Future Generation Computer Systems

A Q-learning Approach for the Autoscaling of Scientific Workflows in the Cloud

--Manuscript Draft--

Manuscript Number:	FGCS-D-21-00508R2
Article Type:	Full Length Article
Keywords:	Cloud Computing; Autoscaling; workflow; reinforcement learning
Corresponding Author:	Yisel Garí ARGENTINA
First Author:	Yisel Garí, BSc
Order of Authors:	Yisel Garí, BSc
	David Monge, Ph.D
	Cristian Mateos, Ph.D
Abstract:	Autoscaling strategies aim to exploit the elasticity, resource heterogeneity and varied prices options of a Cloud infrastructure to improve efficiency in the execution of resource-hungry applications such as scientific workflows. Scientific workflows represent a special type of Cloud application with task dependencies, high-performance computational requirements and fluctuating workloads. Hence, the amount and type of resources needed during workflow execution changes dynamically over time. The well-known autoscaling problem comprises (i) scaling decisions, for adjusting the computing capacity of a virtualized infrastructure to meet the current demand of the application and (ii) task scheduling decisions, for assigning tasks to specific acquired Cloud resources for execution. Both are highly complex sub-problems, even more because of the uncertainty inherent to the Cloud. Reinforcement Learning (RL) provides a solid framework for decision-making problems in stochastic environments. Therefore, RL offers a promising perspective for designing Cloud autoscaling strategies based on an online learning process. In this work, we propose a novel formulation for the problem of infrastructure scaling in the Cloud as a Markov Decision Process, and we use the Q-learning algorithm for learning scaling policies, while demonstrating that considering the specific characteristics of workflow applications when taking autoscaling decisions can lead to more efficient workflow executions. Thus, our RL-based scaling strategy exploits the information available about workflow dependency structures. Simulations performed on four well-known workflows demonstrate significant gains (25%-55%) of our proposal in comparison with a similar state-of-the-art proposal.
Suggested Reviewers:	José Luis Vázquez Poletti, Dr. Universidad Complutense de Madrid Facultad de Informatica jlvazquez@fdi.ucm.es
	Javid Taheri, Dr. Karlstads Universitet javid.taheri@kau.se
	Andrés Neyem, Dr. Pontificia Universidad Católica de Chile: Pontificia Universidad Catolica de Chile aneyem@ing.puc.cl
Response to Reviewers:	The answers were uploaded as a separate file

Dear Editor:

Attached please find the revised version of the article entitled *"A Q-learning Approach for the Autoscaling of Scientific Workflows in the Cloud"* previously submitted for review purposes to the journal Future Generation Computer Systems. This manuscript is the authors' original work and has not been published, submitted or uploaded elsewhere in its present form to any journal or website. Besides, all authors have checked the manuscript and have agreed to the submission.

Sincerely, Yisel Garí et al.

Dear Editor:

We the authors hereby declare the roles played in preparing this paper:

- Yisel Garí (Writing - Original Draft, Writing - Review & Editing, Investigation, Funding acquisition)
- David Monge (Writing - Review & Editing, Investigation, Conceptualization)
- Cristian Mateos (Writing - Review & Editing, Supervision, Funding acquisition)

Sincerely, Yisel Garí et al. (on behalf of all authors)

Change Log: Future Generation Computer Systems - Manuscript FGCS-D-21-00508R1

Associate Editor

– Authors' response: Dear Associate Editor, thank you for this new opportunity to revise the paper. We have indeed prepared a revised version considering the below-mentioned concerns.

Reviewer 2

The revised paper is improved in various aspects and the authors addressed most of the reviewers issues.

– Authors' response: We thank the reviewer for his/her encouraging comments and valuable feedback.

(1) A major weakness of this paper is in the reward (and performance evaluation) metric that combines execution times and monetary costs without any normalization. In the review comment response the authors provide some explanation: "normalizing these partial values with any estimated maximum value of makespan and cost respectively makes reward values too small, affecting the learning process". The authors also state that this issue will be part of future experiments, nevertheless I believe that, although extreme value of makespans and monetary costs range are very large, some form of normalization (e.g., average cases) should be used. Similar considerations hold also for the evaluation metric (i.e., aggregateMC).

– Authors' response: We thank the reviewer for pointing out the importance of normalization in multi-objective optimization.

We are aware of the issues associated with the combination of various objectives on different scales. For that reason, to circumvent these issues, we decided to use the reward components of makespan and cost using the measurement units of hours and USD.

This decision proved to be a simple way of having the values of the two components with comparable magnitudes (See Figure 1 and paragraph "Reward component distributions"). In addition, we added the delta parameter which serves as a finer weight control of the relative importance of each reward component. To set the proper value of this parameter we conducted a tuning process that allowed finding the proper balance for each workflow.

We think that this is not a general solution and hence this should be addressed more generically. However, it is important to note that in the reviewed literature most of the works use multiple objectives in the reward without any form of normalization [3]. Therefore, the investigation of a more general solution for this issue seems like an attractive future work to focus on.

Reward component distributions

Figure 1 shows for each workflow, the histogram of the makespan and cost components of the total reward for each learning episode. These values were observed for the 30 policies learned by the strategy Prop-ECU (Sub-figure 1a) and Prop-Price (Sub-figure 1b). It can be seen that makespan and cost distributions are quite close, in the sense that in general mean values do not differ much more than twice in the worst case (Sub-figure 1a, workflow SIPHT).

Then, this depicts that the reward function is being driven by both optimization objectives. Unfortunately, we can't include these figures in the paper due to page restrictions. Even though it is known that these distributions can variate with a different workflow or a new price configuration, it is possible to adjust the reward components with a delta value specifically computed for each case.

Meanwhile, in the reviewed literature, most of the works use multiple objectives in the reward without any form of normalization. [3]. Anyway, as we indicated above, we are nevertheless interested in investigating a more general solution to this issue in the future.

Aggregate metric

Regarding the aggregate metric used for evaluation, as it is not part of the learning process, it is true that it could be computed using normalized values. This is because after all simulations are done it is possible to normalize makespan and cost values with the limit values observed per workflow.

We thank the reviewer for making us aware of it, but at this point this option involves recomputing/updating all the results and analysis of the paper. Given that in the experiments reported –which use state-of-the-art workflows– makespan and cost values have comparable magnitudes, we decided to keep the current definition of the metric and we will take not only metric but more importantly reward normalization into account for a follow-up big set of experiments.

---

(2) There are some minor issues:
(a) Math formulas should not be justified by adding extra spaces

---

– Authors' response: Fixed (Page 6).

(b) On page 6, right column, remove the extra spacers in "no CPUs available , under-provisioned,"

– Authors' response: Fixed.

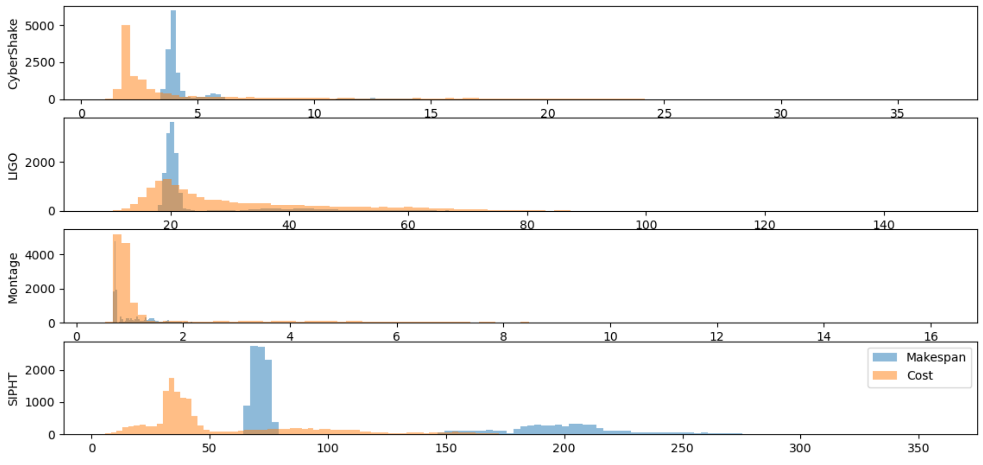
(c) On page 11, left column, "We used a version of the workflows comprising 100 tasks each" should be "We used the versions of the workflows comprising 100 tasks each"

– Authors' response: Fixed.

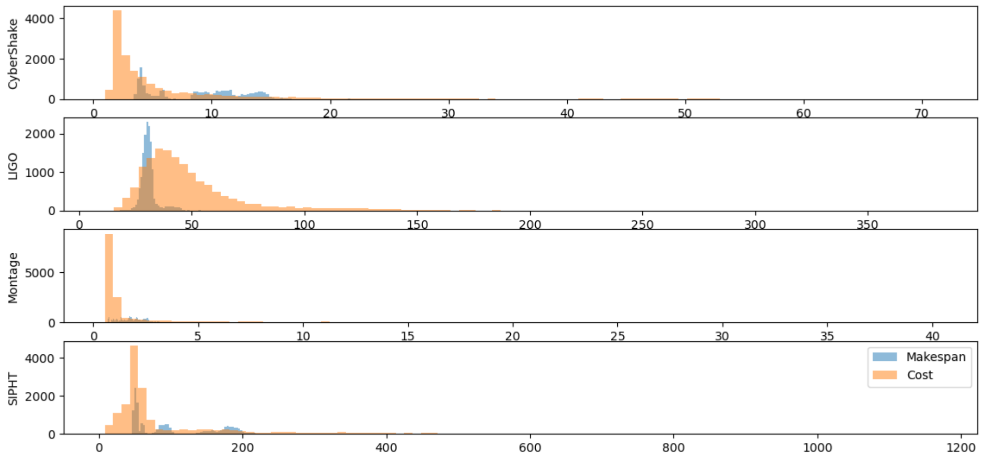
(d) On page 12, left column, remove the extra space in "($\delta \in 0.3, 0.5, 0.7$)"

– Authors' response: Fixed.

(e) In Fig.4 and 5 in the plot x labels, "Aggregate MC" should be "AggregateMC"



(a) Prop-ECU



(b) Prop-Price

Figure 1: Histogram of the makespan and cost components in the total reward observed for each episode.

– Authors' response: Fixed.

(f) In Fig. 6 (and in the text, near the reference) a very short description of the colored bands should be provide (e.g., they represents the 95% CI)

– Authors' response: We now include the following description: Translucent error bands represent the 95% confidence interval around the mean.

(g) In Fig. 6 a brief description of what represent the vertical bars should be provided.

– Authors' response: We now include the following description: Discrete error bars represent the 95% confidence interval around the mean.

Reviewer 3

Thanks for the revision and the response to the comments. I think the revised version is now in a much better shape, and I am happy with most of your response and revision.

– Authors' response: We thank the reviewer for his/her encouraging comments and valuable feedback.

---

(1) One minor issue is still related to the evaluation part. The authors responded that the reason they did not consider Min-Min is because the information about task durations is not easy to achieve in practice, and their method did not need any performance information. Similar to the use if SIAA (as the state-of-the-art) method for comparsion with RL based methods. It is a valid point. However, the authors also agree to the fact that scientific workflows are generally rerun with different input data due to its experimental nature, which means it is relatively easy to come up with a performance prediction model for scientific workflows (as in many non-RL works mentioned by the authors). Therefore, it is not very convincing to claim the advantage of RL based methods over other (prediction-based/performance model based) non-RL methods. If we can get performance information and achieve better results, why not? The authors need to provide more justifications on this issue.

– Authors' response: That is a very good observation. It is true that after 500 executions it is possible to have execution times that could be used for generating a performance model. However, that approach has some scalability limitations in a real setting compared to our RL-based approach.

In the first place, given that each workflow usually comprises multiple task types according to the intended the application logic (see the task types of each studied workflow in Figure 2), we would need to have a performance model per type of task in contrast with our approach, which requires a single policy (model) per workflow, i.e. irrespective of the type of tasks in the workflow. All in all, the complexity of having to build/learn and possibly to tune [1] multiple performance models is way larger than having to learn a single policy.

In the second place, the negative implications of a performance-model–driven approach clearly surface when new virtual machines are available as it requires that every model is updated accordingly. This has an additional complication, because in order to get good performance models, time estimations need to be related to the task inputs, i.e., parameters and/or (meta-) data. Therefore, the space of task inputs should be properly "sampled" for each new available type of VM, which again, is not scalable in the number of models (task types) as we concluded in our previous work [1]. In other words, every time a new VM type is available, all the existing models should be updated. This implies that, for each performance model, we have enough execution examples of the corresponding type covering its input space.

Note that this discussion is mainly focused on performance models based on machine-learning/statistical methods, which provide an automated approach on the problem. The problem is even worst for approaches involving less automated work like, for example, benchmarking tools, analytical modeling, and the like.

As a summary, our formulation avoids both problems by using the information in the structure of dependencies instead of using the performance data of tasks.

Unfortunately, we can't include all arguments in the paper due to page restrictions. Then, we included the main idea in the last paragraph of Section 1.4.2 named SIAA, as follows: "It is worth pointing out that the use of strategies based on performance information is an assumption that restricts its application in real environments, because of the complexity of having to build/learn and possibly to tune [1] multiple performance models (i.e. one per workflow task type)."

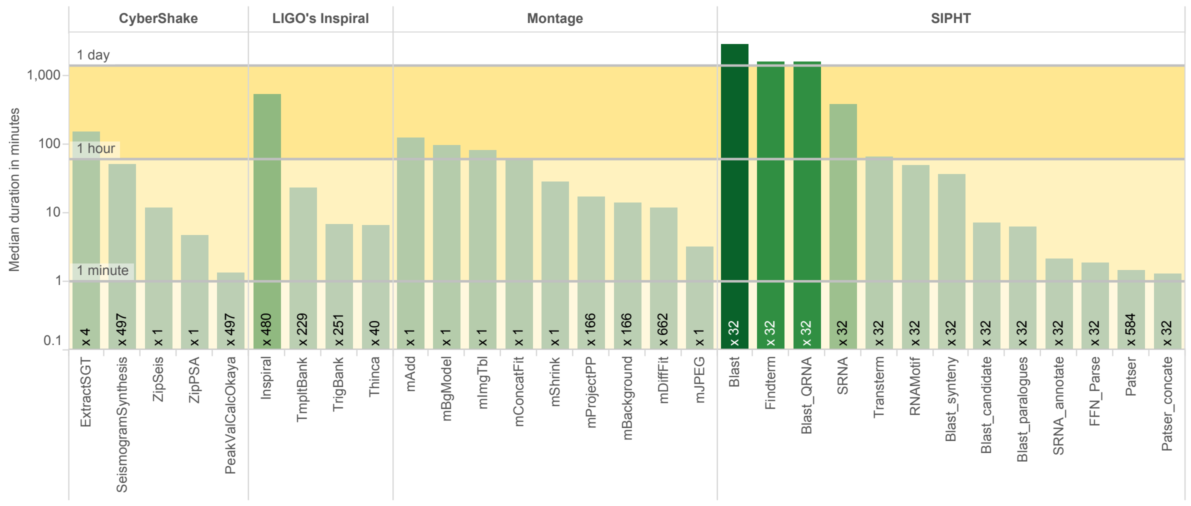


Figure 2: Tasks profile of the studied workflows in a version of 1000 tasks each. Labels on each bar represent the number of tasks for each task type (adapted from the original SIAA paper [2])

References

[1] David A. Monge, Matěj Holec, Filip Železný, and Carlos García Garino. Ensemble learning of runtime prediction models for gene-expression analysis workflows. *Cluster Computing 18(4)*, 13171329, 2015.

[2] David A. Monge, Yisel Garí, Cristian Mateos, and Carlos García Garino. Autoscaling scientific workflows on the cloud by combining on-demand and spot instances. *Journal of Computer Systems Science and Engineering*, 32(4), 2017.

[3] Yisel Garí, David A. Monge, Elina Pacini, Cristian Mateos, and Carlos García Garino. Reinforcement learning-based application autoscaling in the cloud: A survey. *Engineering Applications of Artificial Intelligence*, 102:104288, 2021.

Research highlights:

- A novel MDP formulation for the problem of infrastructure scaling in the Cloud for workflows considering innovative aspects not present in other works.
- Using Q-learning algorithm for learning scaling policies that achieve more efficient workflow executions in terms of makespan and execution cost.
- An in-depth evaluation of the proposed scaling strategy considering 4 well-known workflow applications and a state-of-the-art method as baseline for comparisons.
- Simulation experiments demonstrate significant gains (25%-55%) of our proposal.

A Q-learning Approach for the Autoscaling of Scientific Workflows in the Cloud

Yisel Garí[a,*], David A. Monge[a], Cristian Mateos[b]

[a]ITIC - Universidad Nacional de Cuyo. Mendoza, Argentina
[b]ISISTAN-UNICEN-CONICET. Tandil, Buenos Aires, Argentina

Abstract

Autoscaling strategies aim to exploit the elasticity, resource heterogeneity and varied prices options of a Cloud infrastructure to improve efficiency in the execution of resource-hungry applications such as scientific workflows. Scientific workflows represent a special type of Cloud application with task dependencies, high-performance computational requirements and fluctuating workloads. Hence, the amount and type of resources needed during workflow execution changes dynamically over time. The well-known autoscaling problem comprises *(i)* scaling decisions, for adjusting the computing capacity of a virtualized infrastructure to meet the current demand of the application and *(ii)* task scheduling decisions, for assigning tasks to specific acquired Cloud resources for execution. Both are highly complex sub-problems, even more because of the uncertainty inherent to the Cloud. Reinforcement Learning (RL) provides a solid framework for decision-making problems in stochastic environments. Therefore, RL offers a promising perspective for designing Cloud autoscaling strategies based on an online learning process. In this work, we propose a novel formulation for the problem of infrastructure scaling in the Cloud as a Markov Decision Process, and we use the Q-learning algorithm for learning scaling policies, while demonstrating that considering the specific characteristics of workflow applications when taking autoscaling decisions can lead to more efficient workflow executions. Thus, our RL-based scaling strategy exploits the information available about workflow dependency structures. Simulations performed on four well-known workflows demonstrate significant gains (25%-55%) of our proposal in comparison with a similar state-of-the-art proposal.

Key words: Cloud Computing; Autoscaling; Workflow; Reinforcement Learning

1. Introduction

Workflow technologies have become essential for scientific research since they facilitate in-silico experimentation (i.e., experiments conducted via computer simulations). Scientific workflows specify, in a declarative way, the flow of processing steps (i.e., tasks) involved in a complex experiment. Thus, by adopting workflows, it is possible for scientists from different disciplines to abstract from a great amount of technical computing aspects required for experiment execution, which are out of their area of expertise.

Moreover, the execution of scientific workflows generally demands a large amount of computational resources (i.e., computing power, high-speed networks, storage capacity, etc.) and the workload usually fluctuates over time, determining periods where a greater or lesser capacity is required. The Cloud Computing paradigm offers an excellent opportunity for achieving efficient workflow executions because it enables for reliable and affordable access to a wide spectrum of virtualized resources that can be dynamically scaled up and down as needed. There has been many efforts in developing autoscaling strategies that exploit Cloud elasticity for optimizing the execution of workflows, most of them based on heuristics [1]. Nevertheless, this is a very challenging problem considering that scaling and scheduling decisions need to be wisely taken in this inherently uncertain environment. This uncertainty comes mainly from the unpredictable performance of a Cloud infrastructure due to dynamic changes in the workload of a virtualized infrastructure, which in turn is due to multiple virtual machines competing for the same physical resources.

Reinforcement Learning (RL) is a general pur-

*Corresponding author.
Email address: ygari@uncu.edu.ar (Yisel Garí)

pose framework for decision making that has demonstrated a great potential in complex stochastic environments [2, 3]. RL proposes a goal-directed learning from interaction where an agent can sense the *state* of the environment and select *actions* that influence future observations. Feedback of the chosen actions is given to the agent as a scalar *reward* signal, and the main goal of the agent is maximizing the total reward in the long run. The agent learns a near optimal behavior by sensing the environment and acting on it through a trial and error approach. The mentioned elements -*states*, *actions*, and *rewards*- are the key components in the formulation of any RL problem, and the framework used to define them is called a Markov Decision Process (MDP).

Considering the potential of RL for online learning and dealing with uncertainty, various recent studies address the problem of autoscaling in the Cloud from an RL perspective [4]. However, very few of those works consider workflow applications. Even more importantly, none of the works using workflows are focused on the scaling problem specifically deciding number and type of VMs to acquire, but instead they focus on the scheduling problem (i.e., mapping tasks with already acquired VMs) or they use high-level scaling actions (e.g., percentage of budget used for cheaper but unreliable instances). Then, the nature of the decisions and hence the kind of policies learned in most cases are different in comparison with the policies learned in the present study (See Section 7). Our proposal is focused on learning scaling policies (deciding number and types of VMs to acquire) aiming to optimize workflow executions in terms of makespan -total workflow execution time- and economic cost. To this end, we exploit the information related to the dependency structures in workflow applications, which are an important source of workload variations. It is worth noting that our approach does not require performance information of any type (e.g., the duration of tasks), therefore, it can be more easily ported to actual Cloud environments than strategies that require such information to operate. The specific contributions of this work are listed below:

- A novel MDP formulation for the problem of infrastructure scaling in the Cloud for workflows. This formulation has two innovative aspects not present in other works. These are: *(i)* the consideration of workflow dependency structures for the definition of states and *(ii)* a reward function that considers makespan and execution cost simultaneously.
- An in-depth evaluation of the proposed scaling strategy considering 4 well-known workflow applications and a state-of-the-art method [5] as baseline for comparisons. We selected the strategy as close to ours as possible considering the specific problem addressed and the RL technique used (See Section 4.1.1). We compared them in terms of the performance *(i)* using the same workflows for learning and evaluation *(ii)* and also in their ability to scale to workflows of larger size than those used during learning. This is a desirable ability for the implementation of the techniques in an actual Cloud.

The remainder of this article is organized as follows. First, Section 2 presents the main concepts behind workflows applications (Subsection 2.1) and describes the autoscaling problem in the Cloud (Subsection 2.2). Second, Section 3 summarizes the basics of RL (Subsection 3.1) and describes the proposed RL-based autoscaling approach (Subsection 3.2). Third, Section 4 explains the elements considered for evaluation, namely baseline strategies, complementary scheduling heuristics, evaluation metrics, and experimental settings. Then, Section 5.1 presents and discusses the results obtained. After that, Section 6 discuss the main threats to the validity of our study. Later, Section 7 analyzes the relevant works that apply RL to the problem of autoscaling in the Cloud. Finally, Section 8 concludes this work and highlights future research opportunities.

2. Workflow Autoscaling in the Cloud

In this section we present the background necessary to better understand the rest of this work, in terms of the kind of application and the tackled problem. First, we discuss the distinctive characteristics of workflow applications (Subsection 2.1). Then, we explain the problem of autoscaling in the Cloud (Subsection 2.2).

2.1. Workflow Applications

Workflows technology enable the development of large applications using already defined software components. In general, a workflow outlines a complex objective through the composition of a set of individual *tasks* and their *dependencies*. Specifically, scientific workflows are widely used in the modeling of complex research experiments and they are usually represented as a Directed Acyclic Graph (DAG). Formally, a DAG workflow W is defined as a pair of sets $W = (T, D)$, where T represents the set of tasks and D represents the dependencies between tasks in T.

Bharathi et al. [6] early catalogued different basic structures that can be found in workflows (see Figure 1), and particularly scientific ones. These structures are often repeated multiple times creating more complex patterns especially in workflows composed of hundreds or thousands of tasks. The study [6] is based on well-known scientific workflows (i.e CyberShake, LIGO's Inspiral, Montage and SIPHT). *Process* structure is the simplest one because it only consists of a task that takes an input and produces an output. The *pipeline* structure is quite common in workflows and consist of a task that uses as input the output of the previous task, in turn, each output is the input to the next task. Furthermore, there is the *distribution* structure where the tasks are used for two types of objectives: generate data which is consumed by multiple tasks, or inversely divide a very large data input into smaller chunks to be processed by other subsequent tasks. These types of structures can consume a lot of time and computing resources, but then they usually allow the workflow to reach higher levels of parallelism (in terms of number of parallel workflow tasks). Then, the *aggregation* structure links and processes the outputs of several individual tasks, generating a combined output. These types of tasks also tend to consume a lot of time and resources, and moreover, represent a reduction in the parallelism degree of the workflow. In some cases, the previously aggregated data is then redistributed. This type of *redistribution* structure represents a synchronization point from the perspective of data processing. Although redistribution tasks usually represent potential bottlenecks, parallelism increases again in the next state of the workflow.

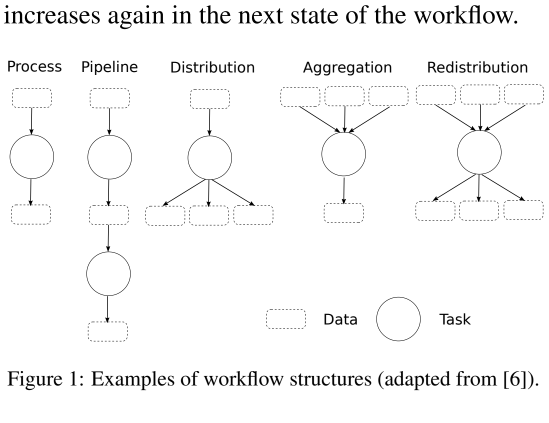


Figure 1: Examples of workflow structures (adapted from [6]).

The execution of scientific workflows, which in general are classified as data-intensive or CPU-intensive, requires a large amount of computational hardware and software resources that include computing power, high-speed networks, storage capacity, techniques and sophisticated administration tools, among others. In addition, as discussed above, the structures that describe the workflow dependencies directly impact the variability of the workload during execution. For example, during workflow execution, periods of potential high parallelism might arise, followed by bottlenecks or long sequential tasks periods, and vice-versa. Such variability determines time windows where an infrastructure with greater or lesser capacity is required, and renting money can be saved. In this context, the elasticity of the Cloud computing model makes it an excellent candidate to meet the computational requirements of this type of applications.

2.2. Autoscaling in the Cloud

Cloud Computing fits efficient workflow execution due to its resources scalability as well as reliability, availability and affordability. Public Cloud providers such as Amazon EC2, Microsoft Azure and Google Cloud Platform facilitate the ubiquitous access to a wide spectrum of robust hardware/software configurations under a pay-per-use scheme.

Particularly, the IaaS Cloud service model, which relies on virtualization technologies, allows users to dynamically create and destroy different types of Virtual Machine (VM) instances. From now on, when we refer to a Cloud infrastructure we are talking about a virtual infrastructure (i.e., a set of VMs). Each VM type represents an specific configuration of virtualized resources (i.e., CPU, RAM, storage, network bandwidth, etc.) and hence offers an expected performance, in terms of storage and processing capacity, with a predefined price. Typically, prices differ according to the computing capacities of each VM type. But also, Cloud providers also offer different price models that determine certain rules in the use and behavior of the instances. For example, Amazon offers on-demand (higher and fixed prices), reserved (in-advance payment, medium prices) and spot instances (fluctuating and lower prices subject to sudden failures).

Moreover several VMs compete for the resources of the physical machine where they coexist, the actual performance of each VM is unpredictable. Performance variability in public, pay-per-use Clouds is a known issue that has been studied in the past and reflected in the literature through benchmarking studies [8, 9]. Variability is regarded as undesirable, because as applications are executed, users might not get what they paid for at the time of renting Cloud resources. Moreover, from the major Cloud providers, Amazon is the one that still varies the most [9]. Indeed, early in 2010, [8] thoroughly analyzed performance variability when using EC2 instances by benchmarking performance metrics such as CPU speed, memory latency, disk read/write latency, network latency, and so on. A major finding is that, compared to a physical cluster used as baseline, performance varied between a minimum of 7-8% and a maximum of 24%. While variability depends on many aspects, including processor type, EC2 availability zone, instance type, weekday, variability values seem to oscillate roughly around 10% and 20%.

Hence, performance variability becomes one of the main sources of uncertainty facing the Cloud Computing model. Then, a Cloud infrastructure can be dynamically adjusted according to the variations in resource demands. Such elasticity is exploited for autoscaling strategies aiming for optimizing workflow executions in terms of several variables (makespan and economic cost for the purposes of this research).

Autoscaling strategies dynamically solve two interdependent problems: infrastructure scaling and task scheduling. The *scaling problem* consists of provisioning or releasing virtualized resources to make an efficient use of the Cloud infrastructure and meet the current computational demand of the application. Scaling can be horizontal, i.e. focused on adjusting the number of VM instances of each type, or vertical, i.e. when the VM resource settings (CPU, memory, I/O) of a single instance are dynamically modified. This work is focused in horizontal scaling. Moreover, the *scheduling problem* consists of assigning tasks for execution in the acquired VM instances. Both sub-problems are NP-hard and, therefore, the solutions proposed to date are mainly based on heuristics and meta-heuristics [1, 10].

Making the most appropriate scaling and scheduling decisions requires that autoscaling strategies run periodically, sensing the state of the infrastructure and the state of the workflow execution. This is due to *(i)* the variable workload patters during the workflow execution which determine dynamic changes in computational requirements and *(ii)* the variable performance of virtualized resources (i.e., CPU, storage, network) in a Cloud Infrastructure, which makes it difficult to determine the best decisions beforehand. Then, in each autoscaling period the strategy aims to optimize the execution based on certain objectives.

The term *makespan* corresponds to the total execution time of the workflow and it depends on the start time and the duration of the tasks involved. Given the set T of tasks of an executed workflow, the makespan value can be computed as:

$$\text{makespan} = \max_{t \in T}\{startT_t + duration_t\} - \min_{t' \in T}\{startT_{t'}\}. \tag{1}$$

Because of the dependencies in a workflow, the start time of each task depends on the completion of all its predecessors. Then, $\forall t \in T, startT_t >= \max_{p \in parents(t)} \{startT_p + duration_p\}$. Also, once a task is ready to run, the actual start time can be delayed if the infrastructure does not have sufficient computing capacity at the moment (i.e scaling issues) or because the scheduling logic decides to do so. Depending on each specific workflow and the particularities of its execution, some tasks can be delayed without affecting the makespan and some other tasks (named critical) needs to be executed as soon as possible. Besides, the duration of a task depends on the computing capabilities of the VM instance to which it was assigned and the current overall load of the Cloud.

The *economic cost* refers to the amount of money spent during the workflow execution in a pay-per-use Cloud infrastructure. The total value depends on the price and the life time (i.e. time since the instance is started until it is finalized) of all the VM instances that were acquired until the workflow completion. Notice that cheaper VMs instances offer less computing capacity, which can usually lead to larger execution times, setting a clear trade-off. Given the set I of VM instances acquired during a workflow execution, the total cost value can be computed as $\sum_{i \in I} price_i \cdot lifeTime_i$.

As can be perceived, workflow autoscaling in the Cloud can be seen as a decision making problem in an stochastic environment. Uncertainty mainly comes, as said, from the performance variability of the Cloud infrastructure. In each autoscaling period, scaling and scheduling actions are taken changing the course of the workflow execution. Thus, to achieve efficient workflow executions, an autoscaling strategy needs to consider the implication of taken decisions in the long term and also being able to deal with the uncertainty inherent to the Cloud.

In a Cloud setting, RL offers a great opportunity to automatically learn near-to-optimal autoscaling policies in an online fashion. Such policies stand out for three important characteristics: they are *transparent* (i.e., independent of human intervention or a deep domain knowledge), *dynamic* (i.e., independent of previously computed static plans) and *adaptable* (i.e., up to date with the possible changes in the dynamic of the environment) [4]. Motivated by this, we address the workflow autoscaling problem in the Cloud with a novel RL-based approach that is presented in the next section.

3. A Reinforcement Learning-based Approach

Next, we present an RL-based autoscaling approach that considers typical workflow characteristics (i.e., dependency structures) aiming to achieve more efficient workflow executions. Although each autoscaling decision comprises both specific scaling and scheduling sub-decisions, in this work we focus in learning intelligent scaling behaviors. Thus, we consider existing heuristics for the scheduling sub-problem in order to properly scope our research. First, we introduce the basics of RL (Subsection 3.1) and then we present our approach for learning scaling policies (Subsection 3.2).

3.1. Theoretical Foundations

Reinforcement Learning (RL) is a subarea of machine learning. It proposes a computational approach that allows an agent to learn a near optimal behavior to achieve a goal by interacting with a stochastic environment. The agent generates its own training data through *trial and error*, thus it learns the consequences of its actions through experience rather than being told the correct actions beforehand. At first the agent does not know how to behave correctly, but it refines its understanding over time.

Markov decision processes (MDP) provide a formal framework for the sequential decision making problems addressed in RL. In other words, the MDP framework is an abstraction of the problem of goal-directed learning from interaction. MDPs uses 3 signals passing back and forth between the agent and its environment: states, actions and reward. Periodically, the agent observes the current state of the environment, and after that it takes one of the possible known actions. Then, the agent gets into a new state receiving a reward for it. The reward is a numerical signal allowing the agent to evaluate the impact of its actions in relation with the goal. The agent is capable of improving its behavior pursuing the maximization of the total reward received over time. Because the current action can influence subsequent states and therefore future rewards, the agent needs to consider the implication of its actions in the long run.

When the agent-environment interaction could break into sub sequences these are called *episodes*. Each episode ends in a terminal state and can be reset to a starting state. In this work, we are dealing with an episodic problem as each workflow execution is performed independently of the others, in other words, each episode corresponds with the execution of a single workflow. An episode begins a new workflow is ready to start executing and it ends when all the workflow tasks have finished.

There are four fundamental elements in an RL learning process. The *policy* $\Pi : S \longrightarrow A$ is the behavior of the agent, mapping each perceived state of the environment with a preferred action. The *reward signal* $R(s, a)$, is a numerical evaluation of the immediate effect for each state-action pair considering the goal in the RL problem faced. The *action-value function* $Q(s, a)$, is the expected gain to be obtained starting from the state s and taking the action a. These values must be estimated and re-estimated from the sequences of observations an agent makes over its entire lifetime and it is usually represented in a tabular form. The *model of the environment*, represent the dynamic that determines the next state and the next reward after taking an action. A model allows learning by considering possible future situations before they are experienced (in offline mode) using model-based methods. In many problems an accurate model is not always available or the dynamic of the environment is prone to change over time. Then, learning is possible by trial and error (in a online mode) using model-free methods.

Temporal-Difference (TD) learning is one of the central ideas in RL [11] and relies on learning a prediction from another prediction. TD methods can learn directly from raw experience without a model of the environment, so they are model-free methods. Moreover, TD methods use the estimated values of successors states for computing the value of the current state, a technique called bootstrapping. The combination of these two features (model-free and bootstrapping) makes TD methods capable of learning in an online fashion. From these, Q-learning is a TD method widely used in RL. The distinctive characteristic of Q-learning is that it updates $Q(s, a)$ considering the action that maximizes the gain for the next step, independent of the policy being followed. Then, the learned action-value function $Q(s, a)$ directly approximates an optimal value function. In this work we used Q-learning for learning scaling policies in the context of workflow executions in the Cloud.

3.2. Learning Scaling Policies

In this section we formalize the scaling sub-problem as an MDP as well as our own approach for learning scaling policies.

An *scaling episode* is the process of completing one single workflow execution through a sequence of scaling actions. An episode begins with all the workflow tasks waiting to be executed, and it ends when all tasks have been finished. Over the course of an episode, a new *scaling step* starts each time the agent has to take a scaling action (i.e., modifying the number of VMs in the virtual infrastructure). Then, the workflow execution progress until reaching the next decision point (e.g., when there are new tasks ready to run).

To achieve efficient workflow executions the agent needs a proper scaling policy Π, which maps each observed state with the preferred scaling action. Besides, considering that the dynamic of the environment is unknown and also it can change over the time, an *online* learning process is suitable to derive and continually update scaling policies.

Considering that the scaling agent has to decide how many VMs to acquire of each type, it may be relevant to know the relation between the capacity of the Cloud infrastructure and the current workload. Moreover, distinct VM types offer different computational capacity and price options. In this sense it may be convenient for the agent to decide to envision the upcoming workload in the near future (i.e, parallel or sequential workflow stage). Different types of workloads could lean the agent towards acquiring one type of VM or another.

States. To characterize the environment we observe the current infrastructure load as well as the task workload associated with each dependency structure type. We consider the dependency structures analyzed in Section 2.1: *pipeline*, *aggregation* (from now referred as *join*) and *distribution* (from now referred as *fork*). The *redistribution* structure can be seen as a composition of a fork and a joint structure, then we are covering all the structures in [6]. The infrastructure load represents the relation between the amount of current tasks (i.e., running or ready) and the total amount of virtual CPUs in the current Cloud infrastructure. Notice that, during each episode (i.e., a workflow execution), there is always at least one task that is running or ready to run. Features related to the workflow dependency structures represent the proportion of the current tasks involved on each structure. These features give us a sense of the upcoming workload. Given: $T^* \subseteq T$ the set of tasks currently running or ready for execution (T^* is never empty), $F \subseteq T^*$ the set of tasks in a fork structure ($t \in F$ if $|children(t)| > 1$), $J \subseteq T^*$ the set of tasks in a join structure ($t \in J$ if $\exists\, i \in children(t) : |parents(i)| > 1$), $P \subseteq T^*$ the set of tasks in a pipeline structure ($t \in P$ if $|children(t)| = 1, t \notin J$) and $CPUs$ the total number of virtual CPUs in the current Cloud infrastructure.

The state space is defined as the 4-tuple $(infrLoad, forkLoad, joinLoad, pipelineLoad)$, where:

- $infrLoad = \begin{cases} |T^*|/CPUs, & \text{if } CPUs > 0 \\ -1, & \text{otherwise} \end{cases}$
- $forkLoad = |F|/|T^*|$,
- $joinLoad = |J|/|T^*|$ and
- $pipelineLoad = |P|/|T^*|$.

Large states and action spaces pose a problem in RL tabular methods, not only because of the memory required for large tables, but also because of the time and data required to fill them accurately [11]. Hence, to manage a simpler state space we discretized the range of the variable $infrLoad$ to four values (equal to -1, below 1, equal to 1, above 1). Note that -1 represents the situation in which the load is indefinite as the number of CPUs is 0. These represent four relevant points to characterize infrastructure load (no CPUs available, under-provisioned, sufficient capacity and over-provisioned respectively). This kind of discretization is common in the proposals of the state-of-the-art [4].

In a similar way, we discretized the ranges of the variables $forkLoad$, $joinLoad$, and $pipelineLoad$ to four values also (equal to 0, up to 1/3, up to 2/3, up to 1). Then, four variables with four possible values each, configure a state space comprising 256 possible states.

Actions. A scaling action can be represented as $a = (VMType_1, VMType_2, ..., VMType_N)$ where $\forall i \in [1, N]$, $VMType_i$ is the number of VMs of type i that will be acquired for the next execution period. Even considering a limited number of VMs, the actions space comprise all possible value combinations. Then, we divide this high-level action in a set of individual scaling sub-actions. In a Cloud environment with N VM types, the set of scaling sub-actions is defined as

$A = \{+VMType_1, ..., +VMType_N, Maintain\}$ where $+VMType_i$ represents the increment of the infrastructure capacity with exactly one VM of type i and *Maintain* is a fictitious action that represents the completion of the scaling action. Note that the scaling action a is implicit in the sequence of selected scaling sub-actions (i.e., the number of $+VMType_i$ sub-actions selected $\forall i \in [1, N]$ before the *Maintain* sub-action is selected). Also, we do not consider actions that scale down the infrastructure because idle instances are automatically removed after each task finalization event. Thus, we simplify the actions set and hence the learning process.

Reward. The reward is defined as a penalization regarding the duration (i.e execution time) and cost involved in the execution period corresponding to the current scaling step. Specifically, we compute the reward as a linear combination of both components. Formally, the reward in the step i is $reward_i = -(\delta \cdot duration_i + (1 - \delta) \cdot cost_i)$, where $duration_i$ is the duration of the execution period i and $cost_i$ is the cost of maintaining the instances used in the period, and δ can be used to adjust the importance degree of each metric according to the user's needs. Note that the sum of the $duration_i$ and the sum of the $cost_i$ over all the scaling steps are equal to the makespan and the total cost, respectively.

We are interested in obtaining scaling agents able to minimize the objectives of cost and makespan. Therefore, for the reward computation, makespan and cost values are combined using the δ parameter allowing to adjust the impact of each individual component on the reward. Note that determining the proper value of δ is crucial for the agent to learn policies oriented to our goal. This is an important aspect that needs to be addressed for obtaining good quality policies.

Algorithm 1 shows the proposed Q-learning based scaling procedure. First, the Q-Table is initialized with zero for each possible state-action pair. One episode represents the completion of a workflow execution through a sequence of scaling steps. On each step, a scaling sub-action a is selected considering the current state and the ε-greedy policy derived from $Q(s, a)$. The initial allocated resources are zero (i.e. no VMs) upon starting the procedure. Also, we consider a maximum of 60 running instances for the scaling decisions. On the one hand, the ε-greedy policy selects a random action with probability ε allowing the agent to explore new situations. On the other hand, with probability $1 - \varepsilon$, the policy selects the action that maximizes the gain for the next step, exploiting the current knowledge. The ε value is set with an initial value (ε_0) and then, on each episode this value is decreased using exponential decay until reaching a target value close to zero (ε_N) in the N episode. In episode number i the ε value is updated according to the law $\varepsilon_i = \varepsilon_0 \cdot e^{-\lambda \cdot i}$, where λ is a predefined decay constant. In this way, the exploration rate is reduced over time whereas the exploitation rate increases. Taking sub-action a increases the infrastructure by at most one VM. If a corresponds with sub-action Maintain, one high-level scaling action is completed and it is time for tasks scheduling, assigning as much ready tasks as possible to the available instances and continuing with the workflow execution. In case of a Maintain sub-action is chosen, the next state and the reward will be observed as soon as the execution of any running task has been completed, if there is at least one task ready to execute (new decision point). In the other case, (i.e., $+VMType_i$ sub-action) the next state and the reward (equals to zero) will be observed immediately, continuing with the selection of the next scaling sub-action.

Algorithm 1 Q-learning Scaling Procedure

```
1:  procedure QL-SCALE($S, A, R, \gamma, \alpha, \varepsilon$):
2:    Initialize $Q(s, a) = 0 \ \forall s \in S, a \in A$
3:    for each (episode $e$) do
4:      Initialize $s$
5:      repeat:
6:        Select $a$ from $s$ using an $\varepsilon$-greedy policy from Q
7:        $\varepsilon \leftarrow$ ExponentialDecay($\varepsilon, e$)
8:        $I \leftarrow$ instances after take scaling action $a$
9:        $T \leftarrow$ ready tasks
10:       if $a$ is Maintain then Schedule($T$,$I$,$priority$)
11:       Observe $r$,$s'$
12:       $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_a Q(s', a) - Q(s, a)]$
13:       $s \leftarrow s'$
14:     until S is terminal
```

Figure 2 illustrates through a simplified example the scaling actions selected during the learning process. Note that each episode comprise a sequence of high-level scaling actions (i.e number of VMs of each type: Medium and Small), each one determined by a sequence of scaling sub-actions (+VMTypeS, +VMTypeM, Maintain).

Figure 3 shows an example of the state of the environment observed by the agent in a decision point during the execution of a workflow (left side) and the effect of a possible general scaling action in the infrastructure (right side). The status of the

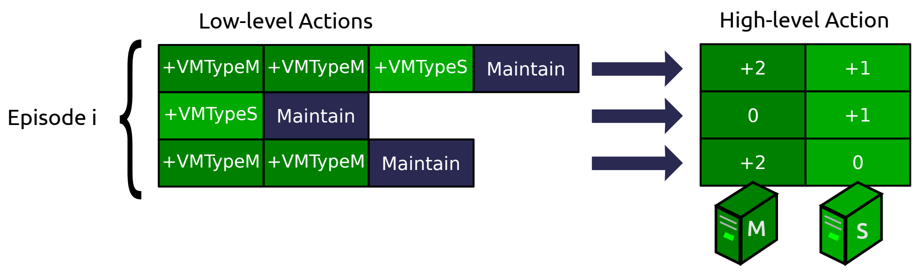


Figure 2: Simplified example of the scaling actions (low-level and high-level) selected on each episode during the learning process.

workflow tasks (finished, running, ready, waiting) were highlighted with different colors. Also, a recently finished task is marked with a red square because it changes the status of three of their dependent tasks (green color is used to represent ready-to-run tasks). At this point the agent must decide a sequence of scaling sub-actions. The Cloud infrastructure is composed by one VM of type Medium (2 virtual CPUs) and one VM of type Small (1 virtual CPU). These VMs are currently running three tasks of the workflow. Then, the agent can observe that the infrastructure load is high (no virtual CPUs available and three tasks ready to execute). Also, the agent observes that all the tasks running or ready belongs to a joint structure and there are no tasks in a fork or pipeline structure. A possibility in the current autoscaling step could be acquiring one more VM of each type (Medium and Small). Considering the corresponding actions, the Cloud infrastructure is scaled up by acquiring 2 extra VMs (3 more available CPUs). Then, ready tasks will be assigned to the available virtual CPUs and they change to the running status.

4. Evaluation

In this section we discuss various aspects considered for evaluating the performance of the proposed strategy. First, we explain two strategies: an RL-based scaling strategy from the state of the art that was selected as baseline for comparisons and a none-RL strategy able to provide good reference values for comparisons. Second, we present three scheduling heuristics that will be combined with the learned scaling policies using our approach to provide a full-fledged autoscaling process. Then, we define the specific variants of the proposal and baseline strategies that will be studied, as well as the metrics used for the performance evaluation of our approach. Finally, a summary of the experimental settings is presented.

Broadly, to evaluate the performance of our proposal we conducted the following studies:

Study A. We explore the impact of different reward configurations in the performance of our proposal, and then we compare different variants of the proposed and baseline RL-based strategies.

Study B. We perform a scalability analysis of the RL-based strategies considering different workflow sizes in the learning process.

Study C. We compare the performance of the RL-based strategies using as reference values the results obtained by a none-RL ideal strategy.

4.1. Scaling strategies

This subsection discuses two state-of-the-art autoscaling strategies. The first one, the work of Wei et al [5], which is also an RL-based strategy, is used as baseline for comparisons. The second one, the Spot instances Aware Autoscaling (SIAA) [1], an heuristic autoscaling strategy with several design advantages, which is used as a reference to a closer-to-ideal approach.

4.1.1. Baseline Strategy: Wei et al.

Considering the RL-based scaling solutions of the state of the art, we selected [5], the strategy closer as possible to our proposal. Major differences between our proposal and other relevant works are discussed in Section 7. The common elements in both proposals are: *(i)* the addressed problem is horizontal scaling in the Cloud, *(ii)* the use of a heterogeneous infrastructure and *(iii)* the online learning process using the Q-learning algorithm. Despite the baseline strategy was designed for Cloud service applications, where tasks have no dependencies to other tasks, we made the adjustments required for workflow applications. In the following paragraphs we specify the performed adjustments.

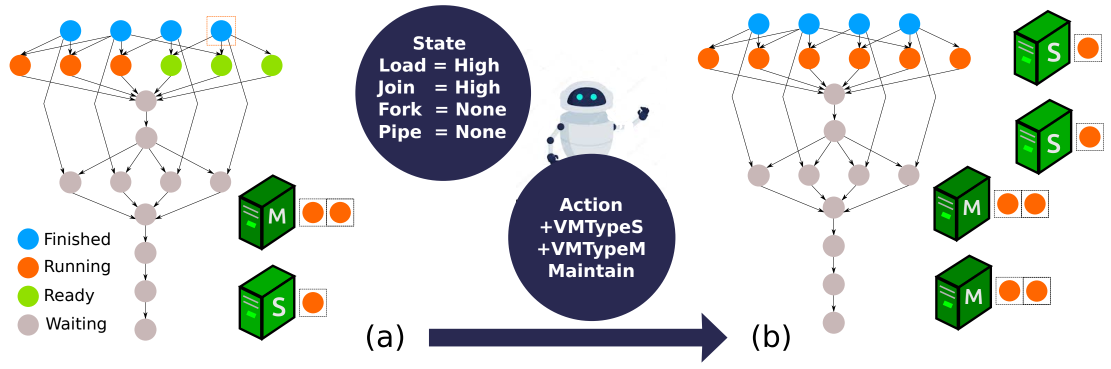


Figure 3: Example of changes in the state of the environment (workflow and infrastructure) in a decision point during the execution. Left side (a) shows what the agent can perceive and right side (b) shows the effect of a possible general scaling action.

States. In the characterization of the environment, the authors used the following features [5]:

CustomerWorkload: The average customer workload in the last decision period. This feature is equivalent to the infrastructure load in our context. Note that in workflow application the workload is determined by the current tasks (i.e., the tasks ready to execute and the tasks already running). Then, we use the $infrLoad$ variable defined in Section 3.2 which represents the ratio between the number of current tasks and the number of CPUs in the current Cloud infrastructure.

VmInstances: The number of VMs instances of each type rented at present (with a maximum of 28 rented VM instances for each type). Then, we use the variable $instances = (VMType_1, VMType_2, ..., VMType_N)$ where $\forall i \in [1, N]$, $VMType_i$ is the number of on-demand VMs of type i in the current infrastructure.

Time: A timestamp within the workload period, since in Cloud service applications, customer workloads normally have certain regularity within a long period. This characteristic does not apply to our context because the workload patterns in the execution of the workflows are not related to time but to the structure of the workflow. Then, we exclude this feature from the state definition.

Actions. Actions decide the number of VMs in a Cloud environment with N types of VMs and two pricing models (on-demand and reserved instances). Considering only on-demand instances, as in the present study, a scaling action can be represented as $a = (VMType_1, VMType_2, ..., VMType_N)$ where $\forall i \in [1, N]$, $VMType_i$ is the number of on-demand VMs of type i that will be acquired for the next execution period. A limitation of this scheme is that the number of actions depends on the maximum number of VMs that can be added on each time step.

Reward. In [5], the reward is computed based on two metrics: *(i)* the profit that SaaS provider earns by providing service to end-users, and *(ii)* the performance (the gain of application performance), which depends on the resource utilization levels. If a SaaS provider owns sufficient VM instances to execute customer workloads, a positive reward will be received. In contrast, a penalty will be caused if application processing capacity is lower than the customer's demands. The value of reward or penalty is related to the distance between the offered processing capacity and the real customer workload. The idea of maximizing the profit for SaaS providers is equivalent to the idea of minimizing cost in our case, and then we replaced the profit component in the reward by a penalty for the cost involved. Also, we adjusted the performance computation formula by considering the workload relative to the workflow execution. Formally, the reward in the step i is $reward_i = -cost_i + perf_i$, where $cost_i$ is the economic cost of the instances used in period i,

$$perf_i = \begin{cases} \beta \cdot workload_i, & \text{if } workload_i \leq 1 \\ P \cdot workload_i, & \text{otherwise} \end{cases}$$

is a measure of the execution performance in this period and $workload_i$ is the workload resulting right after taking the scaling action i and it is computed as the ratio between the amount of tasks (i.e., ready and running) and the amount of CPUs in the infrastructure. The values of the bonus β (2.0) and the penalty P (-1.0) were set according to the values defined in the original work [5].

4.1.2. SIAA

The Spot instances Aware Autoscaling (SIAA) [1] is a heuristic strategy for autoscaling workflows in the Cloud. The scaling heuristic adopted by SIAA estimates the future consumption of the tasks to determine the type and number of instances of each type that will be required. To do this, the heuristic relies on a performance model that allows the estimation of the duration of tasks and thereby characterize the future workload. This feature makes it easier for SIAA to obtain scaling plans that better fit to future workload needs.

Undoubtedly, this is an advantage over the RL-based strategies analyzed in this work. The RL-based strategies do not know the duration of tasks, and they only operate based on the actual load and the structure of workflow dependencies.

It is clear that SIAA has design characteristics but mostly assumptions that allows the approach to obtain scaling plans with a better fit to the workload as it considers the future load requirements. For this reason, it is expected that it will obtain better performances than the studied RL-based methods, at the expense of being harder to adopt in practice. Based on this and from the fact that SIAA has also shown better performance compared to other similar heuristics [1] it is natural to consider it as a reference to measure how far the RL methods are from a closer-to-ideal performance.

It is worth pointing out, that the use of strategies based on performance information is an assumption that restricts its application in real environments, because of the complexity of having to build/learn and possibly to tune [26] multiple performance models (i.e. one per workflow task type). In contrast, strategies that do not require performance information (such as those of RL in this work) can be more easily implanted in real Cloud environments. This is one of the main advantages of our approach. Conversely, SIAA is good as a reference point but less convenient for its use in actual Cloud settings.

4.2. Scheduling Options

Along with the scaling decisions it is also fundamental the scheduling of the workflow tasks. In this work we use two basic heuristics for addressing the scheduling sub-problem. Each heuristic is biased towards one of the addressed optimization objectives (i.e., makespan or cost). Also, we use a third heuristic aimed at fulfilling both objectives alike. Algorithm 2 shows the general scheduling procedure. First, available instances are sorted considering a specific priority criteria determined by the selected heuristic. Then, each ready tasks is assigned to the first available instance until there are no more tasks or no more available instances. If an instance runs out of free CPUs, it is considered no longer available. The heuristics share the same general scheduling procedure but they differ in the priority criteria for instance selection.

Best-ECU. The idea behind this heuristic is to prioritize the use of the best performing instances to reduce makespan. Then, Best-ECU generates a scheduling scheme by first allocating the free CPUs of the instances having the best ECU (i.e., instances are sorted by their relative computing power).

Best-Price. The idea behind this heuristic is to make intensive use of the cheapest instances, facilitating the release of the most expensive ones and thus reducing the economic cost. Then, Best-Price generates a scheduling scheme by first allocating the free CPUs of the instances with lowest price (i.e., instances are sorted by price).

Random. The idea behind this heuristic is to explore the space between both extremes (optimizing only makespan with Best-ECU and optimizing only cost with Best-Price). Then, Random generates a scheduling scheme allocating the free CPUs of the available instances selected randomly.

Algorithm 2 General Scheduling Procedure

```
1: procedure SCHEDULE($T, I, priority$):
2:   $sort(I, priority)$
3:   while $size(T) > 0$ and $size(I) > 0$ do
4:     $t \leftarrow$ first task in $T$
5:     $i \leftarrow$ first instance in $I$
6:     Include $(t, i)$ in the scheduling scheme
7:     Remove $t$ from $T$
8:     if $noFreeCPUs(i)$ then Remove $i$ from $I$
```

4.3. Studied Strategies and Performance Metrics

In the present study we evaluate six RL-based autoscaling strategies: three variants of our scaling proposal (Prop-ECU, Prop-Price and Prop-Rand) in comparison with three variants of the scaling approach selected as baseline (Base-ECU, Base-Price and Base-Rand). Each strategy comprises an RL-based scaling approach (proposal or baseline) combined with a scheduling heuristic (Best-ECU, Best-Price, or Random).

To assess the performance of the strategies we measure the balance between the makespan and the economic cost involved in the workflow execution. Then, we use the aggregateMC metric, defined as the L_2 norm of the vector $x^{\pi} = (\text{makespan}^{\pi}, \text{cost}^{\pi})$ which represents the makespan and cost values achieved by the policy π.

4.4. Experimental Settings

To extensively analyze the performance of the studied strategies, we used four well–known scientific workflows benchmarks commonly used in the literature, namely CyberShake, Montage, LIGO's Inspiral and SIPHT. We used the versions of the workflows comprising 100 tasks each. In this work, we are not explicitly modeling data transfer operations, since we consider them as part of the operations carried out by each task. Different tasks types and dependency structures present in these workflows determine different workload patterns that are suitable for assessing the performance of the autoscaling strategies.

We selected two different instance types from the Amazon EC2 service considering the price and the performance options belonging to the us-west (Oregon) region. Notice that we use the EC2 instances only as a reference model for simulations purpose with CloudSim, not in a real EC2 Cloud environment. The instance types are t2.micro (with one vCPU, ECU = 1, 0.013 USD) and c3.2xlarge (with eight vCPU, ECU = 3.5, 0.42 USD). vCPU is the number of virtual CPUs, ECU is the relative performance of one of the CPUs, and the last characteristic denotes the price in US dollars (USD) of one hour of computation. On the one hand, *t2.micro* is a cheap but lower performance VM type. Using an instance of this type implies that it will take a bit longer to complete a single task but it is also an interesting option in terms of cost savings. On the other hand, *c3.2xlarge* is more expensive but exhibits much better performance. Instances of this type allow 8 task executing in parallel and with a faster completion rate, so it could be suitable for moments of higher computational demands. This two options allows us to explore scaling actions with different impact in both optimization directions (i.e., makespan and cost).

For the learning process of each policy we run 500 episodes. Notice that in a real environment the policy is learned online, improving the performance over time as the number of episodes increase. This number of episodes (500) allows us to maintain a manageable total amount of simulations and also represent a considerable number of solutions to explore on each case. We use an ε-greedy policy starting with $\varepsilon_0 = 0.1$ and decreasing its value to $\varepsilon_{500} = 0.001$ over the time with an exponential decay (decay constant $\lambda = -921.034\text{E-}5$ computed for 500 episodes). This strategy determines the exploration/exploitation trade-off during the learning process, reducing the exploration and increasing the exploitation as the number of episodes increases. We use the discount factor $\gamma = 1$, giving equal importance to long and short term rewards (this is a parameter value allowed for episodic problems) and the learning rate $\alpha = 0.5$. We use some of the RL parameters values (i.e., epsilon initial ε and learning rate α) defined in the original study of the strategy selected as a baseline for comparisons [5].

To ensure the statistical robustness of results, 30 policies were learned for each experimental scenario (i.e., strategy and workflow). To represent a more realistic environment considering the performance variability of a Cloud infrastructure (as explained in Section 2.2), task durations were affected by a ± 10% deviation.

Table 1 summarizes the settings used for the experiments that will be discussed in the next section. As we mentioned earlier, to evaluate our proposal we conducted three studies: (A) exploring the impact of different reward configurations in the performance of our proposal and a comparison of different variants of the RL-based strategies, (B) a scalability analysis of the RL-based strategies and (C) a performance comparison of the RL-based strategies in relation with a none-RL ideal solution. These settings give a total of 3120 learned policies, 30 evaluations of SIAA and 1,560,030 workflow simulations using the CloudSim simulator [12].

5. Results and Discussion

This section presents the results obtained by evaluating the performance and the scalability (from a learning perspective) of the proposed strategy. The runnable version of our experiments can be accessed from `https://bitbucket.org/yiselgari/fgcs2021-experiments/src/master/`. The source code is available upon request.

5.1. Analysis of RL Strategies

This subsection focuses on the performance of the studied RL strategies. First, we explore different rewards in our proposal and select the best configuration for each studied workflow. Then, we discuss a performance comparison of the proposed

Table 1: Experimental settings, resulting in 3,120 learned policies, 30 evaluations of SIAA and 1,560,030 workflow simulations.

Setting	Values
Study	**Global Settings**
Workflow	CyberShake, LIGO's Inspiral, Montage and SIPHT
RL Parameters	500 episodes, $\varepsilon = 0.1$, $\gamma = 1$, $\alpha = 0.5$
RL Testing	100 (wf. size)
Repetitions	30
Cloud	±10% perf. variability
Study	**A. Performance**
Scaling	Proposal and Baseline
Scheduling	Best-ECU, Best-Price and Random
Parameter (δ)	$\{0.3, 0.5, 0.7\}$ (proposal strategy)
RL Training	100 (wf. size)
Study	**B. Scalability**
Scaling	Proposal and Baseline
Scheduling	Best-ECU
Parameter (δ)	0.5 (proposal strategy)
RL Training (wf. size)	$\{30, 40, 50, 60, 70, 80, 90, 100\}$
Study	**C. Ideal Reference**
Scaling	Proposal, Baseline, and SIAA
Scheduling	Best-ECU
Parameter (δ)	$\{0.3, 0.5, 0.7\}$ (proposal strategy)
RL Training	100 (wf. size)

and baseline strategies considering multiple scaling policies and four workflow applications. Finally, we compare the learning processes involved in obtaining the scaling policies evaluated.

5.1.1. Study of Reward Configurations

We explore our RL-based scaling strategy considering different rewards to select the best performing configuration for each strategy and workflow. In the reward function of our strategy, the value of the δ parameter determines the balance between the components of the reward relative to each optimization objective (i.e., makespan and cost). In this section we study the impact of varying the value of this parameter ($\delta \in \{0.3, 0.5, 0.7\}$) in the performance of our strategy. Because of the definition of the reward function, a larger value of δ should lean the strategy towards reducing makespan and a lower value of δ should lean the strategy towards reducing cost.

Figure 4 shows a performance comparison of the scaling strategies Prop-ECU, Prop-Rand and Prop-Price considering the mean aggregateMC values of 30 policies learned by each strategy. A reference line marks the minimum value achieved in each case. Even though there is a best configuration for each strategy and workflow, in 7 out of 12 cases, a higher value of δ (i.e., giving more importance to makespan) leads to better results in terms of the aggregated values of makespan and cost. In 3 cases the medium value was selected and only in 2 cases the lowest value was better. This makes sense considering that the execution cost depends on the VM prices and the execution time. Thus, reducing makespan can also lead to cost savings, but this relationship is not direct and the theoretical trade-off between time and cost is still present. Moreover, using the heuristic Best-ECU almost always leads to the minimum aggregated value. In other words, in terms of the aggregateMCmetric and the proposed strategy, the scheduling heuristic Best-ECU performs better in comparison with Random and Best-Price.

5.1.2. Performance Comparisons

Figure 5 shows a performance comparison of the alternatives for each workflow, considering the mean aggregateMC values of 30 policies learned by each strategy. It can be observed that our strategy outperforms the baseline strategy in all cases. In case of Prop-ECU and Base-ECU, both strategies use the heuristic Best-ECU for the sake of scheduling. Then, lower makespan values are expected because of an intensive use of VMs with higher computing capacity. In case of Prop-Rand and Base-Rand, the scheduling heuristic Random, represents a balance between the use of instances with higher computing capacity and lower price. Finally, in case of Prop-Price and Base-Price, the scheduling heuristic Best-Price, prioritize the use of instances with lower price.

We applied the Mann–Whitney U test in order to check the statistical significance of results. This is a non-parametric test designed to verify whether two samples are derived from the same population mean by comparing median values. Table 2 presents the raw and percentual improvements of our strategies with respect to the baseline strategies as well as the results of the *Mann–Whitney U* test (i.e., the U statistic and the p-values). In our case, a result is significant when $p-$value $< 1.0\text{E}{-}05$. We can observe that our strategies full outperformed the competing strategies with a wide margin (from 25% to 55%). Notice that there was only two non-significant improvement, both in LIGO's Inspiral. First, the comparison between Prop-ECU and Base-ECU (15%) and second, the comparison between Prop-Price and Base-Price (5%). In these

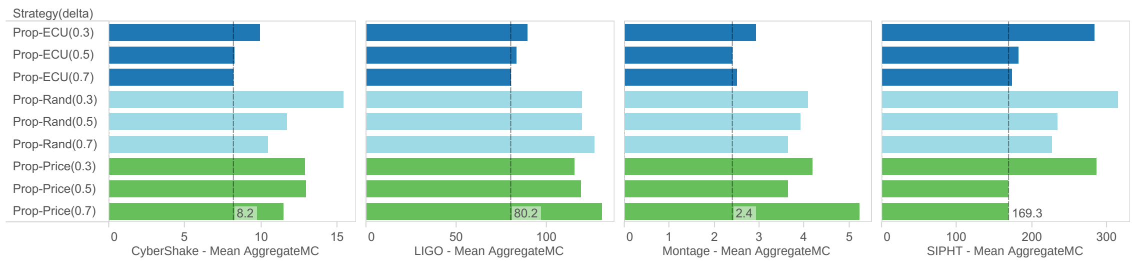


Figure 4: Performance comparisons of the strategies Prop-ECU, Prop-Rand and Prop-Price for each workflow considering different reward configurations ($\delta \in \{0.3, 0.5, 0.7\}$). The reference line marks the minimum value achieved in each case.

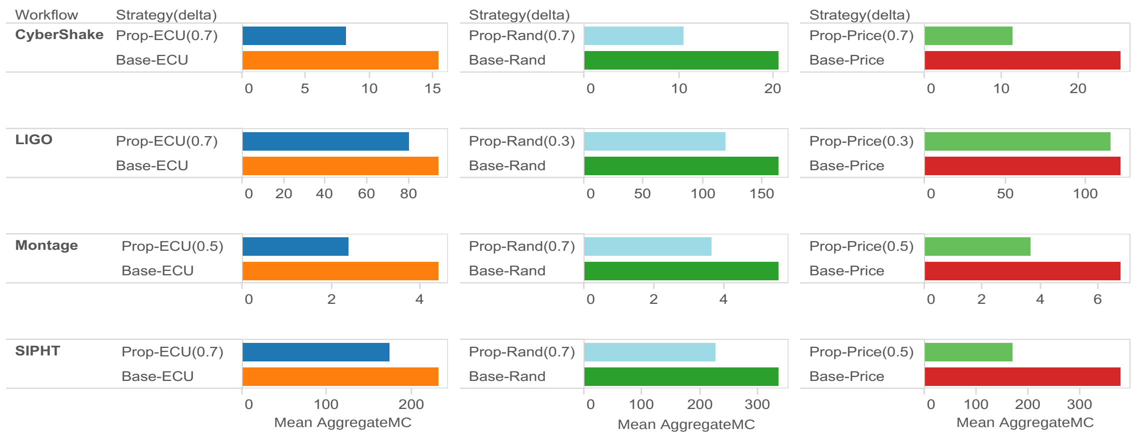


Figure 5: Performance comparisons of the strategies Prop-ECU, Base-ECU, Prop-Rand, Base-Rand Prop-Price and Base-Price, for each workflow considering 30 learned policies.

cases, the baseline strategy performs slightly better in makespan and the proposal performs slightly better in cost, leading to similar results in terms of the aggregateMC metric. Hence, the values of joint gains represented by the aggregated metric were not different enough and deemed not significant.

Table 2: Performance improvements of the proposed strategies in comparison with the corresponding RL baseline strategy (Base-ECU, Base-Price or Base-Rand) considering the aggregateMC metric. Statistically significant results are in bold.

Workflow	Strategy	Improvement
CyberShake	Prop-ECU (0.7)	**47%**
LIGO	Prop-ECU (0.7)	15%
Montage	Prop-ECU (0.5)	**45%**
SIPHT	Prop-ECU (0.7)	**25%**
CyberShake	Prop-Price (0.7)	**55%**
LIGO	Prop-Price (0.3)	5%
Montage	Prop-Price (0.5)	**46%**
SIPHT	Prop-Price (0.5)	**55%**
CyberShake	Prop-Rand (0.7)	**49%**
LIGO	Prop-Rand (0.3)	**27%**
Montage	Prop-Rand (0.7)	**35%**
SIPHT	Prop-Rand (0.7)	**33%**

5.1.3. Learning Curves

In this section we compare the learning process of the studied strategies considering the aggregateMC values. We used the aggregated metric instead of the total reward for two main reasons, *(i)* the reward values of each strategy are not comparable by definition and also *(ii)* in this way, we can observe the evolution along the learning process of each strategy considering the same metric used for performance comparison.

Figure 6 shows the mean aggregateMC value by episode considering 30 policies of the baseline and the proposed strategies for each workflow. Translucent error bands represent the 95% confidence interval around the mean.

First, Figure 6a compares the strategies Prop-ECU and Base-ECU for each workflow. In general, both strategies converge as the number of episodes increases, which means that the agents are learning to reduce both makespan and cost. It can be observed that Prop-ECU tends to perform much worse than Base-ECU at the beginning, but at some point (different for each workflow) Prop-ECU converges to a lower value. The presence of noise in the results is because of the exploration of random actions during the learning process, which is reduced over the time with an exponential decay. First, the agent must try out a variety of actions that leads to different results and progressively favor those that appear to be the most appropriate.

Then, Figure 6b compares the strategies Prop-Price and Base-Price for each workflow. This time, learning curves tend to flatten out faster (taking less than 100 episodes in CyberShakeand Montage). Also, in all workflows, the initial performance of the strategies Base-Price and Prop-Price, is considerable worse (see range of values) in comparison with the initial performance of the strategies Base-ECU and Prop-ECU (see Figure 6a). Finally, in all cases Prop-Price converge to a lower value than Base-Price, which means that our strategy is learning a better-performing scaling policy.

In summary, the performance in all strategies tends to improve as the number of episodes increases (i.e., they are learning). Moreover, proposed strategies (Prop-ECU and Prop-Price) converge to lower values than baseline strategies (Base-ECU and Base-Price). Notice that each strategy (baseline and proposal) presents a different MDP formulation (i.e., states, actions and rewards). Then, each agent takes a distinct path in the learning process that leads to their own preferred scaling policy. Our proposal has a reduced action space, which speeds up the learning process and our reward function represents in a more direct way the optimization objectives (i.e makespan and cost). In addition, although not quite effective as having precise task runtime information beforehand, the information related to the dependency structures gives our agent a better perspective of the type of workload that can be expected in the near future.

5.2. *Scalability Analysis*

Below we evaluate the performance of the studied strategies varying the size of the workflow used in the learning process, and then applying the learned policies on larger workflows for which an individual policy is not learnt. We explore 8 workflow sizes (in the range of 30-100 tasks per workflow), and learn 30 policies for each strategy and workflow size. Then, we evaluate the policies learnt using the largest workflow application (i.e., 100 tasks). To reduce the number of simulations, we select one variant for each the proposal (Prop-ECU(0.5)) and the baseline (Base-ECU) strategies. These settings give a total of 1920 learned policies and 960,000 workflow simulations.

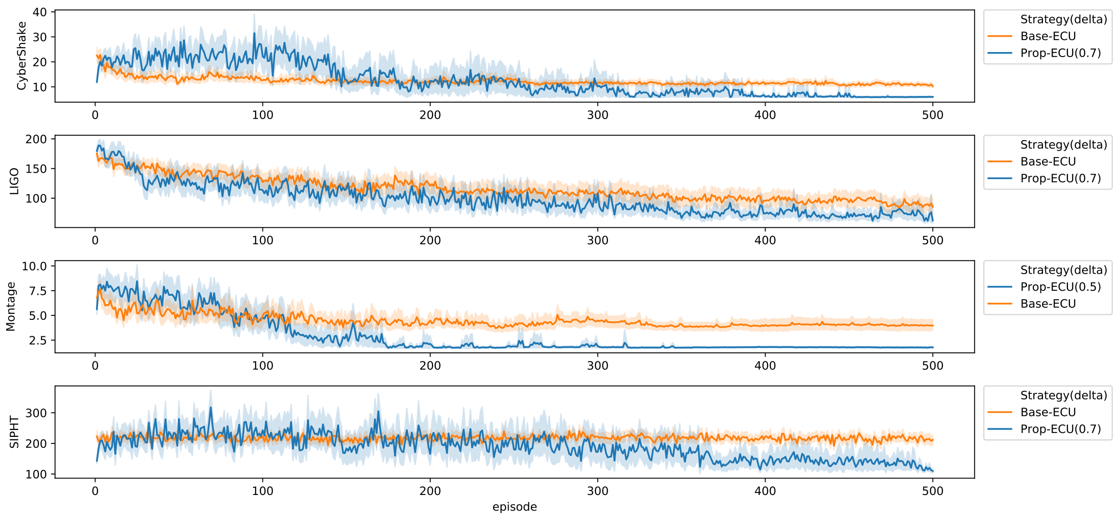
Figure 7 shows a performance comparison of the strategies Prop-ECU and Base-ECU for each workflow, considering different workflow sizes in the learning process. Discrete error bars represent the 95% confidence interval around the mean. In most cases performance tends to improve as workflow size increases, this is due to the circumstances faced by the agent during the learning process are closer to those faced during the evaluation. Also it can be observed that in all cases our strategy full outperformed the competing strategy achieving lower values of the aggregateMC metric and less dispersion in most cases.

5.3. *Comparison with SIAA*

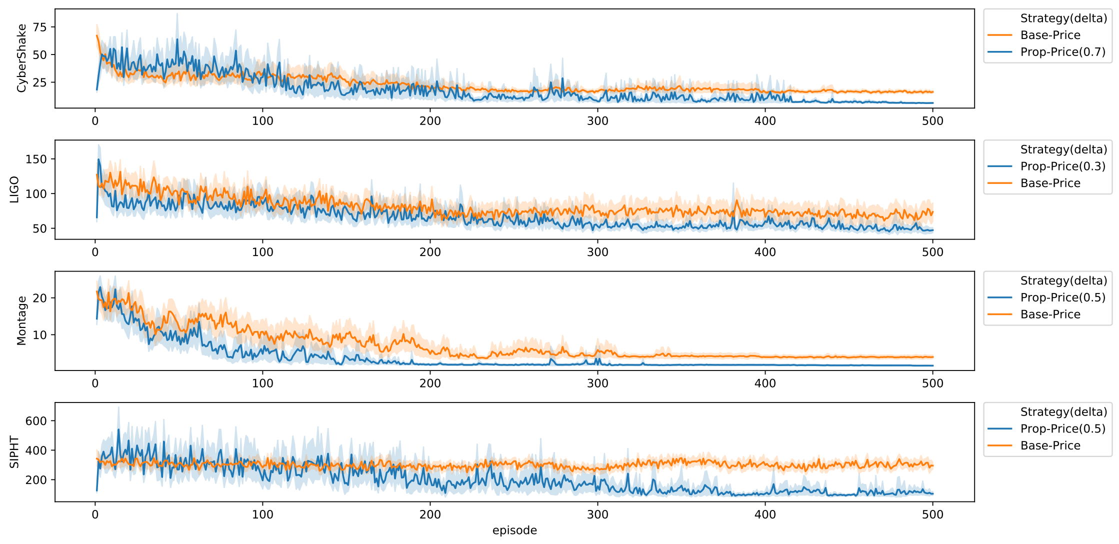
This section studies how the RL-based methods compare against SIAA [1]. As explained in Section 4.1.2, this strategy has some advantages that make it a good reference point as an ideal technique for the purposes of this paper. Because of that reason it is expected that SIAA outperforms the RL-based strategies, but our goal is to explore how close is each RL-based strategy to SIAA. For a meaningful comparison, we analyzed the version of SIAA that uses on-demand instances only.

Figure 8 shows the percentage improvement of the strategies Prop-ECU and Base-ECU in comparison with SIAA, considering the mean aggregateMC values of 30 policies learned by each strategy and workflow. Positive values represent gains and negative values represent losses. We can observe that, SIAA outperformed both RL-based strategies in almost all cases (3/4 workflows). In the case of the baseline strategy, SIAA achieves improvements with margins of 55%, 74%, 108%, and 126% in Montage, SIPHT, LIGO's Inspiral, and CyberShake respectively. In case of our proposal, SIAA achieves margins of 20%, 30% and 77% in CyberShake, SIPHT and LIGO's Inspiral respectively. On the other hand, our proposal outperformed SIAA in Montage with a margin of 16%. Note that in this case, our proposal shows the best behavior among all the techniques (see Figure 6a). The learning curve flattens early in episode 200 and remains almost without variability from episode 300. It is interesting to note that in the case of LIGO's Inspiral and SIPHT (see Figure 6a) the learning curves of the proposal shows more noise in the last episodes in comparison with CyberShake-and Montage. Notice that Base-ECU present wider losses than our proposal in all cases.

These are encouraging results considering that our proposal, unlike the heuristic, does not assume any performance information for decision making. As we mention in section 4.1.2, this kind of as-



(a) Prop-ECU and Base-ECU



(b) Prop-Price and Base-Price

Figure 6: Learning curves of the strategies for each workflow considering the mean of the aggregateMC metric of 30 learned policies. Translucent error bands represent the 95% confidence interval around the mean.

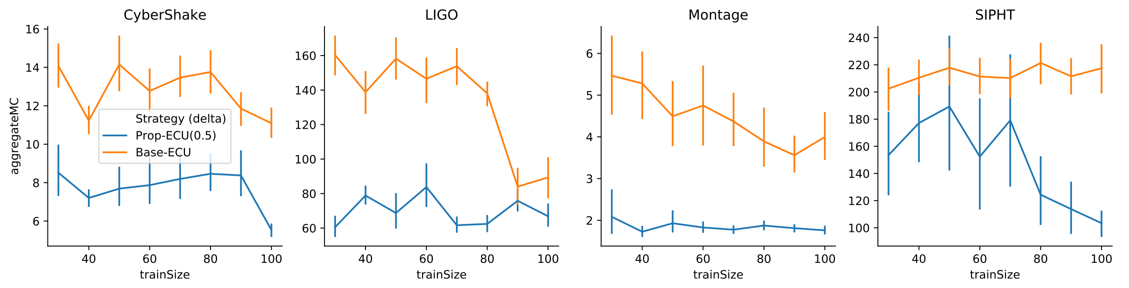


Figure 7: Performance comparison of the strategies Prop-ECU(0.5) and Base-ECU considering different workflow sizes in the learning process. Discrete error bars represent the 95% confidence interval around the mean.

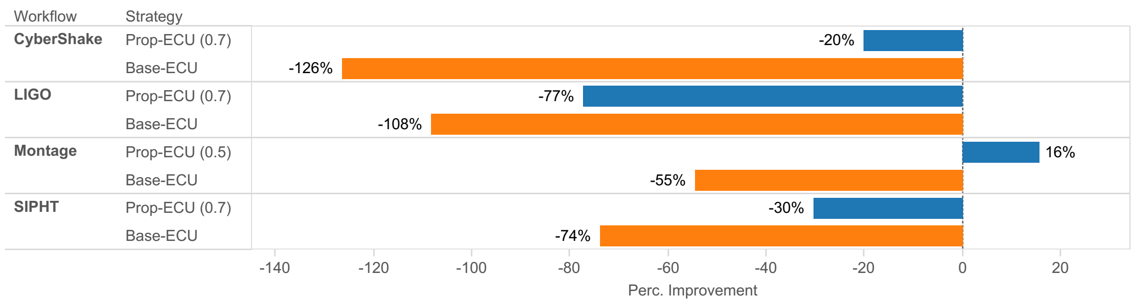


Figure 8: Percentage Improvement of the aggregateMC values for the strategies Prop-ECU and Base-ECU in comparison with SIAA.

sumptions restricts the application of autoscaling strategies in real Cloud environments.

6. Threats to Validity

To minimize the threats to internal validity in this study we used mathematical models known in the literature as well as a simulator widely used in the area. Also, the learning and evaluation of the different policies were repeated numerous times to guarantee the statistical validity of our results.

Regarding threats to external validity, four benchmark workflow applications were considered in this study. Although these applications are well known and widely used in the literature, it is necessary to extend the study to a larger set of applications. Similarly, this study is limited to experimentation with two types of virtual machines. Although this number is consistent with the number of VM types used in other works in the area (up to five VM types at most) [4], it is desirable to extend the spectrum of virtual machine types to improve validation. The extension of the experiment to new types of applications and virtual machines will reduce the impact of these limitations and is something that will be done in future work.

Another important point is the applicability of our method in a real environment. For this, a scalability study was carried out in order to evaluate the ability of the technique to perform against workflow of larger sizes than those used for learning. It was deliberately decided that the performance of the policies be evaluated using the same applications used during the learning in order to limit the scope of this study. But it is necessary to extend the experiments to evaluate the performance of the policies with applications other than those used during the learning in order to further corroborate the generality of the proposal.

7. Related Work

Various works in the literature apply RL to the autoscaling problem in the Cloud and we recently surveyed those proposals from major venues [4]. In a previous work [13] we proposed a model-based approach for learning budget allocation policies for the autoscaling of workflows in the Cloud. Unlike the proposal in the present paper, our previous policies partially solve the scaling problem (i.e., they decide the proportion of the budget to acquire spot and on-demand instance in the next execution period) and the solution depends on a heuristic strategy for completing the scaling decisions (i.e., number and type of VMs to acquire). Another difference with the current approach is that the policies in [13] were derived offline using the Value Iteration algorithm and the data generated by multiple workflow executions. Alternatively, now we are proposing an online learning process using the Q-learning algorithm, which is more suitable for Cloud environments than offline solutions [4].

There are other three approaches [14–16] addressing workflow applications but the RL strategies proposed are focused on scheduling policies (i.e., where and/or when executing tasks). Notice that the nature of the actions is different when learning scaling policies (i.e., number and types of VMs to acquire) as in our current proposal.

Furthermore, Dutreilh and Kirgizov [17] presented VirtRL, a Q-learning approach for scaling Cloud service applications to reduce economic cost while maintaining the Service-Level Agreement (SLA). Unlike our proposal, the authors do not consider different VM types, then, the scaling actions adjust the number of homogeneous VMs allocated to the application. Moreover, Ghobaei-Arani et al. [18] also presented an approach for scaling Cloud service applications based on Q-learning. This strategy aims to achieve a satisfactory compensation between SLA and economic costs. Considering the reward function used, the agent learns to adjust the infrastructure to balance the workload. In the case of workflow applications, focusing only in balancing the workload could limit the possibility of finding solutions with a better trade-off between makespan and cost. Horovitz and Arian [19] proposed Q-Threshold, an innovative Q-Learning approach for scaling Cloud services applications. This threshold-based approach is quite different to our proposal because it comprises higher-level actions with no specific decisions about the amount of instances of each VM type that should be acquired. Finally, we selected the proposal of Wei et al. [5] as a baseline strategy for assessing the performance of our strategy. This is a RL-based scaling approach to help SaaS providers making optimal resource allocation decisions in a Cloud environment. [5] considers an heterogeneous infrastructure with different VM types and price models. The goal of the strategy is to minimize renting expenses while providing sufficient processing capacity to meet customer demands. Also, they use a pure RL solution based on the Q-learning algorithm. Details about the definition of the states, actions and reward were discussed in Section 4.1.1. Both, our proposal and the baseline, formalized the problem of horizontal scaling in a heterogeneous Cloud infrastructure as an MDP and used Q-learning algorithm for learning the scaling policies. The fundamental differences lie in the MDP formulation proposed in each case. First, our strategy takes advantage of a scheme to figure out the upcoming workload for taking decisions, because of the inclusion of information related to the workflow dependency structures in the states definition. Second, our proposal works with reduced state and action spaces to simplify the learning process. And finally, the reward function defined in our proposal consider both makespan and execution cost.

Some other works are focused on vertical scaling [20], accelerating the learning process [21, 22] or combining RL with Neural Networks [16, 23, 24] or Fuzzy Logic [20, 25] (to manage complex state spaces). Our proposal is focused on sequential learning, as we are exploring the impact of other elements in the learning process (i.e., states, actions and reward). Also, we defined a reduced state space by discretizing the values of the variables involved, then we rely in a pure and simpler RL solution.

8. Concluding Remarks

This work addresses the problem of autoscaling scientific workflows in the Cloud to minimize makespan and economic cost. We propose a RL-based scaling method with a novel MDP formulation and we use the Q-learning algorithm for learning scaling policies. One distinctive feature of our proposal is allowing the scaling agent to consider information related to the dependency structures in the workflow. Additionally, we define a reward function that directly corresponds to such objectives. We evaluated the proposed strategy considering three different heuristics for scheduling decisions and a RL baseline strategy from the state of the art closer as possible to our proposal. Also, we used an aggregated metric to measure the balance between makespan and cost achieved by each strategy on four well-known workflow applications.

First, we explored different configurations of our reward function varying the level of importance of each objective and evaluating the impact on the aggregated results. The optimal configuration is specific for each workflow but in many of the cases (7/12) giving more importance to makespan leads to better results. Then, we evaluated multiple policies learned for each workflow and each strategy. Experimental results show that our proposal outperforms the competitor. Our proposal presents significant gains (ranging between 25% and 55%) in comparison with the baseline. This shows that our proposed MDP formulation leads to improved results in performance. We also performed two complementary studies, namely evaluating the scalability of the RL-based strategies and a comparison with a state-of-the-art none-RL strategy. First, the analysis of scalability using different workflow sizes for the learning and the evaluation process, shows that our proposal scale wide better than the baseline strategy for all workflows and almost all workflow size configurations. Second, our proposal, when compared to a high-performance and informed heuristic strategy that assumes task runtime information as input, present losses of less than 31% in 2 workflows, 77% in one workflow (with a difficult learning process) and gains of 16% in another. Whilst baseline strategy present losses in all workflows (ranging between 55% and 126%). These are encouraging results because our proposal, unlike the heuristic, does not consider any performance information for decision making.

To continue improving our proposal, we are interested in extending our study in the following directions. We will study the possibility of learning autoscaling policies completely based on RL (scaling and scheduling decisions). Then, we could fully exploit the benefits related to the online learning process (i.e., policies are transparent, dynamic, and adaptable). Moreover, we will expand our study to exploit more elements of a Cloud environment (e.g., more VM types and price models). This kind of extension could lead to scalability issues related to the dimension of the state and/or the action space. In this sense it could be necessary to explore RL techniques based on function approximations [11], which will allow us to cope with dimensionality limitations. Also, it would be very useful to validate our proposal in a real Cloud environment.

Acknowledgements

We acknowledge the grants by CONICET (PIP 11220170100490CO) and SIIP-UNCuyo (B082).

References

[1] D. A. Monge, Y. Garí, C. Mateos, and C. García Garino. Autoscaling scientific workflows on the cloud by combining on-demand and spot instances. *Journal of Computer Systems Science and Engineering*, 32(4), 2017.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529 EP –, 2015.

[4] Y. Garí, D. A. Monge, E. Pacini, C. Mateos, and C. García Garino. Reinforcement learning-based application autoscaling in the cloud: A survey. *Engineering Applications of Artificial Intelligence*, 102:104288, 2021.

[5] Y. Wei, D. Kudenko, S. Liu, L. Pan, L. Wu, and X. Meng. A reinforcement learning based auto-scaling approach for saas providers in dynamic cloud environment. *Mathematical Problems in Engineering*, pages 1–11, 2019.

[6] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi. Characterization of scientific workflows. *3rd Workshop on Workflows in Support of Large-Scale Science*, pages 1–10, 2008.

[7] D. A. Brown, P. R. Brady, A. Dietz, J. Cao, B. Johnson, and J. McNabb. *A Case Study on the Use of Workflow Technologies for Scientific Analysis: Gravitational Wave Data Analysis*, pages 39–59. Springer London, 2007.

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

[9] J. Ericson, M. Mohammadian, and F. Santana. Analysis of performance variability in public cloud computing. *2017 IEEE International Conference on Information Reuse and Integration*, pages 308–314, 2017.

[10] D. A. Monge, E. Pacini, C. Mateos, E. Alba, and C. García Garino. CMI: An online multi-objective genetic autoscaler for scientific and engineering workflows in cloud infrastructures with unreliable virtual machines. *Journal of Network and Computer Applications*, 149:102464, 2020.

[11] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, USA, 2018.

[12] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya. 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.

[13] Y. Garí, D. A. Monge, C. Mateos, and C. García Garino. Learning budget assignment policies for autoscaling scientific workflows in the cloud. *Cluster Computing*, pages 23:87–105, 2020.

[14] E. Barrett, E. Howley, and J. Duggan. A learning architecture for scheduling workflow applications in the cloud. *9th IEEE European Conference on Web Services*, pages 83–90, 2011.

[15] M. Soualhia, F. Khomh, and S. Tahar. A Dynamic and Failure-aware Task Scheduling Framework for Hadoop. *IEEE Transactions on Cloud Computing*, 8(2):1–16, 2018.

[16] J. L. Mingxi Cheng and S. Nazarian. DRL-Cloud : Deep Reinforcement Learning-Based Resource Provisioning and Task Scheduling for Cloud Service Providers. *23rd Asia and South Pacific Design Automation Conference*, pages 129–134, 2018.

[17] X. Dutreilh and S. Kirgizov. Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow. *7th International Conference on Autonomic and Autonomous Systems*, pages 67–74, 2011.

[18] M. Ghobaei-Arani, S. Jabbehdari, and M. A. Pourmina. An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach. *Future Generation Computer Systems*, 78:191–210, 2018.

[19] S. Horovitz and Y. Arian. Efficient cloud auto-scaling with SLA Objective Using Q-Learning. *6th International Conference on Future Internet of Things and Cloud*, pages 85–92, 2018.

[20] T. Veni and S. M. Saira Bhanu. Auto-scale: automatic scaling of virtualised resources using neuro-fuzzy reinforcement learning approach. *International Journal of Big Data Intelligence*, 3(3), 2016.

[21] J. V. Bibal Benifa and D. Dejey. RLPAS: Reinforcement Learning-based Proactive Auto-Scaler for Resource Provisioning in Cloud Environment. *Mobile Networks and Applications*, pages 1–16, 2018.

[22] S. Mohammad Reza Nouri, H. Li, S. Venugopal, W. Guo, M. He, and W. Tian. Autonomic decentralized elasticity based on a reinforcement learning controller for cloud applications. *Future Generation Computer Systems*, 94:765–780, 2019.

[23] Z. Tong, H. Chen, X. Deng, K. Li, and K. Li. A scheduling scheme in the cloud computing environment using deep Q-learning. *Information Sciences*, 512:1170 – 1191, 2020.

[24] B. Du, C. Wu, and Z. Huang. Learning resource allocation and pricing for cloud profit maximization. *2019 AAAI Conference on Artificial Intelligence*, volume 33, pages 7570–7577, 2019.

[25] H. Arabnejad, C. Pahl, P. Jamshidi, and G. Estrada. A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling. *17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 64–73, 2017.

[26] D. A. Monge, M. Holec, F. Železnỳ, and C. García Garino. Ensemble learning of runtime prediction models for gene-expression analysis workflows. *Cluster Computing 18(4)*, 1317–1329, 2015.

Author biographies:

- Yisel Garí is pursuing a PhD in Computer Science at the UNICEN University since 2016. She has a BSc. in Computer Science from the University of La Habana, Cuba. Her PhD thesis is about autoscaling techniques for Cloud environments.
- David A. Monge received the engineering degree in information systems from Universidad Tecnológica Nacional, Mendoza, Argentina, in 2007 and received the Ph.D. degree in computer science at UNICEN, Tandil, Buenos Aires, Argentina in 2013. He is member of the ITIC Research Institute, Universidad Nacional de Cuyo, Argentina. His research interests include Machine Learning, Optimization, Workflow applications, Grid and Cloud Computing.
- Cristian Mateos received a PhD degree in Computer Science from the UNICEN University in 2008. He is a full-time Adjunct Professor at the UNICEN and member of ISISTAN-CONICET. His main research interest are parallel/distributed programming, Grid/Cloud middlewares and Service-oriented Computing.

Author Photo

Click here to access/download;Author Photo;cmateos.png

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Dear Editor:

We the authors hereby declare the roles played in preparing this paper:

- Yisel Garí (Writing - Original Draft, Writing - Review & Editing, Investigation, Funding acquisition)
- David Monge (Writing - Review & Editing, Investigation, Conceptualization)
- Cristian Mateos (Writing - Review & Editing, Supervision, Funding acquisition)

Sincerely, Yisel Garí et al. (on behalf of all authors)

A Q-learning Approach for the Autoscaling of Scientific Workflows in the Cloud

Yisel Garí[a,*], David A. Monge[a], Cristian Mateos[b]

[a]*ITIC - Universidad Nacional de Cuyo. Mendoza, Argentina*
[b]*ISISTAN-UNICEN-CONICET. Tandil, Buenos Aires, Argentina*

Abstract

Autoscaling strategies aim to exploit the elasticity, resource heterogeneity and varied prices options of a Cloud infrastructure to improve efficiency in the execution of resource-hungry applications such as scientific workflows. Scientific workflows represent a special type of Cloud application with task dependencies, high-performance computational requirements and fluctuating workloads. Hence, the amount and type of resources needed during workflow execution changes dynamically over time. The well-known autoscaling problem comprises *(i)* scaling decisions, for adjusting the computing capacity of a virtualized infrastructure to meet the current demand of the application and *(ii)* task scheduling decisions, for assigning tasks to specific acquired Cloud resources for execution. Both are highly complex sub-problems, even more because of the uncertainty inherent to the Cloud. Reinforcement Learning (RL) provides a solid framework for decision-making problems in stochastic environments. Therefore, RL offers a promising perspective for designing Cloud autoscaling strategies based on an online learning process. In this work, we propose a novel formulation for the problem of infrastructure scaling in the Cloud as a Markov Decision Process, and we use the Q-learning algorithm for learning scaling policies, while demonstrating that considering the specific characteristics of workflow applications when taking autoscaling decisions can lead to more efficient workflow executions. Thus, our RL-based scaling strategy exploits the information available about workflow dependency structures. Simulations performed on four well-known workflows demonstrate significant gains (25%-55%) of our proposal in comparison with a similar state-of-the-art proposal.

Keywords: Cloud Computing; Autoscaling; Workflow; Reinforcement Learning

1. Introduction

Workflow technologies have become essential for scientific research since they facilitate in-silico experimentation (i.e., experiments conducted via computer simulations). Scientific workflows specify, in a declarative way, the flow of processing steps (i.e., tasks) involved in a complex experiment. Thus, by adopting workflows, it is possible for scientists from different disciplines to abstract from a great amount of technical computing aspects required for experiment execution, which are out of their area of expertise.

Moreover, the execution of scientific workflows generally demands a large amount of computational resources (i.e., computing power, high-speed networks, storage capacity, etc.) and the workload usually fluctuates over time, determining periods where a greater or lesser capacity is required. The Cloud Computing paradigm offers an excellent opportunity for achieving efficient workflow executions because it enables for reliable and affordable access to a wide spectrum of virtualized resources that can be dynamically scaled up and down as needed. There has been many efforts in developing autoscaling strategies that exploit Cloud elasticity for optimizing the execution of workflows, most of them based on heuristics [1]. Nevertheless, this is a very challenging problem considering that scaling and scheduling decisions need to be wisely taken in this inherently uncertain environment. This uncertainty comes mainly from the unpredictable performance of a Cloud infrastructure due to dynamic changes in the workload of a virtualized infrastructure, which in turn is due to multiple virtual machines competing for the same physical resources.

Reinforcement Learning (RL) is a general pur-

[*]Corresponding author.
Email address: `ygari@uncu.edu.ar` (Yisel Garí)

pose framework for decision making that has demonstrated a great potential in complex stochastic environments [2, 3]. RL proposes a goal-directed learning from interaction where an agent can sense the *state* of the environment and select *actions* that influence future observations. Feedback of the chosen actions is given to the agent as a scalar *reward* signal, and the main goal of the agent is maximizing the total reward in the long run. The agent learns a near optimal behavior by sensing the environment and acting on it through a trial and error approach. The mentioned elements -*states*, *actions*, and *rewards*- are the key components in the formulation of any RL problem, and the framework used to define them is called a Markov Decision Process (MDP).

Considering the potential of RL for online learning and dealing with uncertainty, various recent studies address the problem of autoscaling in the Cloud from an RL perspective [4]. However, very few of those works consider workflow applications. Even more importantly, none of the works using workflows are focused on the scaling problem specifically deciding number and type of VMs to acquire, but instead they focus on the scheduling problem (i.e., mapping tasks with already acquired VMs) or they use high-level scaling actions (e.g., percentage of budget used for cheaper but unreliable instances). Then, the nature of the decisions and hence the kind of policies learned in most cases are different in comparison with the policies learned in the present study (See Section 7). Our proposal is focused on learning scaling policies (deciding number and types of VMs to acquire) aiming to optimize workflow executions in terms of makespan -total workflow execution time- and economic cost. To this end, we exploit the information related to the dependency structures in workflow applications, which are an important source of workload variations. It is worth noting that our approach does not require performance information of any type (e.g., the duration of tasks), therefore, it can be more easily ported to actual Cloud environments than strategies that require such information to operate. The specific contributions of this work are listed below:

- A novel MDP formulation for the problem of infrastructure scaling in the Cloud for workflows. This formulation has two innovative aspects not present in other works. These are: *(i)* the consideration of workflow dependency structures for the definition of states and *(ii)* a reward function that considers makespan and execution cost simultaneously.
- An in-depth evaluation of the proposed scaling strategy considering 4 well-known workflow applications and a state-of-the-art method [5] as baseline for comparisons. We selected the strategy as close to ours as possible considering the specific problem addressed and the RL technique used (See Section 4.1.1). We compared them in terms of the performance *(i)* using the same workflows for learning and evaluation *(ii)* and also in their ability to scale to workflows of larger size than those used during learning. This is a desirable ability for the implementation of the techniques in an actual Cloud~~environment~~.

The remainder of this article is organized as follows. First, Section 2 presents the main concepts behind workflows applications (Subsection 2.1) and describes the autoscaling problem in the Cloud (Subsection 2.2). Second, Section 3 summarizes the basics of RL (Subsection 3.1) and describes the proposed RL-based autoscaling approach (Subsection 3.2). Third, Section 4 explains the elements considered for evaluation, namely baseline strategies, complementary scheduling heuristics, evaluation metrics, and experimental settings. Then, Section 5.1 presents and discusses the results obtained. After that, Section 6 discuss the main threats to the validity of our study. Later, Section 7 analyzes the relevant works that apply RL to the problem of autoscaling in the Cloud. Finally, Section 8 concludes this work and highlights future research opportunities.

2. Workflow Autoscaling in the Cloud

In this section we present the background necessary to better understand the rest of this work, in terms of the kind of application and the tackled problem. First, we discuss the distinctive characteristics of workflow applications (Subsection 2.1). Then, we explain the problem of autoscaling in the Cloud (Subsection 2.2).

2.1. Workflow Applications

Workflows technology enable the development of large applications using already defined software components. In general, a workflow outlines a complex objective through the composition of a set of individual *tasks* and their *dependencies*. Specifically, scientific workflows are widely used in the modeling of complex research experiments and they are usually represented as a Directed Acyclic Graph (DAG). Formally, a DAG workflow W is defined as a pair of sets $W = (T, D)$, where T represents the set of tasks and D represents the dependencies between tasks in T.

Bharathi et al. [6] early catalogued different basic structures that can be found in workflows (see Figure 1), and particularly scientific ones. These structures are often repeated multiple times creating more complex patterns especially in workflows composed of hundreds or thousands of tasks. The study [6] is based on well-known scientific workflows (i.e CyberShake, LIGO's Inspiral, Montage and SIPHT). *Process* structure is the simplest one because it only consists of a task that takes an input and produces an output. The *pipeline* structure is quite common in workflows and consist of a task that uses as input the output of the previous task, in turn, each output is the input to the next task. Furthermore, there is the *distribution* structure where the tasks are used for two types of objectives: generate data which is consumed by multiple tasks, or inversely divide a very large data input into smaller chunks to be processed by other subsequent tasks. These types of structures can consume a lot of time and computing resources, but then they usually allow the workflow to reach higher levels of parallelism (in terms of number of parallel workflow tasks). Then, the *aggregation* structure links and processes the outputs of several individual tasks, generating a combined output. These types of tasks also tend to consume a lot of time and resources, and moreover, represent a reduction in the parallelism degree of the workflow. In some cases, the previously aggregated data is then redistributed. This type of *redistribution* structure represents a synchronization point from the perspective of data processing. Although redistribution tasks usually represent potential bottlenecks, parallelism increases again in the next state of the workflow.

The execution of scientific workflows, which in general are classified as data-intensive or CPU-intensive, requires a large amount of computational hardware and software resources that include computing power, high-speed networks, storage capacity, techniques and sophisticated administration tools, among others. In addition, as discussed above, the structures that describe the workflow dependencies directly impact the variability of the workload during execution. For example, during workflow execution, periods of potential

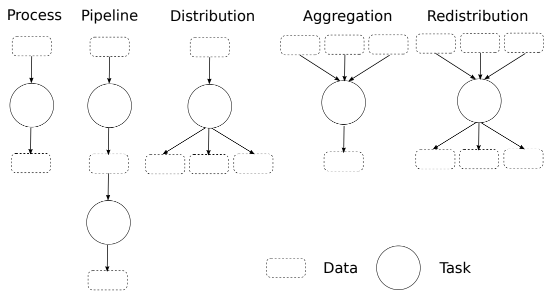


Figure 1: Examples of workflow structures (adapted from [6]).

high parallelism might arise, followed by bottlenecks or long sequential tasks periods, and vice-versa. Such variability determines time windows where an infrastructure with greater or lesser capacity is required, and renting money can be saved. In this context, the elasticity of the Cloud computing model makes it an excellent candidate to meet the computational requirements of this type of applications.

2.2. *Autoscaling in the Cloud*

Cloud Computing fits efficient workflow execution due to its resources scalability as well as reliability, availability and affordability. Public Cloud providers such as Amazon EC2, Microsoft Azure and Google Cloud Platform facilitate the ubiquitous access to a wide spectrum of robust hardware/software configurations under a pay-per-use scheme.

Particularly, the IaaS Cloud service model, which relies on virtualization technologies, allows users to dynamically create and destroy different types of Virtual Machine (VM) instances. From now on, when we refer to a Cloud infrastructure we are talking about a virtual infrastructure (i.e., a set of VMs). Each VM type represents an specific configuration of virtualized resources (i.e., CPU, RAM, storage, network bandwidth, etc.) and hence offers an expected performance, in terms of storage and processing capacity, with a predefined price. Typically, prices differ according to the computing capacities of each VM type. But also, Cloud providers also offer different price models that determine certain rules in the use and behavior of the instances. For example, Amazon offers on-demand (higher and fixed prices), reserved (in-advance payment, medium prices) and spot instances (fluctuating and lower prices subject to ~~possible~~ sudden failures).

Moreover several VMs compete for the resources of the physical machine where they coexist, the actual performance of each VM is unpredictable. Performance variability in public, pay-per-use Clouds is a known issue that has been studied in the past and reflected in the literature through benchmarking studies [8, 9]. Variability is regarded as undesirable, because as applications are executed, users might not get what they paid for at the time of renting Cloud resources. Moreover, from the major Cloud providers, Amazon is the one that still varies the most [9]. Indeed, early in 2010, [8] thoroughly analyzed performance variability when using EC2 instances by benchmarking performance metrics such as CPU speed, memory latency, disk read/write latency, network latency, and so on. A major finding is that, compared to a physical cluster used as baseline, performance varied between a minimum of 7-8% and a maximum of 24%. While variability depends on many aspects, including processor type, EC2 availability zone, instance type, weekday, variability values seem to oscillate roughly around 10% and 20%.

Hence, performance variability becomes one of the main sources of uncertainty facing the Cloud Computing model. Then, a Cloud infrastructure can be dynamically adjusted according to the variations in resource demands. Such elasticity is exploited for autoscaling strategies aiming for optimizing workflow executions in terms of several variables (makespan and economic cost for the purposes of this research).

Autoscaling strategies dynamically solve two interdependent problems: infrastructure scaling and task scheduling. The *scaling problem* consists of provisioning or releasing virtualized resources to make an efficient use of the Cloud infrastructure and meet the current computational demand of the application. Scaling can be horizontal, i.e. focused on adjusting the number of VM instances of each type, or vertical, i.e. when the VM resource settings (CPU, memory, I/O) of a single instance are dynamically modified. This work is focused in horizontal scaling. Moreover, the *scheduling problem* consists of assigning tasks for execution in the acquired VM instances. Both sub-problems are NP-hard and, therefore, the solutions proposed to date are mainly based on heuristics and meta-heuristics [1, 10].

Making the most appropriate scaling and scheduling decisions requires that autoscaling strategies run periodically, sensing the state of the infrastructure and the state of the workflow execution. This is due to *(i)* the variable workload patters during the workflow execution which determine dynamic changes in computational requirements and *(ii)* the variable performance of virtualized resources (i.e., CPU, storage, network) in a Cloud Infrastructure, which makes it difficult to determine the best decisions beforehand. Then, in each autoscaling period the strategy aims to optimize the execution based on certain objectives~~of interest~~.

The term *makespan* corresponds to the total execution time of the workflow and it depends on the start time and the duration of the tasks involved. Given the set T of tasks of an executed workflow, the makespan value can be computed as:

$$\text{makespan} = \max_{t\in T}\{startT_t + duration_t\} - \min_{t'\in T}\{startT_{t'}\}. \quad (1)$$

Because of the dependencies in a workflow, the start time of each task depends on the completion of all its predecessors. Then, $\forall t \in T, startT_t >= \max_{p\in parents(t)}\{startT_p + duration_p\}$. Also, once a task is ready to run, the actual start time can be delayed if the infrastructure does not have sufficient computing capacity at the moment (i.e scaling issues) or because the scheduling logic decides to do so. Depending on each specific workflow and the particularities of its execution, some tasks can be delayed without affecting the makespan and some other tasks (named critical) needs to be executed as soon as possible. Besides, the duration of a task depends on the computing capabilities of the VM instance to which it was assigned and the current overall load of the Cloud.

The *economic cost* refers to the amount of money spent during the workflow execution in a pay-per-use Cloud infrastructure. The total value depends on the price and the life time (i.e. time since the instance is started until it is finalized) of all the VM instances that were acquired until the workflow completion. Notice that cheaper VMs instances offer less computing capacity, which can usually lead to larger execution times, setting a clear trade-off. Given the set I of VM instances acquired during a workflow execution, the total cost value can be computed as $\sum_{i\in I} price_i \cdot lifeTime_i$.

As can be perceived, workflow autoscaling in the Cloud can be seen as a decision making problem in an stochastic environment. Uncertainty mainly comes, as said, from the performance variability of the Cloud infrastructure.

In each autoscaling period, scaling and scheduling actions are taken changing the course of the workflow execution. Thus, to achieve efficient workflow executions, an autoscaling strategy needs to consider the implication of taken decisions in the long term and also being able to deal with the uncertainty inherent to the Cloud.

In a Cloud setting, RL offers a great opportunity to automatically learn near-to-optimal autoscaling policies in an online fashion. Such policies stand out for three important characteristics: they are *transparent* (i.e., independent of human intervention or a deep domain knowledge), *dynamic* (i.e., independent of previously computed static plans) and *adaptable* (i.e., up to date with the possible changes in the dynamic of the environment) [4]. Motivated by this, we address the workflow autoscaling problem in the Cloud with a novel RL-based approach that is presented in the next section.

3. A Reinforcement Learning-based Approach

Next, we present an RL-based autoscaling approach that considers typical workflow characteristics (i.e., dependency structures) aiming to achieve more efficient workflow executions. Although each autoscaling decision comprises both specific scaling and scheduling sub-decisions, in this work we focus in learning intelligent scaling behaviors. Thus, we consider existing heuristics for the scheduling sub-problem in order to properly scope our research. First, we introduce the basics of RL (Subsection 3.1) and then we present our approach for learning scaling policies (Subsection 3.2).

3.1. Theoretical Foundations

Reinforcement Learning (RL) is a subarea of machine learning. It proposes a computational approach that allows an agent to learn a near optimal behavior to achieve a goal by interacting with a stochastic environment. The agent generates its own training data through *trial and error*, thus it learns the consequences of its actions through experience rather than being told the correct actions beforehand. At first the agent does not know how to behave correctly, but it refines its understanding over time.

Markov decision processes (MDP) provide a formal framework for the sequential decision making problems addressed in RL. In other words, the MDP framework is an abstraction of the problem of goal-directed learning from interaction. MDPs uses 3 signals passing back and forth between the agent and its environment: states, actions and reward. Periodically, the agent observes the current state of the environment, and after that it takes one of the possible known actions. Then, the agent gets into a new state receiving a reward for it. The reward is a numerical signal allowing the agent to evaluate the impact of its actions in relation with the goal. The agent is capable of improving its behavior pursuing the maximization of the total reward received over time. Because the current action can influence subsequent states and therefore future rewards, the agent needs to consider the implication of its actions in the long run.

When the agent-environment interaction could break into sub sequences these are called *episodes*. Each episode ends in a terminal state and can be reset to a starting state. In this work, we are dealing with an episodic problem as each workflow execution is performed independently of the others, in other words, each episode corresponds with the execution of a single workflow. An episode begins a new workflow is ready to start executing and it ends when all the workflow tasks have finished.

There are four fundamental elements in an RL learning process. The *policy* $\Pi : S \longrightarrow A$ is the behavior of the agent, mapping each perceived state of the environment with a preferred action. The *reward signal* $R(s, a)$, is a numerical evaluation of the immediate effect for each state-action pair considering the goal in the RL problem faced. The *action-value function* $Q(s, a)$, is the expected gain to be obtained starting from the state s and taking the action a. These values must be estimated and re-estimated from the sequences of observations an agent makes over its entire lifetime and it is usually represented in a tabular form. The *model of the environment*, represent the dynamic that determines the next state and the next reward after taking an action. A model allows learning by considering possible future situations before they are experienced (in offline mode) using model-based methods. In many problems an accurate model is not always available or the dynamic of the environment is prone to change over time. Then, learning is possible by trial and error (in a online mode) using model-free methods.

Temporal-Difference (TD) learning is one of the central ideas in RL [11] and relies on learning a prediction from another prediction. TD methods can learn directly from raw experience without a model of the environment, so they are model-free methods. Moreover, TD methods use the estimated values of successors states for computing the value of the current state, a technique called bootstrapping. The combination of these two features (model-free and bootstrapping) makes TD methods capable of learning in an online fashion. From these, Q-learning is a TD method widely used in RL. The distinctive characteristic of Q-learning is that it updates $Q(s, a)$ considering the action that maximizes the gain for the next step, independent of the policy being followed. Then, the learned action-value function $Q(s, a)$ directly approximates an optimal value function. In this work we used Q-learning for learning scaling policies in the context of workflow executions in the Cloud.

3.2. Learning Scaling Policies

In this section we formalize the scaling subproblem as an MDP as well as our own approach for learning scaling policies.

An *scaling episode* is the process of completing one single workflow execution through a sequence of scaling actions. An episode begins with all the workflow tasks waiting to be executed, and it ends when all tasks have been finished. Over the course of an episode, a new *scaling step* starts each time the agent has to take a scaling action (i.e., modifying the number of VMs in the virtual infrastructure). Then, the workflow execution progress until reaching the next decision point (e.g., when there are new tasks ready to run).

To achieve efficient workflow executions the agent needs a proper scaling policy Π, which maps each observed state with the preferred scaling action. Besides, considering that the dynamic of the environment is unknown and also it can change over the time, an *online* learning process is suitable to derive and continually update scaling policies.

Considering that the scaling agent has to decide how many VMs to acquire of each type, it may be relevant to know the relation between the capacity of the Cloud infrastructure and the current workload. Moreover, distinct VM types offer different computational capacity and price options. In this sense it may be convenient for the agent to decide to envision the upcoming workload in the near future (i.e, parallel or sequential workflow stage). Different types of workloads could lean the agent towards acquiring one type of VM or another.

States. To characterize the environment we observe the current infrastructure load as well as the task workload associated with each dependency structure type. We consider the dependency structures analyzed in Section 2.1: *pipeline*, *aggregation* (from now referred as *join*) and *distribution* (from now referred as *fork*). The *redistribution* structure can be seen as a composition of a fork and a joint structure, then we are covering all the structures in [6]. The infrastructure load represents the relation between the amount of current tasks (i.e., running or ready) and the total amount of virtual CPUs in the current Cloud infrastructure. Notice that, during each episode (i.e., a workflow execution), there is always at least one task that is running or ready to run. Features related to the workflow dependency structures represent the proportion of the current tasks involved on each structure. These features give us a sense of the upcoming workload. Given: $T^* \subseteq T$ the set of tasks currently running or ready for execution (T^* is never empty), $F \subseteq T^*$ the set of tasks in a fork structure ($t \in F$ if $|children(t)| > 1$), $J \subseteq T^*$ the set of tasks in a join structure ($t \in J$ if $\exists\, i \in children(t) : |parents(i)| > 1$), $P \subseteq T^*$ the set of tasks in a pipeline structure ($t \in P$ if $|children(t)| = 1, t \notin J$) and $CPUs$ the total number of virtual CPUs in the current Cloud infrastructure.

The state space is defined as the 4-tuple $(infrLoad, forkLoad, joinLoad, pipelineLoad)$, where:

- $infrLoad = \begin{cases} |T^*|/CPUs, & \text{if } CPUs > 0 \\ -1, & \text{otherwise} \end{cases}$
- $forkLoad = |F|/|T^*|$,
- $joinLoad = |J|/|T^*|$ and
- $pipelineLoad = |P|/|T^*|$.

Large states and action spaces pose a problem in RL tabular methods, not only because of the memory required for large tables, but also because of the time and data required to fill them accurately [11]. Hence, to manage a simpler state space we discretized the range of the variable $infrLoad$ to four values (equal to -1, below 1, equal to 1, above 1). Note that -1 represents the situation in which the load is indefinite as the number of CPUs is 0. These represent four relevant points to characterize infrastructure load (no CPUs available, under-provisioned, sufficient capacity and over-provisioned respectively). This kind of discretization is common in the proposals of the state-of-the-art [4].

In a similar way, we discretized the ranges of the variables $forkLoad$, $joinLoad$, and $pipelineLoad$ to four values also (equal to 0, up to 1/3, up to 2/3, up to 1). Then, four variables with four possible values each, configure a state space comprising 256 possible states.

Actions. A scaling action can be represented as $a = (VMType_1, VMType_2, ..., VMType_N)$ where $\forall i \in [1, N]$, $VMType_i$ is the number of VMs of type i that will be acquired for the next execution period. Even considering a limited number of VMs, the actions space comprise all possible value combinations. Then, we divide this high-level action in a set of individual scaling sub-actions. In a Cloud environment with N VM types, the set of scaling sub-actions is defined as $A = \{+VMType_1, ..., +VMType_N, Maintain\}$ where $+VMType_i$ represents the increment of the infrastructure capacity with exactly one VM of type i and *Maintain* is a fictitious action that represents the completion of the scaling action. Note that the scaling action a is implicit in the sequence of selected scaling sub-actions (i.e., the number of $+VMType_i$ sub-actions selected $\forall i \in [1, N]$ before the *Maintain* sub-action is selected). Also, we do not consider actions that scale down the infrastructure because idle instances are automatically removed after each task finalization event. Thus, we simplify the actions set and hence the learning process.

Reward. The reward is defined as a penalization regarding the duration (i.e execution time) and cost involved in the execution period corresponding to the current scaling step. Specifically, we compute the reward as a linear combination of both components. Formally, the reward in the step i is $reward_i = -(\delta \cdot duration_i + (1 - \delta) \cdot cost_i)$, where $duration_i$ is the duration of the execution period i and $cost_i$ is the cost of maintaining the instances used in the period, and δ can be used to adjust the importance degree of each metric according to the user's needs. Note that the sum of the $duration_i$ and the sum of the $cost_i$ over all the scaling steps are equal to the makespan and the total cost, respectively.

We are interested in obtaining scaling agents able to minimize the objectives of cost and makespan. Therefore, for the reward computation, makespan and cost values are combined using the δ parameter allowing to adjust the impact of each individual component on the reward. Note that determining the proper value of δ is crucial for the agent to learn policies oriented to our goal. This is an important aspect that needs to be addressed for obtaining good quality policies.

Algorithm 1 shows the proposed Q-learning based scaling procedure. First, the Q-Table is initialized with zero for each possible state-action pair. One episode represents the completion of a workflow execution through a sequence of scaling steps. On each step, a scaling sub-action a is selected considering the current state and the ε-greedy policy derived from $Q(s, a)$. The initial allocated resources are zero (i.e. no VMs) upon starting the procedure. Also, we consider a maximum of 60 running instances for the scaling decisions. On the one hand, the ε-greedy policy selects a random action with probability ε allowing the agent to explore new situations. On the other hand, with probability $1 - \varepsilon$, the policy selects the action that maximizes the gain for the next step, exploiting the current knowledge. The ε value is set with an initial value (ε_0) and then, on each episode this value is decreased using exponential decay until reaching a target value close to zero (ε_N) in the N episode. In episode number i the ε value is updated according to the law $\varepsilon_i = \varepsilon_0 \cdot e^{-\lambda \cdot i}$, where λ is a predefined decay constant. In this way, the exploration rate is reduced over time whereas the exploitation rate increases. Taking sub-action a increases the infrastructure by at most one VM. If a corresponds with sub-action Maintain, one high-level scaling action is completed and it is time for tasks scheduling, assigning as much ready tasks as possible to the available instances and continuing with the workflow execution. In case of a Maintain sub-action is chosen, the next state and the reward will be observed as soon as the execution of any running task has been completed, if there is at least one task ready to execute (new decision point). In the other case, (i.e., $+VMType_i$ sub-action) the next state and the reward (equals to zero) will be observed immediately, continuing with the selection of the next scaling sub-action.

Algorithm 1 Q-learning Scaling Procedure

```
 1: procedure QL-SCALE($S, A, R, \gamma, \alpha, \varepsilon$):
 2:   Initialize $Q(s, a) = 0 \; \forall s \in S, a \in A$
 3:   for each (episode $e$) do
 4:     Initialize $s$
 5:     repeat:
 6:       Select $a$ from $s$ using an $\varepsilon$-greedy policy from Q
 7:       $\varepsilon \leftarrow$ ExponentialDecay($\varepsilon, e$)
 8:       $I \leftarrow$ instances after take scaling action $a$
 9:       $T \leftarrow$ ready tasks
10:       if $a$ is Maintain then Schedule($T$,$I$,$priority$)
11:       Observe $r$,$s'$
12:       $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_a Q(s', a) - Q(s, a)]$
13:       $s \leftarrow s'$
14:     until S is terminal
```

Figure 2 illustrates through a simplified example the scaling actions selected during the learning process. Note that each episode comprise a sequence of high-level scaling actions (i.e number of VMs of each type: Medium and Small), each one determined by a sequence of scaling sub-actions (+VMTypeS, +VMTypeM, Maintain).

Figure 3 shows an example of the state of the environment observed by the agent in a decision

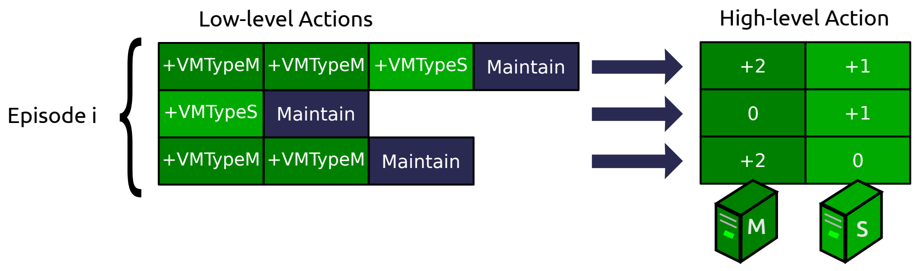


Figure 2: Simplified example of the scaling actions (low-level and high-level) selected on each episode during the learning process.

point during the execution of a workflow (left side) and the effect of a possible general scaling action in the infrastructure (right side). The status of the workflow tasks (finished, running, ready, waiting) were highlighted with different colors. Also, a recently finished task is marked with a red square because it changes the status of three of their dependent tasks (green color is used to represent ready-to-run tasks). At this point the agent must decide a sequence of scaling sub-actions. The Cloud infrastructure is composed by one VM of type Medium (2 virtual CPUs) and one VM of type Small (1 virtual CPU). These VMs are currently running three tasks of the workflow. Then, the agent can observe that the infrastructure load is high (no virtual CPUs available and three tasks ready to execute). Also, the agent observes that all the tasks running or ready belongs to a joint structure and there are no tasks in a fork or pipeline structure. A possibility in the current autoscaling step could be acquiring one more VM of each type (Medium and Small). Considering the corresponding actions, the Cloud infrastructure is scaled up by acquiring 2 extra VMs (3 more available CPUs). Then, ready tasks will be assigned to the available virtual CPUs and they change to the running status.

4. Evaluation

In this section we discuss various aspects considered for evaluating the performance of the proposed strategy. First, we explain two strategies: an RL-based scaling strategy from the state of the art that was selected as baseline for comparisons and a none-RL strategy able to provide good reference values for comparisons. Second, we present three scheduling heuristics that will be combined with the learned scaling policies using our approach to provide a full-fledged autoscaling process. Then, we define the specific variants of the proposal and baseline strategies that will be studied, as well as the metrics used for the performance evaluation of our approach. Finally, a summary of the experimental settings is presented.

Broadly, to evaluate the performance of our proposal we conducted the following studies:

Study A. We explore the impact of different reward configurations in the performance of our proposal, and then we compare different variants of the proposed and baseline RL-based strategies.

Study B. We perform a scalability analysis of the RL-based strategies considering different workflow sizes in the learning process.

Study C. We compare the performance of the RL-based strategies using as reference values the results obtained by a none-RL ideal strategy.

4.1. Scaling strategies

This subsection discuses two state-of-the-art autoscaling strategies. The first one, the work of Wei et al [5], which is also an RL-based strategy, is used as baseline for comparisons. The second one, the Spot instances Aware Autoscaling (SIAA) [1], an heuristic autoscaling strategy with several design advantages, which is used as a reference to a closer-to-ideal approach.

4.1.1. Baseline Strategy: Wei et al.

Considering the RL-based scaling solutions of the state of the art, we selected [5], the strategy closer as possible to our proposal. Major differences between our proposal and other relevant works are discussed in Section 7. The common elements in both proposals are: *(i)* the addressed problem is horizontal scaling in the Cloud, *(ii)* the use of a heterogeneous infrastructure and *(iii)* the online learning process using the Q-learning algorithm. Despite the baseline strategy was designed for Cloud service applications, where tasks have no dependencies to other tasks, we made the adjustments required for workflow applications. In the following paragraphs we specify the performed adjustments.

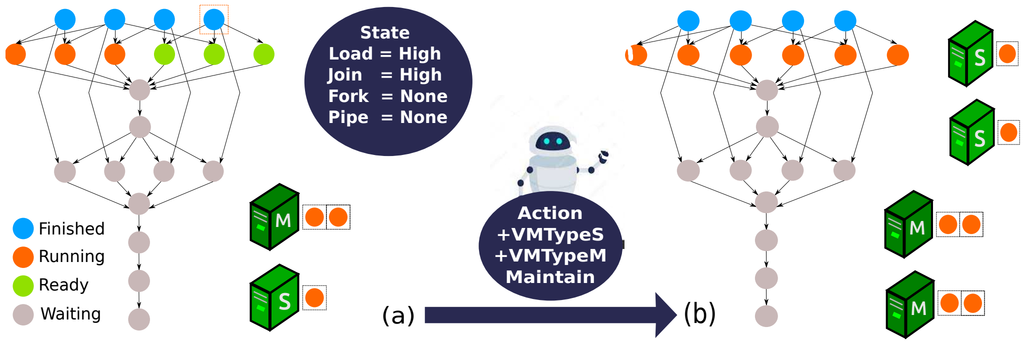


Figure 3: Example of changes in the state of the environment (workflow and infrastructure) in a decision point during the execution. Left side (a) shows what the agent can perceive and right side (b) shows the effect of a possible general scaling action.

States. In the characterization of the environment, the authors used the following features [5]:

CustomerWorkload: The average customer workload in the last decision period. This feature is equivalent to the infrastructure load in our context. Note that in workflow application the workload is determined by the current tasks (i.e., the tasks ready to execute and the tasks already running). Then, we use the $infrLoad$ variable defined in Section 3.2 which represents the ratio between the number of current tasks and the number of CPUs in the current Cloud infrastructure.

VmInstances: The number of VMs instances of each type rented at present (with a maximum of 28 rented VM instances for each type). Then, we use the variable $instances = (VMType_1, VMType_2, ..., VMType_N)$ where $\forall i \in [1, N]$, $VMType_i$ is the number of on-demand VMs of type i in the current infrastructure.

Time: A timestamp within the workload period~~. This is because~~ , since in Cloud service applications, customer workloads normally have ~~some~~ certain regularity within a long period. This characteristic does not apply to our context because the workload patterns in the execution of the workflows are not related to time but to the structure of the workflow. Then, we exclude this feature from the state definition.

Actions. Actions decide the number of VMs in a Cloud environment with N types of VMs and two pricing models (on-demand and reserved instances). Considering only on-demand instances, as in the present study, a scaling action can be represented as $a = (VMType_1, VMType_2, ..., VMType_N)$ where $\forall i \in [1, N]$, $VMType_i$ is the number of on-demand VMs of type i that will be acquired for the next execution period. A limitation of this scheme is that the number of actions depends on the maximum number of VMs that can be added on each time step.

Reward. In [5], the reward is computed based on two metrics: *(i)* the profit that SaaS provider earns by providing service to end-users, and *(ii)* the performance (the gain of application performance), which depends on the resource utilization levels. If a SaaS provider owns sufficient VM instances to execute customer workloads, a positive reward will be received. In contrast, a penalty will be caused if application processing capacity is lower than the customer's demands. The value of reward or penalty is related to the distance between the offered processing capacity and the real customer workload. The idea of maximizing the profit for SaaS providers is equivalent to the idea of minimizing cost in our case, and then we replaced the profit component in the reward by a penalty for the cost involved. Also, we adjusted the performance computation formula by considering the workload relative to the workflow execution. Formally, the reward in the step i is $reward_i = -cost_i + perf_i$, where $cost_i$ is the economic cost of the instances used in period i,

$$perf_i = \begin{cases} \beta \cdot workload_i, & \text{if } workload_i \leq 1 \\ P \cdot workload_i, & \text{otherwise} \end{cases}$$

is a measure of the execution performance in this period and $workload_i$ is the workload resulting right after taking the scaling action i and it is computed as the ratio between the amount of tasks (i.e., ready and running) and the amount of CPUs in the infrastructure. The values of the bonus β (2.0) and the penalty P (-1.0) were set according to the values defined in the original work [5].

4.1.2. SIAA

The Spot instances Aware Autoscaling (SIAA) [1] is a heuristic strategy for autoscaling workflows in the Cloud.

The scaling heuristic adopted by SIAA estimates the future consumption of the tasks to determine the type and number of instances of each type that will be required. To do this, the heuristic relies on a performance model that allows the estimation of the duration of tasks and thereby characterize the future workload. This feature makes it easier for SIAA to obtain scaling plans that better fit to future workload needs.

Undoubtedly, this is an advantage over the RL-based strategies analyzed in this work. The RL-based strategies do not know the duration of tasks, and they only operate based on the actual load and the structure of workflow dependencies.

It is clear that SIAA has design characteristics but mostly assumptions that allows the approach to obtain scaling plans with a better fit to the workload as it considers the future load requirements. For this reason, it is expected that it will obtain better performances than the studied RL-based methods, at the expense of being harder to adopt in practice~~due to the assumptions made~~. Based on this and from the fact that SIAA has also shown better performance compared to other similar heuristics [1] it is natural to consider it as a reference to measure how far the RL methods are from a closer-to-ideal performance.

It is worth pointing out, that the use of strategies based on performance information is an assumption that restricts its application in real environments, because of the complexity of having to build/learn and possibly to tune [26] multiple performance models (i.e. one per workflow task type). In contrast, strategies that do not require performance information (such as those of RL in this work) can be more easily implanted in real Cloud environments. This is one of the main advantages of our approach. Conversely, SIAA is good as a reference point but less convenient for its use in actual Cloud settings.

4.2. Scheduling Options

Along with the scaling decisions it is also fundamental the scheduling of the workflow tasks. In this work we use two basic heuristics for addressing the scheduling sub-problem. Each heuristic is biased towards one of the addressed optimization objectives (i.e., makespan or cost). Also, we use a third heuristic aimed at fulfilling both objectives alike. Algorithm 2 shows the general scheduling procedure. First, available instances are sorted considering a specific priority criteria determined by the selected heuristic. Then, each ready tasks is assigned to the first available instance until there are no more tasks or no more available instances. If an instance runs out of free CPUs, it is considered no longer available. The heuristics share the same general scheduling procedure but they differ in the priority criteria for instance selection.

Best-ECU. The idea behind this heuristic is to prioritize the use of the best performing instances to reduce makespan. Then, Best-ECU generates a scheduling scheme by first allocating the free CPUs of the instances having the best ECU (i.e., instances are sorted by their relative computing power).

Best-Price. The idea behind this heuristic is to make intensive use of the cheapest instances, facilitating the release of the most expensive ones and thus reducing the economic cost. Then, Best-Price generates a scheduling scheme by first allocating the free CPUs of the instances with lowest price (i.e., instances are sorted by price).

Random. The idea behind this heuristic is to explore the space between both extremes (optimizing only makespan with Best-ECU and optimizing only cost with Best-Price). Then, Random generates a scheduling scheme allocating the free CPUs of the available instances selected randomly.

Algorithm 2 General Scheduling Procedure

```
1: procedure SCHEDULE($T, I, priority$):
2:   $sort(I, priority)$
3:   while $size(T) > 0$ and $size(I) > 0$ do
4:     $t \leftarrow$ first task in $T$
5:     $i \leftarrow$ first instance in $I$
6:     Include $(t, i)$ in the scheduling scheme
7:     Remove $t$ from $T$
8:     if $noFreeCPUs(i)$ then Remove $i$ from $I$
```

4.3. Studied Strategies and Performance Metrics

In the present study we evaluate six RL-based autoscaling strategies: three variants of our scaling proposal (Prop-ECU, Prop-Price and Prop-Rand) in comparison with three variants of the scaling approach selected as baseline (Base-ECU, Base-Price and Base-Rand). Each strategy comprises an RL-based scaling approach (proposal or baseline) combined with a scheduling heuristic (Best-ECU, Best-Price, or Random).

To assess the performance of the strategies we measure the balance between the makespan and the economic cost involved in the workflow execution. Then, we use the aggregateMC metric, defined as the L_2 norm of the vector $x^\pi = (\text{makespan}^\pi, \text{cost}^\pi)$ which represents the makespan and cost values achieved by the policy π.

4.4. *Experimental Settings*

To extensively analyze the performance of the studied strategies, we used four well–known scientific workflows benchmarks commonly used in the literature, namely CyberShake, Montage, LIGO's Inspiral and SIPHT. We used ~~a version~~ the versions of the workflows comprising 100 tasks each. In this work, we are not explicitly modeling data transfer operations, since we consider them as part of the operations carried out by each task. Different tasks types and dependency structures present in these workflows determine different workload patterns that are suitable for assessing the performance of the autoscaling strategies.

We selected two different instance types from the Amazon EC2 service considering the price and the performance options belonging to the us-west (Oregon) region. Notice that we use the EC2 instances only as a reference model for simulations purpose with CloudSim, not in a real EC2 Cloud environment. The instance types are t2.micro (with one vCPU, ECU = 1, 0.013 USD) and c3.2xlarge (with eight vCPU, ECU = 3.5, 0.42 USD). vCPU is the number of virtual CPUs, ECU is the relative performance of one of the CPUs, and the last characteristic denotes the price in US dollars (USD) of one hour of computation. On the one hand, *t2.micro* is a cheap but lower performance VM type. Using an instance of this type implies that it will take a bit longer to complete a single task but it is also an interesting option in terms of cost savings. On the other hand, *c3.2xlarge* is more expensive but exhibits much better performance. Instances of this type allow 8 task executing in parallel and with a faster completion rate, so it could be suitable for moments of higher computational demands. This two options allows us to explore scaling actions with different impact in both optimization directions (i.e., makespan and cost).

For the learning process of each policy we run 500 episodes. Notice that in a real environment the policy is learned online, improving the performance over time as the number of episodes increase. This number of episodes (500) allows us to maintain a manageable total amount of simulations and also represent a considerable number of solutions to explore on each case. We use an ε-greedy policy starting with $\varepsilon_0 = 0.1$ and decreasing its value to $\varepsilon_{500} = 0.001$ over the time with an exponential decay (decay constant $\lambda = -921.034\text{E-}5$ computed for 500 episodes). This strategy determines the exploration/exploitation trade-off during the learning process, reducing the exploration and increasing the exploitation as the number of episodes increases. We use the discount factor $\gamma = 1$, giving equal importance to long and short term rewards (this is a parameter value allowed for episodic problems) and the learning rate $\alpha = 0.5$. We use some of the RL parameters values (i.e., epsilon initial ε and learning rate α) defined in the original study of the strategy selected as a baseline for comparisons [5].

To ensure the statistical robustness of results, 30 policies were learned for each experimental scenario (i.e., strategy and workflow). To represent a more realistic environment considering the performance variability of a Cloud infrastructure (as explained in Section 2.2), task durations were affected by a ± 10% deviation.

Table 1 summarizes the settings used for the experiments that will be discussed in the next section. As we mentioned earlier, to evaluate our proposal we conducted three studies: (*A*) exploring the impact of different reward configurations in the performance of our proposal and a comparison of different variants of the RL-based strategies, (*B*) a scalability analysis of the RL-based strategies and (*C*) a performance comparison of the RL-based strategies in relation with a none-RL ideal solution. These settings give a total of 3120 learned policies, 30 evaluations of SIAA and 1,560,030 workflow simulations using the CloudSim simulator [12].

5. Results and Discussion

This section presents the results obtained by evaluating the performance and the scalability (from a learning perspective) of the proposed strategy. The runnable version of our experiments can be accessed from `https://bitbucket.org/yiselgari/fgcs2021-experiments/src/master/`. The source code is available upon request.

5.1. *Analysis of RL Strategies*

This subsection focuses on the performance of the studied RL strategies. First, we explore different rewards in our proposal and select the best configuration for each studied workflow. Then, we

Table 1: Experimental settings, resulting in 3,120 learned policies, 30 evaluations of SIAA and 1,560,030 workflow simulations.

Setting	Values
Study	**Global Settings**
Workflow	CyberShake, LIGO's Inspiral, Montage and SIPHT
RL Parameters	500 episodes, $\varepsilon = 0.1$, $\gamma = 1$, $\alpha = 0.5$
RL Testing	100 (wf. size)
Repetitions	30
Cloud	±10% perf. variability
Study	**A. Performance**
Scaling	Proposal and Baseline
Scheduling	Best-ECU, Best-Price and Random
Parameter (δ)	$\{0.3, 0.5, 0.7\}$ (proposal strategy)
RL Training	100 (wf. size)
Study	**B. Scalability**
Scaling	Proposal and Baseline
Scheduling	Best-ECU
Parameter (δ)	0.5 (proposal strategy)
RL Training (wf. size)	$\{30, 40, 50, 60, 70, 80, 90, 100\}$
Study	**C. Ideal Reference**
Scaling	Proposal, Baseline, and SIAA
Scheduling	Best-ECU
Parameter (δ)	$\{0.3, 0.5, 0.7\}$ (proposal strategy)
RL Training	100 (wf. size)

discuss a performance comparison of the proposed and baseline strategies considering multiple scaling policies and four workflow applications. Finally, we compare the learning processes involved in obtaining the scaling policies evaluated.

5.1.1. *Study of Reward Configurations*

We explore our RL-based scaling strategy considering different rewards to select the best performing configuration for each strategy and workflow. In the reward function of our strategy, the value of the δ parameter determines the balance between the components of the reward relative to each optimization objective (i.e., makespan and cost). In this section we study the impact of varying the value of this parameter (~~$\delta \in 0.3, 0.5, 0.7$~~ $\delta \in \{0.3, 0.5, 0.7\}$) in the performance of our strategy. Because of the definition of the reward function, a larger value of δ should lean the strategy towards reducing makespan and a lower value of δ should lean the strategy towards reducing cost.

Figure 4 shows a performance comparison of the scaling strategies Prop-ECU, Prop-Rand and Prop-Price considering the mean aggregateMC values of 30 policies learned by each strategy. A reference line marks the minimum value achieved in each case. Even though there is a best configuration for each strategy and workflow, in 7 out of 12 cases, a higher value of δ (i.e., giving more importance to makespan) leads to better results in terms of the aggregated values of makespan and cost. In 3 cases the medium value was selected and only in 2 cases the lowest value was better. This makes sense considering that the execution cost depends on the VM prices and the execution time. Thus, reducing makespan can also lead to cost savings, but this relationship is not direct and the theoretical trade-off between time and cost is still present. Moreover, using the heuristic Best-ECU almost always leads to the minimum aggregated value. In other words, in terms of the aggregateMCmetric and the proposed strategy, the scheduling heuristic Best-ECU performs better in comparison with Random and Best-Price.

5.1.2. *Performance Comparisons*

Figure 5 shows a performance comparison of the alternatives for each workflow, considering the mean aggregateMC values of 30 policies learned by each strategy. It can be observed that our strategy outperforms the baseline strategy in all cases. ~~First, in~~ In case of Prop-ECU and Base-ECU, both strategies use the heuristic Best-ECU for the sake of scheduling. Then, lower makespan values are expected because of an intensive use of VMs with higher computing capacity. ~~Then, in~~ In case of Prop-Rand and Base-Rand, the scheduling heuristic Random, represents a balance between the use of instances with higher computing capacity and ~~the use of instances with~~ lower price. Finally, in case of Prop-Price and Base-Price, the scheduling heuristic Best-Price, prioritize the use of instances with lower price.

We applied the Mann–Whitney U test in order to check the statistical significance of results. This is a non-parametric test designed to verify whether two samples are derived from the same population mean by comparing median values. Table 2 presents the raw and percentual improvements of our strategies with respect to the baseline strategies as well as the results of the *Mann–Whitney U* test (i.e., the U statistic and the p-values). In our case, a result is significant when p−value < 1.0E−05. We can observe that our strategies full outperformed the competing strategies with a wide margin (from 25% to 55%). Notice that there was only two non-significant improvement, both in LIGO's Inspiral. First, the comparison between Prop-ECU

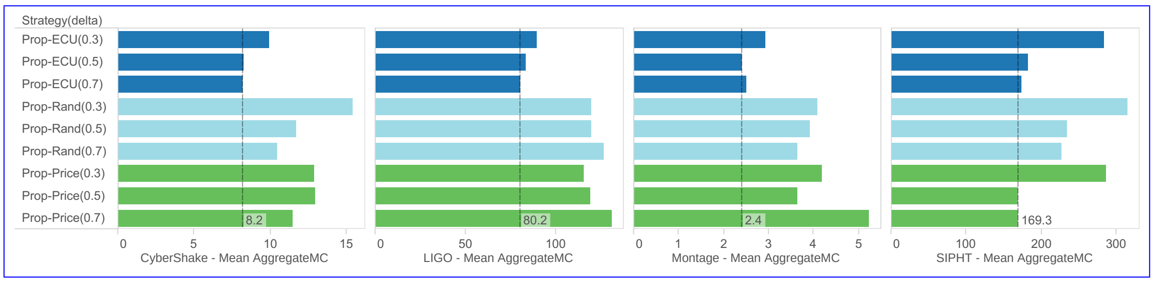


Figure 4: Performance comparisons of the strategies Prop-ECU, Prop-Rand and Prop-Price for each workflow considering different reward configurations ($\delta \in \{0.3, 0.5, 0.7\}$). The reference line marks the minimum value achieved in each case.

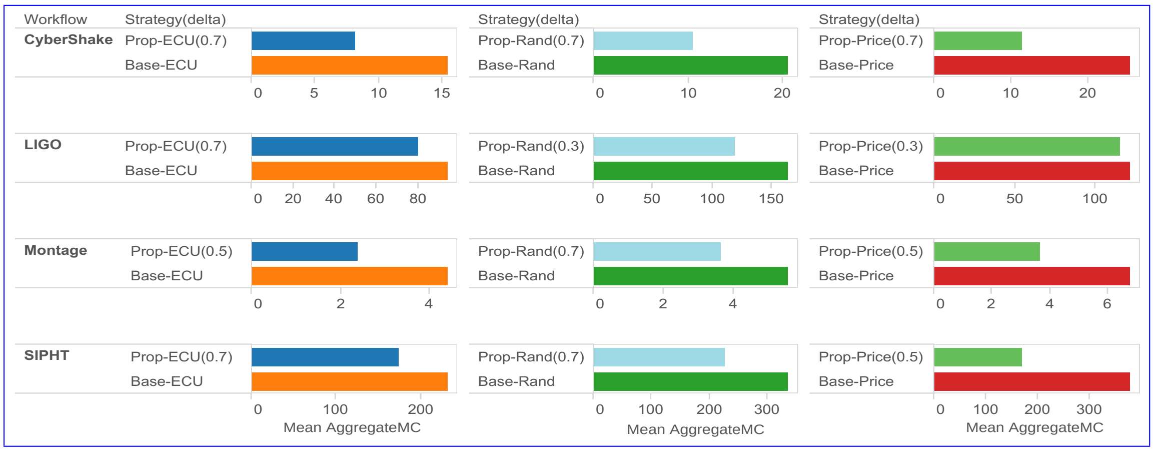


Figure 5: Performance comparisons of the strategies Prop-ECU, Base-ECU, Prop-Rand, Base-Rand Prop-Price and Base-Price, for each workflow considering 30 learned policies.

and Base-ECU (15%) and second, the comparison between Prop-Price and Base-Price (5%). In these cases, the baseline strategy performs slightly better in makespan and the proposal performs slightly better in cost, leading to similar results in terms of the aggregateMC metric. Hence, the values of joint gains represented by the aggregated metric were not different enough and ~~they are considered as~~ deemed not significant.

Table 2: Performance improvements of the proposed strategies in comparison with the corresponding RL baseline strategy (Base-ECU, Base-Price or Base-Rand) considering the aggregateMC metric. Statistically significant results are in bold.

Workflow	Strategy	Improvement
CyberShake	Prop-ECU (0.7)	**47%**
LIGO	Prop-ECU (0.7)	15%
Montage	Prop-ECU (0.5)	**45%**
SIPHT	Prop-ECU (0.7)	**25%**
CyberShake	Prop-Price (0.7)	**55%**
LIGO	Prop-Price (0.3)	5%
Montage	Prop-Price (0.5)	**46%**
SIPHT	Prop-Price (0.5)	**55%**
CyberShake	Prop-Rand (0.7)	**49%**
LIGO	Prop-Rand (0.3)	**27%**
Montage	Prop-Rand (0.7)	**35%**
SIPHT	Prop-Rand (0.7)	**33%**

5.1.3. *Learning Curves*

In this section we compare the learning process of the studied strategies considering the aggregateMC values. We used the aggregated metric instead of the total reward for two main reasons, *(i)* the reward values of each strategy are not comparable by definition and also *(ii)* in this way, we can observe the evolution along the learning process of each strategy considering the same metric used for performance comparison.

Figure 6 shows the mean aggregateMC value by episode considering 30 policies of the baseline and the proposed strategies for each workflow. Translucent error bands represent the 95% confidence interval around the mean.

First, Figure 6a compares the strategies Prop-ECU and Base-ECU for each workflow. In general, both strategies converge as the number of episodes increases, which means that the agents are learning to reduce both makespan and cost. It can be observed that Prop-ECU tends to perform much worse than Base-ECU at the beginning, but at some point (different for each workflow) Prop-ECU converges to a lower value. The presence of noise in the results is because of the exploration of random actions during the learning process, which is reduced over the time with an exponential decay. First, the agent must try out a variety of actions that leads to different results and progressively favor those that appear to be the most appropriate.

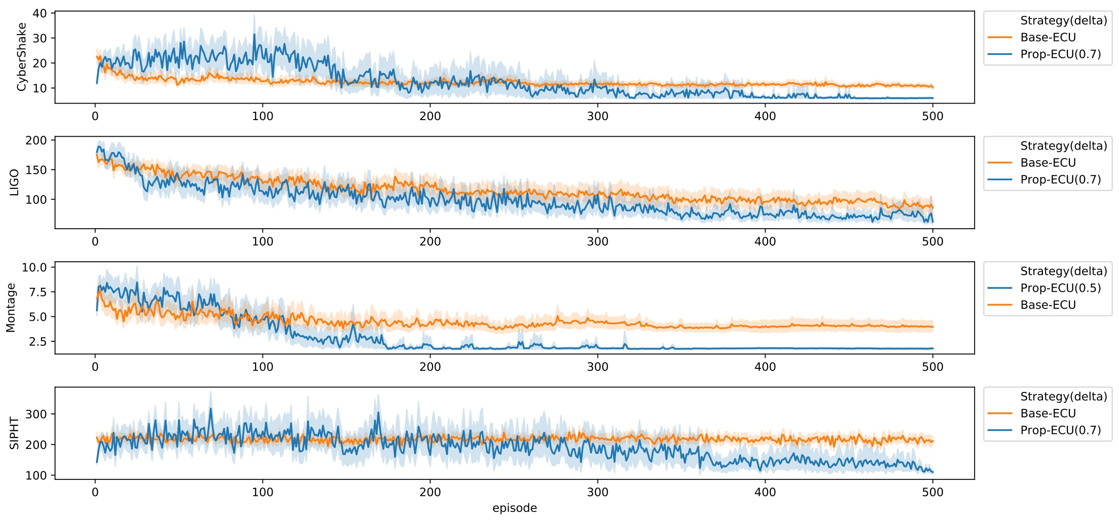
Then, Figure 6b compares the strategies Prop-Price and Base-Price for each workflow. This time, learning curves tend to flatten out faster (taking less than 100 episodes in CyberShakeand Montage). Also, in all workflows, the initial performance of the strategies Base-Price and Prop-Price, is considerable worse (see range of values) in comparison with the initial performance of the strategies Base-ECU and Prop-ECU (see Figure 6a). Finally, in all cases Prop-Price converge to a lower value than Base-Price, which means that our strategy is learning a better-performing scaling policy.

In summary, ~~it can be seen that~~ the performance in all strategies tends to improve as the number of episodes increases (i.e., they are learning). Moreover, proposed strategies (Prop-ECU and Prop-Price) converge to lower values than baseline strategies (Base-ECU and Base-Price). Notice that each strategy (baseline and proposal) presents a different MDP formulation (i.e., states, actions and rewards). Then, each agent takes a distinct path in the learning process that leads to their own preferred scaling policy. Our proposal has a reduced action space, which speeds up the learning process and our reward function represents in a more direct way the optimization objectives (i.e makespan and cost). In addition, although not quite effective as having precise task runtime information beforehand, the information related to the dependency structures gives our agent a better perspective of the type of workload that can be expected in the near future.

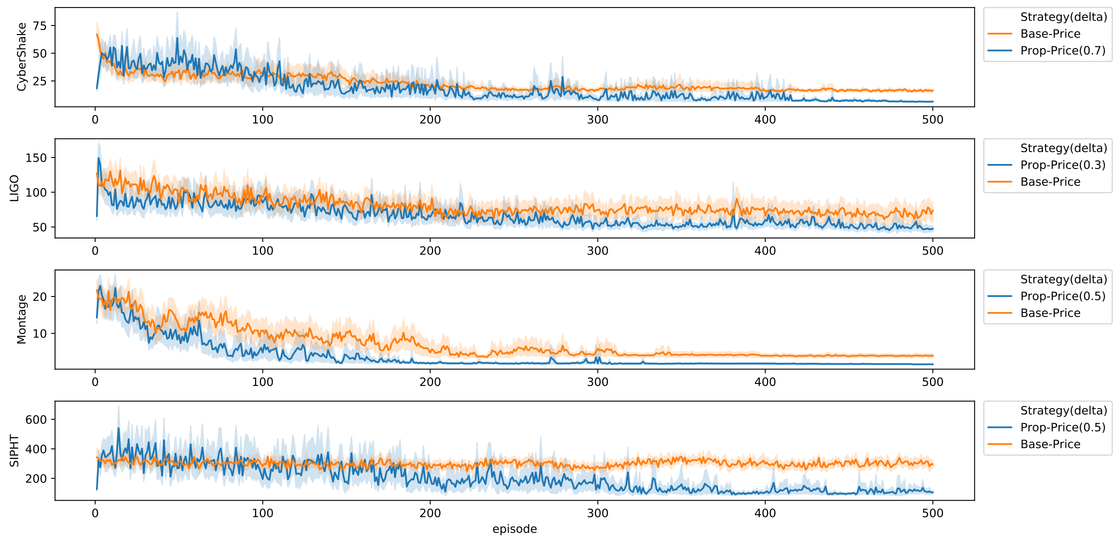
~~Performance comparison of the strategies Prop-ECU(0.5) and Base-ECU for each workflow considering different workflow sizes in the learning process.~~

5.2. *Scalability Analysis*

Below we evaluate the performance of the studied strategies varying the size of the workflow used in the learning process, and then applying the learned policies on larger workflows for which an individual policy is not learnt. We explore 8 workflow sizes (in the range of 30-100 tasks per workflow), and learn 30 policies for each strategy and workflow size. Then, we evaluate the policies learnt using the largest workflow application (i.e., 100 tasks). To reduce the number of simulations,



(a) Prop-ECU and Base-ECU



(b) Prop-Price and Base-Price

Figure 6: Learning curves of the strategies for each workflow considering the mean of the aggregateMC metric of 30 learned policies. Translucent error bands represent the 95% confidence interval around the mean.

we select one variant for each the proposal (Prop-ECU(0.5)) and the baseline (Base-ECU) strategies. These settings give a total of 1920 learned policies and 960,000 workflow simulations.

Figure 7 shows a performance comparison of the strategies Prop-ECU and Base-ECU for each workflow, considering different workflow sizes in the learning process. Discrete error bars represent the 95% confidence interval around the mean. In most cases performance tends to improve as workflow size increases, this is due to the circumstances faced by the agent during the learning process are closer to those faced during the evaluation. Also it can be observed that in all cases our strategy full outperformed the competing strategy achieving lower values of the aggregateMC metric and less dispersion in most cases.

5.3. Comparison with SIAA

This section studies how the RL-based methods compare against SIAA [1]. As explained in Section 4.1.2, this strategy has some advantages that make it a good reference point as an ideal technique for the purposes of this paper. Because of that reason it is expected that SIAA outperforms the RL-based strategies, but our goal is to explore how close is each RL-based strategy to SIAA. For a meaningful comparison, we analyzed the version of SIAA that uses on-demand instances only.

Figure 8 shows the percentage improvement of the strategies Prop-ECU and Base-ECU in comparison with SIAA, considering the mean aggregateMC values of 30 policies learned by each strategy and workflow. Positive values represent gains and negative values represent losses. We can observe that, SIAA outperformed both RL-based strategies in almost all cases (3/4 workflows). In the case of the baseline strategy, SIAA achieves improvements with margins of 55%, 74%, 108%, and 126% in Montage, SIPHT, LIGO's Inspiral, and CyberShake respectively. In case of our proposal, SIAA achieves margins of 20%, 30% and 77% in CyberShake, SIPHT and LIGO's Inspiral respectively. On the other hand, our proposal outperformed SIAA in Montage with a margin of 16%. Note that in this case, our proposal shows the best behavior among all the techniques (see Figure 6a). The learning curve flattens early in episode 200 and remains almost without variability from episode 300. It is interesting to note that in the case of LIGO's Inspiral and SIPHT (see Figure 6a) the learning curves of the proposal shows more noise in the last episodes in comparison with CyberShake- and Montage. Notice that Base-ECU present wider losses than our proposal in all cases~~and not a single case of gain~~.

These are encouraging results considering that our proposal, unlike the heuristic, does not assume any performance information for decision making. As we mention in section 4.1.2, this kind of assumptions restricts the application of autoscaling strategies in real Cloud environments.

6. Threats to Validity

To minimize the threats to internal validity in this study we used mathematical models known in the literature as well as a simulator widely used in the area. Also, the learning and evaluation of the different policies were repeated numerous times to guarantee the statistical validity of our results.

Regarding threats to external validity, four benchmark workflow applications were considered in this study. Although these applications are well known and widely used in the literature, it is necessary to extend the study to a larger set of applications. Similarly, this study is limited to experimentation with two types of virtual machines. Although this number is consistent with the number of VM types used in other works in the area (up to five VM types at most) [4], it is desirable to extend the spectrum of virtual machine types to improve validation. The extension of the experiment to new types of applications and virtual machines will reduce the impact of these limitations and is something that will be done in future work.

Another important point is the applicability of our method in a real environment. For this, a scalability study was carried out in order to evaluate the ability of the technique to perform against workflow of larger sizes than those used for learning. It was deliberately decided that the performance of the policies be evaluated using the same applications used during the learning in order to limit the scope of this study. But it is necessary to extend the experiments to evaluate the performance of the policies with applications other than those used during the learning in order to further corroborate the generality of the proposal.

7. Related Work

Various works in the literature apply RL to the autoscaling problem in the Cloud and we recently surveyed those proposals from major venues [4]. In a previous work [13] we proposed a model-based approach for learning budget allocation policies for

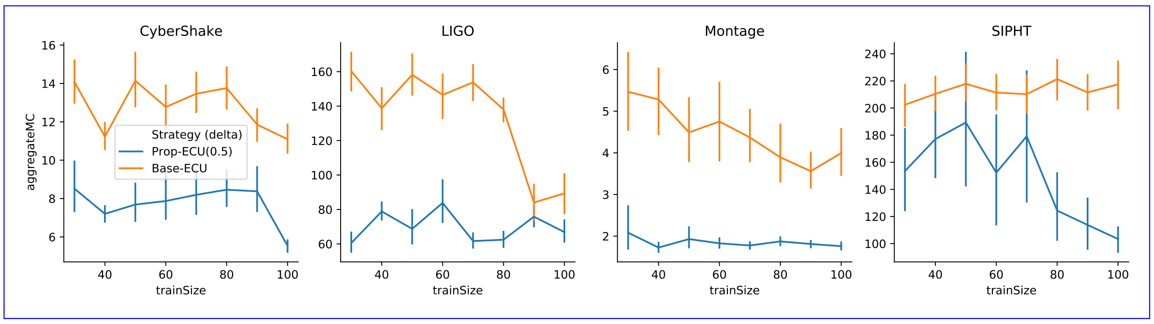


Figure 7: Performance comparison of the strategies Prop-ECU(0.5) and Base-ECU considering different workflow sizes in the learning process. Discrete error bars represent the 95% confidence interval around the mean.

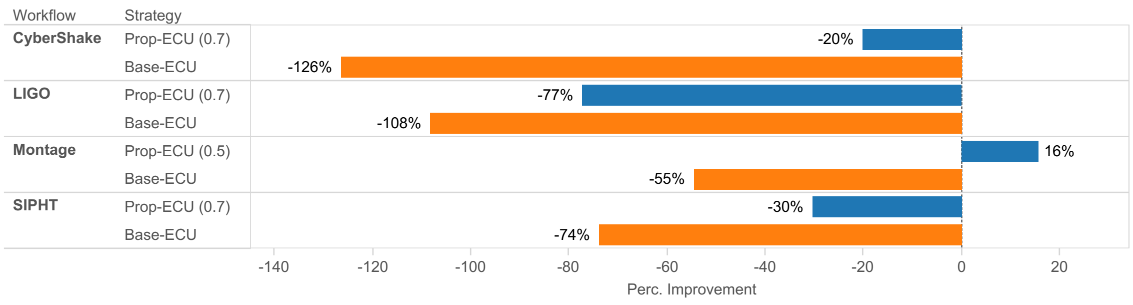


Figure 8: Percentage Improvement of the aggregateMC values for the strategies Prop-ECU and Base-ECU in comparison with SIAA.

the autoscaling of workflows in the Cloud. Unlike the proposal in the present paper, our previous policies partially solve the scaling problem (i.e., they decide the proportion of the budget to acquire spot and on-demand instance in the next execution period) and the solution depends on a heuristic strategy for completing the scaling decisions (i.e., number and type of VMs to acquire). Another difference with the current approach is that the policies in [13] were derived offline using the Value Iteration algorithm and the data generated by multiple workflow executions. Alternatively, now we are proposing an online learning process using the Q-learning algorithm, which is more suitable for Cloud environments than offline solutions [4].

There are other three approaches [14, 15, 16] addressing workflow applications but the RL strategies proposed are focused on scheduling policies (i.e., where and/or when executing tasks). Notice that the nature of the actions is different when learning scaling policies (i.e., number and types of VMs to acquire) as in our current proposal.

Furthermore, Dutreilh and Kirgizov [17] presented VirtRL, a Q-learning approach for scaling Cloud service applications to reduce economic cost while maintaining the Service-Level Agreement (SLA). Unlike our proposal, the authors do not consider different VM types, then, the scaling actions adjust the number of homogeneous VMs allocated to the application. Moreover, Ghobaei-Arani et al. [18] also presented an approach for scaling Cloud service applications based on Q-learning. This strategy aims to achieve a satisfactory compensation between SLA and economic costs. Considering the reward function used, the agent learns to adjust the infrastructure to balance the workload. In the case of workflow applications, focusing only in balancing the workload could limit the possibility of finding solutions with a better trade-off between makespan and cost. Horovitz and Arian [19] proposed Q-Threshold, an innovative Q-Learning approach for scaling Cloud services applications. This threshold-based approach is quite different to our proposal because it comprises higher-level actions with no specific decisions about the amount of instances of each VM type that should be acquired. Finally, we selected the proposal of Wei et al. [5] as a baseline strategy for assessing the performance of our strategy. This is a RL-based scaling approach to help SaaS providers making optimal resource allocation decisions in a Cloud environment. [5] considers an heterogeneous infrastructure with different VM types and price models. The goal of the strategy is to minimize renting expenses while providing sufficient processing capacity to meet customer demands. Also, they use a pure RL solution based on the Q-learning algorithm. Details about the definition of the states, actions and reward were discussed in Section 4.1.1. Both, our proposal and the baseline, formalized the problem of horizontal scaling in a heterogeneous Cloud infrastructure as an MDP and used Q-learning algorithm for learning the scaling policies. The fundamental differences lie in the MDP formulation proposed in each case. First, our strategy takes advantage of a scheme to figure out the upcoming workload for taking decisions, because of the inclusion of information related to the workflow dependency structures in the states definition. Second, our proposal works with reduced state and action spaces to simplify the learning process. And finally, the reward function defined in our proposal consider both makespan and execution cost.

Some other works are focused on vertical scaling [20], accelerating the learning process [21, 22] or combining RL with Neural Networks [16, 23, 24] or Fuzzy Logic [25, 20] (to manage complex state spaces). Our proposal is focused on sequential learning, as we are exploring the impact of other elements in the learning process (i.e., states, actions and reward). Also, we defined a reduced state space by discretizing the values of the variables involved, then we rely in a pure and simpler RL solution.

8. Concluding Remarks

This work addresses the problem of autoscaling scientific workflows in the Cloud to minimize makespan and economic cost. We propose a RL-based scaling method with a novel MDP formulation and we use the Q-learning algorithm for learning scaling policies. One distinctive feature of our proposal is allowing the scaling agent to consider information related to the dependency structures in the workflow~~(fork, join, pipeline)~~. Additionally, we define a reward function that directly corresponds ~~with the objective of minimizing makespan and economic cost~~to such objectives. We evaluated the proposed strategy considering three different heuristics for scheduling decisions and a RL baseline strategy from the state of the art closer as possible to our proposal. Also, we used an aggregated metric to measure the balance between makespan and ~~execution~~ cost achieved by each strategy on four well-known workflow applications.

First, we explored different configurations of our reward function varying the level of importance of each objective and evaluating the impact on the aggregated results. The optimal configuration is specific for each workflow but in many of the cases (7/12) giving more importance to makespan leads to better results. Then, we evaluated multiple policies learned for each workflow and each strategy. Experimental results show that our proposal outperforms the competitor. Our proposal presents significant gains (ranging between 25% and 55%) in comparison with the baseline. This shows that our proposed MDP formulation leads to improved results in performance. ~~Regarding the learning process, in various cases our strategy performs worse at the beginning but always ends up finding a better performing policy.~~ We also performed two complementary studies, namely evaluating the scalability of the RL-based strategies and a comparison with a state-of-the-art none-RL strategy. First, the analysis of scalability using different workflow sizes for the learning and the evaluation process, shows that our proposal scale wide better than the baseline strategy for all workflows and almost all workflow size configurations. Second, our proposal, when compared to a high-performance and informed heuristic strategy that assumes task runtime information as input, present losses of less than 31% in 2 workflows, 77% in one workflow (with a difficult learning process) and gains of 16% in another. Whilst baseline strategy present losses in all workflows (ranging between 55% and 126%). These are encouraging results because our proposal, unlike the heuristic, does not consider any performance information for decision making.

To continue improving our proposal, we are interested in extending our study in the following directions. We will study the possibility of learning autoscaling policies completely based on RL (scaling and scheduling decisions). ~~In this sense~~Then, we could fully exploit the benefits related to the online learning process (i.e., policies are transparent, dynamic, and adaptable). Moreover, we will expand our study to exploit more elements of a Cloud environment (e.g., more VM types and price models). This kind of extension could lead to scalability issues related to the dimension of the state and/or the action space. In this sense it could be necessary to explore RL techniques based on function approximations [11], which will allow us to cope with dimensionality limitations. Also~~as future work~~, it would be very useful to validate our proposal in a real Cloud environment.

Acknowledgements

We acknowledge the ~~financial support by CONICET , grant PIP 2017-2019~~ grants by

CONICET (PIP 11220170100490CO~~, and by the~~) and SIIP-UNCuyo ~~project No.~~ (B082~~.~~).

References

[1] D. A. Monge, Y. Garí, C. Mateos, and C. García Garino. Autoscaling scientific workflows on the cloud by combining on-demand and spot instances. *Journal of Computer Systems Science and Engineering*, 32(4), 2017.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529 EP –, 2015.

[4] Y. Garí, D. A. Monge, E. Pacini, C. Mateos, and C. García Garino. Reinforcement learning-based application autoscaling in the cloud: A survey. *Engineering Applications of Artificial Intelligence*, 102:104288, 2021.

[5] Y. Wei, D. Kudenko, S. Liu, L. Pan, L. Wu, and X. Meng. A reinforcement learning based auto-scaling approach for saas providers in dynamic cloud environment. *Mathematical Problems in Engineering*, pages 1–11, 2019.

[6] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi. Characterization of scientific workflows. ~~In *Third*~~ *3rd Workshop on Workflows in Support of Large-Scale Science*, pages 1–10, 2008.

[7] D. A. Brown, P. R. Brady, A. Dietz, J. Cao, B. Johnson, and J. McNabb. *A Case Study on the Use of Workflow Technologies for Scientific Analysis: Gravitational Wave Data Analysis*, pages 39–59. Springer London, 2007.

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

[9] J. Ericson, M. Mohammadian, and F. Santana. Analysis of performance variability in public cloud computing. ~~In *Proceedings of the*~~ *2017 IEEE International Conference on Information Reuse and Integration*, pages 308–314, 2017.

[10] D. A. Monge, E. Pacini, C. Mateos, E. Alba, and C. García Garino. CMI: An online multi-objective genetic autoscaler for scientific and engineering workflows in cloud infrastructures with unreliable virtual machines. *Journal of Network and Computer Applications*, 149:102464, 2020.

[11] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, USA, 2018.

[12] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya. 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.

[13] Y. Garí, D. A. Monge, C. Mateos, and C. García Garino. Learning budget assignment policies for autoscaling scientific workflows in the cloud. *Cluster Computing*, pages 23:87–105, 2020.

[14] E. Barrett, E. Howley, and J. Duggan. A learning architecture for scheduling workflow applications in the cloud. ~~*Proceedings of the*~~ *9th IEEE European Conference on Web Services*, pages 83–90, 2011.

[15] M. Soualhia, F. Khomh, and S. Tahar. A Dynamic and Failure-aware Task Scheduling Framework for Hadoop. *IEEE Transactions on Cloud Computing*, 8(2):1–16, 2018.

[16] J. L. Mingxi Cheng and S. Nazarian. DRL-Cloud : Deep Reinforcement Learning-Based Resource Provisioning and Task Scheduling for Cloud Service Providers. ~~In~~ *23rd Asia and South Pacific Design Automation Conference*, pages 129–134, 2018.

[17] X. Dutreilh and S. Kirgizov. Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow. ~~In~~ *7th International Conference on Autonomic and Autonomous Systems*, pages 67–74, 2011.

[18] M. Ghobaei-Arani, S. Jabbehdari, and M. A. Pourmina. An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach. *Future Generation Computer Systems*, 78:191–210, 2018.

[19] S. Horovitz and Y. Arian. Efficient cloud auto-scaling with SLA Objective Using Q-Learning. ~~In~~ *6th International Conference on Future Internet of Things and Cloud*, pages 85–92, 2018.

[20] T. Veni and S. M. Saira Bhanu. Auto-scale: automatic scaling of virtualised resources using neuro-fuzzy reinforcement learning approach. *International Journal of Big Data Intelligence*, 3(3), 2016.

[21] J. V. Bibal Benifa and D. Dejey. RLPAS: Reinforcement Learning-based Proactive Auto-Scaler for Resource Provisioning in Cloud Environment. *Mobile Networks and Applications*, pages 1–16, 2018.

[22] S. Mohammad Reza Nouri, H. Li, S. Venugopal, W. Guo, M. He, and W. Tian. Autonomic decentralized elasticity based on a reinforcement learning controller for cloud applications. *Future Generation Computer Systems*, 94:765–780, 2019.

[23] Z. Tong, H. Chen, X. Deng, K. Li, and K. Li. A scheduling scheme in the cloud computing environment using deep Q-learning. *Information Sciences*, 512:1170 – 1191, 2020.

[24] B. Du, C. Wu, and Z. Huang. Learning resource allocation and pricing for cloud profit maximization. ~~In *Proceedings of the*~~ *2019 AAAI Conference on Artificial Intelligence*, volume 33, pages 7570–7577, 2019.

[25] H. Arabnejad, C. Pahl, P. Jamshidi, and G. Estrada. A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling. ~~In *Proceedings of the*~~ *17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 64–73, 2017.

[26] D. A. Monge, M. Holec, F. Železnỳ, and C. García Garino. Ensemble learning of runtime prediction models for gene-expression analysis workflows. *Cluster Computing 18(4)*, 1317–1329, 2015.