Regarding MW-DBS, it performs better with on-demand instances (i.e., for *-O configuration) if compared to *-OBS and *-OB configurations, due to two factors: (a) higher completion rate of tasks; and (b) lower average execution times. Similarly, ECPTR performs better with *-O and *-OB than with *-OBS configuration because of: (a) the lower average execution times; and (b) the higher completion rate of tasks.

Figure 3 identifies the number of schedules per instance pricing model, aiming at describing the instance purchasing models by which workflow tasks were scheduled. It is evident that all algorithms show a preference for resources of lower monetary cost (excepting the case of ECPTR-OB configuration, which is unable to manage spot block resources). However, the flip side is that the cheaper instances are also the less reliable ones. MW-DBS is a scheduling algorithm agnostic to the reliability of SIs, and both MW-DBS and ECPTR ignore the eventual high delay in acquiring spots which is of paramount importance to consider when executing deadline constrained tasks. This explains why the performance of MW-DBS degrades substantially as spots are chosen to allocate tasks. Similar behavior in terms of performance degradation occurs for ECPTR regarding SIs. In fact, ECPTR was designed to exploit redundancy in executing tasks to deal with the problem of premature interruption of SIs, but it is incapable of dealing with the delay in acquiring spots. Moreover, the results obtained for the case of MW-DBS-OBS configuration are in line with those of [14], in which case tasks were scheduled onto Amazon EC2 SIs. Diversely, MISER makes a better usage of unreliable SIs and spot block instances, as was shown for the cases of *-OB and *-OBS. By spending more on renting powerful yet cheaper and less reliable resources, MISER strongly accelerates the execution of tasks. Tasks executing faster are less susceptible to interruption of SIs, as time goes by. Additionally, the increasing number of instances as configuration evolves from *-O, to *-OB, and then to *-OBS, evidences that tasks initially scheduled to spot resources might need to be rescheduled onto eventually new on-demand instances because spot requests take too long to be fulfilled. Moreover, spot block instances are restricted to running for six hours at most, which can make impossible the scheduling of long tasks onto free slots of already running instances. In turn, the scheduling of tasks onto SIs is limited by their reliability, aiming at avoiding interruption, which forces allocation of tasks onto new spots over free slots of long-running ones.

Table 3 shows a detailed characterization of MISER, in the configurations of *-OB and *-OBS, in terms of unfulfilled SIs and spot block requests, additional number of schedules in relation to MISER-O configuration, and number of tasks migrated away from SIs reclaimed by EC2. The number of unfulfilled spot requests was higher for *-OBS configuration than that of *-OB, since the delay in acquiring spots affected both spot block and SIs. Additionally, migration of tasks was performed for *-OBS only because MISER had to deal with the interruption of SIs. This justifies the higher number of additional schedules observed for the MISER-OBS configuration. Considering that 190 workflows, totaling 5370 tasks, were submitted to the cloud, the additional schedules and the number of migration of tasks remained relatively small. Moreover, MISER has shown itself to efficiently use spots in order to strongly reduce the task execution time without compromising costs, thereby successfully completing 100% of tasks for almost all experiments.

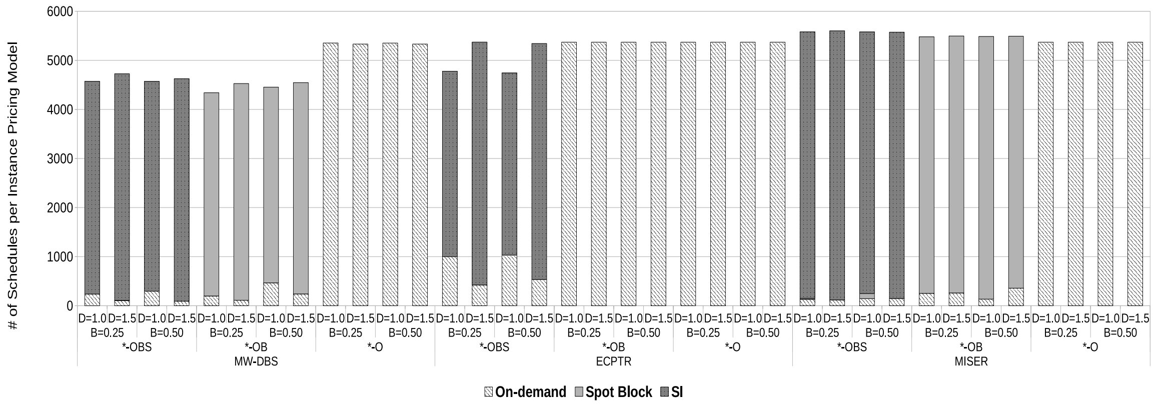


Figure 3. Characterization of the schedules regarding the instance pricing model.

Table 3. Results for MISER in terms of number of occurrences of unfulfilled spot requests, for both *-OB and *-OBS configurations; migration of tasks, for *-OBS configuration; and additional schedules compared to *-O configuration, for both *-OB and *-OBS configurations, and for diverse deadline and budget QoS constraints.

	Budget = 0.25		**Budget = 0.50**	
	Deadline = 1.0	**Deadline = 1.5**	**Deadline = 1.0**	**Deadline = 1.5**
Unfulfilled Spot Requests (*-OB)	110	126	118	122
Unfulfilled Spot Requests (*-OBS)	175	184	175	174
Migration of Tasks (*-OBS)	36	47	35	30
Additional Schedules (*-OB)	2.05%	2.35%	2.18%	2.27%
Additional Schedules (*-OBS)	3.93%	4.30%	3.89%	3.80%

6. Conclusions

In this paper, we have proposed a multi-workflow scheduling algorithm, allied with a container migration-based mechanism, to dynamically construct and readjust virtual clusters on top of non-reserved Amazon EC2 pricing model instances. Our objective is to address the unreliable behavior of Amazon EC2 spots and make it possible to use these instances to execute workflow applications constrained by user-defined deadline and budget QoS parameters. This objective implies an increase in the rate of completed tasks and in the rate of completed workflows. To achieve said objective, we developed MISER, a best-effort scheduling algorithm, which: (a) leverages containers to ease migration of tasks between spots (to deal with premature termination of SIs); and (b) dynamically re-allocates tasks that were previously scheduled to spots and whose requests were not timely fulfilled. The tests were conducted by injecting a set of 190 Montage, CyberShake, and Ligo workflow applications. Various budget and deadline limits for each workflow application were defined in order to analyze their impacts on the performances of the scheduling algorithms. The proposed solution completed almost 100% of tasks and more than 99.7% of workflows, which improves the rate of completed tasks by almost 20%, and the rate of completed workflows by at least 30%, compared with other state-of-the-art algorithms, for a worse-case scenario. What is more, MISER algorithm was able to maintain its performance with diverse deadline and budget constraints, and diverse EC2 instance pricing models. Additional interesting findings related to MISER are: (a) the quality of service provided is improved as spots are included in the scheduling process (i.e., the ratio of successfully finished tasks is kept at almost 100% while the service cost perceived by end users is improved); (b) the efficient management of diverse EC2 instances to reduce costs and execution times (i.e., the product of average task runtime and the monetary costs related to instances renting decreases as spots are introduced in the scheduling process); and (c) the small number of migrations observed, to deal with SIs interruptions (only 0.86% of a maximum of 5483 tasks scheduled onto SIs needed to be migrated).

Author Contributions: Conceptualization, A.M.S.; methodology, A.M.S. and J.G.B.; software, A.M.S.; validation, A.M.S. and J.G.B.; formal analysis, A.M.S. and J.G.B.; investigation, A.M.S. and J.G.B.; resources, A.M.S.; data curation, A.M.S. and J.G.B.; writing—original draft preparation, A.M.S.; writing—review and editing, A.M.S. and J.G.B.; visualization, A.M.S.; supervision, J.G.B.; project administration, A.M.S. and J.G.B.; funding acquisition, A.M.S. and J.G.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Fox, A.; Griffith, R.; Joseph, A.; Katz, R.; Konwinski, A.; Lee, G.; Patterson, D.; Rabkin, A.; Stoica, I. Above the clouds: A berkeley view of cloud computing. *Dept. Electr. Eng. Comput. Sci. Univ. Calif. Berkeley Rep. UCB/EECS* **2009**, *28*, 2009.
2. Sampaio, A.M.; Barbosa, J.G. Towards high-available and energy-efficient virtual computing environments in the cloud. *Future Gener. Comput. Syst.* **2014**, *40*, 30–43. [CrossRef]
3. Vallee, G.; Naughton, T.; Engelmann, C.; Ong, H.; Scott, S.L. System-level virtualization for high performance computing. In Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing, Toulouse, France, 13–15 February 2008; IEEE: Piscataway, NJ, USA, 2008; pp. 636–643.
4. Juve, G.; Deelman, E. Scientific workflows in the cloud. In *Grids, Clouds and Virtualization*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 71–91.
5. Schulte, S.; Janiesch, C.; Venugopal, S.; Weber, I.; Hoenisch, P. Elastic Business Process Management: State of the art and open challenges for BPM in the cloud. *Future Gener. Comput. Syst.* **2015**, *46*, 36–50. [CrossRef]
6. Challa, S.; Das, A.K.; Gope, P.; Kumar, N.; Wu, F.; Vasilakos, A.V. Design and analysis of authenticated key agreement scheme in cloud-assisted cyber–physical systems. *Future Gener. Comput. Syst.* **2020**, *108*, 1267–1286. [CrossRef]
7. Sun, G.; Zhou, R.; Sun, J.; Yu, H.; Vasilakos, A.V. Energy-efficient provisioning for service function chains to support delay-sensitive applications in network function virtualization. *IEEE Internet Things J.* **2020**, *7*, 6116–6131. [CrossRef]
8. Huang, M.; Liu, A.; Xiong, N.N.; Wang, T.; Vasilakos, A.V. An effective service-oriented networking management architecture for 5G-enabled internet of things. *Comput. Netw.* **2020**, *173*, 107208. [CrossRef]
9. Wu, F.; Wu, Q.; Tan, Y. Workflow scheduling in cloud: A survey. *J. Supercomput.* **2015**, *71*, 3373–3418. [CrossRef]
10. Deelman, E.; Gannon, D.; Shields, M.; Taylor, I. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Gener. Comput. Syst.* **2009**, *25*, 528–540. [CrossRef]
11. Agmon Ben-Yehuda, O.; Ben-Yehuda, M.; Schuster, A.; Tsafrir, D. Deconstructing amazon ec2 spot instance pricing. *ACM Trans. Econ. Comput.* **2013**, *1*, 16. [CrossRef]
12. Chen, L.; Li, X.; Ruiz, R. Idle block based methods for cloud workflow scheduling with preemptive and non-preemptive tasks. *Future Gener. Comput. Syst.* **2018**, *89*, 659–669. [CrossRef]
13. Poola, D.; Ramamohanarao, K.; Buyya, R. Enhancing reliability of workflow execution using task replication and spot instances. *ACM Trans. Auton. Adapt. Syst. (TAAS)* **2016**, *10*, 30. [CrossRef]
14. Pham, T.P.; Ristov, S.; Fahringer, T. Performance and Behavior Characterization of Amazon EC2 Spot Instances. In Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 2–7 July 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 73–81.
15. Garey, M.R.; Johnson, D.S. Computers and intractability: A guide to the theory of npcompleteness (series of books in the mathematical sciences), ed. In *Computers Intractability*; WH Freeman and Company: New York, NY, USA, 1979; Volume 340.
16. Merkel, D. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.* **2014**, *2014*, 2.

17. Sampaio, A.M.; Barbosa, J.G. Enhancing the Reliability of Compute Environments on Amazon EC2 Spot Instances. In Proceedings of the 2019 International Conference on High Performance Computing & Simulation (HPCS), Dublin, Ireland, 15–19 July 2019.
18. Mateescu, G.; Gentzsch, W.; Ribbens, C.J. Hybrid computing—Where HPC meets grid and cloud computing. *Future Gener. Comput. Syst.* **2011**, *27*, 440–453. [CrossRef]
19. Sampaio, A.M.; Barbosa, J.G. A comparative cost analysis of fault-tolerance mechanisms for availability on the cloud. *Sustain. Comput. Inform. Syst.* **2018**, *19*, 315–323. [CrossRef]
20. Zhou, A.C.; He, B.; Liu, C. Monetary cost optimizations for hosting workflow-as-a-service in IaaS clouds. *IEEE Trans. Cloud Comput.* **2016**, *4*, 34–48. [CrossRef]
21. Sharma, P.; Lee, S.; Guo, T.; Irwin, D.; Shenoy, P. Spotcheck: Designing a derivative iaas cloud on the spot market. In Proceedings of the Tenth European Conference on Computer Systems, Bordeaux, France, 21–24 April 2015; ACM: New York, NY, USA, 2015; p. 16.
22. Williams, D.; Jamjoom, H.; Weatherspoon, H. The Xen-Blanket: Virtualize once, run everywhere. In Proceedings of the 7th ACM European Conference on Computer Systems, Bern, Switzerland, 10–13 April 2012; ACM: New York, NY, USA, 2012; pp. 113–126.
23. Sharma, P.; Chaufournier, L.; Shenoy, P.; Tay, Y. Containers and virtual machines at scale: A comparative study. In Proceedings of the 17th International Middleware Conference, Trento, Italy, 12–16 December 2016; ACM: New York, NY, USA, 2016; p. 1.
24. Shastri, S.; Irwin, D. HotSpot: Automated server hopping in cloud spot markets. In Proceedings of the 2017 Symposium on Cloud Computing, Santa Clara, CA, USA, 24–27 September 2017; ACM: New York, NY, USA, 2017; pp. 493–505.
25. Yi, S.; Andrzejak, A.; Kondo, D. Monetary cost-aware checkpointing and migration on amazon cloud spot instances. *IEEE Trans. Serv. Comput.* **2011**, *5*, 512–524. [CrossRef]
26. Voorsluys, W.; Buyya, R. Reliable provisioning of spot instances for compute-intensive applications. In Proceedings of the 2012 IEEE 26th International Conference on Advanced Information Networking and Applications, Fukuoka, Japan, 26–29 March 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 542–549.
27. Tang, S.; Yuan, J.; Wang, C.; Li, X.Y. A framework for amazon ec2 bidding strategy under sla constraints. *IEEE Trans. Parallel Distrib. Syst.* **2013**, *25*, 2–11. [CrossRef]
28. Abundo, M.; Di Valerio, V.; Cardellini, V.; Presti, F.L. QoS-aware bidding strategies for VM spot instances: A reinforcement learning approach applied to periodic long running jobs. In Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), Ottawa, ON, Canada, 11–15 May 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 53–61.
29. Mishra, A.K.; Umrao, B.K.; Yadav, D.K. A survey on optimal utilization of preemptible VM instances in cloud computing. *J. Supercomput.* **2018**, *74*, 5980–6032. [CrossRef]
30. Arabnejad, H.; Barbosa, J.G. Maximizing the completion rate of concurrent scientific applications under time and budget constraints. *J. Comput. Sci.* **2017**, *23*, 120–129. [CrossRef]
31. Son, J.H.; Kim, J.S.; Kim, M.H. Extracting the workflow critical path from the extended well-formed workflow schema. *J. Comput. Syst. Sci.* **2005**, *70*, 86–106. [CrossRef]
32. Chen, H.; Zhu, J.; Wu, G.; Huo, L. Cost-efficient reactive scheduling for real-time workflows in clouds. *J. Supercomput.* **2018**, *74*, 6291–6309. [CrossRef]
33. Xavier, M.G.; Neves, M.V.; Rossi, F.D.; Ferreto, T.C.; Lange, T.; De Rose, C.A. Performance evaluation of container-based virtualization for high performance computing environments. In Proceedings of the 2013 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), Belfast, UK, 27 February–1 March 2013; IEEE: Piscataway, NJ, USA, 2013; pp. 233–240.
34. Ruan, B.; Huang, H.; Wu, S.; Jin, H. A performance study of containers in cloud environment. In Proceedings of the Asia-Pacific Services Computing Conference, Zhangjiajie, China, 16–18 November 2016; Springer: Cham, Switzerland, 2016; pp. 343–356.
35. Kozhirbayev, Z.; Sinnott, R.O. A performance comparison of container-based technologies for the cloud. *Future Gener. Comput. Syst.* **2017**, *68*, 175–182. [CrossRef]
36. Helsley, M. LXC: Linux Container Tools. 2017. Available online: https://developer.ibm.com/tutorials/l-lxc-containers/ (accessed on 5 January 2019).

37. Chen, J.; Guan, Q.; Liang, X.; Vernon, L.J.; McPherson, A.; Lo, L.T.; Chen, Z.; Ahrens, J.P. Docker-Enabled Build and Execution Environment (BEE): An Encapsulated Environment Enabling HPC Applications Running Everywhere. *arXiv* **2017**, arXiv:1712.06790.
38. Al-Dhuraibi, Y.; Paraiso, F.; Djarallah, N.; Merle, P. Autonomic vertical elasticity of docker containers with elasticdocker. In Proceedings of the 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), Honolulu, CA, USA, 25–30 June 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 472–479.
39. Machen, A.; Wang, S.; Leung, K.K.; Ko, B.J.; Salonidis, T. Live service migration in mobile edge clouds. *IEEE Wirel. Commun.* **2018**, *25*, 140–147. [CrossRef]
40. Ma, L.; Yi, S.; Carter, N.; Li, Q. Efficient Live Migration of Edge Services Leveraging Container Layered Storage. *IEEE Trans. Mob. Comput.* **2018**, *18*, 2020–2033. [CrossRef]
41. Zheng, C.; Thain, D. Integrating containers into workflows: A case study using makeflow, work queue, and docker. In Proceedings of the 8th International Workshop on Virtualization Technologies in Distributed Computing, Portland, OR, USA, 15–16 June 2015; ACM: New York, NY, USA, 2015; pp. 31–38.
42. Topcuoglu, H.; Hariri, S.; Wu, M.Y. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.* **2002**, *13*, 260–274. [CrossRef]
43. Bharathi, S.; Chervenak, A.; Deelman, E.; Mehta, G.; Su, M.H.; Vahi, K. Characterization of scientific workflows. In Proceedings of the 2008 Third Workshop on Workflows in Support of Large-Scale Science, Austin, TX, USA, 17 November 2008; IEEE: Piscataway, NJ, USA, 2008; pp. 1–10.
44. Deelman, E.; Singh, G.; Livny, M.; Berriman, B.; Good, J. The cost of doing science on the cloud: The montage example. In Proceedings of the SC'08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, Austin, TX, USA, 15–21 November 2008; pp. 1–12.
45. Abrishami, S.; Naghibzadeh, M.; Epema, D.H. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *Future Gener. Comput. Syst.* **2013**, *29*, 158–169. [CrossRef]
46. Shi, J.; Luo, J.; Dong, F.; Zhang, J.; Zhang, J. Elastic resource provisioning for scientific workflow scheduling in cloud under budget and deadline constraints. *Clust. Comput.* **2016**, *19*, 167–182. [CrossRef]
47. Amazon EC2 T2 Instances. Available online: https://aws.amazon.com/ec2/instance-types/t2/ (accessed on 27 July 2020).
48. Deelman, E.; Singh, G.; Su, M.H.; Blythe, J.; Gil, Y.; Kesselman, C.; Mehta, G.; Vahi, K.; Berriman, G.B.; Good, J.; et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Sci. Program.* **2005**, *13*, 219–237. [CrossRef]