Meeting Deadlines of Scientific Workflows in Public Clouds with Tasks Replication

Rodrigo N. Calheiros, *Member, IEEE*, and Rajkumar Buyya, *Senior Member, IEEE*

Abstract—The elasticity of Cloud infrastructures makes them a suitable platform for execution of deadline-constrained workflow applications, because resources available to the application can be dynamically increased to enable application speedup. Existing research in execution of scientific workflows in Clouds either try to minimize the workflow execution time ignoring deadlines and budgets or focus on the minimization of cost while trying to meet the application deadline. However, they implement limited contingency strategies to correct delays caused by underestimation of tasks execution time or fluctuations in the delivered performance of leased public Cloud resources. To mitigate effects of performance variation of resources on soft deadlines of workflow applications, we propose an algorithm that uses idle time of provisioned resources and budget surplus to replicate tasks. Simulation experiments with four well-known scientific workflows show that the proposed algorithm increases the likelihood of deadlines being met and reduces the total execution time of applications as the budget available for replication increases.

Index Terms—Cloud computing, scientific workflows, task replication, soft deadline

1 Introduction

Among the programming paradigms available for development of scientific applications, the *workflow* model is extensively applied in diverse areas such as astronomy, bioinformatics, and physics. Scientific workflows are described as direct acyclic graphs (DAGs) whose nodes represent tasks and vertices represent dependencies among tasks. Because a single workflow can contain hundreds or thousands of tasks [1], this type of application can benefit of large-scale infrastructures.

Among such infrastructures, the one of special interest in this paper is public Cloud [2]. This is because these infrastructures are available in a pay-per-use system and can provide dynamic scaling in response to the needs of the application (a propriety known as *elasticity*). Therefore, resources for execution of the workflow can be provisioned on demand, and their number can be increased if there is enough budget to support it. This Cloud utilization model, where users obtain hardware resources such as virtual machines where they deploy their own applications, is called *Infrastructure as a Service*—IaaS. For the sake of simplicity, throughout this paper we refer Cloud IaaS providers as Cloud providers.

This capability of Clouds make them a suitable platform to host *deadline-constrained* scientific workflows. In this class of workflows, a specific *soft deadline* for completion of the workflow is assigned, along with the workflow specification, during the application submission. A soft deadline is a deadline that, when unmet, does not render the computation useless [3]. Thus, although the maximal value of the computation is achieved when the deadline is met, investment is not lost if the deadline is missed by small margins.

As the execution of the workflow in the Cloud incurs financial cost, the workflow may also be subject to a *budget constraint*. Although the budget constraint of a workflow may be a limiting factor to its capability of being scaled across multiple resources in Clouds, its structure itself also imposes significant limitation. This is because dependencies among tasks and the number of tasks ready for execution in a given time may limit the amount of resources being used in parallel for executing the workflow. Because Cloud resource providers charge resource utilization by integer time intervals, such a limitation in the workflow scalability causes situations where Cloud resources are available (i.e., their allocation time interval is paid and there are still time before it expires) but no task is ready to be executed.

To being able to schedule the workflow in such a way that it completes before its deadline, the workflow scheduler needs an estimation of the run time of applications, which may be available, for example, via analysis of historical data. However, typical Cloud environments do not present regular performance in terms of execution and data transfer times. This is caused by technological and strategic factors and can cause performance variation of up to 30 percent for execution times and 65 percent for data transfer time, as reported by Jackson *et al.* [4]. Fluctuations in performance of Cloud resources delay tasks execution, what also delays such tasks' successors. If the delayed tasks were part of the critical path of the workflow, it will delay its completion time, and may cause its deadline to be missed. Workflow execution is also subject to delays if one or more of the virtual machines fail during task execution. However, typical Cloud infrastructures offer availability

• *The authors are with the Department of Computing and Information Systems, The University of Melbourne, Australia. E-mail: {rnc, rbuyya}@unimelb.edu.au.*

Manuscript received 20 Nov. 2012; revised 15 July 2013; accepted 10 Sept. 2013. Date of publication 19 Sept. 2013; date of current version 13 June 2014. Recommended for acceptance by S. Papavassiliou.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2013.238

above 99.9 percent[1], therefore performance degradation is a more serious concern than resource failures in such environments.

Previous research in workflow scheduling in the context of Clusters and Grids typically ignore costs related to utilization of the infrastructure, and also have limitations in the capacity of taking advantage of elastic infrastructures. Existing research in execution of scientific workflows in Clouds either tries to minimize the workflow execution time ignoring deadlines and budgets or focus on the minimization of cost while trying to meet the application deadline. Furthermore, previous research implements limited contingency strategies to correct delays caused by underestimation of tasks execution time or fluctuations in the delivered performance of leased Cloud resources.

To address limitations of previous research, we propose an algorithm that uses idle time of provisioned resources to replicate workflow tasks to mitigate effects of performance variation of resources so that soft deadlines can be met. The amount of ''idle time'' available for replication can be increased if there is budget available for replica execution, which allows provisioning of more resources than the minimal necessary for meeting the deadline. Therefore, the proposed algorithm applies a deadline constraint and variable budget for replication to achieve its goals. Simulation experiments with four well-known scientific workflows show that the proposed algorithm increases the chance of deadlines being met and, for the majority of studied scenarios, reduces the total execution time of workflows as the replication budget increases.

2 Related Work

Scheduling of workflows (also referred as Direct Acyclic Graphs—DAGs) in parallel and distributed systems has been subject of extensive research. Kwok and Ahmad [5] presented a survey describing and comparing different algorithms for static scheduling of workflows in parallel systems. Shi and Dongarra [6] present an algorithm for scheduling of workflows in heterogeneous clusters. These algorithms can be used in the context of Cloud computing if the machines are provisioned a priori. However, they cannot decide the optimal number of resources to reduce the cost of use of the infrastructure, and therefore they are not suitable for elastic Cloud environments using a pay-per-use economic model.

In the field of Grid computing, Yu *et al.* [7] and Hirales-Carbajal *et al.* [8] presented surveys comparing and describing algorithms for scheduling of workflows in Grid computing environments. These algorithms either assume a cooperation model free of financial cost or are based on an economic model that differs from the model adopted by Cloud providers. Because of that, the budget calculated by such algorithms would differ from the amount actually charged by Cloud providers, and thus they cannot be used in the context of Cloud computing.

Lin and Lu [9] developed an algorithm for scheduling of workflows in service-oriented environments. Unlike algorithms for Grid systems, it is able to utilize dynamic provisioning of resources. However, it misses the capability of considering the cost for utilization of resources required for its utilization in Cloud environments.

Recent research focused on algorithms that are aware of the intricacies of Cloud environments when using them to schedule workflow applications. Reynolds *et al.* [10] proposed the utilization of Clouds to complement desktop Grid resources. However, Cloud resources are deployed with the goal of replicating slow tasks to increase the chance of an early completion of the workflow. The proposed method is not optimized either for budget or for execution time; instead, it operates in a best-effort basis when late tasks are detected. Xu *et al.* [11] and Mao and Humphrey [12] proposed algorithms for scheduling multiple workflows in Clouds. Rahman *et al.* [13] proposed an algorithm for hybrid Clouds, where at least part of the resources can be used without cost and with higher level of control over their performance. Different from this approach, we focus on the scheduling of single workflows in pure Cloud environments.

Byun *et al.* [14] proposed the Partitioned Balanced Time Scheduling (PBTS) algorithm for cost-optimized and deadline-constrained execution of workflow applications on Clouds. The PBTS algorithm considers only one type of Cloud resource, chosen a priori, for its provisioning and scheduling decision. Abrishami *et al.* [15] proposed two algorithms for cost-optimized, deadline-constrained execution of workflows in Clouds. As explained in Section 5, these algorithms do not consider all data transfer times during provisioning and scheduling, increasing the execution budget. Our proposed algorithm is based on one of such algorithms (called IC-PCP), but also accounts for data transfer times and Cloud resources boot time during the provisioning and scheduling process. Furthermore, it explores possibility of tasks replication to increase the probability of meeting application deadlines.

The topic of replication of tasks has been extensively explored in the context of Grid systems [16], [17], [18], [19], [20] without addressing the issue of cost for resource utilization. Plankensteiner and Prodan [21] proposed an algorithm for scheduling of workflow applications on Grids with tasks replication to meet deadline constraints. Our algorithm is inspired by such method by considering particularities of Cloud infrastructures, such as cost and capacity of dynamic provisioning, when making replication decisions in Cloud environments.

3 Application and System Models

A scientific workflow application is modeled as a Direct Acyclic Graph (DAG) $G = (T, E_T)$, where T is the set of tasks that compose the workflow and E_T is the set of dependencies between tasks. Dependencies are in the form of edges $e_{i,j} = (t_i, t_j), t_i, t_j \in T$ that establish that a task t_j depends on data generated by t_i for its execution, and therefore t_j cannot start before execution of t_i completes and data generated by the latter is transferred to the location where t_j will execute. Task t_i is a *parent task* of t_j and t_j is a *child task* of t_i. Tasks without parents are called *entry tasks* and tasks without children are called *exit tasks*. For the correct operation of the proposed algorithm, we

1. http://www.cloudharmony.com/status

assume that a workflow can have only one entry task and one exit task. This can be achieved with the insertion of ''dummy'' tasks t_{entry} and t_{exit} that have execution time equals to 0. All the actual entry tasks are children of t_{entry} and all the actual exit tasks are parents of t_{exit}.

The sets of parents and children of a task t_j are given respectively by functions $parents(t_j)$ and $children(t_j)$. Each workflow G has a *soft deadline* $dl(G)$ associated to it. It determines the time to complete its execution, counted from the moment it is submitted to the *workflow scheduler*. The latter manages the execution of the workflow, makes decision on allocation of virtual machines, and schedules and dispatches tasks for execution in the Cloud.

A Cloud provider offers a set of n virtual machine (VM) types denoted by $\overrightarrow{VM} = vm_1, vm_2, \ldots, vm_n$. Each VM type offers different amount of resources, and incurs a different cost per use. Let $\overrightarrow{C} = c_1, c_2, \ldots, c_n$ be the *cost* vector associated with the use of each VM. VMs are charged per integer amount of time units, and partial utilization of a time period incurs charge for the whole period. Therefore, if the time period is one hour, utilization of a VM per 61 minutes incurs in the payment of two hours (two time periods). There is no limit imposed on the number of VMs of each type that can be running in any moment for execution of the workflow.

Runtime of each task is defined in the *runtime matrix* R. An element r_{jk} of R specifies the estimated runtime of task t_j in a VM of type vm_k. The *minimum runtime* $R_{min}(t_i)$ of a task t_i is the smallest runtime for such a task in the matrix R. Notice that $r_{entryk} = r_{exitk} = 0$ for all k. Tasks cannot be preempted or checkpointed. Therefore, if the execution of a task fails or if a task is canceled by the scheduler, it has to be restarted.

Each edge $e_{i,j}$ of G has an associated *data transfer time* $D(i, j)$. This is the amount of time required to transfer the data required by the non-entry and non-exit task t_j from the VM where t_i is running to the VM where t_j is running. Notice that, if both t_i and t_j are running on the same VM, $D(i, j) = 0$. The existence of data transfer time among different VMs implies that, for each task t_j to be executed in a given VM vm_k, vm_k is deployed before the data transfer from parents of t_j start, and is decommissioned after all the data transfers to its children are completed.

Important parameters of tasks are the *early start time* (est) and *latest finish time* (lft). The former represents the earliest time a task is able to start, which happens when all its parent tasks finish as early as possible and the latter represents the latest time a task can finish without missing the deadline, which happens when all the children of a task are executed as late as possible. Formally, est and lft are defined as:

$$est(t_j) = \begin{cases} 0, & \text{if } t_j = t_{entry} \\ \max\limits_{t_a \in parents(t_j)} \big(est(t_a) + R_{min}(t_a) + D(e_{a,j})\big), & \text{otherwise} \end{cases}$$

$$Ift(t_j) = \begin{cases} dl(G), & \text{if } t_j = t_{exit} \\ \max\limits_{t_s \in children(t_j)} \big(lft(t_s) - R_{min}(t_s) - D(e_{j,s})\big), & \text{otherwise.} \end{cases}$$

The *schedule time* $st(t_j)$ of a task t_j is the time on which the task has been scheduled for execution. This parameter is defined during the scheduling process, and can assume any value between $est(t_j)$ and $lft(t_j)$.

The problem addressed in this paper consists in the execution of a workflow G in the Cloud on or before $dl(G)$ (i.e., *deadline-constrained*) at the smaller possible cost (i.e., *cost-optimized*). Furthermore, because the workflows are subject to a soft deadline, a bigger budget can be invested for execution of G if it increases the likelihood of the deadline being met. The extra budget is expected to be proportional to the importance of the application to complete by its deadline.

For this problem to be solved, two subproblems have to be solved, namely *provisioning* and *scheduling*. The *provisioning* problem consists in the determination of the optimal number and type of VMs that can complete the workflow within its deadline. The *scheduling* problem consists in the determination of the placement and order of execution of the different tasks that compose the workflow in the VMs selected during the provisioning stage. The provisioning and scheduling problems are interconnected, as a different decision in types and number of machines may result in a different scheduling of tasks.

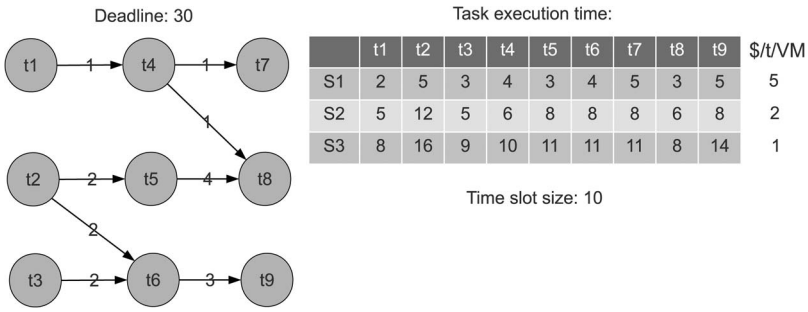
We assume that the workflow application executes in a single Cloud data center. Since more predictable execution and data transfer times are paramount for meeting application deadlines, keeping the workflow in a single data center eliminates one possible source of execution delay. It also eliminates the cost incurred by data transfer among data centers. We also ignore overheads incurred by the workflow management system. This is because they are strongly dependent on the particular technology for workflow management in use, varying from constant time [22] (which could be modeled as additional execution time of each task) to cyclical regarding the number of tasks managed [23].

4 Motivation and Example

The system and application model described in the previous section captures the basic features of typical Cloud computing environments, but they make too optimistic assumptions about the performance exhibited by the underlying Cloud infrastructure. Utilization of the runtime matrix R in the model implies a known, stable execution time for each task that composes the workflow. Similarly, the data transfer time function D assumes a stable data transfer time between VMs.

However, Cloud environments do not present regular performance in terms of execution and data transfer times. Jackson *et al.* [4] report performance variation of up to 30 percent in execution time and up to 65 percent in data transfer time when High Performance Computing (HPC) applications are executed in public Clouds. This demands countermeasures to be applied at the application provisioning and scheduling stage to enable soft deadlines to be met. Therefore, we propose the application of task replication as a means to mitigate the effect of performance variation of Cloud resources in the workflow execution time.

As a motivational example, consider the workflow and Cloud VM model proposed by Abrishami *et al.* [15] depicted in Fig. 1. The figure contains the description of



	t1	t2	t3	t4	t5	t6	t7	t8	t9	$/t/VM
S1	2	5	3	4	3	4	5	3	5	5
S2	5	12	5	6	8	8	8	6	8	2
S3	8	16	9	10	11	11	11	8	14	1

Fig. 1. Example of workflow and available VMs [15]. Numbers in arcs denote data transfer time, whereas the table presents execution times of tasks on three different VM types S1, S2, and S3, as well as the cost per time unit of each VM. Workflow's deadline is 30 time units and allocation slot is 10 time units.

tasks, data transfer time between tasks if running in different VMs (depicted in the arcs), and execution time of tasks in three different VM types (labeled S1, S2, and S3). The deadline for execution of such workflow is 30 time units and the allocation interval is 10 time units. On these settings, the IC-PCP algorithm, which is the state-of-the-art algorithm for provisioning and scheduling of workflows in Clouds, generates the schedule presented in Fig. 2a [15]. Hatched areas denote idle time slots of provisioned virtual machines, whereas the arrow in the figure denotes data dependency between VMs that requires both VMs to be simultaneously available to enable the data transfer.

The schedule represented in Fig. 2a assumes that Cloud resources are able to sustain their ideal performance for the duration of the workflow execution, and therefore the execution is able to complete by its deadline. Without extra cost, the schedule depicted in Fig. 2b is able to meet the application deadline even if either T4 or its replica T4' is delayed because of poor Cloud resource performance. Finally, Fig. 2c presents a schedule that tolerates delays of three tasks or two virtual machines at the cost of two extra VM time unit allocations. The extra time enables replication of one long duration task, what further increases likelihood of completion of the workflow before its deadline.

The EIPR algorithm also enables the provisioning of more VMs than the required for the execution of the workflow within the deadline to further increase the likelihood of deadline meeting. This is achieved with the enhancement of the model described in the previous section with the maximum number of replicas allowed for a single task and the replication budget $rb(G)$, which defines the amount, in relation to the estimated cost of executing the workflow determined during the original provisioning and scheduling, that can be used to provision extra VMs to enable further tasks replication. Notice that, if $rb(G) = 0$, only opportunistic replication caused by idle time slots is applied, and therefore the algorithm operates like existing cost minimization approaches [14] with the advantage of opportunistic tasks replication.

The EIPR algorithm addresses performance variation in public Clouds, not fault tolerance. This is because public Cloud resources offer high availability and reliability. In fact, most public Cloud providers offer guarantees only in terms of *availability* of resources, without any guarantees in terms of *performance* of such resources. Although task replication may also help in avoiding delays in execution caused by resource failures, this topic is not investigated in this paper.

5 The EIPR Algorithm

The goal of the proposed Enhanced IC-PCP with Replication (EIPR) algorithm is increasing the likelihood of completing the execution of a scientific workflow application within a user-defined deadline in a public Cloud environment, which typically offers high availability but significant performance variation, with the use of task replication. In a high level, the proposed algorithm performs three distinct steps:

Step 1. Combined provisioned of Cloud resources and task scheduling (Section 5.1);
Step 2. Data transfer-aware provisioning adjust (Section 5.2); and
Step 3. Task replication (Section 5.3).

These steps are detailed on the next sections. The full listing of the EIPR algorithm can be found in Section 1 of

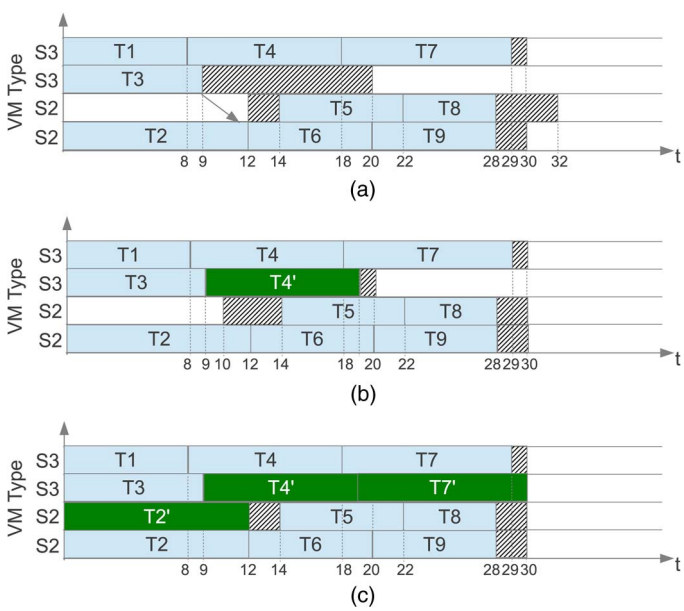


Fig. 2. Schedule of the workflow from Fig. 1 with different task replication strategies. (a) Original scheduling enabled by the IC-PCP algorithm [15]. (b) Utilization of an available idle slot for replication of T4 (no extra cost incurred). (c) Allocation of VMs for two extra time units for replication of T2 and T7.

the supplementary material of this paper which is available in the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.238.

5.1 Combined Provisioning and Scheduling

The first step of the EIPR algorithm consists in the determination of the number and type of VMs to be used for workflow execution as well as start and finish time of each VM (provisioning) and the determination of ordering and placement of tasks on such allocated resources (scheduling).

The provisioning and scheduling problems are closely related, because the availability of VMs affects the scheduling, and the scheduling affects finish time of virtual VMs. Therefore, a more efficient scheduling and provisioning can be achieved if both problems are solved as one rather than independently.

Among the existing approaches for combined provisioned and scheduling of workflow applications in public Cloud environments, the IC-PCP (IaaS Cloud Partial Critical Path) algorithm [15] works with the closest assumptions to the system and application models described in Section 3. It is a cost-minimizer with deadline constraint algorithm that operates via assignment of all the tasks of a *partial critical path* (PCP) of the workflow to the same virtual machine. The PCP is initially composed of one of the parents of t_{exit} or, if it was already scheduled, the task t_o with the latest lft that has not been assigned to a PCP. The next element to be part of the PCP is the parent t_p of such task with latest $est(t_p) + R_{\min}(t_p) + D(e_{p,o})$, i.e., the parent with longer execution and data transfer time. The procedure for definition of a PCP is detailed in Algorithm 1. Algorithm 1 is initially executed to t_{exit} and then recursively for each task of the resulting PCP, starting from the first. The process is repeated for each parent of t_{exit}. The whole PCP is assigned to a machine, as detailed below, and est, lft, and st of all tasks affected by the scheduling of the PCP are updated.

Algorithm 1 Assignment of the PCP of a task t [15].

```
1: Data: U: Unscheduled tasks of G.
2: pcp ← ∅;
3: while U ⊇ parents(t) do
4:   for each t_p ∈ parents(t) do
5:     if t_p ⊂ U then
6:       readyTime ← −1;
7:       if est(t_p) + R_min(t_p) + D(t_p, t) > readytime then
8:         readyTime ← est(t_p) + R_min(t_p) + D(t_p, t);
9:         parent ← t_p;
10:      end if
11:    end if
12:  end for
13:  pcp → parent ∪ pcp;
14:  Remove parent from U;
15:  t ← parent;
16: end while
17: return pcp;
```

For assigning partial critical paths to VMs, we adopt the method from the IC-PCP algorithm, which examines the available VMs starting from the cheapest (Algorithm 3, in the supplementary material available online, Lines 2-30) to the more expensive to find one able to execute each task of the path before its lft. The first VM able to meet such requirement is selected. If none of the existing machines can meet this requirement, a new VM is provisioned (Line 32). The type of this VM is the cheapest one that can execute each task of the critical path before the task's lft.

When evaluating the suitability of a VM for receiving a partial critical path, EIPR inserts the new path in the beginning of the scheduling, if it does not violate lft nor dependencies of previously assigned tasks (Lines 5-11). Otherwise, it schedules the path at the end of the schedule (Lines 13-19). When evaluating each case, the case is considered invalid if any of the following conditions occur: 1) at least one scheduled task has its lft violated if it is preempted; 2) at least one task of the path has its lft violated if the path is scheduled in the considered position; and 3) it will require the VM to be executed for an extra time slot.

The IC-PCP algorithm disregards deployment and boot time of virtual machines, by assuming that the earliest start time of the entry task is 0. However, virtual machines provisioned from public Cloud providers are not immediately available for task execution; VMs need to be properly initialized and this time is not negligible. To better model effects of such non-negligible deployment and boot times of virtual machines in the workflow scheduling process, the EIPR algorithm assigns the average boot time of virtual machines, rather than 0, to $est(t_{entry})$, and $st(t_{entry})$ before calculating est and lft of each task.

5.2 Data-Transfer Aware Provisioning Adjust

The combined provisioning and scheduling detailed in the previous section does not dictate the start and stop times of VMs. To determine both values, the algorithm has to consider not only start and end time of scheduled tasks, but also the data transfers to the first scheduled task and from the last scheduled task. If the first scheduled task in a VM is not an entry task, data from parent tasks have to be moved to the virtual machine before the task can run, and thus, the VM needs to be provisioned before the start time of its first task. This affects the start time of tasks and the total provisioning time of the VM, and may cause workflow execution delay and execution of VMs for an extra billing period.

In the second step of the EIPR algorithm, the provisioning decision performed in Step 1 is adjusted to account for the aforementioned factors. For each non-entry task scheduled as first task of a virtual machine, and for each non-exit task scheduled as the last task of a virtual machine, the algorithm meets the required communication time by setting the start time of the machine $D(i, j)$ earlier than st of the first task, and/or setting the end time of the machine $D(i, j)$ later than the finish time of the last task, depending on where the extra time is required. Finally, the beginning of the first allocation slot of each virtual machine is anticipated by the estimated deployment and

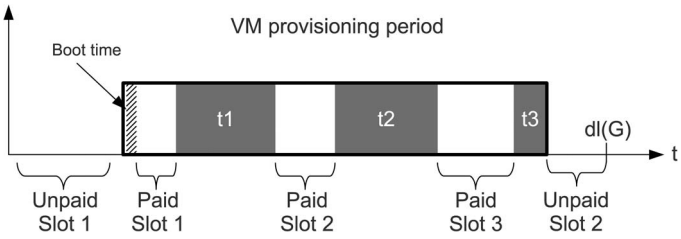


Fig. 3. Paid and unpaid idle time slots of provisioned VMs.

boot time for virtual machines, which was accounted for during the scheduling process as described in the previous section.

5.3 Task Replication

The aforementioned corrections enable virtual machines to be ready to receive data and tasks in the moment that they are required to meet times estimated during the scheduling process. However, it does not account for delays in the tasks execution caused by poor performance of public Cloud resources. EIPR tries to mitigate such effects with the utilization of task replication in idle slots of provisioned VMs or on new VMs allocated for enabling extra replication (if the replication budget allows). Notice that, because the goal of this replication is increasing performance rather than fault tolerance, *space replication* is the target of EIPR. Therefore, tasks are only replicated on different VMs, oppositely to a time replication approach where the same task could be scheduled multiple times in a single VM to increase fault-tolerance. The process is listed in Algorithm 5 of the supplementary material available online.

Idle slots exist in the scheduling because of two reasons. The first is dependencies between tasks, which may lead to periods where the next task scheduled to a virtual machine has to wait for data being generated during execution of another task in another machine. The other cause of idleness is adjusts applied by the EIPR algorithm to the provisioning times of VMs. As the corrected times may cause the allocation time to exceed the billing period, in some situations some billing periods will be only partially used, and the excess can be used for task replication purposes.

The task replication process works as a semi-active replication technique for fault tolerance [24], with the difference that here tasks are replicated across performance-independent hosts rather than failure-independent locations. As in the case of replication for fault tolerance, in our approach replication is also managed by a single entity. The process operates as follows. Initially, the algorithm generates new VMs based on the available replication budget (Lines 8-18).

For selection of the type of VM to be used for the new VM, the list of already provisioned VMs is sorted in descending order of number of scheduled tasks (Line 7). Starting from the beginning of the sorted list, the first VM type whose cost is smaller than the available budget is chosen (Line 9). The chosen VM is replicated (Line 13), provisioning information is updated (Line 14), the VM is moved to the end of the list (Line 16), the available budget is updated (Line 17), and the search is restarted. The process is repeated while the budget admits new instantiations. The new provisioned VM and the correspondent time slot reproduce the same start and end times than the original VM.

Next, a list of all possible idle slots is created. This list includes both *paid slots*, which are slots were the VM is actually provisioned and no task is scheduled on it; and *unpaid slots*, which are the times between the start time and the deadline were the VM is not provisioned. Fig. 3 depicts the difference between these two types of slots. Once paid and unpaid slots are identified, the slots are sorted in increasing order of size (Line 19). The first unpaid slot on the list is inserted after the last paid slot (Line 20), so paid time slots have precedence over unpaid in the slots selection.

The next step consists in defining an order for tentative replication of tasks in the available time slots (Line 21). Tasks are sorted in their ''replication precedence order'', which is based on three criteria, as follows:

1. **Ratio between execution time and lag time**. Task t_i has precedence over task t_j if $st(t_i) + r_{ij}/lt(t_i, v_i) < st(t_j) + r_{jj}/lt(t_j, v_j)$. The *lag time* $lt(t, v)$ of a task t scheduled to a VM v depends on the task's latest finish time, start time, and runtime:
$$lt(t, v) = lft(t) - st(t) + r_{tv};$$
2. **Execution time**. For tasks t_i and t_j respectively scheduled in VMs u and v and with the same ratio, t_i has precedence over t_j if $r_{iu} > r_{jv}$;
3. **Number of children**. For tasks t_i and t_j with the same ratio and execution times, t_i has precedence over t_j if $|children(t_i)| > |children(t_j)|$, where $|children(t)|$ represents the cardinality (number of elements) of the set of children of a task.

The EIPR algorithm prioritize tasks whose proportion between execution time and available time is bigger; then larger tasks (in terms of execution time), and finally tasks that have many children (and thus its delay will cause delays of a bigger number of other tasks).

Once the priority of time slots and priority of tasks is defined, the algorithm, for each time slot (Lines 22-39), iterates over the task lists (Lines 26-38), assigning the first task to fit in the slot (Lines 27-37). A task is considered to fit a slot if

1. it can complete its execution before the end of the time slot;
2. it can complete its execution before its latest finish time;
3. the maximum number of replicas allowed for a single task was not reached for this task;
4. a replica of the same task is not scheduled in this machine;
5. it does not violate tasks dependencies (i.e., in this slot, it does not run before an predecessor or after a successor).

If the task fits the slot, a replica is created (Line 28) and scheduled to start either at beginning of the slot or at its est

(whatever occurs later), and information about available slots is updated (Line 30), as the schedule of the replica itself can create a new, smaller, time if the task execution time is smaller than the slot size. The task is also moved to the end of the task list (Line 31), to reflect a lower priority for further replication, and the current number of replicas of the task is updated.

The process for unpaid slots is similar, but before considering utilization of the slot, the algorithm checks the available budget (Line 23). If there is budget for allocation of new slot, it is allocated and deemed paid. Provisioning information of the virtual machine is then updated to reflect the choice for allocation of a new time slot. After that, its utilization follows the same method applied for other paid slots (Lines 34-35).

At the end of this process, it is possible that some VMs provisioned at this stage for enabling extra replication did not have any replica assigned to them. In this case, such unused machines are removed from the list of VMs to be provisioned (Line 40). The result of this step is a provisioning and scheduling plan containing type, start time, and end time of virtual machines and start times for tasks and their replicas. This plan respects the deadline constraint and replication budget set for execution of the workflow.

6 Performance Evaluation

In this section, we describe the experiments we conducted to evaluate the EIPR algorithm. Experiments were conducted with the CloudSim toolkit [25]. The simulation testbed consists of a data center containing 500 hosts. Each host has 256 GB of RAM and 8 cores. The data center models Amazon AWS EC2 standard instance types, and the parameters relevant for the experiments are presented in Table 1. The billing period is 60 minutes.

Four workflow applications were used in these tests. They are Montage (generation of sky mosaics), CyberShake (earthquake risk characterization), LIGO (detection of gravitational waves), and SIPHT (bioinformatics).[2] These applications were characterized by Juve *et al.* [1]. Readers are referred to Fig. 5 of the supplementary material available online for a visual representation of the applications and to Juve *et al.* [1] for their full characterization. The execution time defined in such files, with a positive or negative variation between 0 and 10 percent uniformly applied to each task, is assumed to be achieved by VMs of type m1.medium.

TABLE 1
VM Types Used in the Experiments

Type	Memory (GB)	Core speed (ECU)	Cores	Cost ($)
m1.small	1.7	1	1	0.06
m1.medium	3.75	2	1	0.12
m1.large	7.5	2	2	0.24
m1.xlarge	15	2	4	0.48
m3.xlarge	15	3.25	4	0.50
m3.xxlarge	30	3.25	8	1.00

2. The XML files describing the applications are available via the Pegasus project: https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator

TABLE 2
Number of Tasks of Each Application for Each of the Three Configuration Sizes Evaluated

Application	Medium	Large	Very large
Montage	50	100	1000
CyberShake	50	100	1000
LIGO	50	100	1000
SIPHT	60	100	1000

Because users of IaaS services do not have access to the underlying physical infrastructures to be able to infer or model precisely the performance degradation of the platform, we used a performance degradation model based on reported observations of the phenomena in public Clouds. Reduction of performance was modeled after the analysis presented by Jackson *et al.* [4] regarding performance of HPC applications in public Clouds. Loss in performance observed in a VM for a specific scheduling period is sampled from a Normal distribution with average of 15 percent loss and standard deviation of 10 percent loss. Similarly, loss in each data transfer performance is modeled

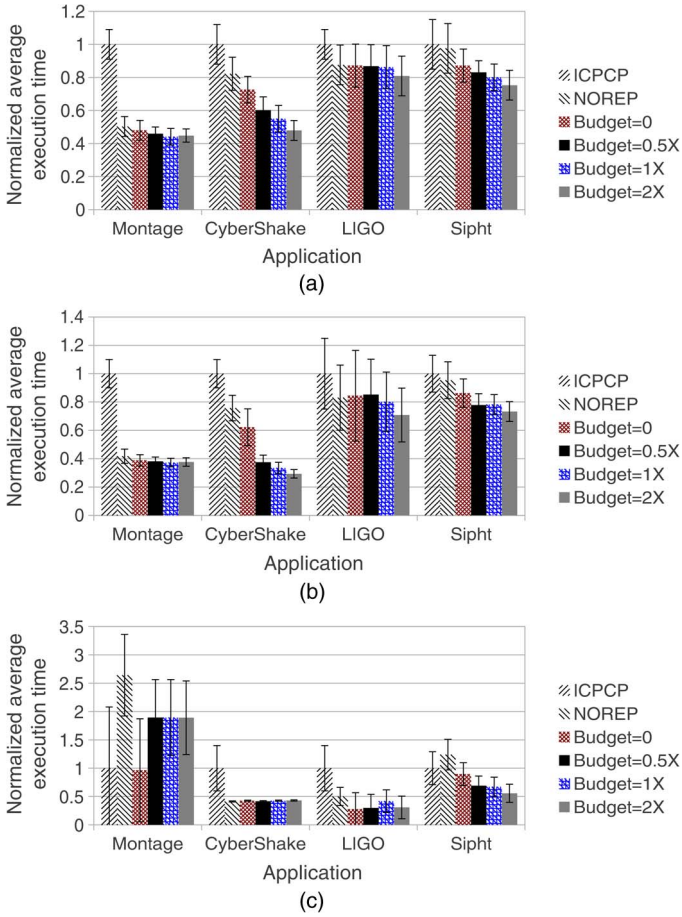


Fig. 4. Normalized average execution time of applications for different application sizes. NO REP stands for utilization of the EIPR policy without the replication stage. The budget represents the extra budget available for replication on the EIPR algorithm in relation to the amount spent before the replication. (a) Medium application size (as defined in Table 2). (b) Large application size. (c) Extra large application size.

TABLE 3
Average Cost (U$) of Workflows Execution. Standard Deviation for Each Case Is Presented in Parenthesis. NO REP Stands for Utilization of the EIPR Policy without the Replication Stage. The Budget on EIPR Represents the Extra Budget Available for Replication, in Relation to the Amount Spent before the Replication

Workflow		ICPCP	NO REP	EIPR			
				budget=0	budget=0.5X	budget=1X	budget=2X
Montage	Medium	0.06 (0)	1.44 (0.19)	1.44 (0.19)	2.14 (0.29)	2.88 (0.39)	4.33 (0.59)
	Large	0.06 (0)	3.25 (0.34)	3.25 (0.34)	4.87 (0.52)	6.51 (0.69)	9.77 (1.04)
	Very large	20.09 (18.81)	46.61 (9.2)	22.53 (19.92)	61.54 (11.3)	83.29 (17.59)	119.46 (15.73)
CyberShake	Medium	0.47 (0.46)	0.38 (0.06)	0.38 (0.06)	0.56 (0.08)	0.76 (0.12)	1.15 (0.18)
	Large	1.13 (0.14)	0.93 (0.1)	0.92 (0.16)	1.38 (0.15)	1.87 (0.21)	2.81 (0.32)
	Very large	46.22 (7.2)	58.86 (4.16)	58.86 (4.16)	88.26 (6.24)	117.72 (8.33)	176.59 (12.49)
LIGO	Medium	0.83 (0.13)	1.21 (0.61)	1.31 (0.75)	1.88 (1.09)	2.42 (1.31)	3.18 (1.71)
	Large	2.1 (0.24)	2.47 (0.45)	2.86 (0.88)	4.23 (0.81)	5.24 (0.98)	6.72 (1.24)
	Very large	41.26 (17.69)	29.09 (5.96)	23.05 (20.79)	39.54 (26.19)	63.63 (23.02)	67.48 (30.87)
SIPHT	Medium	0.76 (0.06)	0.92 (0.05)	1.09 (0.12)	1.44 (0.14)	1.98 (0.17)	2.77 (0.22)
	Large	1.22 (0.09)	1.38 (0.07)	1.55 (0.19)	2.14 (0.22)	2.85 (0.29)	3.99 (0.33)
	Very large	14.51 (0.96)	14.48 (0.76)	16.32 (1.56)	21.75 (1.85)	28.32 (2.27)	39.8 (3.3)

with a uniform distribution with average 30 percent and standard deviation 15 percent.

Each workload was evaluated with three different number of tasks (referred as *application size*). Table 2 details the number of tasks composing each application in each of the sizes: medium, large, and very large.

For each application and each size, a soft deadline was generated. The deadline was defined as follows (adapted from Abrishami *et al.* [15]). The ''base'' runtime is defined as the execution time obtained with a provisioning and scheduling policy that assigns each workflow task to a instance of the most powerful virtual machine (m3.xxlarge). Even though the execution time obtained with such a policy is not optimal, because data transfer times could be reduced by mapping tasks that require large data transfer time to the same VM, the schedule obtained with such a policy is still very efficient, as only data transfers limit the performance of the schedule. The base value obtained with such a policy is then multiplied by the number of tasks of the workflow, so we can scale the time proportionally to the number of tasks of the workflow. The deadline is then defined as 12.5 percent of the proportional base value for task runtime.

Each application with each size was subject to provisioning and scheduling via the IC-PCP algorithm (as a baseline for the experiment) and the EIPR algorithm, with the VM type offers defined in Table 1. The EIPR algorithm was evaluated with four different replication budget values: 0 (when only gaps between tasks can be utilized for replication purposes), 0.5, 1, and 2 times the base budget. We also executed the EIPR algorithm without performing the replication step (labeled *NO REP* in the result figures and tables) to evaluate in isolation the benefits of the task replication and the benefits for accounting for VM deployment and boot time and data transfer times for the output metrics.

The simulator used the provisioning and scheduling information to instantiate resources for the application and to execute it. The whole experiment was repeated 50 times and the averages of output metrics are reported in this section. For each of the 50 repetitions, the same random number generation seeds were used for each policy, what guarantees that the conditions (including performance losses) faced by each algorithm are exactly the same regarding the infrastructure behavior. The observed output metrics are execution time normalized with the correspondent value obtained with the IC-PCP algorithm and the average cost. The total number of violations, i.e., executions were the deadline was not meet, is also reported for each metric.

6.1 Results and Analysis

Fig. 4 presents the normalized execution time and Table 3 presents the costs incurred for each workflow application and for each application size. The average values observed for 50 executions are presented along with the standard deviation (in parenthesis). In most scenarios, EIPR drastically reduces the execution time of applications compared to IC-PCP. Results show reduction in up to 59 percent of execution time when compared to the execution time with the IC-PCP algorithm.

The case where EIPR underperformed is the Montage very large. However, even in this case EIPR met all the application deadlines. The reason for the underperformance on this application is the conservative scheduling policy of EIPR, which caused more VMs to be allocated. This in turn increased the need of data transfers, increasing the application execution time while still meeting the application deadline. As IC-PCP applies more aggressive co-scheduling (by potentially inserting a new PCP in between an already scheduled PCP), it was able to reuse more VMs and reduce the execution time. The more aggressive reuse of VMs resulted in smaller amount of data transfer, what it turn resulted in significantly smaller cost for execution of the workflow by IC-PCP compared to EIPR.

Fig. 4 also shows the effect that increased replication budget has on the applications execution time. Utilization of opportunistic replication (i.e, utilization of the available gaps in the allocated VMs, without deployment of extra VMs) introduces, in most cases, performance improvements in the application execution times. As expected, increased replication budget tends to further reduce execution time, although the amount of performance improvement is application-dependent. For example, the CyberShake large workflow experienced significant performance gains with budget increases.

TABLE 4
Total Number of Violations in the Application Deadlines after Execution of 50 Instances of Each Application. NO REP Stands for Utilization of the EIPR Policy without the Replication Stage. The Budget on EIPR Represents the Extra Budget Available for Replication, in Relation to the Amount Spent before the Replication

Workflow		ICPCP	NO REP	EIPR			
				budget=0	budget=0.5X	budget=1X	budget=2X
Montage	Medium	22	0	0	0	0	0
	Large	20	0	0	0	0	0
	Very large	0	0	0	0	0	0
CyberShake	Medium	26	6	0	0	0	0
	Large	3	0	0	0	0	0
	Very large	0	0	0	0	0	0
LIGO	Medium	50	3 1	31	30	28	21
	Large	4	2	6	3	0	0
	Very large	0	0	0	0	0	0
SIPHT	Medium	1	1	0	0	0	0
	Large	0	0	0	0	0	0
	Very large	0	0	0	0	0	0

Regarding cost-benefit of replication, all the applications experience a ''saturation point'' where extra replication budget does not introduce significant performance improvement. The budget point where it occurs varies depending on the application and on the size of the workflow. We can also notice that the estimated extra budget for replication may be different from the actual budget spent in the workflow execution. This happens because the estimated budget assumes a ''perfect scheduling'' where there will be no delays in the application execution. Because data transfer times and performance of Cloud resources vary, it is impossible to determine beforehand the exact execution time of the application (and consequently the total cost of workflow execution). Nevertheless, the estimated extra budget approximates the observed expenditure in most cases.

Table 4 presents the total number of violations in the deadline observed for 50 executions of each workflow. The table shows that EIPR either completely eliminates violations or significantly reduces them compared to IC-PCP. Although in some cases just the improved provisioning and scheduling already sufficed for meeting deadlines, other cases required some level of replication. The required level of replication required varied from opportunistic replication, as in the cases of CyberShake and SIPHT medium—where opportunistic replication eliminated completely deadline violations—to the case of LIGO, where 1X budget was required for the large workflow and 2X budget was required to enable the number of violations to be reduced to 21, whereas in the same case IC-PCP could not meet any of the deadlines.

From the above discussion, the following conclusions can be drawn from the experiments:

- Although rarely, replication can increase the execution time of applications (without compromising deadlines) when the structure of the workflow causes low utilization of provisioned VMs. In this case, replication can delay the tasks originally scheduled to the machine. If the delayed task is a data distribution task, its successors may be delayed;
- The structure of the workflow is more important than its class (i.e., CPU, memory, or I/O-intensive) for the effectiveness of EIPR. This is because memory and CPU requirements can be adjusted with the provisioning of more powerful machines. However, amount of data transfer and utilization of gaps in the scheduling, which were found to affect the performance of EIPR, are dependent on the particular organization of the workflow;
- Replication has also the additional benefit of reducing the variance of execution time caused by performance fluctuations. Even in cases where none of the policies cause violations, increase in replication budget increases the stability of the solution. Therefore, our solution is beneficial not only when applications need to complete before a deadline, but also in situations when more certainty about the execution time is required, such as when the provisioning process is assisted by predictive algorithms.

7 Conclusion and Future Work

Scientific workflows present a set of characteristics that make them suitable for execution in Cloud infrastructures, which offer on-demand scalability that allows resources to be increased and decreased to adapt to the demand of applications. However, public Clouds experience variance in actual performance delivered by resources. Thus, two resources with the same characteristics may have different performance in a given time, what results in variation in the execution time of tasks that may lead to delays in the workflow execution.

To reduce the impact of performance variation of public Cloud resources in the deadlines of workflows, we proposed a new algorithm, called EIPR, which takes into consideration the behavior of Cloud resources during the scheduling process and also applies replication of tasks to increase the chance of meeting application deadlines. Experiments using four well-known scientific workflow applications showed that the EIPR algorithm increases the chance of deadlines being met and reduces the total execution time of workflows as the budget available for replication increases.

As future work, we will increase the capabilities of the EIPR algorithm by enabling replication of tasks across multiple Clouds. Another point that can be further explored is new criteria for ranking candidate tasks for replication and also workflow structure-aware scheduling

of replicas, where the structure of the workflow application is considered not only during the selection of candidates for replication but also during the replica's scheduling. We will also investigate how the replication-based approach can be used when the provisioning and scheduling process is performed for multiple workflows whose requests arrive at different rates.

ACKNOWLEDGMENT

The authors thank A.V. Dastjerdi and the anonymous reviewers for their insights and comments for improvements of this paper. This work is supported by research grants from the Australian Research Council and the University of Melbourne.

REFERENCES

[1] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, ''Characterizing and Profiling Scientific Workflows,'' *Future Gener. Comput. Syst.*, vol. 29, no. 3, pp. 682-692, Mar. 2013.
[2] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, ''Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility,'' *Future Gener. Comput. Syst.*, vol. 25, no. 6, pp. 599-616, June 2009.
[3] R. Abbott and H. Garcia-Molina, ''Scheduling Real-Time Transactions,'' *ACM SIGMOD Rec.*, vol. 17, no. 1, pp. 71-81, Mar. 1988.
[4] K.R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H.J. Wasserman, and N.J. Wright, ''Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud,'' in *Proc. 2nd Int'l Conf. CloudCom*, 2010, pp. 159-168.
[5] Y.-K. Kwok and I. Ahmad, ''Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors,'' *ACM Comput. Surveys*, vol. 31, no. 4, pp. 406-471, Dec. 1999.
[6] Z. Shi and J.J. Dongarra, ''Scheduling Workflow Applications on Processors with Different Capabilities,'' *Future Gener. Comput. Syst.*, vol. 22, no. 6, pp. 665-675, May 2006.
[7] J. Yu, R. Buyya, and K. Ramamohanarao, ''Workflow Scheduling Algorithms for Grid Computing,'' in *Metaheuristics for Scheduling in Distributed Computing Environments*, F. Xhafa and A. Abraham, Eds. New York, NY, USA: Springer-Verlag, 2008.
[8] A. Hirales-Carbajal, A. Tchernykh, R. Yahyapour, J.L. González-García, T. Röblitz, and J.M. Ramírez-Alcaraz, ''Multiple Workflow Scheduling Strategies with User Run Time Estimates on a Grid,'' *J. Grid Comput.*, vol. 10, no. 2, pp. 325-346, June 2012.
[9] C. Lin and S. Lu, ''SCPOR: An Elastic Workflow Scheduling Algorithm for Services Computing,'' in *Proc. Int'l Conf. SOCA*, 2011, pp. 1-8.
[10] C.J. Reynolds, S. Winter, G.Z. Terstyanszky, T. Kiss, P. Greenwell, S. Acs, and P. Kacsuk, ''Scientific Workflow Makespan Reduction through Cloud Augmented Desktop Grids,'' in *Proc. 3rd Int'l Conf. CloudCom*, 2011, pp. 18-23.
[11] M. Xu, L. Cui, H. Wang, and Y. Bi, ''A Multiple QoS Constrained Scheduling Strategy of Multiple Workflows for Cloud Computing,'' in *Proc. Int'l Symp. ISPA*, 2009, pp. 629-634.
[12] M. Mao and M. Humphrey, ''Auto-Scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows,'' in *Proc. Int'l Conf. High Perform. Comput., Netw., Storage Anal. (SC)*, 2011, p. 49.
[13] M. Rahman, X. Li, and H. Palit, ''Hybrid Heuristic for Scheduling Data Analytics Workflow Applications in Hybrid Cloud Environment,'' in *Proc. IPDPSW*, 2011, pp. 966-974.
[14] E.-K. Byun, Y.-S. Kee, J.-S. Kim, and S. Maeng, ''Cost Optimized Provisioning of Elastic Resources for Application Workflows,'' *Future Gener. Comput. Syst.*, vol. 27, no. 8, pp. 1011-1026, Oct. 2011.
[15] S. Abrishami, M. Naghibzadeh, and D. Epema, ''Deadline-Constrained Workflow Scheduling Algorithms for IaaS Clouds,'' *Future Gener. Comput. Syst.*, vol. 29, no. 1, pp. 158-169, Jan. 2013.
[16] W. Cirne, F. Brasileiro, D. Paranhos, L.F.W. Góes, and W. Voorsluys, ''On the Efficacy, Efficiency and Emergent Behavior of Task Replication in Large Distributed Systems,'' *Parallel Comput.*, vol. 33, no. 3, pp. 213-234, Apr. 2007.
[17] G. Kandaswamy, A. Mandal, and D.A. Reed, ''Fault Tolerance and Recovery of Scientific Workflows on Computational Grids,'' in *Proc. 8th Int'l Symp. CCGrid*, 2008, pp. 777-782.
[18] R. Sirvent, R.M. Badia, and J. Labarta, ''Graph-Based Task Replication for Workflow Applications,'' in *Proc. 11th Int'l Conf. HPCC*, 2009, pp. 20-28.
[19] M. Dobber, R. van der Mei, and G. Koole, ''Dynamic Load Balancing and Job Replication in a Global-Scale Grid Environment: A Comparison,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 2, pp. 207-218, Feb. 2009.
[20] X. Tang, K. Li, G. Liao, and R. Li, ''List Scheduling with Duplication for Heterogeneous Computing Systems,'' *J. Parallel Distrib. Comput.*, vol. 70, no. 4, pp. 323-329, Apr. 2010.
[21] K. Plankensteiner and R. Prodan, ''Meeting Soft Deadlines in Scientific Workflows Using Resubmission Impact,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 5, pp. 890-901, May 2012.
[22] R. Prodan and T. Fahringer, ''Overhead Analysis of Scientific Workflows in Grid Environments,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 3, pp. 378-393, Mar. 2008.
[23] W. Chen and E. Deelman, ''WorkflowSim: A Toolkit for Simulating Scientific Workflows in Distributed Environments,'' in *Proc. 8th Int'l Conf. E-Science*, 2012, pp. 1-8.
[24] P. Chevochot and I. Puaut, ''Scheduling Fault-Tolerant Distributed Hard Real-Time Tasks Independently of the Replication Strategies,'' in *Proc. 6th Int'l Conf. RTCSA*, 1999, p. 356.
[25] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F.D. Rose, and R. Buyya, ''CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms,'' *Softw., Pract. Exper.*, vol. 41, no. 1, pp. 23-50, Jan. 2011.

Rodrigo N. Calheiros is a Post-Doctoral Research Fellow in the Cloud Computing and Distributed Systems Laboratory (CLOUDS Lab) in the Department of Computing Information Systems, University of Melbourne, Australia. His research interests include Cloud and Grid computing and simulation and emulation of distributed systems. He is a member of the IEEE.

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

Supplementary Material for "Meeting Deadlines of Scientific Workflows in Public Clouds with Tasks Replication"

Rodrigo N. Calheiros, *Member, IEEE* and Rajkumar Buyya, *Senior Member, IEEE*

1 The EIPR Algorithm

This section contains the full listing of the EIPR algorithm. Algorithm 2 contains the high-level logic of the algorithm, whereas Algorithms 3 to 5 contain the description of each stage of the algorithm.

Algorithm 2 High-level operation of the EIPR algorithm.

1: **Data:** G: DAG representing the application.
2: **Data:** $deadline(G)$ and $rb(G)$: Deadline and budgets for DAG execution.
3: **Data:** $\overrightarrow{VM}, \overrightarrow{C}$: List of VM types and their costs.
4: **Data:** R: Matrix containing execution times of tasks on different VM types.
5: Add t_{entry} and t_{exit} to G;
6: $est(t_{entry}) \leftarrow$ VM boot time;
7: $st(t_{entry}) \leftarrow$ VM boot time;
8: $lft(t_{exit}) \leftarrow deadline(G)$;
9: Calculate $est(t_i), lft(t_i)$ of each task $t_i \in G$;
10: ▷ `Step 1: Combined provisioned of Cloud resources and task scheduling`
11: $pcp \leftarrow$ result of Algorithm 3 on t_{exit};
12: Apply the combined provisioning and scheduling (Algorithm 3) on pcp;
13: ▷ `Step 2: Provisioning adjust (based on [1])`
14: Apply the data transfer-aware provisioning adjust Algortihm 4);
15: ▷ `Step 3: Task replication`
16: Apply the task replication strategy (Algorithm 5);

Algorithm 3 Combined provisioning and scheduling in the EIPR algorithm.

1: **Data**: pcp: A PCP, as returned by Algorithm 1.
2: **for each** provisioned VM sorted by ascending order of cost **do**
3: valid ← true;
4: ▷ `Try to put` pcp `at the beginning of the schedule`
5: **if** Scheduling pcp before previously assigned PCPs violates any $lft(t)$ **then**
6: valid ← false;
7: **end if**
8: **if** Scheduling pcp before previously assigned PCPs violates execution order **then**
9: valid ← false;
10: **end if**
11: **if** valid=false **then**
12: ▷ `Try to put` pcp `at the end of the schedule`
13: **if** Scheduling pcp after previously assigned PCPs violates any $lft(t)$ **then**
14: valid ← false;
15: **end if**
16: **if** Scheduling pcp after previously assigned PCPs violates execution order **then**
17: valid ← false;
18: **end if**
19: **end if**
20: **if** valid=true **then**
21: ▷ `A potential schedule was found`
22: **if** Scheduling of PCP requires allocation of extra time slots of the VM **then**
23: valid ← false;
24: **end if**
25: **end if**
26: **if** valid=true **then**
27: Assign the PCP in the chosen position in the VM;
28: Break;
29: **end if**
30: **end for**

if PCP is not scheduled **then**
  Instantiate the cheapest VM able to execute each task t of pcp before $lft(t)$;
  Schedule the PCP in the created instance;
end if
for each task t from pcp **do**
  Update est and lft of tasks;
  Apply Algorithm 1 on t;
  Recursively apply this algorithm on t;
end for

Algorithm 4 Provisioning adjusts in the EIPR algorithm.

Data: VM: List of provisioned VMs.
for each virtual machine vm in VM **do**
  $t_{start} \leftarrow$ first task scheduled to vm;
  $t_{last} \leftarrow$ last task scheduled to vm;
  vm.startTime $\leftarrow st(t_{start})$;
  transferTime $\leftarrow 0$;
  for each parent t_p of t_{start} **do**
    if $D(e_{p,start}) >$ transferTime **then**
      transferTime $\leftarrow D(e_{p,start})$;
    end if
  end for
  vm.startTime $\leftarrow$ vm.startTime $-$ transferTime $-$ bootTime;
  for each child t_c of t_{end} **do**
    if $D(e_{end,c}) >$ transferTime **then**
      transferTime $\leftarrow D(e_{end,c})$;
    end if
  end for
  vm.endTime $\leftarrow$ vm.endTime + transferTime;
end for

Algorithm 5 Task replication in the EIPR algorithm.

rb: Replication budget.
VM: List of provisioned VMs.
T: Set of tasks from G.
P: List of paid idle slots.
U: List of unpaid idle slots.
$extraVms \leftarrow \lfloor |VM| \times vmRate \rfloor$;
V $\leftarrow$ sorted list containing VMs from VM in descending order of scheduled tasks;
while $rb > 0.00$ **do**
  $vm \leftarrow$ first VM from V whose cost is smaller than rb;
  if no VM was found **then**
    Break;
  end if
  Replicate vm;
  Add the new machine to VM;
  Add a slot representing the new machine in P;
  Move vm to the end of V;
  $rb \leftarrow$ cost of vm;
end while
Sort P and U in increasing order of size;
Add U at the end of P;
Sort tasks from T in ascending order of *replication precedence*;
for each slot s in P **do**
  if s is unpaid and there is no available budget **then**
    Break;
  end if
  for each task t in T **do**
    if t fits s **then**
      Create replica t' of t;
      Schedule t' in s, respecting $est(t)$;
      Update slots information;
      Move t to the end of T;
      if s is unpaid **then**
        Update budget information;
        Update paid status of slots deriving from s;
        Update provisioning information of VM that contains s;
      end if
    end if
  end for
end for
Remove from VM machines that are not executing any task or replica;

2 WORKFLOWS

Figure 5 contains the visual representation of the workflows used in the performance evaluation of the EIPR algorithm.

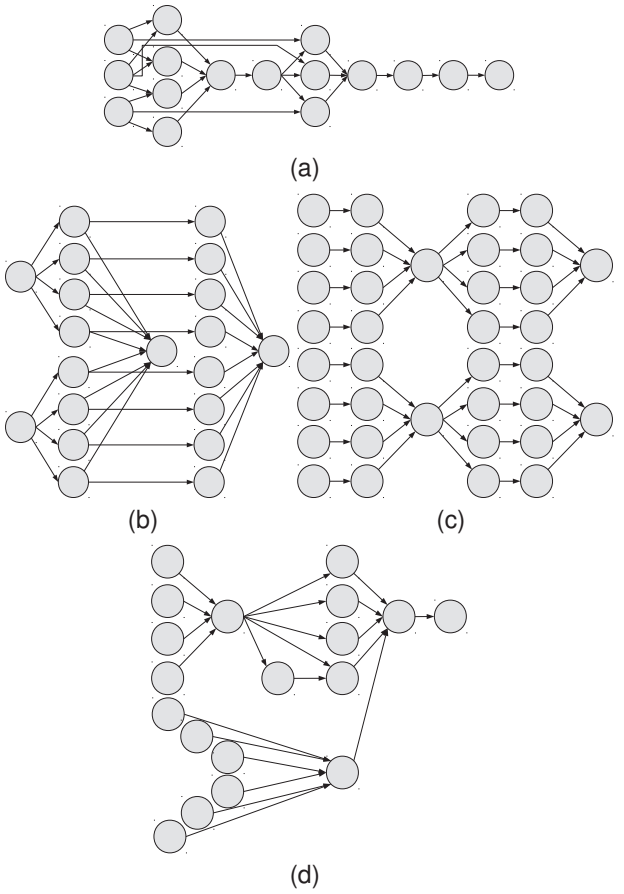


Fig. 5. Applications used for performance evaluation. Graphs represent execution dependency only. Full characterization of the applications, including input and output data analysis and typical execution times of tasks is available in Juve *et al.* [2]. (a) Montage. (b) CyberShake. (c) LIGO. (d) SIPHT.

REFERENCES

[1] S. Abrishami, M. Naghibzadeh, and D. Epema, "Deadline-constrained workflow scheduling algorithms for IaaS clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, Jan. 2013.

[2] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *Future Generation Computer Systems*, vol. 29, no. 3, pp. 682–692, Mar. 2013.