2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)

A Reinforcement Learning Scheduling Strategy for Parallel Cloud-based Workflows

André Nascimento[1], Victor Olimpio[1], Vítor Silva[2], Aline Paes[1], Daniel de Oliveira[1]
[1] Institute of Computing, Fluminense Federal University, Niterói, RJ, Brazil
[2] Federal University of Rio de Janeiro, Rio de Janeiro, RJ, Brazil

***Abstract*—Scientific experiments can be modeled as Workflows. Such Workflows are usually computing- and data-intensive, demanding the use of High-Performance Computing environments such as clusters, grids, and clouds. This latter offers the advantage of elasticity, which allows for increasing and/or decreasing the number of Virtual Machines (VMs) on demand. Workflows are typically managed using Scientific Workflow Management Systems (SWfMS). Many existing SWfMSs offer support for cloud-based execution. Each SWfMS has its own scheduler that follows a well-defined cost function. However, such cost functions must consider the characteristics of a dynamic environment, such as live migrations and/or performance fluctuations, which are far from trivial to model. This paper proposes a novel scheduling strategy, named $ReASSIgN$, based on Reinforcement Learning (RL). By relying on an RL technique, one may assume that there is an optimal (or sub-optimal) solution for the scheduling problem, and aims at learning the best scheduling based on previous executions in the absence of a mathematical model of the environment. For this, an extension of a well-known workflow simulator WorkflowSim is proposed to implement an RL strategy for scheduling workflows. Once the scheduling plan is generated, the workflow is executed in the cloud using SciCumulus SWfMS. We conducted a thorough evaluation of the proposed scheduling strategy using a real astronomy workflow.**

***Index Terms*—Scientific Workflow, Reinforcement Learning, Cloud**

I. Introduction

Scientific workflows can be considered a *de facto* standard for modeling simulation-based scientific experiments that are commonly compute- and data-intensive [1], [2]. Workflows are abstractions that represent the dataflow among activities, *i.e.,* program invocations that perform operations such as data loading, data processing, and data aggregation.

A workflow can be formally defined as a directed acyclic graph $W(A, Dep)$, where the set of nodes $A = \{a_1, a_2, ..., a_n\}$ are the activities and the edges Dep represent the data dependencies among the activities in A [3]. Given $a_i \mid (1 \leq i \leq n)$, let us define $I = \{i_1, i_2, .., i_m\}$ the possible set of input data for an activity a_i, then $input(a_i) \in I$. Moreover, let us consider O as the set of outputs produced by a_i, then $Output(a_i) \in O$. The dependency between two activities a_i and a_j can be represented as $dep(a_i, a_j) \leftrightarrow \exists O_k \in input(a_j) | O_k \in output(a_i)$.

Let us also define as *activation* [4] the smallest unit of work that can be processed in parallel and consumes a specific data chunk [5]. Let us consider $Ac = \{ac_1, ac_2, ..., ac_k\}$ as the set of activations of the workflow W. Each ac_i is associated with a specific activity a_i that is represented as $act(ac_i) = a_i$. Activations also present data dependencies, thus $input(ac_i) \in I$ and $output(ac_i) \in O$ and the dependency between two activations ac_i and ac_j can be represented as $dep(ac_i, ac_j) \leftrightarrow \exists r \in input(ac_j)$.

Scientific Workflows are currently applied in many fields, such as IoT [6], [7], biology and astronomy [1], [8], and they are managed by complex engines called Scientific Workflow Management Systems (SWfMS), such as Pegasus [9], Swift/T [10] and SciCumulus [11] (the latter SWfMS is used in this paper). SWfMSs are responsible for executing, monitoring and collecting provenance data [12], which represent the execution history of the workflow. Over the last years, many SWfMSs started to provide ways to execute data- and compute-intensive workflows in the cloud [13] using Virtual Machines (VMs) and other resources such as distributed storage and elastic mechanisms. Clouds have already demonstrated its suitability to execute workflows that demand HPC capabilities [14]. This way, to execute a workflow in the cloud, all activations have to be scheduled and executed in parallel on a set of VMs $VM = \{vm_1, vm_2, ..., vm_d\}$ that may present heterogeneous characteristics.

In this context, several algorithms have been proposed from traditional scheduling algorithms (HEFT [15], Min-Min, Max-Min, *etc.*), to more complex algorithms that consider performance prediction [16]–[18], multi-site clouds [19], *etc*. These algorithms have to be implemented in a scheduler component in each SWfMS. Each scheduler works to minimize specific criteria such as makespan, financial cost, *etc*. Thus, in the aforementioned approaches, each algorithm follows a specific cost model that aims at formalizing the characteristics of the cloud environment in order to schedule all activations in Ac to VM. Although these approaches represent a step forward, they do not consider all characteristics of the cloud environment when scheduling activations, such as live migrations [20], performance fluctuations, *etc*. In fact, it is difficult, or sometimes even impossible, to model all these characteristics because they are usually Cloud provider or application-dependent [17]. In addition, these algorithms are commonly based on activation performance estimations. Estimating activations performance in VMS is also a difficult task to accomplish. Without accurate performance prediction and considering characteristics such as VM migrations and performance fluctuations, the scheduler may work inaccurately and inefficiently. In this way, the ideal would be that the scheduler of the SWfMSs would be adaptive to the environment instead of the need to model or formalize

978-1-5386-5555-9/19/$31.00 ©2019 IEEE
DOI 10.1109/IPDPSW.2019.00134

cloud characteristics.

One possible solution to schedule workflow activations in clouds without the need to model all characteristics of the environment beforehand is to use an algorithm based on Machine Learning [21]. One of these techniques is Reinforcement Learning [22] (RL), which is an area of Machine Learning that allows for this adaptability, as it is based on the existence of an agent capable of learning how to make more efficient decision-making in dynamic environments through prior trial and error. In the context of this paper, the agent in the RL-based scheduling algorithm checks the performance of each activation execution in a specific VM, and then receives a *reward* when the execution is considered a "good" decision or a *punishment* in the negative case. This type of scheduling algorithm does not need to be aware of all characteristics of the environment neither to predict the activation performance. The only requirement is that the agent should be able to evaluate if the current schedule is "better" or "worse" than a previous one. There are some approaches that apply RL to scheduling [23], [24] but they are decoupled from the concept of scientific workflows.

In this paper, we make the following contributions. First, we propose a novel RL-based cloud activation scheduling algorithm named $ReASSIgN$. Second, we propose an extension of WorkflowSim [25] that implements the proposed RL-based scheduling algorithm. Third, we propose the SciCumulus-RL, an extension of the SciCumulus SWfMS that executes workflows in cloud environments using RL with provenance support. Finally, we carry out an experimental evaluation, based on the implementation of SciCumulus-RL in Amazon AWS using Montage workflow [26].

This paper is organized as follows. Section 2 presents the background of Reinforcement Learning. Section 3 proposes the scheduling algorithm and discusses the design of RL-SciCumulus SWfMS. Section 4 presents the experimental evaluation. Section 5 discusses related work, and, finally, Section 6 concludes the paper and points out future work.

II. Background on Reinforcement Learning

Reinforcement Learning [22] is a sub-area of Machine Learning that focuses on building intelligent agents capable of behaving in dynamic environments by repeatedly executing actions and observing their results. While learning, the agent has to perceive the environment and execute an action, after what it receives a reward or punishment (a numeric value) according to the effect of such an action in the environment. The agent tries to maximize the cumulative reward by intercalating the *exploration* and *exploitation* steps: in the first case, the agent chooses any action to try to explore new paths not followed before; in the second case, the agent chooses the best action related to a state as observed so far.

Thus, given a set of states S and a set of actions A, the result of the learning phase is a *policy*, which is a function $\pi S \rightarrow A$ that maps states to actions. In other words, after learning, the agent is going to use the policy to decide which action to take when observing a state. In order to learn the best policy to a given environment, the agent assumes that the environment follows the Markov Decision Process (MDP). An MDP is composed of five components (i) a (finite) set of states S; (ii) a (finite) set of actions A; (iii) a (possibly stochastic) transition function $T : (S \times A) \rightarrow S$ that maps a state to its successor, according to the action taken; (iv) a reward function $R : (S, A) \rightarrow \mathbb{R}$; and (v) a discount factor $0 \leq \gamma \leq 1$ to assess value over future actions compared to the current ones.

By following the Markov property, we can assume that the next state and the received reward depend upon only the previous state and the action taken in a instant t, following the Equation 1:

$$p(s', r|_{s,a}) = Pr\{S_{t+1} = s', R_{t+1} = r|_{S_t=s,A_t=a}\} \quad (1)$$

Where s' is the next state of the environment and r is the reward received by taking the action a when observing the state s.

a) Q-Learning: As mentioned before, the agent is going to try to maximize the cumulative reward received along the time (Equation 6). Since the value of future rewards is not known beforehand, we should use an iteration value function to estimate them. In this paper, we use the classical model-free, off-policy Q-learning algorithm [22] (Algorithm 1) to handle this problem.

$$\sum_{t=0}^{\infty} \gamma^t r_t(s, a) \quad (2)$$

The algorithm has as intrinsic goal to approximate the value returned by the Q function, which maps a state s and an action a to the expected reward of starting in s and executing a. The Equation 3 shows how the Q value is computed and updated.

$$\begin{aligned} Q(s_t, a_t)^+ \leftarrow \; & Q(s_t, a_t) + \alpha^t \times (r(s_t, a_t) + \\ & \gamma^t \times \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \end{aligned} \quad (3)$$

Where $s_t \in S$, $a_t \in a$, r is the reward received after executing a_t when in the state s_t in time t. α is the learning rate, defining how much one would like to update the current value of Q, and γ is the discount factor of the MDP. Notice that both the current reward $r(s_t, a_t)$ and the future estimated reward (the maximum expected value of the next state s_{t+1}, regarding each action, *i.e.*, $\max_a Q(s_{t+1}, a)$) are taken into account when computing the value of the Q function.

The exploration/exploitation dilemma is faced by the Q-learning through the *ϵ-greedy* policy: at each iteration, with a probability ϵ, the best action is taken, according to the value returned by the Q function; otherwise, an action is selected at random. The whole procedure is executed until some termination criteria is met: this could be, for example, the number of times that an entire sequence of states and actions are followed, until the goal state is reached.

III. RL-based Activation Scheduling Approach

In this section, we propose our RL-based scheduling approach for cloud-based scientific workflow execution, *i.e.*

Input: $S, A, \gamma, \alpha, \epsilon$
Output: A matrix $Q : S \times A$
Start $Q(s, a) \forall s, a, s \in S, a \in A$ at random
repeat
  foreach *episode* e **do**
    $t \leftarrow 0$ **while** *has not reached a terminal state* **do**
      observe the state s
      with probability ϵ choose a as the best action to s according to $Q(s, a)$; otherwise choose a at random
      execute a into the environment
      collect the reward $r \leftarrow r(s, a)$
      observe the next state s'
      $\delta \leftarrow (r + \gamma^t \times \max'_a Q(s', a') - Q(s, a))$
      $Q(s, a) \leftarrow Q(s, a) + \alpha \times \delta$
      $s \leftarrow s'$
    end
  end
until *termination*;

Algorithm 1: Q-learning Algorithm

$ReASSIgN$ (Rl-based Activation Scheduling of ScIeNtific workflows). Before scheduling workflow activations, it is mandatory to define the environment and how to evaluate it. VMs are the resources the SWfMSs use to execute the activations, so the performance of each VM has to be considered individually after executing an activation as well as globally considering the entire set of VMs in the cloud environment. This is necessary so that we can calculate the reward/punishment for each schedule produced by the proposed RL-based Scheduling Algorithm (*i.e.* the agent). In this paper, the scheduling algorithm considers the performance of each VM when executing workflow activations. We use as metrics the execution and queue times of activations.

We extended/adapted the scheduling approach proposed by Costa *et al.* [27] that schedules standalone applications in grid environments. Particularly, we benefit from the reward function defined by Costa *et al.* and extended it to the cloud-based workflow environment. Next, we first define the space of states and actions regarding the goal of finishing all the workflow activations successfully. Then, we define the reward function used to leverage the Q-learning algorithm. Finally, we devise our version of the Q-learning algorithm to solve the scheduling problem in cloud-based workflows.

A. States and Actions of the Workflow

Since the overall workflow execution depends upon its several activations, which may be scheduled to several VMs, we define the space of states of a workflow based on the state of the activations. Let us consider that each activation is in a state $s_{ac} \in SAC$ = {*ready, locked, running, succesfully finished, finished with a failure*}. We would like that at the end of the process all the activations are in the state *successfully finished*, which corresponds to a successful execution. However, it may be the case that the activation finishes with a failure due to a problem in the hardware or other issues. The *running* state represents that the activation is running on some VM. The *locked* state means that the activation ac_i is waiting for another activation ac_j to finish according to $dep(ac_i, ac_j) \leftrightarrow \exists r \in input(ac_j) | r \in output(ac_i) \wedge dep(act(ac_i), act(ac_j))$. Finally, the *ready* state means that the activation is ready to be scheduled to a VM, with no dependencies. A virtual machine vm_k may be in a state $s_{vm} \in SVM = \{idle, busy\}$.

Finally, let $\mathcal{S}$ = {*successfully finished, finished with failure, available, unavailable*} be the possible states of the workflow W that are going to be submitted to the Q function. Both the states *successfully finished* and *finished with failure* are terminal states, meaning that all the activations have finished and there is nothing more to be done. The first one is reached when all the activations $ac_i \in Ac$ have finished without failure, while the second one means that at least one activation $ac_i \in Ac$ has not ended well.

In case there is at least one activation ac_i in the state *ready* and at least one machine in the state $idle$, then W is associated with the *available* state, meaning that it may be able to choose an activation to run at a VM. If all the activations are in the state *running* or *locked*, or all the machines are in the state *busy*, then W is in the state *unavailable*. This shows that there is not a single activation that could be scheduled to a machine, because they are all already running, or because they are waiting for another dependent activation to finish, or still because there is not a single *idle* machine to receive a new activation to execute.

There are only two types of actions allowed in our approach: either we *schedule an activation* ac_x *to a VM* vm_j or we *do nothing*. Notice, however, that we may have k ready activations to be scheduled at m idle machines in an instant of time, creating a space of $k \times m + 1$ possible actions to be taken in the worst case. The *schedule* action is allowed whenever W has the state *available* and the *do nothing* action is chosen when W has the state *unavailable*.

The transition function $T : S \times A \rightarrow S$ defines six possible transitions in the states of the workflow W. The workflow starts at the state *available*. The only action that makes sense in this state is one of the *schedule* type. After that, W may remain at the state *available*, in case there are still some activations to run and some *idle* machine. Otherwise, in case the machines are all *busy* or the activations are all *running* or *locked*, then W goes from *available* to *unavailable*. From the state *unavailable*, the only action allowed is *do nothing*. However, changes in the environment may make W to transit to other states. In case there are no changes in the environment (none of the running activations have finished), then W stays at the *unavailable* state. Otherwise, if at least one activation finishes and there are still other activations in the *ready* state, then W goes from *unavailable* to *available*. Finally, if all the activations have finished and there are no more activations to run, then W goes to one of the two terminal states. In short, each state is defined as follows:

- s_w = *successfully finished* if $\forall\ s_{ac}$ = *successfully finished*

- $s_w =$ *finished with failure* if $\exists\ s_{ac} =$ *finished with failure* $\wedge\ \nexists\ s_{ac}$ = *ready* $\wedge\ \nexists\ s_{ac}$ = *locked* $\wedge\ \nexists\ s_{ac}$ = *running*
- $s_w =$ *available* if $\exists\ s_{ac} =$ *ready*
- $s_w =$ *unavailable* if $\nexists\ s_{ac} =$ *ready*

B. The reward function

To assess the quality of the action $schedule\ ac_x\ to\ vm_j$ and see how it affects the overall scheduling, we need to analyze the performance of vm_j and contrast it with the overall performance of the workflow. To do that, let us define te_i as the execution time of an activation ac_i on the vm_j and tf_i as the queue time of ac_i. Let us also define $tt_i = te_i + tf_i$ (where tet_i is the total execution time of the activation ac_i). Based on these times we can estimate the performance index of vm_j when executing an activation ac_i as $Pi_j = tt_i \times \mu + (1-\mu) \times (tf_i)$, where μ is a parameter to balance the relevance of the total execution time against the queue time. It is worth noticing that Pi_j defines the performance of vm_j for a single activation execution. For short, Pi_j represents how fast a VM is able to execute an activation in relation to the time the activation was left in queue. Next, we need to consider the overall behavior of vm_j. This way, we define

$$\overline{Pi_j} = \overline{te_i} * \mu + (1-\mu) * (\overline{tf_i}) \tag{4}$$

Where $\overline{te_i}$ and $\overline{tf_i}$ are, respectively, the average execution and queue times for all the activations ac_i that have executed at vm_j. Then, we compute the global performance $\overline{Pw}$ of the workflow considering all the VMs as follows:

$$\overline{Pw} = \overline{te} * \mu + (1–\mu) * \overline{tf} \tag{5}$$

Where $\overline{te}$ and $\overline{tf}$ are the average times of execution and queue time, respectively, considering all the activations $ac_i \in Ac$. Next, we compute a partial crisp reward by comparing the average performance index $\overline{Pi_j}$ of vm_j to the average global performance index $\overline{Pw}$. We do that by computing the standard deviation $stdv$ of $\overline{Pi_j}$, and we define the partial reward/punishment as follows:

$$r_i = \begin{cases} -1, & \text{if } \overline{Pi_j} > (\overline{Pw} + stdv) \\ 1, & \text{otherwise} \end{cases} \tag{6}$$

Finally, to compute the reward r^t to update the Q value at the instant t, we take $r^t = r^{t-1} + \rho \times (r_i - r^{t-1})$, where r^{t-1} is the previous reward and ρ is a parameter to measure the relevance of the crisp partial reward compared to the previous reward. This update also provides the intuition that we want to award the scheduling decision that is improving the efficiency of the workflow.

C. The $ReASSIgN$ (Rl-based Activation Scheduling of ScIeNtific workflows) Algorithm

As aforementioned, Q-Learning works with episodes (a sequence of states, actions, and rewards, which ends with the terminal state). Thus, each complete execution/scheduling of the workflow represents an episode for the Q-Learning. In order to interconnect the episodes all relevant learning and analysis information are used in subsequent episodes, such as the action of scheduling an activation to a VM, that is, the action defined by $sched(ac_x, vm_y)$, together with the value obtained through the Q function and the state of the workflow after the scheduling. Thus, at the beginning of each execution of the workflow, all information associated with the previous episodes is loaded allowing the progression of learning.

For the development of the $ReASSIgN$ algorithm, it was necessary to adapt the Q-Learning algorithm to consider the aforementioned characteristics. $ReASSIgN$ receives a list of activations available for execution, but not yet scheduled, along with the list of available VMs. It is worth mentioning that $ReASSIgN$ do not consider activations that are prevented from being executed by dependency issues. The evaluation table Q used by $ReASSIgN$ is represented by an array containing all values of Q for each schedule action between the activation and a VM. The selection of the VM occurs in two ways: if it is in the stage of exploitation, the choice is determined based on the largest value of the evaluation function Q of the Q-Learning algorithm among idle VMs, *i.e.,* with its state in *idle*; on the other hand, if it is in the exploration phase, the choice is made at random based on the probability ϵ. In this phase, the choice of a specific VM among idle VMs also occurs randomly. After that, the current activation is scheduled to the chosen VM.

Next, it takes the execution and queuing times of this activation to calculate the efficiency index Pi_j for scheduling this activation in the chosen VM. Then, the global performance index $\overline{Pi_j}$ of the VM is calculated for all activations performed on this machine and the workflow performance index Pw. Subsequently, the partial reward/punishment r_i, the reward r^t, and the efficiency value of the VM $nrle_i$ are calculated, yielding in the $Q(s,a)$ value, which in turn is based on the Q function of Q-Learning. Finally, it updates the state of the workflow. At the end of the scheduling of all activations, $ReASSIgN$ records all data associated to this episode that they can be used in the next episode, thus contributing continuously to the learning until reaching the desired number of episodes. The Algorithm 2 describes the step-by-step $ReASSIgN$ to obtain a scheduling plan through several sequential episodes.

D. System Implementation

The proposed $ReASSIgN$ algorithm is implemented in the simulator named WorkflowSim [25] and is invoked by the SciCumulus SWfMS. Since the reinforcement learning process is based on trial and error, it may be financially expensive to be entirely executed in a real cloud environment (since the user pay per hour). This way, we propose to schedule activations in 2 stages: the first one is based on simulation using the WorkflowSim (where several episodes are executed) and then execute the generated final scheduling plan in SciCumulus in a real cloud environment. This integration between SciCumulus and WorkflowSim was named SciCumulus-RL. The proposed architecture is presented in Figure 1.

Input: $S, A, T, \gamma, \alpha, \epsilon, \mu, \rho, maxIter, Ac, VMs$
Output: A matrix $Q : S \times A$ representing the scheduling strategy

Start $Q(s,a) \forall s, a, s \in S, a \in A$ at random
$iter \leftarrow 0$
while $iter < maxIter$ **do**
  $s \leftarrow available$
  $t \leftarrow 1$
  $r^t \leftarrow 0$
  while $s \neq successfully\ finished \wedge$
   $s \neq finished\ with\ failure$ **do**
    if $s = unavailable$ **then**
      $a \leftarrow do\ nothing$
    else
      with probability ϵ choose a_{ij} as the best action to s according to $Q(s,a)$; otherwise choose a_{ij} at random
      allocate $ac_i \in Ac$ to $vm_j \in VMs$
      compute $\overline{Pi}(\mu, Ac, VMs)$ according to equation 4
      compute $\overline{Pw}(\mu, Ac, VMs)$ according to equation 5
      compute $r_i(\overline{Pi}, \overline{Pw})$ according to equation 6
      $r^t \leftarrow r^{t-1} + \rho \times (r_i - r^{t-1})$
      $\delta \leftarrow (r^t + \gamma^t \times \max'_a Q(s', a') - Q(s,a))$
      $Q(s,a) \leftarrow Q(s,a) + \alpha \times \delta$
      $t \leftarrow t + 1$
    end
    $s \leftarrow$ the new state s' according to the transition model T
  end
  $iter \leftarrow iter + 1$
end

Algorithm 2: $ReASSIgN$ Algorithm

SciCumulus SWfMS is composed of three modules: $SCStarter$, $SCSetup$, and $SCCore$. $SCSetup$ is responsible for loading the workflow specification (an XML file) to be executed. In SciCumulus-RL, besides loading the data, it invokes WorkflowSim. WorkflowSim is a *de facto* standard for simulating workflows in distributed environments such as clouds. It is composed of a Workflow Mapper to map abstract workflows (XML file) to concrete workflows that are dependent on execution VMs, a Workflow Engine to handle the data dependencies, and a Workflow Scheduler to match activations to VMs. In the context of this paper, the only modification performed in WorkflowSim was to implement $ReASSIgN$ in the scheduler, by replacing the default HEFT algorithm.

Once WorkflowSim executes $ReASSIgN$ and produces the final scheduling plan, this plan is submitted to SciCumulus. By analyzing the produced scheduling plan, $SCStarter$ deploys the necessary VMs in the cloud. After, $SCCore$ executes the activations following the scheduling plan. Note that, $SCCore$ is a MPI-based application, *i.e.* it is executed in all VMs where one $SCMaster$ coordinates the execution of several $SCSlaves$. All data associated with the workflow execution is stored in a provenance database. Such information can be used in future executions of $ReASSIgN$. For more information about SciCumulus, please refer to Oliveira *et al.* [11].

IV. Experimental Evaluation

This section presents the experiments performed to evaluate $ReASSIgN$ and its implementation in SciCumulus-RL . To evaluate $ReASSIgN$, we have modeled Montage workflow [26]. Montage is an astronomy workflow that collects data through telescopes and creates mosaic images from objects in the sky [28]. Montage is based on a toolkit for assembling Flexible Image Transport System (FITS) images into custom mosaics. It is considered a *de facto* benchmark for evaluating SWfMSs. We have obtained traces from Montage in the Workflow Generator web page [29].

A. Environment Setup

For the experiments executed in this paper, we have deployed SciCumulus-RL on top of Amazon AWS cloud. Amazon AWS provides several different types of VMs for users to deploy and use. Each one of them has unique characteristics (CPU power, RAM and storage capacity). There are several types of VMs, but in the experiments performed in this paper we only considered **t2.micro** - 1 vCPU and 1 GB RAM and **t2.2xLarge** - 8 vCPUs and 16 GB RAM. Each deployed VM uses Linux Cent OS (64-bit), and it was configured with the necessary software and libraries like MPI and Montage applications. All VMs were configured to be accessed using SSH without password checking (although this is not recommended due to security issues). In terms of software, all VMs, no matter its type, executes the same programs and configurations. According to Amazon, all VMs were instantiated in the US East—N. Virginia location and follow the pricing rules of that locality. Table I presents the configurations of the environment used in the experiments.

# of VMs	# of VMs t2.micro	# of VMs t2.2xLarge	# of vCPUs
9	8	1	16
11	8	3	32
15	8	7	64

TABLE I: VM configurations used in the experiments

B. Experiment Setup

The experiments were performed using a dataset constructed based on the performance requirements of workflows in real executions of the Montage workflow. To evaluate the scheduling plans, we used information from real executions provided by Workflow Generator and SciCumulus-RL Provenance Database. In the workflow generator page, there are several executions of Montage available, but we considered the one with 50 activations in these experiments (50 node DAX).

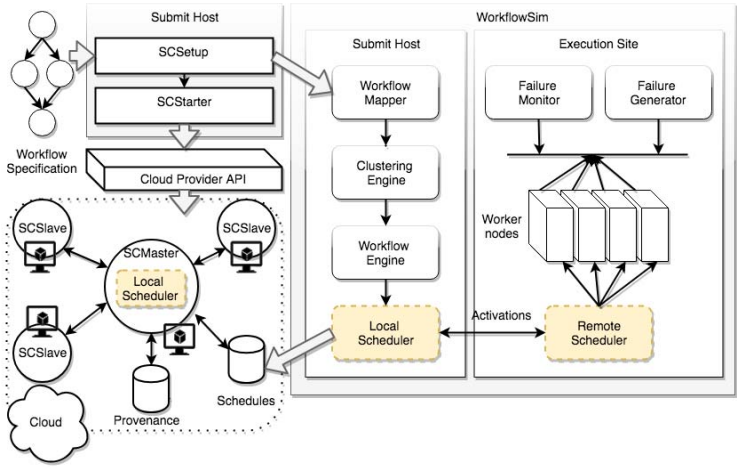

Fig. 1: *Architecture of SciCumulus-RL*

C. Results

To obtain the scheduling plan generated by $ReASSIgN$, each execution of Montage performed 100 episodes with $\mu = 0.5$ and varied the other Q-Learning parameters according to the set of VMs defined in Table I. The 3 parameters of the Q-Learning that were varied are α that corresponds to the learning rate, γ which is the weight attributed to the reward and ϵ that represents the probability of deciding between exploitation or exploration. Each parameter assumed a value in the set $\{0.1, 0.5, 1.0\}$ for each execution. Thus, 27 different schedules were generated for the same configuration of VMs, *i.e.,* 81 executions were performed. For each result, we present the time needed to learn the scheduling plan (Table II) and the simulated execution time of Montage in WorkflowSim (Table III). It is worth noticing that the learning time is acceptable in comparison with the total execution time of the workflow.

After $ReASSIgN$ generates the scheduling plans, they were submitted to SciCumulus-RL for the real execution. Table IV presents the Montage execution times using 16, 32 and 64 vCPUS. We compared the schedules obtained using $ReASSIgN$ with the schedules obtained using HEFT [15] scheduling algorithm. HEFT is an extension of the classical list scheduling algorithm for homogeneous systems which includes heterogeneous systems. By analyzing results presented in Table IV we can state that $ReASSIgN$ presented execution times slightly smaller than the ones of HEFT for 32 and 64 vCPUs. In addition, we can state that a slower learning parameter (*i.e.,* α) can produce better performance for $ReASSIgN$, indicating that a longer history contains good information about resource behavior.

To better understand this comparison, it is necessary to analyze how both algorithms schedule the activations for 16 vCPUS as presented in Table V. Table V presents the ID of the activation (from o to 49), and the ID of the VM where the activation was scheduled (from 0 to 8) by HEFT and $ReASSIgN$. In the case of $ReASSIgN$, C_1 represents the scenario where $\alpha = 1.0$, $\gamma = 1.0$ and $\epsilon = 0.1$, C_2 represents

Parameters			Learning Time		
α	γ	ϵ	16 vCPUs	32 vCPUs	64 vCPUs
0.1	0.1	0.1	93.0	98.0	119.0
0.1	0.1	0.5	90.0	99.0	114.0
0.1	0.1	1.0	88.0	100.0	116.0
0.1	0.5	0.1	92.0	100.0	110.0
0.1	0.5	0.5	88.0	96.0	117.0
0.1	0.5	1.0	96.0	96.0	108.0
0.1	1.0	0.1	83.0	96.0	109.0
0.1	1.0	0.5	88.0	96.0	112.0
0.1	1.0	1.0	86.0	97.0	110.0
0.5	0.1	0.1	91.0	100.0	116.0
0.5	0.1	0.5	91.0	94.0	118.0
0.5	0.1	1.0	87.0	95.0	119.0
0.5	0.5	0.1	93.0	97.0	115.0
0.5	0.5	0.5	93.0	95.0	120.0
0.5	0.5	1.0	89.0	97.0	120.0
0.5	1.0	0.1	78.0	93.0	108.0
0.5	1.0	0.5	99.0	93.0	107.0
0.5	1.0	1.0	82.0	86.0	106.0
1.0	0.1	0.1	91.0	93.0	108.0
1.0	0.1	0.5	90.0	93.0	104.0
1.0	0.1	1.0	87.0	92.0	108.0
1.0	0.5	0.1	89.0	92.0	105.0
1.0	0.5	0.5	80.0	89.0	107.0
1.0	0.5	1.0	89.0	96.0	107.0
1.0	1.0	0.1	79.0	90.0	107.0
1.0	1.0	0.5	92.0	89.0	107.0
1.0	1.0	1.0	86.0	90.0	110.0

TABLE II: Learning time of Montage workflow in WorkflowSim

the scenario where $\alpha = 0.5$, $\gamma = 1.0$ and $\epsilon = 0.1$, and finally C_3 represents the scenario where $\alpha = 0.1$, $\gamma = 1.0$ and $\epsilon = 0.1$.

In the scheduling plan defined by the scheduler HEFT, the initial 9 activations are distributed sequentially among the 8 available virtual machines. Then, it can be seen that the HEFT scheduling algorithm makes no distinction as to how much these activations can be computing-intensive and need to be scheduled in more robust VMs. The same is no longer the case in the scheduling plans obtained by $ReASSIgN$ that tend to make these schedules in the more robust VMs, since for con-

Parameters			Simulated Execution Time		
α	γ	ϵ	16 vCPUs	32 vCPUs	64 vCPUs
0.1	0.1	0.1	663.94153	595.16841	671.06544
0.1	0.1	0.5	848.71421	594.01914	796.02482
0.1	0.1	1.0	752.95727	839.18546	592.58062
0.1	0.5	0.1	749.42662	670.53755	665.96669
0.1	0.5	0.5	905.13440	891.55308	746.14164
0.1	0.5	1.0	852.55245	695.53101	537.25396
0.1	1.0	0.1	259.02589	334.16751	256.52062
0.1	1.0	0.5	542.93981	416.73384	440.44978
0.1	1.0	1.0	829.70283	904.39862	585.58822
0.5	0.1	0.1	822.41175	670.84354	814.45827
0.5	0.1	0.5	923.86923	695.94739	672.85318
0.5	0.1	1.0	924.44982	737.26139	770.37092
0.5	0.5	0.1	845.64167	672.09372	641.31525
0.5	0.5	0.5	824.88087	747.06360	594.10582
0.5	0.5	1.0	928.42364	751.88346	774.29824
0.5	1.0	0.1	486.68072	258.06918	332.70493
0.5	1.0	0.5	697.33701	643.53614	438.81466
0.5	1.0	1.0	830.05115	827.60478	510.21845
1.0	0.1	0.1	642.84260	801.82828	486.07712
1.0	0.1	0.5	898.03071	723.51203	497.66092
1.0	0.1	1.0	775.77633	836.65385	693.54991
1.0	0.5	0.1	817.88112	821.89452	639.98802
1.0	0.5	0.5	847.97787	846.69140	412.29261
1.0	0.5	1.0	905.85435	698.15094	652.81855
1.0	1.0	0.1	259.31386	333.56856	279.72661
1.0	1.0	0.5	663.30534	641.21227	432.01269
1.0	1.0	1.0	752.69346	851.41301	511.38465

TABLE III: Simulated execution time of Montage workflow in WorkflowSim/$ReASSIgN$

Algorithm	vCPUs	α	γ	ϵ	Total Execution Time
HEFT	16	-	-	-	00:03:09.625
$ReASSIgN$	16	1.0	1.0	0.1	00:03:17.483
$ReASSIgN$	16	0.1	1.0	0.1	00:03:29.584
$ReASSIgN$	16	0.5	1.0	0.1	00:03:37.552
$ReASSIgN$	32	0.5	1.0	0.1	00:03:17.395
$ReASSIgN$	32	0.1	1.0	0.1	00:03:41.148
HEFT	32	-	-	-	00:03:48.892
$ReASSIgN$	32	1.0	1.0	0.1	00:03:53.511
$ReASSIgN$	64	0.5	1.0	0.1	00:03:23.26
$ReASSIgN$	64	0.1	1.0	0.1	00:03:35.23
$ReASSIgN$	64	1.0	1.0	0.1	00:03:41.918
HEFT	64	-	-	-	00:03:42.675

TABLE IV: Actual execution time of *Montage* workflow in Amazon AWS

figuration 1, as shown in Table V, it is possible to observe the predominance of schedules of these activations in the VM type 2xLarge. Although results are promising, experiments show that $ReASSIgN$ always presents a better performance, yet very close to HEFT. We believe that $ReASSIgN$ will provide better scheduling plans as more episodes are considered. This way, more experiments with larger instances of Montage and other workflows are still needed.

V. Related Work

There are some works that consider RL to schedule applications in distributed environments [23], [24], [30], [31]. However, none of them is associated with the concept of scientific workflows. Fang *et al.* [30] propose an RL scheduler for scheduling applications in GPUs while maintaining the Quality-of-Service. Li *et al.* [31] proposes a series of schedulers that are based on RL. These schedulers. The main goal of Li *et al.* paper is to use the results to speed up reinforcement learning via reward shaping. Costa *et al.* [23], [24] study the behavior of different scheduling algorithms when performing job orchestration in computational grids.

VI. Conclusions and Future Work

In this paper, we propose a scheduling strategy for cloud-based scientific workflows based on reinforcement learning named $ReASSIgN$, and compared it with a standard scheduling algorithm based on HEFT. One advantage of $ReASSIgN$ in comparison with most schedulers is that it does not need a cost function to perform scheduling. In fact, $ReASSIgN$ only needs to be able to evaluate if each activation schedule is "good"or not. We studied how $ReASSIgN$ is affected by the learning parameters (by varying α, γ and ϵ parameters) and by the queue waiting time.

We executed our experiments (running Montage workflow) in a real cloud (Amazon AWS) after a series of simulated executions of $ReASSIgN$, composed of up to 64 vCPUs, with different kinds of resources. The results show gains in optimizing makespan. In addition, the results show that a slower learning parameter can produce better performance for $ReASSIgN$, indicating that long history of cloud usage for running workflows contains useful information about resource behavior.

Acknowledgments

This work has been partially supported by CAPES, FAPERJ, and CNPq.

References

[1] Y. Zhao, I. Raicu, and I. Foster, "Scientific workflow systems for 21st century, new bottle or new wine?" in *Services-Part I, 2008. IEEE Congress on*. IEEE, 2008, pp. 467–471.

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

[3] D. de Oliveira, K. A. C. S. Ocaña, F. A. Baião, and M. Mattoso, "A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds," *J. Grid Comput.*, vol. 10, no. 3, pp. 521–552, 2012. [Online]. Available: https://doi.org/10.1007/s10723-012-9227-2

[4] E. Ogasawara, J. Dias, D. Oliveira, F. Porto, P. Valduriez, and M. Mattoso, "An algebraic approach for data-centric scientific workflows," *Proc. of VLDB Endowment*, vol. 4, no. 12, pp. 1328–1339, 2011.

[5] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso, "A survey of data-intensive scientific workflow management," *Journal of Grid Computing*, vol. 13, no. 4, pp. 457–493, Dec 2015. [Online]. Available: https://doi.org/10.1007/s10723-015-9329-8

[6] P. Iyenghar and E. Pulvermueller, "A model-driven workflow for energy-aware scheduling analysis of iot-enabled use cases," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4914–4925, Dec 2018.

[7] Y. Shao, C. Li, and H. Tang, "A data replica placement strategy for iot workflows in collaborative edge and cloud environments," *Computer Networks*, vol. 148, pp. 46 – 59, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128618311460

[8] K. A. Ocaña, D. de Oliveira, E. Ogasawara, A. M. Dávila, A. A. Lima, and M. Mattoso, "Sciphy: a cloud-based workflow for phylogenetic analysis of drug targets in protozoan genomes," in *BSB11*. Springer, 2011, pp. 66–70.

[9] E. Deelman, G. Mehta, G. Singh, M.-H. Su, and K. Vahi, "Pegasus: mapping large-scale workflows to distributed resources," in *Workflows for e-Science*. Springer, 2007, pp. 376–394.

Activation ID	HEFT	C_1	C_2	C_3
0	0	4	8	4
1	0	8	8	8
2	1	8	4	8
3	2	8	1	8
4	3	8	8	8
5	4	8	3	8
6	5	8	8	8
7	6	8	8	8
8	7	8	8	8
9	8	8	8	8
10	8	8	8	8
11	0	8	8	8
12	1	8	8	8
13	0	7	7	7
14	1	8	8	8
15	1	8	8	8
16	2	8	8	8
17	3	7	7	7
18	4	8	8	8
19	5	6	6	6
20	6	7	7	7
21	7	5	5	5
22	8	8	8	8
23	8	4	4	4
24	3	5	5	5
25	4	3	3	3
26	2	6	6	6
27	7	2	2	2
28	6	7	7	7
29	5	1	1	1
30	1	8	8	8
31	8	0	0	0
32	1	4	4	4
33	4	2	2	2
34	2	3	3	3
35	3	7	7	7
36	6	8	8	8
37	8	1	1	1
38	8	8	8	8
39	7	6	6	6
40	2	5	5	5
41	3	4	4	4
42	4	3	3	3
43	7	0	0	0
44	5	5	5	5
45	6	6	6	6
46	8	2	2	2
47	1	1	1	1
48	5	7	7	7
49	1	8	8	8

TABLE V: Scheduling plan for 16 vCPUS

[10] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. Von Laszewski, V. Nefedova, I. Raicu, T. Stef-Praun, and M. Wilde, "Swift: Fast, reliable, loosely coupled parallel computation," in *Services, 2007 IEEE Congress on*. IEEE, 2007, pp. 199–206.

[11] D. Oliveira, E. Ogasawara, K. Ocaña, F. Baião, and M. Mattoso, "An adaptive parallel execution strategy for cloud-based scientific workflows," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1531–1550, 2012.

[12] J. Freire, D. Koop, E. Santos, and C. T. Silva, "Provenance for Computational Tasks: A Survey," *Computing in Science & Engineering*, pp. 20–30, Maio/Junho 2008.

[13] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: Towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, Dec. 2008. [Online]. Available: http://doi.acm.org/10.1145/1496091.1496100

[14] I. Sadooghi, J. H. Martin, T. Li, K. Brandstatter, K. Maheshwari, T. P. P. D. L. Ruivo, G. Garzoglio, S. Timm, Y. Zhao, and I. Raicu, "Understanding the performance and potential of cloud computing for scientific applications," *IEEE Trans. Cloud Computing*, vol. 5, no. 2, pp. 358–371, 2017. [Online]. Available: https://doi.org/10.1109/TCC.2015.2404821

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

[16] K. Wang, X. Zhou, T. Li, D. Zhao, M. Lang, and I. Raicu, "Optimizing load balancing and data-locality with data-aware scheduling," in *2014 IEEE International Conference on Big Data, Big Data 2014, Washington, DC, USA, October 27-30, 2014*, 2014, pp. 119–128.

[17] R. Mathá, S. Ristov, and R. Prodan, "A simplified model for simulating the execution of a workflow in cloud," in *Euro-Par 2017: Parallel Processing - 23rd International Conference on Parallel and Distributed Computing, Santiago de Compostela, Spain, August 28 - September 1, 2017, Proceedings*, 2017, pp. 319–331.

[18] H. Arabnejad, J. G. Barbosa, and R. Prodan, "Low-time complexity budget-deadline constrained workflow scheduling on heterogeneous resources," *Future Generation Comp. Syst.*, vol. 55, pp. 29–40, 2016. [Online]. Available: https://doi.org/10.1016/j.future.2015.07.021

[19] J. Liu, E. Pacitti, P. Valduriez, D. de Oliveira, and M. Mattoso, "Multi-objective scheduling of scientific workflows in multisite clouds," *Future Generation Comp. Syst.*, vol. 63, pp. 76–95, 2016. [Online]. Available: https://doi.org/10.1016/j.future.2016.04.014

[20] W. Hu, A. Hicks, L. Zhang, E. M. Dow, V. Soni, H. Jiang, R. Bull, and J. N. Matthews, "A quantitative study of virtual machine live migration," in *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, ser. CAC '13. New York, NY, USA: ACM, 2013, pp. 11:1–11:10. [Online]. Available: http://doi.acm.org/10.1145/2494621.2494622

[21] T. M. Mitchell, *Machine learning*, ser. McGraw Hill series in computer science. McGraw-Hill, 1997.

[22] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[23] B. F. Costa, I. de Castro Dutra, and M. Mattoso, "Rl-based scheduling strategies in actual grid environments," in *IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA 2008, Sydney, NSW, Australia, December 10-12, 2008*, 2008, pp. 572–577.

[24] B. F. Costa, M. Mattoso, and I. de Castro Dutra, "Applying reinforcement learning to scheduling strategies in an actual grid environment," *IJHPSA*, vol. 2, no. 2, pp. 116–128, 2009. [Online]. Available: https://doi.org/10.1504/IJHPSA.2009.032029

[25] W. Chen and E. Deelman, "Workflowsim: A toolkit for simulating scientific workflows in distributed environments," in *8th IEEE International Conference on E-Science, e-Science 2012, Chicago, IL, USA, October 8-12, 2012*, 2012, pp. 1–8.

[26] J. C. Jacob, D. S. Katz, G. B. Berriman, J. Good, A. C. Laity, E. Deelman, C. Kesselman, G. Singh, M. Su, T. A. Prince, and R. Williams, "Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking," *CoRR*, vol. abs/1005.4454, 2010. [Online]. Available: http://arxiv.org/abs/1005.4454

[27] B. F. Costa, M. Mattoso, and I. Dutra, "Applying reinforcement learning to scheduling strategies in an actual grid environment," *International Journal of High Performance Systems Architecture*, vol. 2, no. 2, pp. 116–128, 2009.

[28] R. Sakellariou, H. Zhao, and E. Deelman, "Mapping workflows on grid resources: experiments with the montage workflow," in *Grids, P2P and Services Computing*. Springer, 2010, pp. 119–132.

[29] Pegasus, "Workflow generator," 2010, [Online; last accessed 22-March-2019]. [Online]. Available: https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator

[30] Z. Fang, T. Yu, O. J. Mengshoel, and R. K. Gupta, "Qos-aware scheduling of heterogeneous servers for inference in deep neural networks," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, ser. CIKM '17. New York, NY, USA: ACM, 2017, pp. 2067–2070.

[31] M. Li, T. Brys, and D. Kudenko, "Introspective reinforcement learning and learning from demonstration," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '18. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 1992–1994.