Noname manuscript No.
(will be inserted by the editor)

Autoscaling Scientific Workflows on the Cloud by Combining On-demand and Spot Instances

David A. Monge · Yisel Garí · Cristian Mateos · Carlos García Garino

the date of receipt and acceptance should be inserted later

Abstract Autoscaling strategies achieve efficient and cheap executions of scientific workflows running in the cloud by determining appropriate type and amount of virtual machine instances to use while scheduling the tasks/data . Current strategies only consider *on-demand* instances ignoring the advantages of a mixed cloud infrastructure comprising also *spot* instances. Although the latter type of instances are subject to failures and therefore provide an unreliable infrastructure, they potentially offer significant cost and time improvements if used wisely. This paper discusses a novel autoscaling strategy with two features. First, it combines both types of instances to acquire a better cost-performance balance in the infrastructure. And second, it uses heuristic scheduling to deal with the unreliability of spot instances. Simulated experiments based on 4 scientific workflows showed substantial makespan and cost reductions of our strategy when compared with a reference strategy from the state of the art entitled Scaling First. These promising results represent a step towards new and better strategies for workflow autoscaling in the cloud.

Keywords scientific workflows · cloud computing · autoscaling · scheduling · spot instances

D.A. Monge
ITIC-CONICET & Facultad de Ciencias Exactas y Naturales, Universidad Nacional de Cuyo (UNCuyo), Mendoza, Argentina
E-mail: dmonge@uncu.edu.ar
Tel.: +542614291000

Y. Garí
ITIC-CONICET, Universidad Nacional de Cuyo (UNCuyo), Mendoza, Argentina

C. Mateos
ISISTAN-CONICET, UNICEN, Tandil, Buenos Aires, Argentina

C. García Garino
ITIC & Facultad de Ingeniería, Universidad Nacional de Cuyo (UNCuyo), Mendoza, Argentina

1 Introduction

With the advent of cloud, scientific computing has been empowered with an advantageously paradigm for in-silico research. Cloud computing offers transparent and on-demand access to computational and storage resources that fulfill the great demands of such scientific applications.

Cloud computing enables the acquisition of computing infrastructures on the fly with the help of virtualization technologies [6]. Several types of Virtual Machine (VM) *instances* offer a wide spectrum of hardware and software configurations to the under a pay-per-use scheme. Typically, the prices differ according to the type of instance acquired, but prices may also vary according to their pricing model.

There are at least two different pricing models under which the instances can be acquired. On the one hand, *on-demand* instances can be purchased for a fixed price typically charged by hour of use. This is the most common pricing model across public cloud providers. On the other hand, there are also the so-called *spot* instances whose prices fluctuate dynamically decreasing during low demand periods. Cloud providers use this strategy to increase the utilization of their resources and thus their profits. Amazon was the pioneer on the provisioning of spot instances [2].

Spot instances are generally much cheaper than their on-demand counterpart. In some cases, spot prices show reductions of more than a 50% with respect to the

on-demand prices [2]. Although at first sight spot instances seem very attractive for maximizing the cost-performance of an infrastructure, they involve a compromise between cost and reliability and therefore they must be used wisely. In the spot-instances model the user must bid the maximum price that he/she is willing to pay for each instance. If the spot price overcomes such bid, an out-of-bid error occurs and the instances are terminated, abruptly interrupting the tasks that may be running in these instances.

Many efforts have been proposed to take advantage of cloud-infrastructure services in the domain of scientific workflows [29,23]. Having in mind that scientific workflows usually comprise hundreds or thousands of tasks, and that a wide spectrum of VM-instance types and pricing models are available, determining beforehand the right amount and type of necessary instances for an application is very difficult. First, because the unbalance of task durations and the existing dependencies generate *variable computation workloads* along the application's execution. Second, because of the characteristic performance variability of the cloud [24,10] it may be difficult to *accurately predict the duration* of tasks.

Autoscaling strategies [16,17] have to deal with the dynamic scaling of the infrastructure according to the application needs (i.e. determining the number and type of instances) and the on-line scheduling of such tasks on the running infrastructure. These are two interdependent problems that must be solved simultaneously. Even more, this scheduling problem in which workflow tasks have to be mapped onto a given set of resources (VM instances) is NP-hard, whereby, autoscaling strategies apply heuristics to efficiently find the scaling and scheduling plans. Up to now, autoscaling strategies for workflow applications have focused on the management of multiple workflow applications subject to deadline or budget constraints. Although these strategies represent important advances in the area, none of them take into consideration the advantages of using spot instances.

Motivated by this gap, we present a novel strategy for the autoscaling of scientific workflows aiming for makespan minimization subject to budget constraints, which builds on previous work of our own [18]. Our strategy is called Spot Instances Aware Autoscaling (SIAA). The novelty of SIAA and therefore a major contribution relies on that it is the first strategy addressing the problem of autoscaling of scientific workflows considering a combination of on-demand and spot instances. Implications of using such unreliable *spot* instances in the workflow execution process are profound because the unwise use of them may lead to serious degradations in performance, which is the basis for our second contribution. SIAA implements a new heuristic scheduling algorithm that prioritizes the execution of tasks on their fastest instances to improve workflow makespan. The heuristic also prioritizes the execution of *critical* tasks on reliable on-demand instances to mitigate the negative effects of out-of-bid-errors that might happen from the use of spot instances.

We evaluate the performance of our autoscaling strategy on four widely used benchmark workflow applications. To such end, we analyzed the influence of two bid-price prediction methods considering real data of spot prices. On the other hand, we now evaluate the (separate and joint) contributions based on time, cost and number of instances derived from the use of spot instances and the heuristic scheduling algorithm. Such an analysis was missing in our previous work [18]. In addition, we provide an analysis of the statistical significance of our results and therefore the strength of our claims.

The remainder of this paper is organized as follows. Next section formalizes the problem of scientific workflow autoscaling and presents current efforts regarding cloud-based autoscaling strategies for workflow applications. Section 3 examines the autoscaling strategy used as a baseline for comparisons in our experiments. Section 4 explains the details of our strategy. Then, Section 5 presents the experiments carried out over 4 scientific workflow applications and discusses the results obtained. Section 6 presents the most relevant related work and highlight the differences with our strategy. Finally, Section 7 concludes this work and provides future research directions.

2 Workflow Autoscaling

This section discusses the main concepts behind the problem of scientific workflow autoscaling in clouds, the underlying assumptions, and the existing strategies to tackle the problem.

2.1 Problem Definition

In this paper we address the problem of executing *standalone scientific workflows* in public clouds. The objective of autoscaling strategies discussed in this paper is the minimization of workflow makespan subject to budget constraints.

Workflow applications comprise a set of reusable software components denominated *tasks*. We focus on Directed Acyclic Graph (DAG)-like workflows in which

tasks are represented by nodes, and dependencies between them are represented by edges in the graph. We assume that there is a central storage for data and that transfer operations are carried out as part of the execution of tasks. In other words, dependencies in the graph represent control-flows.

While executing a workflow on the cloud, the *performance* of tasks differs according to the *type* of Virtual Machine (VM) instances used. Tasks have different profiles according to their balance between CPU and I/O operations (both, network demands and data processing) which makes them suitable for some types of instances over other types. Different instance types also differ in their associated price, which we assume is charged by hour as in well–known cloud services like Amazon's Elastic Compute Cloud (EC2).

In practice, scientific workflows model experiments involving a large number of tasks, and several hours of computation and/or process large amounts of data. Autoscaling strategies aim to determine the proper amount and type of resources to acquire in order to fulfill the variable workload patterns of the applications. Autoscaling regards two interrelated problems: (*a*) the determination of the proper number and type of VM instances, and (*b*) the scheduling of the workflow tasks onto the available instances. Both sub-problems are NP-hard and therefore the solutions proposed to date are based on heuristics [16,17,18].

The fact that (*i*) the applications present variable workload patterns, that (*ii*) performance models of tasks are imperfect, and that (*iii*) cloud infrastructures present variable performance, demand that autoscaling strategies follow a *dynamic* approach [16,17]. Such dynamism permits the autoscalers to adapt the execution and mitigate the effect of such discrepancies between the information available (estimations) and the actual progression of the execution. In a broad sense, autoscaling strategies need to constantly monitor the status of the environment (applications plus infrastructure) and take the proper scaling and scheduling decisions during the entire execution of an application.

More formally, autoscaling strategies are executed periodically adjusting the number and type of VM instances used, at the same time that tasks are being scheduled. On each monitoring interval, the autoscaling strategy addresses a makespan minimization problem subject to a budget[1].

Given τ, the set of tasks in the workflow, and T^{vm}, the set of available VM types, such optimization problem is defined as:

$$\begin{aligned} &\min\{\text{makespan}(X^{\text{sca}}, X^{\text{sch}}, S)\} \\ &\text{s.t.}: \quad \text{cost}(X^{\text{sca}}) \leq B, \end{aligned} \tag{1}$$

where $X^{\text{sca}} = \{T^{\text{vm}} \rightarrow \mathbb{N}\}$ is a *scaling plan* that indicates the number of necessary instances of each type for the next hour of computation, $X^{\text{sch}} = \{\tau \rightarrow T^{\text{vm}}\}$ is a *scheduling plan* that maps each task $t \in \tau$ to a type of instance where it will execute, S represents the current status of the infrastructure and the application, and B is the budget for the next hour of computation.

In this way, autoscaling strategies adapt the number of required instances within the budget constraint and minimize workflow makespan by periodically solving the optimization problem presented on Eq. (1).

For each of the optimization problems solved, the objective function makespan$(X^{\text{sca}}, X^{\text{sch}}, S)$ depends on the current status of the application and the infrastructure as well as the estimations of start time and durations of tasks not yet executed. In other words, makespan is computed as:

$$\text{makespan} = \max_{t \in \tau}\{\text{EST}(t) + d_t\} - \min_{t \in \tau}\{\text{EST}(t)\}.$$

Task Durations The duration d_t of a task t can be estimated using some of the existent performance prediction mechanism [19,20]. Here, durations are estimated considering the *preferred instance type* for each task. The preferred instance type for a *waiting* task (i.e. a task waiting for execution) is such that provides the shortest execution time. That information is obtained from the scheduling plan, X^{sch}. In the case of a *running* task (i.e. a task that is executing on a VM instance), the preferred instance type is just the type of the instance where the task is currently executing according to the current status, S. For running tasks, the remaining execution time is estimated by subtracting the time that the task has been running.

Tasks Earliest Start Time The earliest start time is the minimum time at which a task can start its execution considering its predecessors in the associated workflow. The *earliest start time* (EST) of a waiting task t is computed as:

$$\text{EST}(t) = \max_{1 \leq k \leq p}\{\text{EST}(t_k) + d_k\},$$

where t is a waiting task, t_k is one of the p parent tasks of t and d_k is the estimated duration of t_k. For tasks which are *ready* to execute, the EST is set to the current time, i.e. the time at which the autoscaling problem is being solved.

[1] Please note that other optimization objectives are also valid such as the minimization of energy consumption, for example.

Workflow Finish Time The finish time of a workflow is the time at which the application completes its execution and it is computed as:

$$\mathrm{FT} = \max_{1 \leq k \leq n} \{\mathrm{EST}(t_k) + d_k\}, \quad (2)$$

where t_k is one of the n tasks in the application.

Budget The budget B for the execution of each application was fixed in concordance with the work of Mao and Humphrey [17]. To get a reasonable budget estimation instead of using arbitrary values, we computed the average quotient between the cost and makespan considering all instance types and on-demand prices. Formally, the budget B for an application is computed as:

$$B = \operatorname*{avg}_{i \in T^{\mathrm{vm}}} \left(\frac{\sum_{t \in \tau} d_{t,i} \cdot p_i}{\mathrm{makespan}_i} \right), \quad (3)$$

where T^{vm} is the list of virtual machine types, τ is the list of tasks in the workflow, $d_{t,i}$ is the estimated running of executing the task t on an instance of type i, p_i is the on-demand price of (one CPU) of an instance of type i for a period of time equal to one hour, and $\mathrm{makespan}_i$ is the makespan of the application assuming an unrestricted number of instances of type i [2].

2.2 Autoscaling Strategies

In recent years, several strategies for the autoscaling of workflow applications have been designed. Strategies can be focused on cost minimization [16] or makespan minimization [17]. Under the makespan minimization category, two algorithms entitled *Scheduling First* and *Scaling First* have been proposed.

Both strategies, Scheduling First and Scaling First, respond to the same scheme consisting of 4 stages comprising: (*i*) the collection of runtime information, (*ii*) running a particular autoscaling algorithm, (*iii*) the consolidation of instances, and (*iv*) idle-instances shutdown. Autoscaling is performed, following these 4 stages, in an hourly basis to mitigate the effects derived from the discrepancy between the performance models and the actual duration of tasks. This generic scheme is presented in Figure 1. The following sections are dedicated to the description of each of these stages.

[2] This makespan estimation is computed offline and should not be confused with the makespan value to optimize in Eq (1).

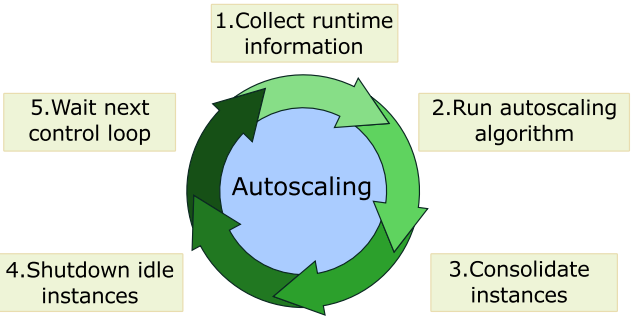


Fig. 1: Generic operational structure of autoscaling strategies.

2.2.1 Collect Runtime Information

On each control loop, the strategy updates information regarding the status of the running instances and information of the execution of workflow tasks. This information is crucial because it permits the adaptation of the strategy to the evolution of the application as it runs [16,17].

2.2.2 Run an Autoscaling Algorithm

The generic autoscaling strategy can be instantiated with a particular autoscaling algorithm that solves the optimization problem presented on Eq. (1).

The obtained scaling plan X^{sca} is used to acquire new instances taking into account those that might be already running due to previous executions of the autoscaling algorithm. The scheduling plan X^{sch} is used to enqueue each task t into the queue associated to the corresponding VM type. In each queue, tasks compete for the available instances. The submission of a given task to one of such instances for their execution is performed by a scheduling algorithm. Two commonplace autoscaling algorithms are briefly described next.

The *Scheduling-First* algorithm [17] first determines the fastest execution plan and then acquires the cloud instances. The details of this algorithm are not going to be further discussed nor will be included in the experimentation of this paper. The arguments supporting such decision are presented in the following section.

In contrast to the previous algorithm, *Scaling First* [17] begins by determining the number and type of instances within the budget constraint, and then schedules the tasks on the acquired resources to minimize the workflow makespan. We adopted this autoscaling algorithm as baseline-comparison in our experiments because in our previous study [18] such algorithm achieved much better makespan reductions than Scheduling First. In such study, we found that Scaling First outperformed

Scheduling First within the range of 43.8% to 85.6% in terms of speedup considering 4 benchmark workflows. We claim that these results are sufficient reason to exclude Scheduling First from this work. Section 3 is entirely dedicated to the analysis of Scaling First.

2.2.3 Consolidate Instances

As the Scheduling-First and Scaling-First algorithms schedule tasks to their fastest resources to minimize the workflow makespan, some periods of unused computing power may arise. In such cases, makespan can be effectively reduced by re-scheduling *ready* tasks (i.e. waiting tasks whose parents completed their execution) to unused instances. If the idle instance is faster than the originally assigned instance type, the task is re-scheduled directly. However, if the idle instance is slower than the originally assigned instance type, the task can be re-scheduled only if no task will be ready before this task terminates.

Instances Consolidation is a process complementary to scheduling for taking advantage of unused computation intervals of instances. When possible, such intervals are harnessed by reassigning tasks to execution queues in different instance types. Instances consolidation in this context should not be confused with VM consolidation, which is a common strategy in the cloud for the co-allocation of multiple instances on the same physical machine as a way to efficiently use the available resources [8].

2.2.4 Shutdown Idle Instances

For avoiding the use of unnecessary instances, and therefore reducing monetary costs, idle instances close to an hour of computation are terminated. Note that this stage is very important because it virtually scales down the infrastructure when the application workload decreases.

3 Scaling-First Autoscaling Strategy

In general terms Scaling First, as its name suggests, first determines the number and type of necessary cloud instances (scaling phase) and then schedules the workflow tasks (scheduling phase) [17].

3.1 Scaling Phase

The main objective of this phase is determining a scaling plan for the acquisition of instances. Then, the necessary amount of instances of each type are requested according to such plan. For a better comprehension of the scaling phase, the following paragraphs discuss an example adapted from [17]. Without loss of generality, we assume 8 tasks (belonging to the same workflow) that will be running during the following hour of computation. Three VM types are available: large (L) with price 0.5 USD/hour, medium (M) with price 0.3 USD/hour and small (S) with price 0.1 USD/hour. Figure 2 illustrates the process. We also assume that the available budget for the next hour is 1.4 USD.

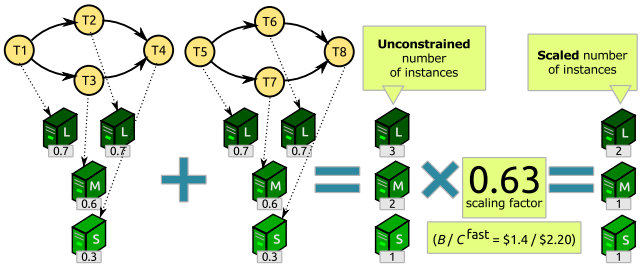


Fig. 2: Scaling phase for the Scaling-First autoscaling strategy. Labels below each instance represent the needed instance-hours. This image has been adapted from [17].

From the estimated duration of tasks on their preferred VM types, the algorithm computes the required instance hours. From all the involved tasks instance-hour requirements are summed up to determine the unconstrained scaling plan which comprises 3 type-L instances, 2 type-M instances and 1 type-S instance, with an associated cost $C^{\text{fast}} = 2.2$ USD. As we have a limited budget, the unconstrained amount of instances must be scaled to fulfill the budget constraint. Therefore the plan is scaled by the factor $r = B/C^{\text{fast}}$ giving a scaling plan comprising two type-L instances one instance for each of the remaining VM types.

The following paragraphs explain the process more formally. Algorithm 1 presents the pseudo-code of the scaling algorithm.

Algorithm 1 Scaling First: scaling algorithm.

```
1: procedure scaleInfrastructure:
2:     tasks ← getTasksInThePeriod()
3:     c ← estimateConsumption(tasks)
4:     c ← scale(c, B/C^fast)
5:     for all t in VMTypes do:                ▷ get instances
6:         requestInstances(c_t^od)
```

To determine the number and type of instances the algorithm estimates the consumption of instances for the next hour assuming that the tasks are executed

on their preferred instance type. This estimated consumption is represented by a vector $\mathbf{c} = [c_1, c_2, \ldots, c_m]$, where each component c_i is the number of instances of the i^{th} VM type to acquire. Each c_i value is estimated by adding the computation hour portions for all the tasks which prefer and instance of the i^{th} VM type. In the case of *running tasks*, the computation load is set to the type of the instance where such task is executing.

A consumption vector is used to compute the cost C^{fast} associated to the needed instances. However, to fit the scaling plan according to the budget B, this consumption vector $\mathbf{c}$ is scaled down by the factor $r = B/C^{\text{fast}}$ as: $\text{scale}(\mathbf{c}, r) = \begin{cases} \text{round}(\mathbf{c} \cdot r) & r < 1 \\ \text{round}(\mathbf{c}) & r \geq 1 \end{cases}$, where $\mathbf{c} \cdot r$ is the product of a vector and a scalar and the round$(\cdot)$ function is applied to each element of the resulting vector.

Instances are then acquired according to the re-scaled consumption vector. It is important to note that in this process only on-demand instances are considered.

3.2 Scheduling Phase

Once the necessary instances have been requested, the task-scheduling process starts. Figure 3 illustrates the scheduling phase process.

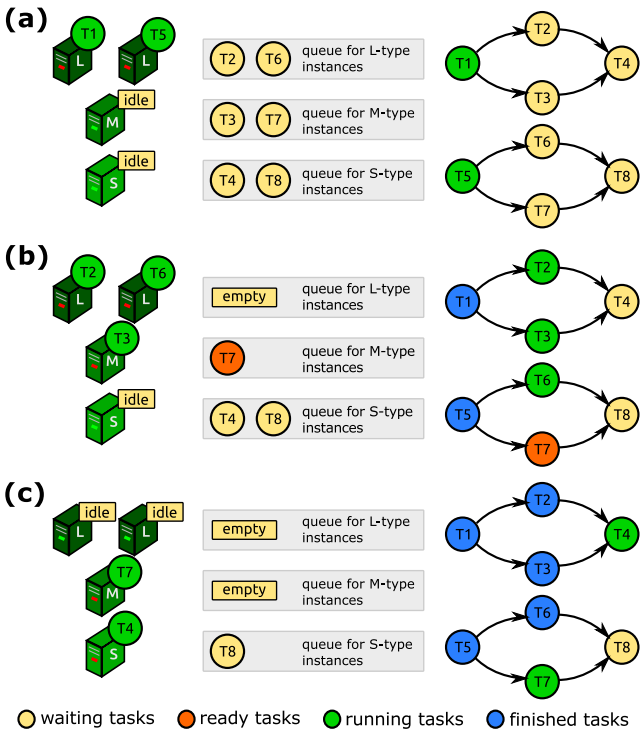


Fig. 3: Task scheduling in the Scaling-First autoscaling strategy.

Following with our example, each of the task which are ready for execution are inserted in the corresponding queue according to their preferred VM type. Ready tasks are taken from the queues according to their priority while there are available instances of such type. From Figure 3a we can see that tasks T1 and T5 (in green) are scheduled to 2 type-L instances, and all their subsequent tasks are waiting for execution in their corresponding queues.

Figure 3b shows the status just after the termination of T1 and T5 (in blue). Tasks T2 and T6 are submitted to the two type-L instances that have recently become idle from the termination of T1 and T5. Note that T3 has been extracted from their corresponding queue and is executing on the only type-M instance available. Although T7 is ready for execution (represented in orange) there are no type-M instances available for submission and therefore the task remains in the queue[3].

Figure 3c shows the status just after the termination of T2, T3 and T6 (now in blue) and that tasks T4 and T7 have been scheduled to their corresponding instances. In this stage, T8 is in the queue waiting for the termination of T7 (its parent task) and the termination of T4 that is running on the only available S-type instance.

The following paragraphs explain the process in detail. Algorithm 2 presents the pseudo-code of the scheduling algorithm implemented by Scaling First.

Algorithm 2 Scaling First: scheduling algorithm.

```
1: procedure scheduleTasks(instances):      ▷ input:lists of available on-demand instances
2:     readyTasks ← getReadyTasks() ▷ tasks ready to run
3:     queue ← sortByPriority(readyTasks)
4:     while notEmpty(instances) and notEmpty(queue) do:
5:         task ← queue.pop()
6:         instance ← preferred(task, instances_od)
7:         schedule(task, instance)    ▷ submit for execution
8:         if allCPUsBusy(instance) then:
9:             remove(instance, instances)
```

The algorithm starts by sorting the ready tasks according to priority (lines 2 and 3). Here, it is important to remark that different workflows might be executing in the cloud and tasks inherit the priority of the workflow they belong to. Therefore this prioritizing scheme focus on establishing a preference between tasks belonging to different workflows. As in this study we focus on

[3] Note that at this point there is an idle type-S instance, then it should be possible evaluate the consolidation of the task as explained in Section 2.2.3.

the management of a single workflow application at a time, the tasks selection procedure turns out to be one that chooses tasks considering the order established by the dependencies between them. Random selection of tasks is used to break ties between two o more competing tasks.

One by one, tasks are scheduled until there are no more ready tasks to schedule or all the instances are busy (line 4). Each selected task is voraciously scheduled on an idle virtual CPU of an available instance of their preferred type (lines 5 through 7). For future references within this paper, we call this algorithm the ≪greedy≫ scheduling algorithm. When a task terminates its execution, the virtual CPU (or processor) used is registered as free. Then, runtime information is updated and the scheduling algorithm is invoked with the intention of allocating more tasks to the recently released (idle) processor.

At this point, it is worth to point out that the scheduling algorithm is invoked in two contexts: (*i*) in an hourly basis, i.e. each time the autoscaling algorithm is executed, or (*ii*) each time a task terminates its execution.

4 SIAA: Autoscaling Using Spot Instances

Our Spot Instances Aware Autoscaling (SIAA) strategy differs from Scaling First in two main aspects. On the one hand, it uses a scaling algorithm that permits the acquisition of a cloud infrastructure comprising on-demand and spot instances. On the other hand, uses an heuristic scheduling algorithm to minimize the workflow makespan and reduce the probability that failing spot instances may interrupt the execution of critical tasks.

SIAA follows the same philosophy as the Scaling-First algorithm, i.e. it starts determining a scaling plan and then schedules the tasks for running on the available instances. Following sections describe both algorithms.

4.1 Scaling Algorithm

SIAA relies on the exploitation of a mixed infrastructure comprising on-demand and spot instances to attain an overall better cost-performance. As said in Section 1, on-demand instances provide a reliable computing platform suitable for the execution of critical tasks but at expenses of a higher cost than spot instances. Conversely, spot instances can be used to extend such infrastructure with unreliable instances offering better cost-efficiency, which are ideal for short duration or non-critical tasks.

A scaling plan in SIAA is defined as $X^{\text{sca}} = \{T^{\text{vm}} \times M \to \mathbb{N}\}$. The plan indicates the amount of instances to acquire for each combination of instance type T^{vm} and pricing model M (i.e. on-demand or spot).

The balance between both types of instances is governed by the *spots ratio* parameter ($\alpha \in [0, 1]$), which is used to determine how to split the original budget B and set the maximum prices to pay for on-demand and spot instances respectively. According to α, the budget for on-demand instances is computed as $B^{\text{od}} = (1 - \alpha) \cdot B$ and the budget for spot instances is computed as $B^{\text{s}} = \alpha \cdot B$. Algorithm 3 presents the process of infrastructure scaling.

Algorithm 3 SIAA: scaling algorithm.

```
1: procedure scaleInfrastructure(biddingStrategy):
                          ▷ step 1: determine on-demand instances
2:     tasks ← getTasksInThePeriod()
3:     c ← estimateConsumption(tasks)                ▷ unbound consumption
4:     c^od ← scale(c, B^od/C^fast)
                          ▷ step 2: determine spot instances
5:     c ← c − c^od                                  ▷ unbound consumption
6:     p ← biddingStrategy.predictBidPrices()
7:     c^s ← scale(c, B^s/C^s)
                          ▷ step 3: request instances
8:     for all t in VMTypes do:
9:         requestInstances(c_t^od)
10:        requestSpotInstances(c_t^s, p)
```

The scaling algorithm follows 3 steps:

1. determine on-demand instances (see Section 4.1.1),
2. determine spot instances (see Section 4.1.2), and
3. acquire instances (see Section 4.1.3).

4.1.1 Step 1: Determine on-demand instances

To generate the scaling plan, the algorithm starts by estimating the computation load for the next hour (lines 2 and 3) as Scaling First does. Then, the consumption vector for the required on-demand instances $\mathbf{c}^{\text{od}}$ is determined by fitting the number of instances according to the available budget B^{od}. The $\mathbf{c}^{\text{od}}$ vector is obtained scaling the unconstrained consumption vector $\mathbf{c}$ by the factor $B^{\text{od}}/C^{\text{fast}}$, where C^{fast} is the total cost of acquiring all the instances in $\mathbf{c}$ using the on-demand instance prices (line 4). Note that this first step in our algorithm is very similar to the scaling algorithm implemented in Scaling First, but the former fits the instances to a portion of the original budget.

4.1.2 Step 2: Determine spot instances

Analogously, the consumption vector of spot instances $\mathbf{c}^{\mathrm{s}}$ is determined by scaling the remaining unconstrained instances: $\mathbf{c} - \mathbf{c}^{\mathrm{od}}$ (line 5). To such end, a vector $\mathbf{p} = [p_1, p_2, \ldots, p_m]$ of bid prices is computed having one value for each instance type. The components p_i of such vector (bid prices) can be obtained using any *bid price prediction method* (line 6). Then the consumption vector $\mathbf{c}$ is scaled by the factor $B^{\mathrm{s}}/C^{\mathrm{s}}$ using the same criteria as before (line 7). Note that C^s is the total cost of acquiring all the instances in $\mathbf{c}^{\mathrm{s}}$ according to the vector of bid prices $\mathbf{p}$.

Returning to bid prediction of spot instances, many methods have been proposed [1,26,27]. This study relies on 2 bidding methods described on Table 1.

Table 1: Bidding methods.

Strategy	Bid value
Current	the current price
Probabilistic	a price over the current one that satisfies that the probability of failure is below ϕ, where $\phi \in \{0.1, 0.05, 0.01\}^{\dagger}$

† These probabilities are computed according to a history of prices that is previous to the current time.

It is important to note that the selection of a bidding method has a great influence on various aspects of SIAA's performance. On the one hand, methods that bid a *low price* increase the probability of task failures derived from out-of-bid errors, but increase the potential number of instances to acquire. On the other hand, methods that bid higher prices are less prone to failures but reduce the number of instances that SIAA can acquire within the budget constraints.

4.1.3 Step 3: Acquire instances

Both consumption vectors ($\mathbf{c}^{\mathrm{od}}$ and $\mathbf{c}^{\mathrm{s}}$) represent the scaling plan X^{sca} generated by SIAA. According to them, the algorithm acquires the necessary instances considering the number of instances already running (lines 10 to 12). For each VM type t, the algorithm requests the necessary amount of on-demand instances to reach the c_t^{od} quantity. Similarly manner, for each VM type t, a number of c_t^{s} spot instances is requested, considering not only the amount of instances already running but also the recently requested on-demand instances of type t.

4.2 Heuristic Task Scheduling

As part of the runtime information collection stage (update of task durations and earliest start time of tasks), SIAA computes the *late start time* and *slack times* to determine which are the critical tasks in the workflow. This information is crucial for an intelligent scheduling of tasks.

The *late start time* (LST) is the maximum start time at which a task can start without delaying any of its successor tasks. The late start time of a task t is computed as:

$$\mathrm{LST}(t) = \min_{1 \le k \le c} \{\mathrm{LST}(t_k) - d\},$$

where t_k is one of the c child tasks of t and d is the estimated duration of t. For each of the n *exit* tasks the LST is computed as $\mathrm{LST}(t) = \mathrm{FT} - d_t$, where FT is finish time of the workflow computed according to Eq. (2).

Critical tasks are those tasks that if delayed, will produce an increment of the application makespan. The *slack* time of a task permits identifying which of the tasks are critical. The slack time of a task t is computed as $\mathrm{slack}(t) = \mathrm{LST}(t) - \mathrm{EST}(t)$. Tasks with a slack time of 0 are critical tasks and should not be delayed. Slack times are used to establish a preference between tasks according to its degree of criticality, i.e. task with shorter slack times should be attended first.

4.2.1 Scheduling Algorithm

At this point in the process, the infrastructure has been fitted to the needs for the next hour of computation for a given application. The scheduling algorithm is now in charge of efficiently executing the workflow considering the available instances. SIAA uses an heuristic scheduling algorithm that optimizes the workflow makespan.

The objective of makespan minimization is accomplished keeping in mind two premises:

1. execute the tasks as fast as possible, and
2. minimize the negative effects of instance failures in the overall makespan.

Algorithm 4 describes the pseudo-code of the scheduling strategy implemented by SIAA.

The scheduling algorithm undertakes the minimization of workflow makespan by reducing the execution time of critical tasks. Slack times are used to determine a priority for the selection of tasks during scheduling. Those tasks ready to execute are sorted by their slack times in increasing order (line 4), i.e. first those tasks that have smaller margin for delay. This criterion first executes first those tasks which potentially have more

Algorithm 4 SIAA: scheduling algorithm.

```
1: procedure ScheduleTasks(instances_od, instances_s):  ▷ input:lists of available on-demand and spot instances
2:     readyTasks ← getReadyTasks() ▷ tasks ready to run
3:     queue ← ∅ ▷ slack priority queue
4:     queue.enqueueAll(readyTasks)
5:     while notEmpty(queue) and notEmpty(instances_od ∪ instances_s) do:
6:         task ← queue.pop() ▷ get tasks by priority
7:         instance ← getECTInstance(task, instances_od)
8:         if no on-demand instance is available then: ▷ check for available spot instances
9:             instance ← getECTInstance(task, instances_s)
10:        schedule(task, instance) ▷ submit for execution
11:        if allCPUsBusy(instance) then:
12:            remove(instance, instances)
```

impact on workflow makespan. One by one, tasks are scheduled until there are no more ready tasks to schedule or all the instances are busy (line 5).

Tasks are scheduled to instances according to two criteria. First, tasks are allocated to on-demand instances (line 7), but if there are no available instances, the algorithm checks the availability of spot instances (line 9). This policy favors that failures, if occur, affect non-critical tasks that could be re-scheduled without a large impact on the overall makespan as they have a wider margin for delays (larger slack times). Second, tasks are allocated to the instances that promise the earliest completion time (ECT). This policy is applied while selecting an on-demand or a spot instance for the task at hand. In this way the algorithm promotes the execution of tasks on the fastest possible (available) instances.

4.2.2 Tasks Termination Handling

As in the case of Scaling First, each time a task sucessfully finishes its execution the processor (virtual CPU) used is marked as free. When a task is terminated due to an out-of-bid error, the task is re-inserted in the corresponding execution queue for its re-scheduling. In both cases, the workflow information is updated and the scheduling algorithm is invoked to continue with the execution of tasks.

As in the case of Scaling First, it is worth to point out that the scheduling algorithm is invoked in two contexts: (*i*) in an hourly basis, each time the autoscaling algorithm is executed, or (*ii*) each time a task terminates its execution or fails due to an out-of-bid error. These events are handled in a separate thread of the autoscaling algorithm.

5 Experimental Settings and Results

For validating our strategy, Scaling First was used as baseline while evaluating performance in terms of number of instances acquired, out-of-bid errors, task failures, speedup and execution cost. SIAA was configured using several settings to evaluate the influence of both the scaling and the scheduling algorithms implemented on it. Experiments consists on the evaluation of such autoscaling strategies/configurations considering a wide number of simulated scenarios using the CloudSim simulator [7] version 3.0. Section 5.1 details the used experimental settings, then, sections 5.2 to 5.6 discuss the results obtained.

5.1 Experimental Settings

Workflow Applications Different scientific workflow applications [4,11] from the areas of Geosciences, Astronomy and Bioinformatics were used to validate the performance of the autoscaling strategies under realistic workload patterns. Mentioned applications are:

- CyberShake (seismic hazard simulation) [13],
- LIGO's Inspiral (detection of gravitational waves) [5],
- Montage (generation of mosaics from the sky) [3], and
- SIPHT (search for small untranslated RNAs) [12].

This selection is based on the fact that such applications have been characterized in depth by recognized researchers in the area [4,11]. These benchmark applications have been used in many studies and they are convenient for replicability of experiments. In addition, these four applications present very dissimilar workload patterns and structure of dependencies appropriate for studying the behavior of the autoscaling strategies under different relevant conditions. Figure 4 shows the tasks profile for the studied workflow applications. The figure presents the median duration in minutes and the number of tasks grouped by task type. Each application comprises 1000 tasks except for SIPHT that comprises 968 tasks.

From the figure can be seen that the applications present very different workload patterns resulting from the different types and number of tasks. CyberShake and Montage comprise many short-duration tasks, and just a few of them (4 and 3 respectively) have durations that exceed one hour of computation. On the other hand, LIGO's Inspiral and SIPHT comprise many long-duration tasks with durations that exceed 1 hour of computation.

The structure of task dependencies on each workflow application are also very dissimilar as can be appreci-

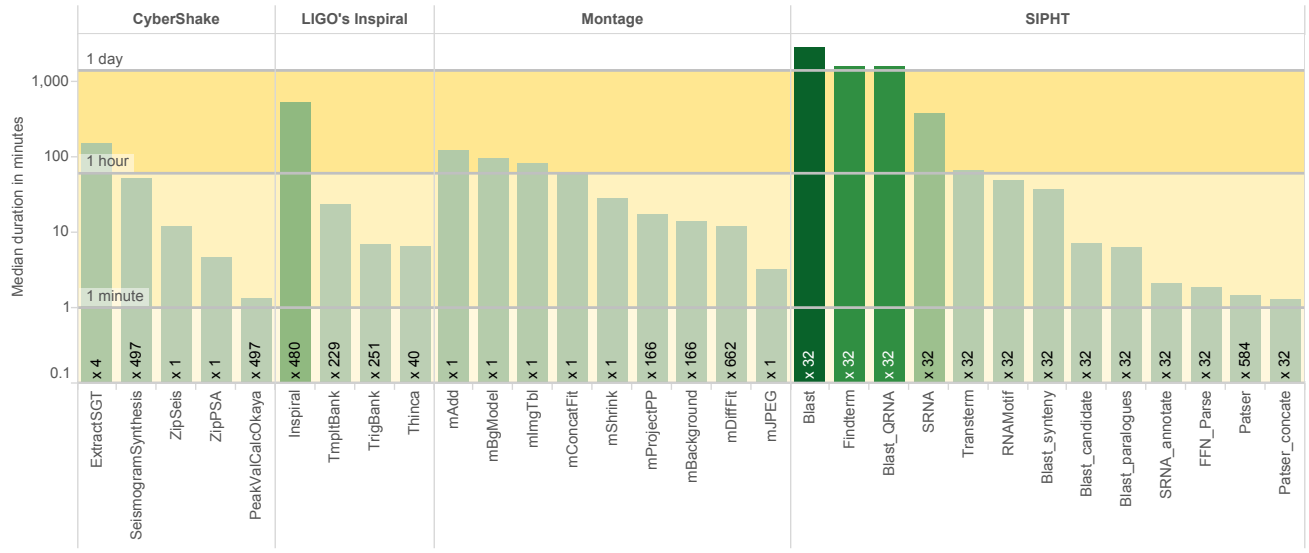


Fig. 4: Tasks profile of the scientific workflows. Bars represent the median task duration for each type in minutes using a logarithmic scale. The label on each bar represents the number of tasks for each type in the corresponding workflow application.

ated from Figure 5. For a more insightful analysis of these applications we invite the reader to refer to other publications [4,11].

CyberShake and Montage present a large number of independent short-duration tasks allowing a greater parallelism. Such applications are prone to be benefited from the exploitation of (cheaper) spot instances as they permit the acquisition of large-size infrastructures. Therefore it is reasonable to think that the use of spot instances will impact positively on makespan and cost.

In opposition to the previously mentioned applications, LIGO's Inspiral and Montage present a number of long-duration tasks, which are more sensitive to out-of-bid errors. Instance failures on such tasks would have a strong negative impact on makespan and cost as tasks need to be restarted and new instances have to be acquired in order to process such tasks. Therefore, it would be expected that using heuristic scheduling will avoid the occurrence of errors affecting such long-duration tasks, leading to superior speedups and cost savings.

VM Instances Table 2 presents the characteristics of each type of *on-demand instances* considered during the experimentation. Values in the table correspond to actual characteristics of Amazon Elastic Compute Cloud (EC2) instances, where vCPU is the number of (virtual) CPUs available for such instance type, ECU^{tot} (acronym for EC2 compute units) are the relative computing power of the instances considering all the virtual

Table 2: Characteristics of the on-demand instances. The data corresponds to instances belonging to the us-west (Oregon) region.

VM type	vCPU	ECU^{tot}	ECU	Price [USD]
t2.micro (GP)	1	1	1	0.013
m3.medium (GP)	1	3	2	0.07
c3.2xlarge (CO)	8	28	3.5	0.42
r3.xlarge (MO)	4	13	3.25	0.35
m3.2xlarge (GP)	8	26	3.25	0.56

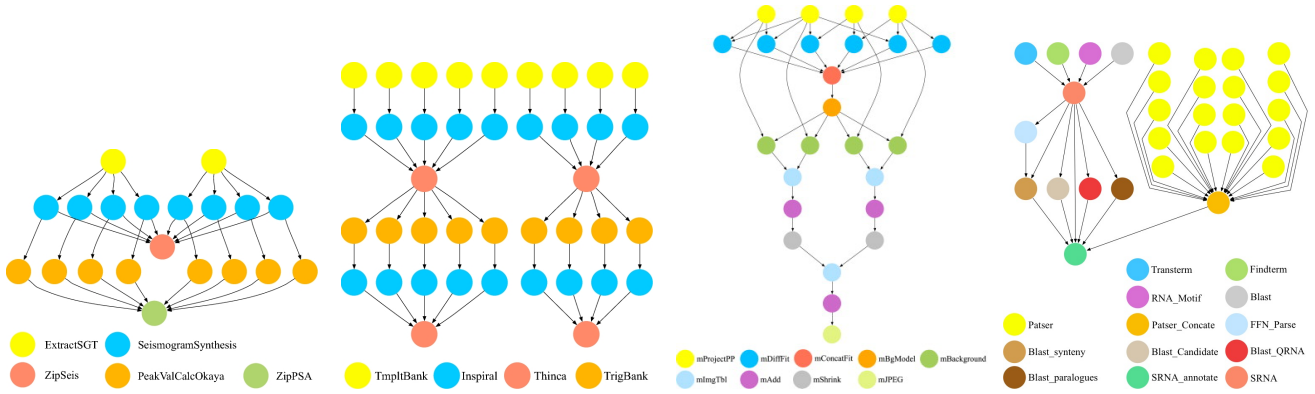
CPUs[4], ECU is the relative performance of one of the CPUs, and the last column denotes the price in US dollars (USD) of an hour of computation. Note that each VM-type name is accompanied by a category label that stands for the type of processes for which the instance is best suitable:

- general purpose (GP): balanced characteristics suitable for most of the tasks,
- compute optimized (CO): best suitable for compute-intensive tasks, and
- memory optimized (MO): best suitable for memory-intensive tasks.

Instance types were selected to provide diverse spectrum of performance and price configurations.

Spot instances have the same characteristics presented on the above table except that their prices vary

[4] One ECU is equivalent to a CPU capacity of a 1.0-1.2 GHz 2007 Opteron.



(a) CyberShake. (b) LIGO's Inspiral. (c) Montage. (d) SIPHT.

Fig. 5: Example of dependency structures on small versions of the studied workflows. Images have been adapted from [4].

over time. For experimentation we used an actual history of Amazon EC2 spot prices that corresponds to the period between March 7$^{\text{th}}$ and June 7$^{\text{th}}$ of 2016 for the US-west region (Oregon).
The first two months of data were used to determine, for each application, the bidding values that satisfy that the probability of failure is below ϕ, where $\phi \in \{0.1, 0.05, 0.01\}$. The data corresponding to the last month of the history was used during the simulations as the actual spot price variations. Using the data in such way allowed us the evaluation of the bidding methods and therefore the performance of the autoscaling strategies while completely ignoring the future evolution of spot prices, which is the real scenario for any EC2 user.

Duration of Tasks For the sake of realism, autoscalers are equipped with a performance model which aims to *estimate* the *actual* duration of tasks. It is worth to point out that the differences between such estimations and the actual duration of tasks produced by the simulator, augments the degree of uncertainty that autoscalers face during the decision making process. It is also important to highlight that, both, the estimated and the actual task durations, are derived from *reference* duration of tasks provided as part of the descriptions of the benchmark workflows [4,11].

The **estimated duration** of tasks is computed using a simple model, which considers the run-time of such task on an instance type used as reference (i.e. m3.2xlarge instance type) and the computing power and disk performance of the instance where the task is going to execute. Formally, the duration d_{ij} of a task i on an instance of type j is estimated as:

$$d_{ij}^{\text{estimated}} = d_i^{\text{ref}} . \mu_j . \rho_{ij},$$

$$\mu_j = \frac{\text{ECU}^{\text{ref}}}{\text{ECU}_j},$$

where, d_i^{ref} is the reference duration of the task i provided in the workflow descriptions, μ_j is the relative CPU performance considering of the ECUs per virtual CPU for an instance of type j (ECU_j) and the ECUs per virtual CPU for the reference instance type (ECU^{ref}), and ρ_{ij} is correction factor that rewards or penalizes the reference duration d_i^{ref} if there is concordance or not between the characteristics of the task (CPU-intensive or memory-intensive) and the characteristics of the instance type (GP, CO or MO), as seen in Table 2.

Although this is a simple model, it is suitable for the simulation experiments described in this study. It is worth to point out that more robust methods could also be applied [19,20], however the study of such methods is beyond the scope of this paper.

The **actual duration** of tasks is computed internally by CloudSim during the simulation. To compute the actual duration of a task, the simulator takes as inputs:

1. the size of the task, which is the number of operations that the task has to perform, and
2. the computing power of the instance used to execute the task (i.e. the ECU for the corresponding instance type).

The size s_i of a task i is computed by sampling from a uniform distribution $U(0.9s_i^{\text{ref}}, 1.1s_i^{\text{ref}})$, where s_i^{ref} is the reference size of task i, which is equal to $s_i^{\text{ref}} = d_i^{\text{estimated}} \cdot \text{ECU}^{\text{ref}}$. This mechanism is implemented by CloudSim as a way to model the *performance variability* of the infrastructure.

During the simulation, the actual duration d_{ij}^{actual} of a task i running on an instance j is:

$$d_{ij}^{\text{actual}} = s_i/\text{ECU}_j,$$

where s_i is the size of task i and and ECU_j is the ECU/vCPU for the instance type j (see table 2). The equation presented gives us a rough approximation of the actual duration of a task, which is ultimately defined also by the evolution of the simulation. For more details on the operation of CloudSim, we invite the reader to refer to the available bibliography and documentation [7].

Budget Allocation For each applicacion, we computed a *fit* budget value, B^{fit}, according to Eq. (3). To generate a larger variety of experimentation scenarios, from each fit budget value we derived *reduced* and *wide* budgets. The reduced and wide budgets are computed from the fit budget (B^{fit}) as $B^{\text{reduced}} = 0.8 \cdot B^{\text{fit}}$ and $B^{\text{wide}} = 1.2 \cdot B^{\text{fit}}$ respectively. Table 3 presents the estimated budgets used for each workflow application.

Table 3: Estimated budgets for each workflow application.

Workflow	B^{reduced}	B^{fit}	B^{wide}
CyberShake	4.04	5.04	6.05
LIGO's Inspiral	7.28	9.09	10.91
Montage	1.39	1.74	2.09
SIPHT	1.48	1.85	2.21

Autoscaling Strategies For performing complete comparison between Scaling First and SIAA we defined different settings of SIAA to evaluate the improvements resulting from the use of (*i*) spot instances, (*ii*) the heuristic scheduling algorithm, and (*iii*) both features together. Table 4 provides a summary of all the autoscaling strategies evaluated and their settings including: if spot instances were considered, which bidding strategy was used and if the heuristic scheduling algorithm was used. As can be seen, 4 families of strategies can be identified, namely Scaling First, SIAA-no-spots, SIAA-greedy, and SIAA-full.

Experimental Scenarios The evaluation of the autoscaling strategies was performed according to a series of scenarios described as follows. Each scenario is defined by (*i*) the autoscaling strategy used (Scaling First, SIAA with all its possible configurations), (*ii*) the workflow at hand (CyberShake, LIGO's Inspiral, Montage, SIPHT), and (*iii*) the budget available (fit, reduced, wide). Each experimental scenario was simulated 30 times (10 for each type of budget) for statistical significance of results using the CloudSim simulator [7]. In all cases, task durations were affected by a 20% error (uniformly distributed, as described in "Duration of Tasks") to provide a more realistic environment according to the performance variability of the cloud. This 20% variability on performance was selected in concordance with the settings used by other colleagues [16,17]. Table 5 summarizes the parameters used in the experiments.

Table 4: Summary of the autoscaling strategies and their configuration. Text in parenthesis P<0.1, P<0.05 and P<0.01 indicate that the failure probability is below 0.1, 0.05 and 0.01.

Name	Spot?	Bidding	Scheduling
Scaling First	no	-	greedy
SIAA-no-spots	no	-	heuristic
SIAA-greedy (current)	yes	current	greedy
SIAA-greedy (P<0.1)	yes	prob.	greedy
SIAA-greedy (P<0.05)	yes	prob.	greedy
SIAA-greedy (P<0.01)	yes	prob.	greedy
SIAA-full (current)	yes	current	heuristic
SIAA-full (P<0.1)	yes	prob.	heuristic
SIAA-full (P<0.05)	yes	prob.	heuristic
SIAA-full (P<0.01)	yes	prob.	heuristic

Table 5: Experimental settings. SIAA was configured without using spot instances, and with spot instances using the current price, and the probabilistic bidding methods with probability of failure P<α and $\alpha \in \{0.1, 0.05, 0.01\}$. These settings give a total of 1200 simulations.

Setting	Value
Strategy	Scaling First, SIAA (all 9 configurations)
Application	CyberShake, LIGO, Montage, SIPHT
Budget	fit, reduced, wide
Perf. var.	20%
Repetitions	30 (10 per type of budget)

5.2 Global Results

Table 6 presents the global results per strategy including the mean values for the metrics of the number of instances, the percentage of spot instances, speedup with respect to the sequential execution of the workflows, cost of execution and number of out-of-bid errors and

task failures. The best performances are highlighted in bold font.

As can be observed form the table, using spot instances always permits the acquisition of a larger amount of instances, being the extreme case, the two flavors of SIAA using the current price bidding method. On the other side of the spectrum we can find the two methods that disregard the use of spot instances, i.e. Scaling First and SIAA-no-spots. The reader can observe that the proportion of spot instances varies from 65.8% to 91.7%.

It is worth to note that acquiring a larger amount of spot instances is not always convenient. We can observe that the use of a naïve bidding strategy such as current price leads to higher rates of task failures which impact notably in the speedup of applications. A task failure implies the re-execution of such task and therefore an overhead in the total execution time[5]. As can be seen from the table, the use of such bidding strategy leads to a performance of the applications, which is below the performance of the baseline strategy (Scaling First).

Considering the cost of execution, the behavior is different and the presence of failures does not imply so profound implications. The main difference relies on the fact that in the case of failure the last partial hour is not charged. Even more, the fact that the majority of the tasks comprised in the studied applications have a duration below 1 hour of execution leads to many situations in which tasks start and finish their execution on instances which are not charged.

Regarding speedup, except for SIAA-full (current) and SIAA-greedy (current), our strategy always outperforms Scaling First. The widest speedup margins are observed for the full configurations of SIAA (i.e. using spot instances and heuristic scheduling). In terms of execution cost, all the SIAA configurations outperform Scaling First. The lowest costs are obtained by SIAA-full followed by SIAA-greedy, SIAA-no-spots and finally Scaling First.

From these results we can say that SIAA-full ($P<0.01$) achieves the fastest and cheapest executions supporting our hypothesis that the wise use of spot instances through proper bidding and scheduling permits the exploitation of better time and cost improvements. The following sections perform a fine-grained analysis to unveil the behavior of strategies per application. The remainder of this section is dedicated to study in detail the speedup, cost of execution and the effect of failures on the two latter mentioned metrics.

[5] Note that such effect could be mitigated by the use of a checkpointing technique.

5.3 Speedup Analysis

Figure 6 presents the average speedups and standard deviations per strategy for each workflow application. Strategies are sorted in increasing order according to speedup.

In almost all the cases SIAA-full (current) and SIAA-greedy (current) present the lowest speedups below the baseline strategy because of the large number of failures that occur.

For CyberShake and Montage (Figures 6a and 6c), for which most of their tasks have durations below 1 hour of computation the use of spot instances with a proper bidding method implies an substantial improvement in terms of speedup. The best performing strategies are SIAA-full and -greedy with probabilistic bidding followed by SIAA-no-spots. In all the cases we can observe also the positive incidence of the heuristic scheduling algorithm because, for a given failure probability, it always happen that SIAA-full outperforms SIAA-greedy[6].

Conversely, for LIGO's Inspiral and SIPHT (Figures 6b and 6d) the most important contributions to speedup improvement come from the usage of the heuristic scheduling algorithm. The reason behind such observation is that for those applications exist many long-duration tasks (LIGO's Inspiral: almost half of the tasks demand 6 hours of computation; SIPHT: 96 tasks that require more than 1 day of execution) and therefore the avoidance of failures on such tasks permits that the degradation of the global execution time is minimum.

Note that for LIGO's Inspiral SIAA-full (probabilistic bidding) and SIAA-no-spots obtain the best speedups indicating the benefits of using the heuristic scheduling and in second place the benefits of using spot instances.

For SIPHT, SIAA-full and SIAA-greedy with probabilistic bidding ($P<0.01$) and SIAA-no-spots it is important to keep the number of out-of-bid errors to the minimum due to the extreme long-duration of some of their tasks.

We can conclude that the usage of spot instances permits achieving better speedups. However, the usage of the scheduling algorithm is very important even if the autoscaling strategy is not using spot instances for execution, i.e. SIAA-no-spots. The heuristic scheduling leads to a better assignment of tasks improving the workflow makespan. In addition, when spot instances are used, the scheduling algorithm avoids the execution of tasks affecting critical tasks. For LIGO's Inspiral and SIPHT, SIAA-full ($P<0.01$) always achieves the high-

[6] Note also that SIAA-no-spots always outperforms Scaling First and that they differ on the presence of the heuristic scheduling algorithm.

Table 6: Summary of results. Each row presents the mean values per strategy considering all the studied applications.

Strategy	Instances	% spot	Out-of-bid errors	Task Failures	Speedup	Cost
Scaling First	15.5	0.0%	**N/A**	**N/A**	57.6	174.9
SIAA-no-spots	15.6	0.0%	**N/A**	**N/A**	68.8	139.4
SIAA-greedy (current)	150.0	91.6%	135.8	755.7	49.2	129.0
SIAA-greedy (P<0.1)	28.8	71.4%	5.9	23.4	70.6	111.7
SIAA-greedy (P<0.05)	26.9	70.2%	3.3	13.2	72.5	105.4
SIAA-greedy (P<0.01)	23.8	65.8%	0.6	3.6	73.5	101.6
SIAA-full (current)	158.2	91.7%	143.9	825.1	52.1	120.2
SIAA-full (P<0.1)	28.6	71.8%	5.1	24.7	73.7	107.4
SIAA-full (P<0.05)	26.9	70.4%	2.5	13.1	76.5	99.2
SIAA-full (P<0.01)	23.8	66.0%	0.6	3.7	**77.8**	**96.1**

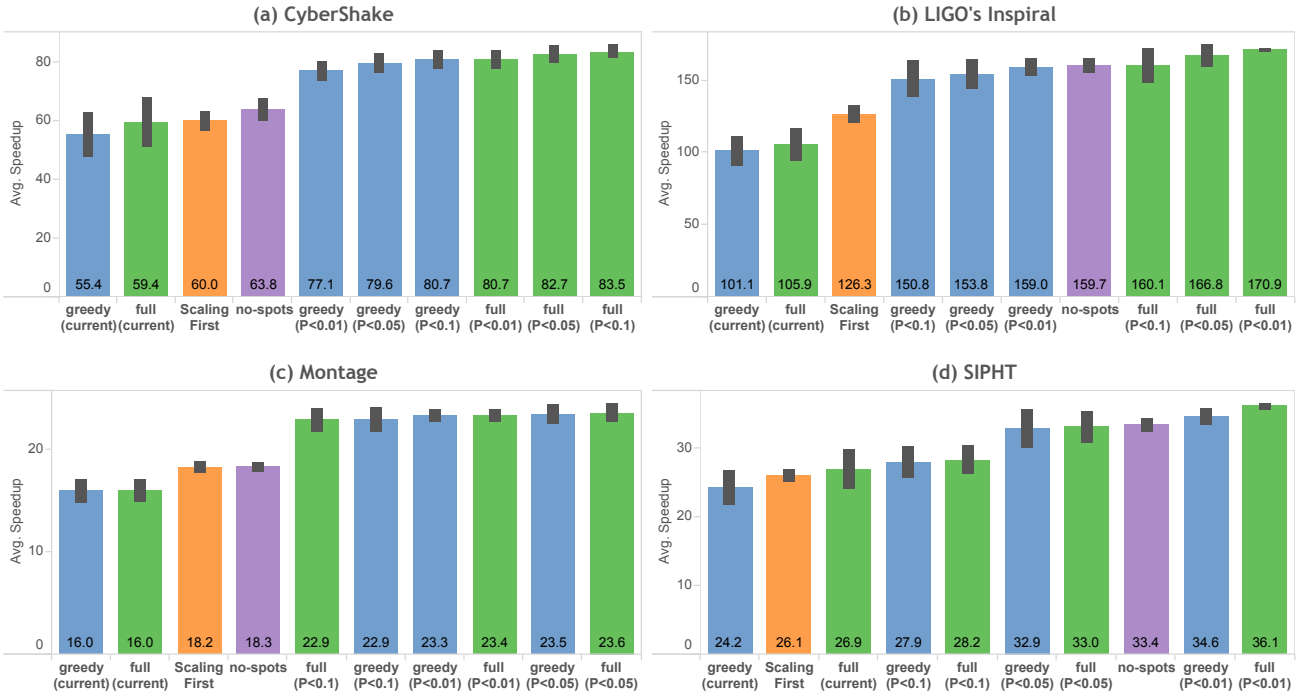


Fig. 6: Speedup comparison for each workflow application. Words in parenthesis (current), (P<0.1), (P<0.05) and (P<0.01) stand for the current and the probabilistic bidding methods.

est speedups. This observation is in concordance with the fact that such bidding strategy produces the lowest number of failures that impact on the long-duration tasks that characterize such applications. For the other two applications, CyberShake and Montage with tasks of shorter duration, less conservative bidding methods (P<0.1 and P<0.05) work properly. This analysis will be deepened in Section 5.5.

5.4 Execution Cost Analysis

Figure 7 presents the average execution costs and standard deviations per strategy for each workflow application. Strategies are sorted in decreasing order according to cost.

For all the applications, using spot instances always leads to cost savings when comparing with the baseline strategy, Scaling First. This behavior is observed even for the two flavors of SIAA that use the current price bidding method which leads to the occurrence of an extremely large number of failures. This observation is explained by the fact that the last partial hour

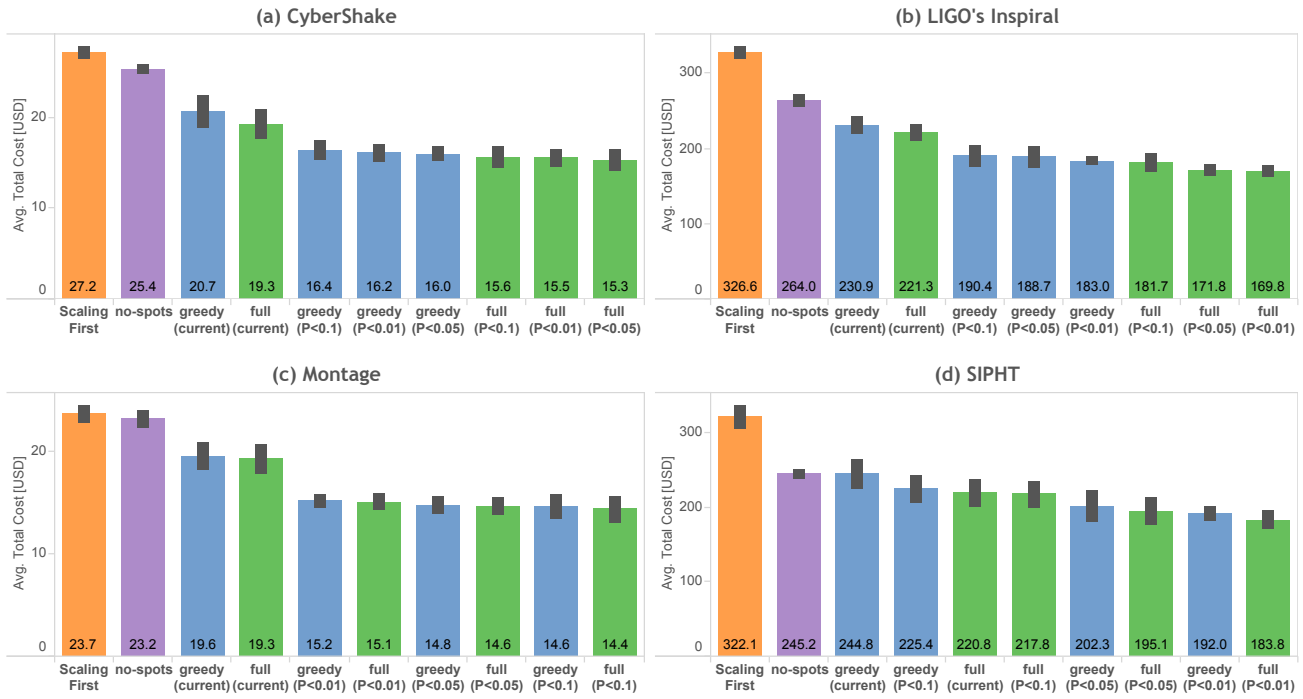


Fig. 7: Execution cost comparison for each workflow application. Words in parenthesis (current), (P<0.1), (P<0.05) and (P<0.01) stand for the current and the probabilistic bidding methods.

of computation for a failing instance is not charged. As the current price bidding method produces many failures within the first hour of computation, the total cost does not increase significantly and thus never achieves the level of strategies that do not consider spot instances (Scaling First and SIAA-no-spots).

It is also important to note that for CyberShake and Montage (Figures 7a and 7c) SIAA-full and SIAA-greedy using the probabilistic bidding method with the three thresholds achieve the lowest costs. It is also important to highlight that such configurations achieved very similar results in terms of execution cost indicating that the use of the heuristic scheduling algorithm is not so important as the usage of spot instances. This observation is coherent with our original hipothesis that the large number of independent short-duration tasks on such applications are most benefited by the use of infrastructures comprising cheap (spot) instances.

Differences in cost become larger between such strategies when they are applied to the LIGO's Inspiral and SIPHT applications (Figures 7b and 7d). However, strategies do not present the same behavior for both applications. In the case of LIGO's Inspiral, the best performing strategy is SIAA-full with probabilistic bidding in comparison with SIAA-greedy with the same bidding method. In the case of SIPHT strategies are ranked in a different manner. SIAA-full always leads to better cost savings than SIAA-greedy for each threshold of the probabilistic bidding method (P<0.01, P<0.05, P<0.1).

Note that SIAA-no-spots is the second most expensive strategy and that in the case of SIPHT the gap in cost is very large. The same observation is true for LIGO's Inspiral. These cost reductions are explained by the fact that the heuristic scheduling algorithm permits better task-to-instance assignments leading to shorter workflow makespans and therefore to the purchase of less instance-hours.

5.5 Effect of Tasks Failures

This section is dedicated to the analysis of task failures in both, cost and speedup of the applications considering all the spot-aware strategies, i.e. SIAA-full and SIAA-greedy with all the bidding methods. Figure 5.5 shows the effect of tasks failures in the speedup (on top) and cost-of-execution metrics (at the bottom) for the mentioned autoscaling strategies.

As can be seen in the figure, the general trend is that a larger amount of task failures leads to lower speedups and higher execution costs. This trend does not completely apply to the case of the Montage application. It is important to point out that by comparing the trend

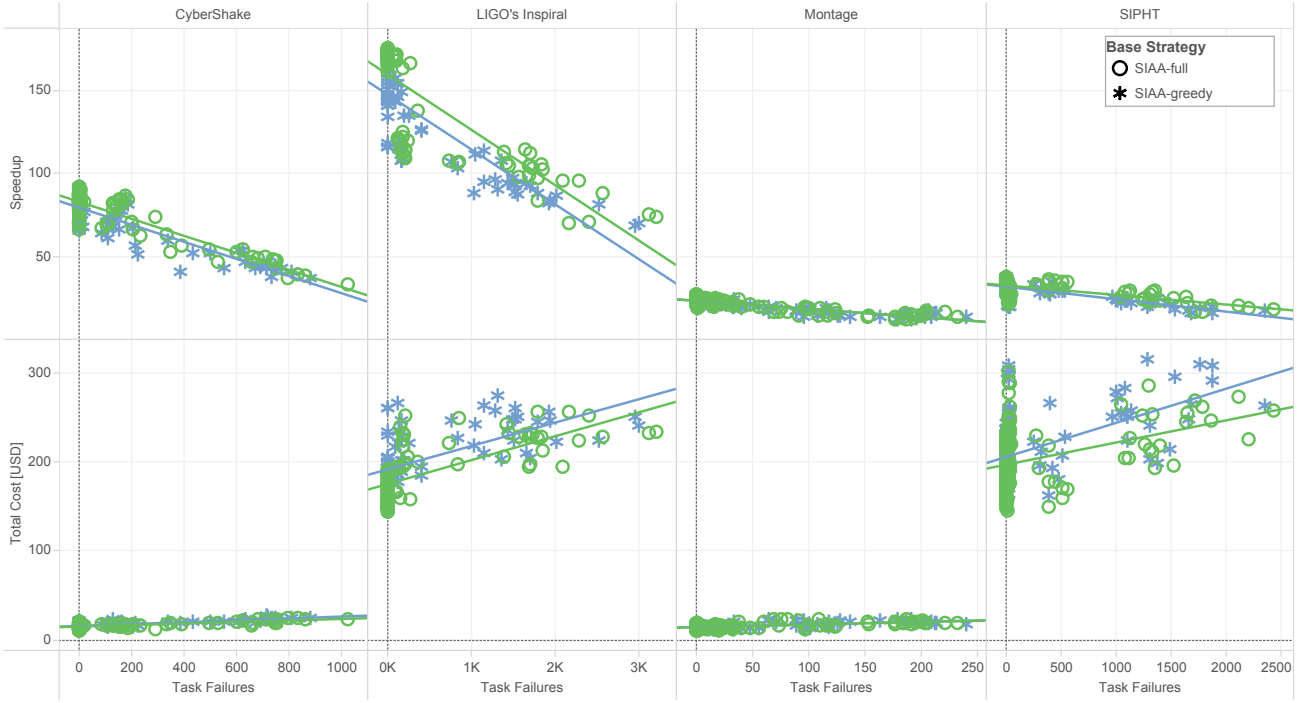


Fig. 8: Effect of tasks failures in the speedup of the applications and the total execution cost. Curves correspond to strategies using all the bidding strategies.

lines, the use of the scheduling algorithm (SIAA-full) leads to better speedups for the same number of task failures. From another point of view, SIAA-full can deal with a higher number of tasks failures and still achieve the same level of speedups than SIAA-greedy thanks to the use of the heuristic scheduling algorithm.

Similar observations can be made when evaluating the cost of execution, however, in this case the cost has a negative correlation with the number of task failures. It can be also seen that cost trend lines for CyberShake and Montage are also minimal and it seems there is no difference in cost considering SIAA-full and SIAA-greedy. The same behavior was observed in Figures 7a and 7c. However, for LIGO's Inspiral and SIPHT it can be seen that SIAA-greedy leads to higher costs than SIAA-full. As in the case of speedup, such difference is caused by the use of the heuristic scheduling algorithm which can deal better with the occurring failures.

5.6 Statistical Significance of Comparisons

From previous results we can observe that SIAA-full in general outperformed the remaining strategies in its various configurations. This section provides an analysis of the speedup and cost improvements achieved and their statistical significance with respect to the Scaling First, our baseline for comparisons. Table 7 shows, for each application the speedup improvements resulting from the comparison with Scaling First. Only the best performing configuration for SIAA-greedy and SIAA-full are presented. The table presents the difference between speedups and the percentage of improvement with respect to the average Scaling First speedup.

To check statistical significance of results we used the Mann–Whitney U test [15] with a confidence level of $\alpha = 0.01$. This is a non-parametric test that evaluates that two samples come from the same population based on a comparison of median values. In our case the significance of results can be asserted when the resulting p-value is less than α, i.e. when we discard the null hypothesis. The table also presents the U statistic, the p-values and if the test indicates that the results are significant. Note that the improvement column is computed as the difference of median values for each sample.

We can observe that for all the applications, but Montage, SIAA with its full configuration outperformed Scaling First with a wide margin in various cases (from 7.98% to 40.02%). As SIAA takes advantage of instances of better cost-performance, it is able to acquire more computing power (more instances) with the same budget. This leads to an increase of the amount of tasks that can be executed in parallel and therefore to a re-

Table 7: Speedup improvements of SIAA (with and without using spot instances) compared with Scaling First. Results are aggregated by workflow application.

Workflow	Strategy	Improvement	Percentage	U statistic	p-value	significant?
CyberShake	SIAA-no-spots	4.91	7.98%	564	0.092	no
	SIAA-greedy (P<0.1)	20.71	33.70%	892	< 0.001	yes
	SIAA-full (P<0.05)	**22.44**	**36.52%**	900	< 0.001	yes
LIGO's Inspiral	SIAA-no-spots	34.70	27.24%	898	< 0.001	yes
	SIAA-greedy (P<0.1)	39.03	30.64%	691	< 0.001	yes
	SIAA-full (P<0.01)	**43.48**	**34.13%**	900	< 0.001	yes
Montage	SIAA-no-spots	-0.02	-0.10%	471	0.756	no
	SIAA-greedy (P<0.05)	**5.87**	**32.52%**	888	< 0.001	yes
	SIAA-full (P<0.05)	5.84	32.35%	890	< 0.001	yes
SIPHT	SIAA-no-spots	8.17	31.81%	900	< 0.001	yes
	SIAA-greedy (P<0.01)	9.94	38.68%	893	< 0.001	yes
	SIAA-full (P<0.01)	**10.28**	**40.02%**	900	< 0.001	yes

duction of the overall makespan. In addition, the heuristic scheduling algorithm permits maximizing the use of such instances and achieve higher performances than those achieved by the remaining methods.

It can be seen that depending on the application, the results of the test present differences. For LIGO's Inspiral and SIPHT, all the reported strategies present statistically significant speedup improvements over Scaling First. For the remaining two applications, CyberShake and Montage, the only strategy that is not significantly better than Scaling First is SIAA-no-spots. These results demonstrate the importance of using at least one of the two features implemented in SIAA in order to achieve speedup improvements.

As in the case of speedup improvements, Table 8 presents the raw and percentual cost savings of SIAA with respect to Scaling First as well as the results of the Mann–Whitney U test.

We can observe that for all the applications, SIAA-full outperformed Scaling First with a wide margin in various cases (from 3.23% to 47.93%). We can observe from the statistical test results that almost all the configurations outperform the baseline strategy, except SIAA-no-spots applied to the Montage workflow. Another interesting result regarding Montage is that SIAA-full and SIAA-greedy show the same cost reduction which is consistent with the results in Figure 7c. The test indicates that, except for the mentioned case, all the reported strategies show statistically significant cost savings indicating that using at least one of the two novel features of our strategy is advantageously. In the case of Montage, significant cost savings are achieved when spot instances are considered during the scaling process. This observation confirms the importance of better prices associated to spot instances.

6 Related Work

In the literature there are many approaches addressing the efficient management of workflow applications on the cloud. Most of them focus on the problem of workflow scheduling, which is a classical problem in distributed computing. For example, in the work of Poola et al. [21] the authors present a robust scheduling algorithm with resource allocation policies that schedules workflow tasks on heterogeneous cloud resources while trying to minimize makespan and cost under budget and deadline constraints. Malawski et al. [14] address the problem of efficient management of workflow ensembles under budget and deadline constraints on the cloud using static and dynamic strategies for both task scheduling and resource provisioning, considering homogeneous resources. In contrast with our paper, both works deal with the problem of scheduling workflow applications under deadline and budget constraints but without taking advantage of spot instances.

Workflow autoscaling has been also tackled under both, deadline and budget constraints. The problem was first addressed by Mao and Humphrey [16]. They proposed an autoscaling strategy for the efficient execution of multiple workflow applications subject to deadline constraints. Later, the same authors moved to the problem of workflow autoscaling but considering budget constraints [17]. The strategies proposed simultaneously address the problems of scaling the infrastructure and scheduling workflow tasks in heterogeneous cloud infrastructures but they did not considered spot instances, which is the main difference with respect to our work.

There are also several scheduling approaches aiming to minimize the execution cost of scientific workflow approaches using mixed infrastructures comprising

Table 8: Cost savings of SIAA (with and without using spot instances) compared with Scaling First. Results are aggregated by workflow application.

Workflow	Strategy	Reduction	Percentage	U statistic	p-value	significant?
CyberShake	SIAA-no-spots	1.6	5.99%	713	< 0.001	yes
	SIAA-greedy (P<0.01)	10.7	40.13%	900	< 0.001	yes
	SIAA-full (P<0.05)	**11.4**	**42.79%**	900	< 0.001	yes
LIGO's Inspiral	SIAA-no-spots	71.5	21.58%	900	< 0.001	yes
	SIAA-greedy (P<0.01)	148.7	44.85%	900	< 0.001	yes
	SIAA-full (P<0.01)	**158.9**	**47.93%**	900	< 0.001	yes
Montage	SIAA-no-spots	0.8	3.23%	528.5	0.246	no
	SIAA-greedy (P<0.1)	**8.8**	**37.80%**	900	< 0.001	yes
	SIAA-full (P<0.1)	**8.8**	**37.80%**	900	< 0.001	yes
SIPHT	SIAA-no-spots	87.1	26.40%	894	< 0.001	yes
	SIAA-greedy (P<0.01)	142.9	43.31%	900	< 0.001	yes
	SIAA-full (P<0.01)	**148.1**	**44.89%**	900	< 0.001	yes

spot and on-demand instances. Poola et al. [22] proposed a fault tolerant and robust scheduling algorithm with a mixed infrastructure to reduce the cost of execution whilst meeting the workflow deadlines. Zhou et al. [28] developed a workflow scheduling system aimed for the minimization of monetary cost using spot and on-demand instances subject to probabilistic deadline constraints. However, the main difference between their work and ours is that we focus on a budget-constrained autoscaling problem while the efforts mentioned focus on solving scheduling problems subject to deadline constraints, thus they are useful in different scenarios.

7 Concluding Remarks

This paper presented a new strategy for the autoscaling of scientific workflows entitled Spots Instances Aware Autoscaling (SIAA) strategy. SIAA incorporates two novel features namely (i) an infrastructure scaling algorithm that uses a combination of on-demand and (unreliable but low-cost) spot instances, and (ii) a heuristic scheduling method for makespan minimization whose aim is to reduce the chance that out-of-bid errors interrupt the execution of critical tasks. Then, our main goal is to take advantage of both types of VM instances while minimizing makespan under a given budget.

Experiments with four real-world scientific workflow applications showed that the combination of those two features permits SIAA to overcome Scaling First (a state-of-the-art autoscaling strategy) from 7.98% to 40.02% in terms of speedup. Results also demonstrated that SIAA led to cost reductions of 3.23% to 47.93%. It is important to highlight that SIAA using only one of the two features at a time lead to improvements that are halfway between Scaling First and the full-fledged version of SIAA. In addition, depending on the application, it might result more important to use spot instances or the use of heuristic scheduling algorithm. All these observations highlight the importance of using both features together to achieve good performance that arises from their synergy.

It is also worth mentioniong that this strategy could be implemented for its operation on real clouds as a utility for the users. Such utility could run as central manager receiving workflow submissions and administrating instances and tasks through the APIs that cloud providers make available (e.g. for creating/destroying instances, scheduling tasks, execution monitoring, etc.). Also, some required mechanisms have to be adopted, like for example the management of applications [9], estimation of tasks running-time [19] and spot prices prediction [25]. Finally it is worth to highlight that such utility could be implemented as a service on the cloud-provider side or it could be part of an external tool.There are some research lines to develop as part of the future work. First, the scaling algorithm in SIAA operates subject to a parameter that determines a balance between on-demand and spot instances that remains static along the execution of the application. It seems a good idea to explore how to dynamically adapt this balance according to the application needs in order to improve speedups and execution costs. Second, it is important to study the applicability of more sophisticated bidding methods. Better bidding methods may help in the reduction of bid failures and therefore help in the achievement of better overall performances. Finally it is crucial to study the incidence of data transfer times and hence networking cost during the autoscaling process beyond data processing times. These features open the door for the design of new autoscaling techniques aimed on big data workflows able to more accurately balance performance and cloud resources rent costs.

Acknowledgements This research is supported by the ANPCyT project No. PICT-2012-2731, and by the SeCTyP-UNCuyo project No. M041. DAM and YG want to thank the CONICET for the granted fellowships. Finally, the authors want to thank the anonymous reviewers for their valuable comments and suggestions that helped to improve the quality of this paper.

Reproducibility of Results The data used in this study and additional results are available in the following URL:

`http://damonge.wordpress.com/research/wfautosca-set2016`

References

1. Agmon Ben-Yehuda, O., Ben-Yehuda, M., Schuster, A., Tsafrir, D.: Deconstructing amazon EC2 spot instance pricing. ACM Transactions on Economics and Computation **1**(3), 16:1–16:20 (2013)
2. Amazon: EC2 spot instances. `http://aws.amazon.com/ec2/purchasing-options/spot-instances/` (2016). [Online; accessed September-2016]
3. Berriman, G.B., Deelman, E., Good, J.C., Jacob, J.C., Katz, D.S., Kesselman, C., Laity, A.C., Prince, T.A., Singh, G., Su, M.H.: Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand. pp. 221–232 (2004). DOI 10.1117/12.550551
4. Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M.H., Vahi, K.: Characterization of scientific workflows. In: Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on, pp. 1–10 (2008)
5. Brown, D.A., Brady, P.R., Dietz, A., Cao, J., Johnson, B., McNabb, J.: A Case Study on the Use of Workflow Technologies for Scientific Analysis: Gravitational Wave Data Analysis, pp. 39–59. Springer London, London (2007)
6. Buyya, R., Yeo, C., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems **25**(6), 599–616 (2009)
7. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A., Buyya, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience **41**(1), 23–50 (2011)
8. Corradi, A., Fanelli, M., Foschini, L.: VM consolidation: A real case based on openstack cloud. Future Generation Computer Systems **32**, 118–127 (2014)
9. Deelman, E., Mehta, G., Singh, G., Su, M.H., Vahi, K.: Pegasus: Mapping large-scale workflows to distributed resources. In: I.J. Taylor, E. Deelman, D.B. Gannon, M. Shields (eds.) Workflows for e-Science: Scientific Workflows for Grids, pp. 376–394. Springer London (2007)
10. Iosup, A., Yigitbasi, N., Epema, D.: On the performance variability of production cloud services pp. 104–113 (2011)
11. Juve, G., Chervenak, A., Deelman, E., Bharathi, S., Mehta, G., Vahi, K.: Characterizing and profiling scientific workflows. Future Generation Computer Systems **29**(3), 682–692 (2013)
12. Livny, J., Teonadi, H., Livny, M., Waldor, M.K.: High-throughput, kingdom-wide prediction and annotation of bacterial non-coding rnas. PLoS ONE **3**(9), 1–12 (2008)
13. Maechling, P., Deelman, E., Zhao, L., Graves, R., Mehta, G., Gupta, N., Mehringer, J., Kesselman, C., Callaghan, S., Okaya, D., Francoeur, H., Gupta, V., Cui, Y., Vahi, K., Jordan, T., Field, E.: SCEC CyberShake workflows-automating probabilistic seismic hazard analysis calculations. In: I.J. Taylor, E. Deelman, D.B. Gannon, M. Shields (eds.) Workflows for e-Science: Scientific Workflows for Grids, pp. 143–163. Springer London (2007)
14. Malawski, M., Juve, G., Deelman, E., Nabrzyski, J.: Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. Future Generation Computer Systems **48**, 1–18 (2015). Special Section: Business and Industry Specific Cloud
15. Mann, H.B., Whitney, D.R.: On a test of whether one of two random variables is stochastically larger than the other. The Annals of Mathematical Statistics **18**(1), 50–60 (1947)
16. Mao, M., Humphrey, M.: Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, p. 49. ACM (2011)
17. Mao, M., Humphrey, M.: Scaling and scheduling to maximize application performance within budget constraints in cloud workflows. In: Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on, pp. 67–78. IEEE (2013)
18. Monge, D.A., García Garino, C.: Adaptive spot-instances aware autoscaling for scientific workflows on the cloud. In: High Performance Computing, *Communications in Computer and Information Science*, vol. 485, pp. 13–27. Springer Berlin Heidelberg (2014)
19. Monge, D.A., Holec, M., Železný, F., Garino, C.G.: Ensemble learning of runtime prediction models for gene-expression analysis workflows. Cluster Computing pp. 1–13 (2015)
20. Pllana, S., Brandic, I., Benkner, S.: A survey of the state of the art in performance modeling and prediction of parallel and distributed computing systems. International Journal of Computational Intelligence Research **4**(1), 279–284 (2008)
21. Poola, D., Garg, S.K., Buyya, R., Yang, Y., Ramamohanarao, K.: Robust scheduling of scientific workflows with deadline and budget constraints in clouds. In: 2014 IEEE 28th International Conference on Advanced Information Networking and Applications, pp. 858–865. IEEE (2014)
22. Poola, D., Ramamohanarao, K., Buyya, R.: Fault-tolerant workflow scheduling using spot instances on clouds. Procedia Computer Science **29**, 523–533 (2014)
23. Rahman, M., Hassan, R., Ranjan, R., Buyya, R.: Adaptive workflow scheduling for dynamic grid and cloud computing environment. Concurrency and Computation: Practice and Experience **25**(13), 1816–1842 (2013)
24. Schad, J., Dittrich, J., Quiané-Ruiz, J.A.: Runtime measurements in the cloud: Observing, analyzing, and reducing variance. Proc. VLDB Endow. **3**(1-2), 460–471 (2010)
25. Turchenko, V., Shultz, V., Turchenko, I., Wallace, R.M., Sheikhalishahi, M., Vazquez-Poletti, J.L., Lucio, G.: Spot price prediction for cloud computing using neural networks. International Journal of Computing **12**(4), 348–359 (2013)
26. Voorsluys, W., Buyya, R.: Reliable provisioning of spot instances for compute-intensive applications. In: Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on, pp. 542–549. IEEE (2012)

27. Wallace, R., Turchenko, V., Sheikhalishahi, M., Turchenko, I., Shults, V., Vazquez-Poletti, J., Grandinetti, L.: Applications of neural-based spot market prediction for cloud computing. In: Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), 2013 IEEE 7th International Conference on, vol. 2, pp. 710–716 (2013)
28. Zhou, A.C., He, B., Liu, C.: Monetary cost optimizations for hosting workflow-as-a-service in iaas clouds. IEEE Transactions on Cloud Computing **4**(1), 34–48 (2014)
29. Zhu, M., Wu, Q., Zhao, Y.: A cost-effective scheduling algorithm for scientific workflows in clouds. In: Performance Computing and Communications Conference (IPCCC), 2012 IEEE 31st International, pp. 256–265 (2012)

Gmail

David A. Monge <damonge27@gmail.com>

SI: Elastic Data Management in Cloud Systems

David A. Monge <damonge27@gmail.com> Wed, Feb 1, 2017 at 12:48 PM
To: Riad MOKADEM <Riad.Mokadem@irit.fr>, hameurlain@irit.fr
Cc: Tigre <cmateos2006@gmail.com>, Yisel Garí <yiselgari@gmail.com>, Carlos García Garino <cgarino04@gmail.com>

Dear Editors-in-Chief, we hope this mail finds you well and sorry for taking a bit of your valuable time.

We are writing you once more to know whether there are news about our paper "Autoscaling Scientific Workflows on the Cloud by Combining On-demand and Spot Instances" submitted to the Special Issue on Elastic Data Management in Cloud Systems (CCSE). The outcome of the revision process was acceptance subject to fixing some minor comments made by the reviewers, based on which a revised version of the paper was re-submitted by December 20, 2016.

One of the authors (Dr. David Monge) is close to apply for a permanent research position in the very same topic at our National Research Council, and thus a final decision regarding the evaluation process would be very helpful to better support his application.

Thank you again for your patience and consideration.

Sincerely,

--
Dr. David A. Monge

ITIC Research Institute
National University of Cuyo
Edificio del Espacio de la Ciencia y la Tecnología (ECT)
Padre Jorge Contreras 1300
M5502JMA Mendoza, Argentina
Room: 906a
Tel: +54 (261) 4291000
E-mail: dmonge@uncu.edu.ar

Gmail

David A. Monge <damonge27@gmail.com>

SI: Elastic Data Management in Cloud Systems

Riad MOKADEM <Riad.Mokadem@irit.fr> Wed, Feb 1, 2017 at 1:20 PM
To: "David A. Monge" <damonge27@gmail.com>
Cc: hameurlain@irit.fr, Tigre <cmateos2006@gmail.com>, Yisel Garí <yiselgari@gmail.com>, Carlos García Garino <cgarino04@gmail.com>

Dear David,

We confirm that your paper "Autoscaling Scientific Workflows on the Cloud by Combining On-demand and Spot Instances" has been definitively accepted and sent to the publisher for publication.

Let us know if you need an official notification.

Regards,

Riad Mokadem
on behalf of Guest editors for the Special Issue on 'Elastic Data Management in Cloud Systems'/ IJCSSE
[Quoted text hidden]