# Reactive Workflow Scheduling in Fluctuant Infrastructure-as-a-Service Clouds Using Deep Reinforcement Learning

Qinglan Peng[1], Wanbo Zheng[2(✉)], Yunni Xia[1(✉)], Chunrong Wu[1], Yin Li[3], Mei Long[4], and Xiaobo Li[5]

[1] Software Theory and Technology Chongqing Key Lab, Chongqing University, Chongqing 400044, China
xiayunni@hotmail.com
[2] Data Science Research Center, Kunming University of Science and Technology, Kunming 650031, China
zwanbo@163.com
[3] Institute of Software Application Technology, Guangzhou & Chinese Academy of Sciences, Guangzhou 511000, China
[4] ZBJ Network Co. Ltd., Chongqing 401123, China
[5] Chongqing Animal Husbandry Techniques Extension Center, Chongqing 401121, China

**Abstract.** As a promising and evolving computing paradigm, cloud computing benefits scientific computing-related computational-intensive applications, which usually orchestrated in terms of workflows, by providing unlimited, elastic, and heterogeneous resources in a pay-as-you-go way. Given a workflow template, identifying a set of appropriate cloud services that fulfill users' functional requirements under pre-given constraints is widely recognized to be a challenge. However, due to the situation that the supporting cloud infrastructures can be highly prone to performance variations and fluctuations, various challenges such as guaranteeing user-perceived performance and reducing the cost of the cloud-supported scientific workflow need to be properly tackled. Traditional approaches tend to ignore such fluctuations when scheduling workflow tasks and thus can lead to frequent violations to Service-Level-Agreement (SLA). On the contrary, we take such fluctuations into consideration and formulate the workflow scheduling problem as a continuous decision-making process and propose a reactive, deep-reinforcement-learning-based method, named DeepWS, to solve it. Extensive case studies based on real-world workflow templates show that our approach outperforms significantly than traditional ones in terms of SLA-violation rate and total cost.

**Keywords:** Workflow scheduling · IaaS cloud · Quality-of-Service · Pay-as-you-go · Reinforcement learning


H. Gao et al. (Eds.): CollaborateCom 2020, LNICST 350, pp. 285–304, 2021.
https://doi.org/10.1007/978-3-030-67540-0_17

## 1 Introduction

As a useful process modeling tool, workflow has been widely applied in scientific computing, big data processing, video rendering, business process modeling, and many other fields [2]. Workflow scheduling refers to an integration of workflow tasks, with varying non-functional properties in terms of Quality-of-Service (QoS), collectively orchestrated to automate a particular task flow or computational process. Scientific workflows, e.g., weather forecasting and high-energy-physics simulating workflows, are usually high time, resource, energy, and storage-requiring. Thus, effective scheduling algorithms and methods are in high demand for guaranteeing cost-effective and timely execution of complex scientific workflows.

However, the fast-growing data and computing requirements demand more powerful computing environments in order to execute large-scale workflows in a reasonable amount of time. Traditional grid and cluster systems are mostly dedicated and statically partitioned per administration policy. These traditional systems can be inefficient in adapting to today's computing demand. Fortunately, with the rapid development of the cloud ecosystem, large-scale workflow applications are able to leverage the dynamically provisioned resources from the cloud instead of using a dedicated server or node in grids or clusters [21]. As the cloud provides resources as maintenance-free and pay-as-you-go services, users are allowed to access resources on-demand at anytime and from anywhere. They only need to identify what type of resource to lease, how long to lease resources, and how much their applications will cost. Nevertheless, according to the study of Schad *et al.* [18], the performance of Virtual Machines (VMs) in Amazon EC2 cloud vary by 24% percentages under high workload. And such fluctuations of VMs' performance have a great possibility to impact the user-perceived quality of cloud systems, especially when the Service-Level-Agreement (SLA) constrains, e.g., workflow execution time, are violated. Therefore, how to schedule workflow tasks into the fluctuant Infrastructure-as-a-Service (IaaS) cloud resources to meet the SLA constraint while reducing the cost as much as possible has become the major problem.

As a practical problem, IaaS cloud-based workflow scheduling becomes a hot research topic since it is widely acknowledged that to schedule multi-task workflow on distributed platforms is a typical NP-hard problem [6], and various existing works [1,5,8,9,15,16] in this direction addressed the multi-objective-multi-constraint cloud workflow scheduling problem by using statistic, stochastic, heuristic, and meta-heuristic methods. However, these traditional solutions might be inefficient and incur high SLA-violation rate when encounter with massive requests and fluctuate cloud performance. Recent progresses in artificial intelligence [11,19] show that learning-based approaches, especially the reinforcement-learning ones, can be highly potential in dealing with such a cloud-based workflow scheduling problem as well. In this paper, we target at the online reactive workflow scheduling problem and propose a deep-reinforcement-learning-based method, shorts for DeepWS, to solve it.

The main contributions of this work are as follows: 1) we formulate the deadline-constrained-cost-minimization IaaS-cloud-workflow scheduling problem into a continuous decision-making process and use a Markov Decision Process (MDP) to model it; 2) we define the state space and action space of the proposed MDP problem, and design an SLA and cost-aware reward function; 3) we propose a deep-reinforcement-learning-based method to solve the reactive workflow scheduling problem on the fly. Extensive case studies based on various real-world workflow templates under different deadline constraint levels are conducted to verify the effectiveness of our approach.

The remainder of this paper is organized as follows. Section 2 introduces the DQN and how it incorporates with our problem. The system model and corresponding MDP problem formulation are described in Sect. 3. We propose a DQN-based method in Sect. 4. Our experimental results and discussions are given in Sect. 5. Section 6 reviews related works. Finally, Sect. 7 concludes this paper.

## 2 Preliminary

Reactive workflow scheduling can be regarded as a continuous decision-making process where the scheduling decisions of tasks are made on the fly, and we use deep-reinforcement-learning to train the scheduling model through a repeated trial-and-error learning strategy. In reinforcement learning (RL), an action-value function $Q(s, a)$ is often used to identify the discounted future reward of taking an action $a$ at state $s$. The reward values are often stored in an action-value mapping table, shorts for Q-Table, and its update process can be expressed as follow:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a')] \tag{1}$$

where $\alpha$ is the learning rate, $r$ the direct reward of action $a$, $s'$ the resulting state of tacking action $a$ at state $s$. To overcome the limitation of state explosions,

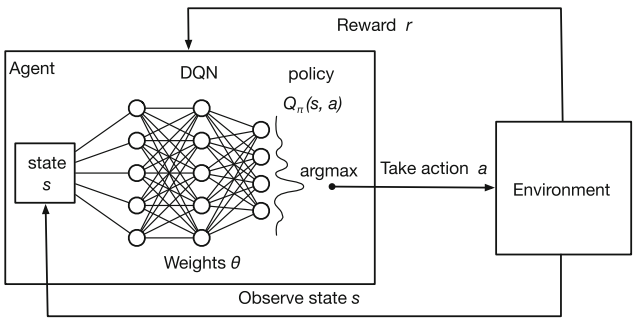



**Fig. 1.** Deep reinforcement learning

the action-value function is usually approximated by neural network for state-space saving:

$$Q(s, a; \theta) \approx Q(s, a) \tag{2}$$

Figure 1 shows the architecture of deep reinforcement learning. In this paradigm, an action-value function $Q(s, a; \theta)$ is often used to identify the discounted future reward of taking an action $a$ at state $s$, and Deep Q-Network (DQN) [11] is used to fit the mapping between actions and reward. It can be trained through minimizing the loss function $L_i(\theta_i)$ at each iteration:

$$L_i(\theta_i) = E_{s,a\sim\rho(*)}[(y_i - Q(s, a; \theta_i))^2] \tag{3}$$

where

$$y_i = E_{s\sim\epsilon}[r + \gamma \max_{a'}(s', a'; \theta_{i-1}|s, a)] \tag{4}$$

where $y_i$ is the target in the $i$-th iteration, $\rho(s, a)$ the probability distribution of state and action, and it can be through an $\epsilon$-greedy method. The gradient can be derived based on the loss function as follows:

$$\begin{aligned} \nabla_{\theta_i} L_i(theta_i) = & E_{(s,q\sim\rho(*);s'\sim\epsilon)}[(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) \\ & - Q(s, a; \theta))\nabla_{\theta_i} Q(s, a; \theta)] \end{aligned} \tag{5}$$

Then, neural networks optimization methods, i.e., Adam, SGD, RMSprop, and so on, can be employed for updating the weights of neural network according to the gradients derived in Eq. (5). In this paper, we regard the status of workflow and VMs as the states, the schedules of tasks as actions, and propose a DQN-based method to train a scheduling model to yield reactive schedules in real-time.

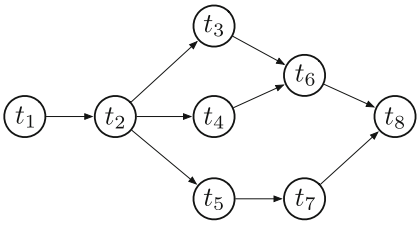
## 3 System Model and MDP Problem Formulation

In this section, we first present our system model and the corresponding reactive deadline-constrained workflow scheduling problem, then we formulate it to a continuous decision-making process using MDP model.

### 3.1 System Model

A workflow can be described as a Directed Acyclic Graph (DAG) $W = (T, E, D)$, where $T = \{t_1, t_2, ...t_n\}$ is the set of tasks, $E = \{e_{ij}|i, j \in n, i \neq j\}$ the set of edges which represent the data dependencies between different tasks, edge $e_{ij}$ indicates that the output of $t_i$ is the input of $t_j$, $D$ the user-defined deadline constraint. A task can only start when all its preceding ones are successfully executed and data transfers between them are accomplished. Figure 2 shows an workflow example with 8 tasks.

Generally, there are different types of VMs provided by IaaS providers available for custom selection, and these different types promise different prices and resource configurations in terms of, e.g., the number of cores of CPU, the size of




| VMtype / Task | *Small* | *Medium* | *Large* | *XLarge* | *2XLarge* |
|---|---|---|---|---|---|
| $t_1$ | 0.81 | 0.65 | 0.42 | 0.31 | 0.27 |
| $t_2$ | 1.76 | 1.31 | 0.94 | 0.65 | 0.57 |
| $t_3$ | 0.94 | 0.72 | 0.58 | 0.42 | 0.39 |
| ... | ... | ... | ... | ... | ... |

**Fig. 2.** An workflow example with 8 tasks and the execution of it tasks on different type of VMs.

memory and allocated storage. Therefore, for the same workflow task, its execution can vary in VMs with different types. The leased VMs can be regarded as a resource pool $P = \{VM_1, VM_2, ..., VM_m\}$. Due to the fact that the VM acquiring time is chargeable, therefore tasks are scheduled into idle VMs preferentially in the resource pool to reduce the cost. And the VMs which are in the status of idle for a certain period will be released to reduce the cost. We consider the violations to SLA are subject to penalties as well, and that penalty as a linear function similar to those proposed in [4,14,22]. Therefore, deadline-constrained-cost-minimized workflow scheduling problem over fluctuant IaaS cloud can be formulated as follow:

$$\begin{aligned} Min: \quad & C = \sum_{i=1}^{n} h(g(i)) \times f_i^{g(i)} \\ s.t: \quad & M \leq D \end{aligned} \tag{6}$$

where $g(i)$ is the function to identify which type of VM that task $t_i$ is scheduled into, $h(j)$ the function to identify cost-per-unit-time of using a type $j$ VM, $f_i^{g(i)}$ the actual execution time of task $t_i$ on the VM with type $g(i)$, thus $C$ denotes the cost of running the scientific workflow. $D$ is the user defined deadline constraint, and $M$ the actual makespan of the workflow. Note that, if a schedule violates the deadline constraint, additional penalty will be incurred to the total cost. Under this situation (i.e., $M > D$), cost $C$ can be calculated as:

$$C = \sum_{i=1}^{n} h(g(i)) \times f_i^{g(i)} + \beta \sum_{t=D}^{M} h_t \tag{7}$$

where $\beta$ the deadline constraint violation (i.e., SLA-violation) penalty rate, and $h_t$ the cost incurred by all hired VMs at time slot $t$.

### 3.2 MDP Problem Formulation

A Markov decision process (MDP) is a discrete time stochastic control process [12], it can be seen as Markov Chain with additional actions and rewards. A MDP can be expressed as a 5-tuples $(S, A, P_a, \gamma)$, where $S$ is the set of finite states, $A$ the set of finite actions, $A_s$ the action set under $s$ state,

$P_a(s, a, s') = \mathbf{Pr}(s_{t+1}|s_t = s, a_t = a)$ the transition probability from state $s$ to $s'$ if action $a$ is taken at time $t$ at state $s$, $R_a(s, a, s')$ the direct reward of the transition from state $s$ to $s'$ if an action a is performed, $\gamma \in [0, 1]$ the discount to decide the discount ratio of future reward to the previous steps. MDP provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision-maker. It is a useful tool for studying a wide range of decision making problems solved via dynamic programming and reinforcement learning.

The workflow scheduling problem can also be formulated as an MDP problem. When a task in a workflow is ready to start, the scheduling agent will decide the action that next step to take based on evaluation of the current state in terms of, e.g., resource utilization and remaining execution time permitted, of the entire workflow. After the decided action is taken, the scheduling agent gets a reward calculated based on the resulting state of the workflow.

The training goal for the MDP-based workflow scheduling problem is thus to find a policy $\pi$ which leads to an capable of maximizing the expected future rewards. It is usually assumed that the future reward is subject to a discount rate $\gamma$. To be specific, the total discounted reward at state $s$ can be calculated as follow:

$$R_s = \sum_{s'=s}^{T} \gamma^{s'-s} R_{s'} \tag{8}$$

where $T$ is the terminate state, which means there are no remaining tasks in the workflow to be scheduled (i.e., the workflow has finished), $R_{s'}$ the reward at state $s^{'}$.

## 4 Reinforcement-Learning-Based Solution

To solve the MDP problem we formulated previously, in this section, we propose a deep-reinforcement-learning-based approach, short for DeepWS, to train the scheduling model to yield reactive schedules in real-time.

### 4.1 MDP Definitions

We first present the detailed MDP definitions, including its state space, observation, action space, and reward function, of the proposed reactive workflow scheduling problem.

**1) State-space:** In DeepWS, the state is defined to the status of the whole system when a task in the workflow is ready to be scheduled. Therefore, every task in a workflow has its corresponding state, and we use $s_i$ to denote the state when task $t_i$ is ready to be scheduled. For example, Fig. 3 shows an example of state transition in a whole scheduling process. Suppose that there is a workflow with 4 tasks to be scheduled, the first state of the system will be $s_1$ because the task $t_1$ is the entry task, and only $t_1$ is able to start at the beginning. Then the agent will schedule $t_1$ to a certain VM to be executed. After $t_1$ has accomplished,

system will transfer into $s_2$ or $s_3$ to schedule $t_2$ and $t_3$. If data transfer $e_{(1,2)}$ is faster than $e_{(1,3)}$, then state will transfer from $s_1$ to $s_2$, otherwise, from $s_1$ to $s_3$. If data transfer $e_{(1,2)}$ and $e_{(1,3)}$ accomplish at the same time, we random choose an available state to step into, note that, the transition order $\{s_1 \rightarrow s_2 \rightarrow s_3\}$ is equal to $\{s_1 \rightarrow s_3 \rightarrow s_2\}$ under such a situation, because the schedule decisions will be made in millisecond level, and they can be regarded as simultaneous operations. Finally, if $t_2$ and $t_3$ have accomplished, system will step into $s_4$, which is obvious the terminal state, and the schedule process will end after $t_4$ is scheduled at $s_4$.

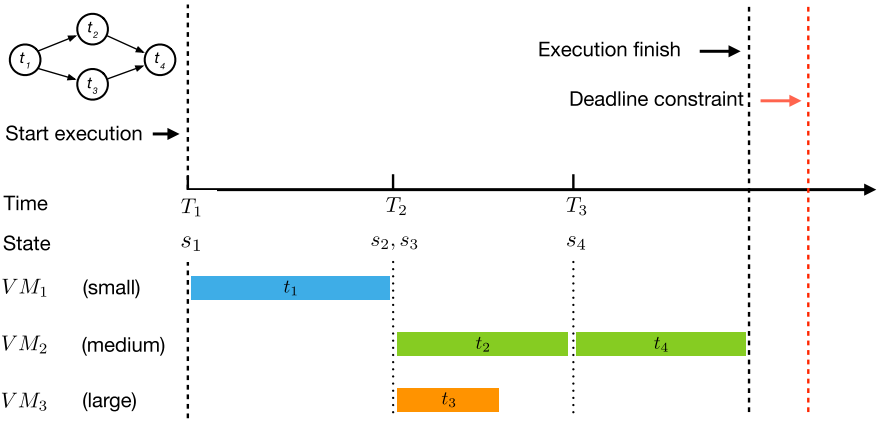



**Fig. 3.** An example of state transition and online scheduling.

**2) Observation:** In DeepWS, observation is the status of whole system at a certain state, it can be represented as a $(l + 5)$-tuple $o_i = \{t_i, \tau_1, \tau_2, ..., \tau_l, G, O, P, H\}$, where $l$ is the number of VM types available for users to choose, $t_i$ the task which need to be scheduled at current state, $\tau_j$ the estimated execution time of $t_i$ in the VM with type $j$ based on current performance of IaaS cloud, $G$ the deadline left time, $O$ the cumulative cost from the beginning to current state, $P$ the time pasted from the beginning, $H$ the longest remain time for task $t_i$ to meet the deadline constraint, and it can be obtained as follow:

$$H = D - P - \sum_{t \in \{U - t_i\}} \tau_t^e \tag{9}$$

where $U$ is the set of tasks in the crucial path of unfinished part of DAG, $\tau_t^e$ the estimated execution time of task $t$ in most expensive type of VM.

When a task is ready to schedule, the agent first gets the observation of the system at the current state. Then it make a decision on next action, i.e., which type of VM this task is going to schedule to with the help of schedule policy $\pi$, i.e, trained scheduling model. Finally, the action will be performed and its corresponding reward will be evaluated to update current policy.

**3) Action space:** As we mentioned previously, we define the action $a_i$ to the type of VM which task should be scheduled into at state $s_i$. For example, there are three types of VM available for custom selection: *small*, *medium*, and *large*, and we use number 1, 2, and 3 to identify them. Suppose that the action made by a agent at state $s_i$ for task $t_i$ is $a_i = 3$, and it indicates that $t_i$ is going to be scheduled into a medium VM to be executed.

**4) Reward function:** The design of the reward function is the key step in reinforcement learning problem formulation. It tells agent that which kind of action under a certain state is proper and encouraging, and which kind of action is discourage and will be punished. The reward function in DeepWS is designed as follow:

$$R(s) = \begin{cases} \epsilon, & s \neq \mathbb{T} \\ \xi, & s = \mathbb{T} \wedge f > D \\ (\frac{f-F}{D-F})^{\zeta}, & s = \mathbb{T} \wedge f \leq D \end{cases} \tag{10}$$

where $s$ is the current state, $\mathbb{T}$ the terminal state, $f$ the final makespan the schedule, $F$ the fastest estimated finish time of a workflow, it can be estimated by assuming that all tasks are scheduled into fastest VM instances, $D$ the user defined deadline, and $\zeta$ the scale coefficient. In DeepWS, we define that only actions made at the terminal state can get nonzero rewards, and these rewards will effect previous actions' rewards in a discount way as shown in Eq. (8). Action series which result in missing deadlines will be punished, we use $\xi$, a negative real number which is infinitely close to 0, to present such a punishment to make a clear comparison with positive rewards got by those meeting deadline ones.

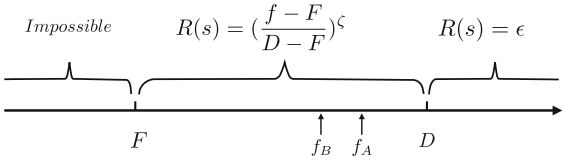



**Fig. 4.** Rewards distribution.

Figure 4 shows an illustrative example of the distribution of the reward function. The makespan of a workflow should be within $[F, +\infty)$ theoretically, once the deadline is missed, the reward will be $\epsilon$, where $\epsilon \to 0^-$. The reward will be in the range of $(0, 1]$ when makespan within $[F, D]$, the more close makespan to $D$, the higher reward will get. Intuitively, this reward reflects how close does a makespan to the deadline constraint when the deadline is met. Figure 4 also shows an example of scheduling policy comparison, for the same workflow to be scheduled, the makespan of workflow scheduled by policy $\pi_A/\pi_B$ are $f_A/f_B$, which both meeting the deadline constraint. We say that policy $\pi_A$ is better than policy $\pi_B$, because the longer time left between actual makespan and deadline, the more likely to employ cheaper VM to reduce cost while meeting the deadline. The scheduling target in this paper is to reduce cost while meeting the deadline constraint, therefore the reward function is design to lead the agent to learn how to make full use of every possible time slot to reduce cost.

### 4.2 Environment Emulator Design

Agents in reinforcement learning learn policies over time through repeated interactions with the environment. For this purpose, we consider a simulated environment through which policies of scheduling workflow over IaaS clouds can gradually evolve. We use *env* to indicate a instance of the environment emulator, which has the following functions:

- *env*.Init($W$): it initializes an environment instance with a workflow request $W$;
- *env*.GetTaskToSchedule(): it returns a set of tasks which are ready to execute/schedule;
- *env*.AssignTaskToVM($t_i$, $P$, $VMType$): it assigns task $t_i$ into a VM with type $VMType$. As shown in Procedure 1, it searches all idle VMs with the desired VM types in the resource pool. Otherwise, it simply releases a new VM with type $VMType$ and allocates $t_i$ to it;
- *env*.isDone(): It returns $True$ if the workflow is finished and otherwise $False$.

The design of emulator consider multiple factors, e.g., pay-as-you-go pricing, unlimited heterogeneous resources, performance fluctuation, VM acquiring and releasing delay, etc. Given the training set(i.e., historical workflow requests), the agent learns scheduling policies through repeatedly interact (e.g., make scheduling decisions) with the environment instance.

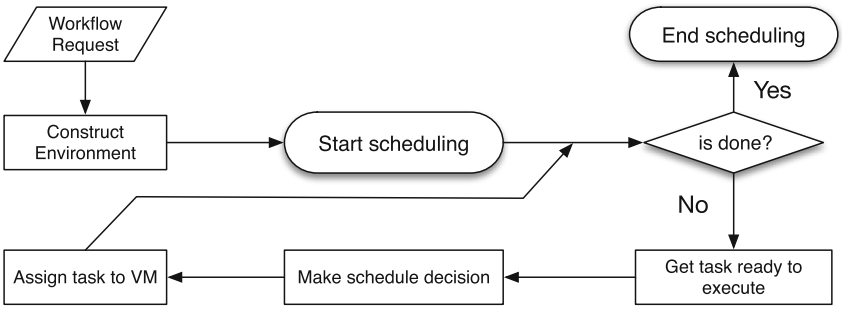



**Fig. 5.** The process of interacting with the emulator for an RL agent.

Figure 5 shows the process of interacting with the emulator for an RL agent, it can be seen that it begin with accepting a certain workflow request. Then, tasks ready to start are and assigned to VMs according to the evolving schedule policy, this process loops until all tasks in a workflow are scheduled.

### 4.3 Training Algorithm

In DeepWS, DQN is employed as an approximate function to map the state to the probability of different actions can be taken. The input of DQN is the

**Procedure 1:** AssignTaskToVM

**Input**: Task to schedule $t$; Resource pool $P$; VM type *VMType*
**Output**: VM instanace *vmi*;

*VMCancidates* ← ∅
**foreach** $vm \in P$ **do**
  **if** *vm*.isIdle() **and** vm.type = *VMType* **then**
    *VMCancidates*.append(*vm*)
**if** *VMCancidates* = ∅ **then**
  Find a new VM instance *vmi* with type of *VMType* and schedule task $t$ into it to execute
**else**
  Select the VM $vmi \in VMCancidates$ with shortest idle time and schedule task $t$ into it to execute
**return** *vmi*

**Algorithm 2:** Training algorithm for DQN

**Input**: Training workflow set $W$
**Output**: Scheduling model $M$

initialize ReplayMemory $RM$ with random scheduling policy
initialize eval-DQN *eNN* with weights $\sim N(0, 0.1)$
initialize target-DQN *tNN* with weights $\sim N(0, 0.1)$
**foreach** $w \in W$ **do**
  $env \leftarrow$ Environment(w)
  **while** env.isDone() is **False do**
    $tasks \leftarrow$ env.getTasksToSchedule()
    **foreach** *task* $t \in tasks$ **do**
      $s_t \leftarrow$ env.getCurrentState()
      $a_t \leftarrow \epsilon$-greedy($\max_a eNN(s_t, a)$, $\epsilon$)
      $r_t \leftarrow$ env.ScheduleTaskToVM($t$, $a_t$)
      $s_{t+1} \leftarrow$ env.getCurrentState()
      RM.put($s_t$, $a_t$, $r_t$, $s_{t+1}$)
      $s_j, a_j, r_j, s_{j+1} \leftarrow$ mini batch randomly select from $RM$
      **if** env.isDone() **then**
        $y = r$
      **else**
        $y = r_j + \gamma \max_{a'} tNN(s_{j+1}, a')$
      $loss \leftarrow (y - eNN(s_j, a_j))^2$
      Perform a gradient descent step on *loss* with respect to the *tNN*
      Every $C$ steps reset $tNN \leftarrow eNN$
**return** *eNN*

observation of the current system, and the output is the probability of actions, i.e., which type of VM that the task should be scheduled into. A complete process

(as shown in Fig. 5) of the scheduling of a workflow can be seen as an episode, and DQN is trained through such iterations of episodes.

To train DQN effectively, memory reply and delayed update [11] are employed to make it immune to the correlations between training set. Memory replay aims at making the agent learn from random experiences. All decision steps $(s_i, a_i, r_i, s_{i+1})$ are recorded into a memory set during training and a mini-batch of steps is randomly selected at each state to train the DQN. The delayed update aims at decreasing the fluctuation of the trained policy due to the frequent change of an action-value function. It maintains two DQN (target-DQN and eval-DQN) at the same time, eval-DQN is responsible for decision making and loss backpropagation, while target-DQN responsible for calculating the loss. The weights of eval-DQN are copied into target-DQN at stated intervals. The detail of the training procedure for workflow scheduling model is shown in Algorithm 2.

## 5 Experiment and Discussion

To verify the effectiveness of DeepWS, we conduct a series case studies to evaluate the performance of our method and its peers in terms of makespan, deadline hit rate and cost.

### 5.1 Experiment Setting

We consider SCS [9], PSO [16] and IC-PCP [1] these three state-of-art workflow scheduling algorithms as baseline algorithms since their scheduling targets are minimizing the cost while meeting the deadline constraint, which is similar with us. SCS is a dynamic heuristic scheduling method, scheduling decisions are pre-generated but task consolidations are performed at run time to reduce the total cost; while PSO is a static meta-heuristic scheduling method, which takes VM performance fluctuation into consideration and uses an overrated tasks execution time to cope with fluctuation; IC-PCP is a static heuristic scheduling method which does not consider VM performance fluctuation.

We consider there are five types of VM available (i.e., *small*, *medium*, *large*, *xlarge*, and *xxlarge*) in our experiments, and their configurations, pricing[1], and benchmark socres[2] (i.e., performance) are shown in Table 1. According to the study of Schad *et al.* [18], the performance of VM can vary up to 24%. In our experiments, the fluctuation of VM performance is set to follow a normal distribution with 15% mean and 20% standard deviation, which is same as the study of Rodriguez *et al.* [16]. To further evaluate the performance of DeepWS and its peers under various levels of the deadline constraint, we set four deadline constraint levels, i.e., *loose*, *moderate*, *medium*, and *tight*, for every cases. The actual deadline $DL$ for a workflow is determined by deadline constraint factor $\alpha \in \{0.2, 0.4, 0.6, 0.8\}$ and it can be calculated as follow:

$$DL = F + (L - F) \times \alpha \tag{11}$$

[1] https://aws.amazon.com/cn/ec2/pricing/on-demand/.
[2] http://browser.geekbench.com/.

**Table 1.** VM configurations.

| VM type | vCPU | Memory | Benchmark | Price (cent per hour) |
|---|---|---|---|---|
| t3.small | 1 | 2G | 4833 | 2.08 |
| t3.medium | 1 | 4G | 4979 | 4.16 |
| c5.large | 2 | 4G | 6090 | 8.5 |
| c5.xlarge | 4 | 8G | 11060 | 17 |
| c5.xxlarge | 8 | 16G | 17059 | 34 |

where $F$ is the fastest estimated finish time of workflow request, $L$ the longest estimated finish time, it can be estimated by assuming only the cheapest type of VM is available. Obviously, deadline constraint will never be met when $\alpha = 0$, thus we set four deadline constraint intervals to identify DeepWS and its peers' performance under different levels of deadline.

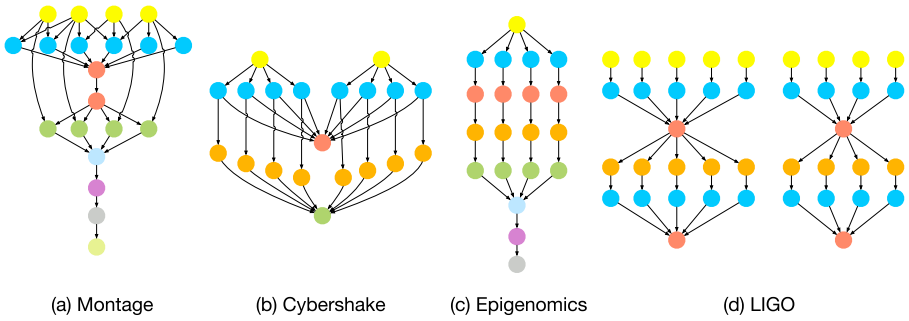



**Fig. 6.** Workflows used in experiments.

Pegasus project[3] has released a lot of real-world scientific workflow for academic research. As shown in Fig. 6, we use Montage, Cybershake, Epig, and LIGO these four cases to conduct experiments. For each case, 100 different workflows are tested by four algorithms and their performance in terms of makespan and cost are recorded.

We consider four layers of DQN with 8 neurons in the input layer, 256 neurons and 128 neurons in hidden layers, and 5 output neurons in out layer. We update DQN's weights by using the RMSProp algorithm with a learning rate of 0.001. The max number of training epochs is set to 200, the replay memory set length is set to 2000, the mini batch size is set to 32, and target network update frequency is set to 20. The $\epsilon$ value which is used in an $\epsilon$-greedy is set to range from 0.3 to 0.05, and the scale coefficient $\zeta$ is set to 3. According to the latest Amazon

[3] https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator.

compute SLA[4], service credits will be compensated to users to cover their future bills if the included services do not meet the service commitment. And the service credit percentage for monthly uptime percentage ranged from 99% to 99.5% is 25%, which covers the most of SLA violation cases[3,10], thus we set the ceiling of the penalty $\beta$ is a quarter of VM real cost in this paper. The training environment emulator is implemented by Python and the training algorithm is implemented with the help of PyTorch. The training set for each case are from the Pegasus project but the task sizes are randomly generated.

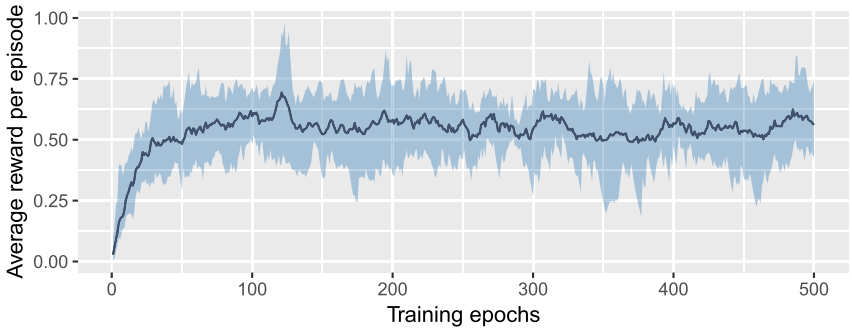



**Fig. 7.** Average reward got by agent during training.

Figure 7 shows the average reward got during the training in Montage workflow. We can see that the average reward shows its convergence after 100 epochs, which indicates the effectiveness of our designed reward function.

### 5.2 Makespan and Deadline Hit Rate Evaluation

We first evaluate the makespan and deadline hit rate. Figure 8 shows a makespan comparison between DeepWS and its peers for different workflows under different deadline constraints. The red dot line represents the average deadline of 100 different workflows for each case. It can be clearly seen that the makespan fluctuations of DeepWS is slight, and its Q3 values are always below but near to the average deadline constraint no matter what constraint levels are put, which guarantees good cost-effectiveness while meeting the deadline constraint.

Table 2 shows the deadline hit rates of different algorithms under different configurations. We can observe that our method, i.e., DeepWS, can achieve higher deadline hit rates in most cases. To be specific, our approach gets 82.5%, 43.75%, 72%, and 89.5% higher deadline hit rates than SCS on average; 61.25%, 33.5%, 50.25%, and 53.5% higher than PSO; and 11.25%, 11.75%, 25.25%, and 13.5% higher than IC-PCP. That is because as a reactive method, DeepWS is able to aware of the VM performance fluctuation at runtime and the scheduling decisions are made based on real-time system status. Besides, benefits from

[4] https://aws.amazon.com/compute/sla/.

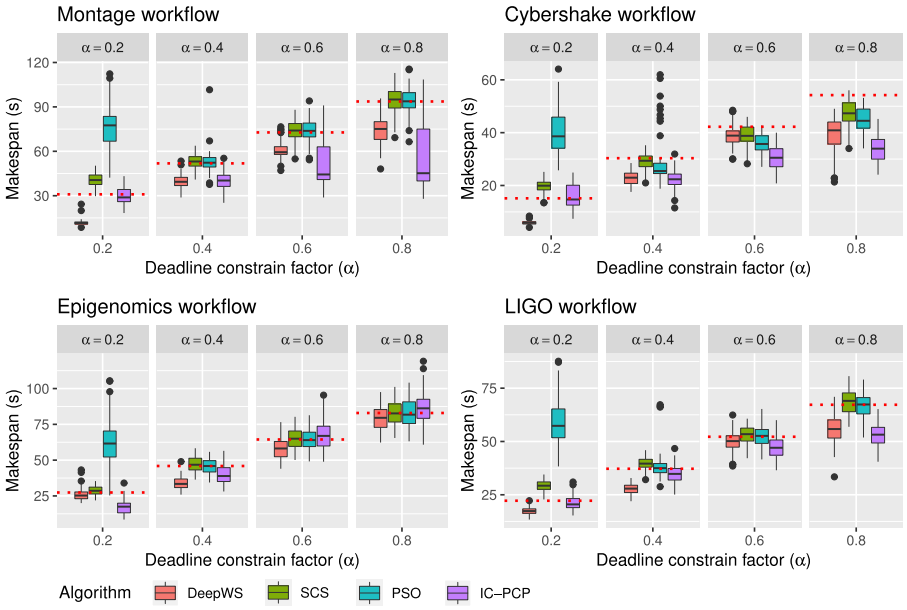



**Fig. 8.** Makespan comparison. (Color figure online)

**Table 2.** Deadline hit rates comparison

| Algorithms | $\alpha$ | Montage | Cybershake | Epigenomics | LIGO |
|---|---|---|---|---|---|
| DeepWS | 0.2 | 88% | 100% | 79% | 100% |
| | 0.4 | 98% | 99% | 77% | 100% |
| | 0.6 | 97% | 91% | 95% | 80% |
| | 0.8 | 100% | 100% | 100% | 100% |
| SCS | 0.2 | 0% | 0% | 0% | 0% |
| | 0.4 | 2% | 61% | 4% | 0% |
| | 0.6 | 10% | 72% | 12% | 6% |
| | 0.8 | 41% | 82% | 47% | 16% |
| PSO | 0.2 | 0% | 0% | 0% | 0% |
| | 0.4 | 47% | 80% | 45% | 54% |
| | 0.6 | 42% | 85% | 48% | 57% |
| | 0.8 | 49% | 91% | 57% | 55% |
| IC-PCP | 0.2 | 59% | 51% | 32% | 72% |
| | 0.4 | 98% | 96% | 41% | 73% |
| | 0.6 | 85% | 96% | 81% | 83% |
| | 0.8 | 96% | 100% | 96% | 98% |

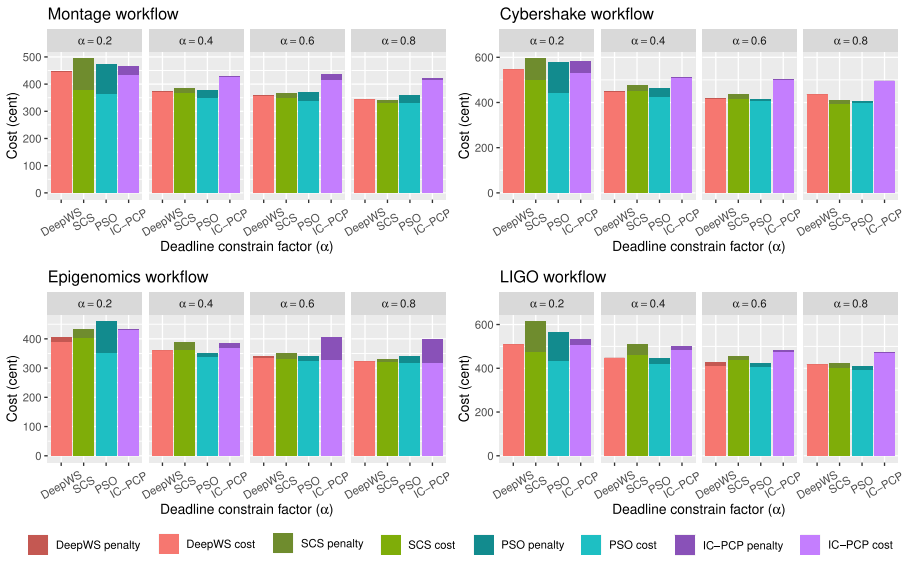



**Fig. 9.** Cost comparison.

deep-reinforcement-learning with strong decision making capability, it can make smarter scheduling decisions than its peers, especially when deadline constraint is tight.

### 5.3 Cost Evaluation

We also compare the total cost of DeepWS and its peers. As shown in Fig. 9, the costs incurred by deadline missing (i.e., SLA-violation) penalty are represented as the dark color, and the costs incurred by leasing VMs are represented as light color. If we ignore SLA-violation penalty, as shown in light-colored bars, DeepWS does not show its superiority in cost saving. But the the SLA-violation penalty is taken into consideration, we find that DeepWS can achieve lower cost in most of the cases. More specifically, our approach gets 3.32%, 4.68%, 2.45%, and 6.71% lower cost than SCS in for workflows on average; 3.21%, 4.54%, 2.06%, and 5.84% lower than PSO; and 6.79%, 8.32%, 4.87%, and 6.89% lower than IC-PCP. That is because DeepWS can always make scheduling decisions based on current VM performance, once the deadline is going to miss, it will employ faster but expensive VMs to avoid it. DeepWS beats IC-PCP because IC-PCP is too pessimistic, which represented in that there is a lot of spare time between workflow executions completed and final deadline as shown in Fig. 8. And that time has a great probability to be further transformed into more cost-saving while meeting the deadline constraint if we employ some proper cheaper VMs. The reward function of DeepWS is designed to make the full use of every possible time slot to reduce cost while meeting the deadline, and that makes our method outperforms than its peers.

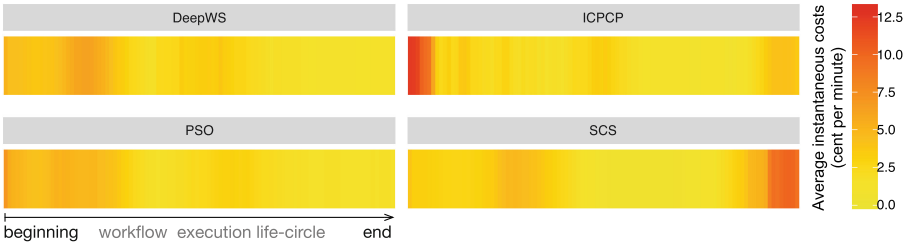



**Fig. 10.** Instantaneous costs comparison.

### 5.4 Further Analysis

To find out the difference in detail and further reveal where does the gain come from, we record the execution details of DeepWS and its peers including how many VMs have been leased to finish the workflows, what is the types of the leased VMs, and how much does their resource pools cost instantaneously.

Figure 10 shows the instantaneous costs of DeepWS and its peers when the deadline constraint level is set to medium, i.e., $\alpha = 0.6$. Instantaneous cost is the total cost incurred by VMs and EBS service in a pricing slot (one minute in this paper), which can be used to measure the pace of workflow execution, the integral of instantaneous cost over time come to the total cost. Intuitively, the more expensive VMs in resource pool are hired, the high instantaneous cost will be, and the faster makespan will be obtained. We can clearly see that the execution of schedules generated by PSO shows an excellent stability feature, where there are no sharp instantaneous cost rise or fall. That is because PSO is a kind of meta-heuristic method which generators schedules on the view of the whole workflow, thus its workflow execution pace presents a smooth process. DeepWS performs closely to PSO, there are no obvious fluctuations on instantaneous cost during the whole execution process. On the contrary, obvious changes are found in IC-PCP and SCS methods. IC-PCP performs too pessimistic at the beginning compared with DeepWS and PSO. VMs with high performance but expensive are leased at the beginning in order to meet the deadline, which leads to the high instantaneous cost at the beginning. Later, it realizes that there is still enough time to meet the deadline and thus decides to slow down the pace of execution to reduce cost. In the contrary, SCS performs too optimistic at the beginning which represented in the low instantaneous cost in the early part of its workflow execution process. Then it finds that it is hard to meet the deadline at the current pace of execution, and thus employs expensive VMs with higher performance to speed up the execution, which represented in the high instantaneous cost at the end of the execution process.

Figure 11 shows the proportions of different VM types hired by different methods. It can be clearly seen that the VMs leased by DeepWS and PSO are relatively cheap, mainly *small* type, *medium* type and *large* type, seldom relatively expensive VMs such as *xlarge* and *xxlarge* ones. On the contrary, these

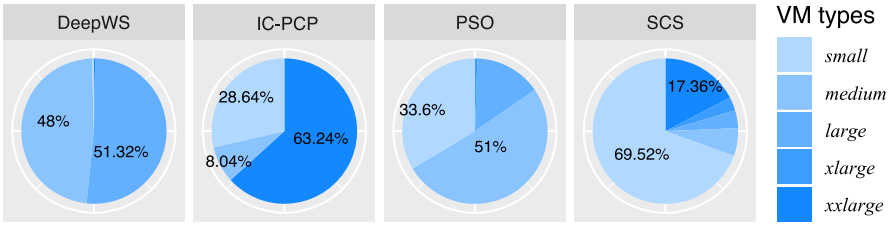



**Fig. 11.** VM type distribution comparison of rented VMs.

expensive VMs account for a considerable portion of VMs hired by IC-PCP and SCS approaches. Generally, expensive VMs with high performance have low cost-effective due to the fact that the price of VMs and the performance is nonlinear (as shown in Table 1). Therefore, the more expensive VMs are used the poor cost-effective a schedule will be. But employing expensive has the ability to speed up the execution to meet the deadline, which is essential especially when constraints are tight. Thus, the smart use of different types of VMs is the key to reduce total cost while meeting the deadline. Compared with baselines, expensive VMs (i.e., *large*, *xlarge*, and *xxlarge*) leased by DeepWS only takes up 0.68% of its resource pool, which is the lowest proportion among its peers, and that indicates that DeepWS is capable of well controlling the execution pace of workflows to achieve higher cost-effective while meeting the deadline constraints.

## 6 Related Work

According to the taxonomy by Rodriguez *et al.* [17], existing workflow scheduling algorithms fall into two major categories: static and dynamic (i.e., reactive). Static algorithms usually generate the schedule plans in advance and then schedule workflow step-by-step according to the plan, while the reactive ones see the decision-making for workflow scheduling as an evolvable and refinable process. We take recent static and reactive studies as related work and review them here.

### 6.1 Static Workflow Scheduling

Static scheduling methods are easy to implement and widely applied in traditional distributed computing systems, grids, cluster-based systems, and cloud-based workflows as well. E.g., Malawski *et al.* [8]. proposed workflow scheduling method for a muti-cloud environment with infinite heterogeneous VMs. They formalized the scheduling problem as an optimization problem aiming at minimization of the total cost under the deadline constraint and developed a Mixed-Integer-Program (MIP) algorithm to solve it. Rodriguez *et al.* [16] proposed Particle-Swarm-Optimization (PSO) based scheduling algorithm. They assumed infinite heterogeneous resources and elastic provisioning and Virtual Machine (VM) performance variation are considered in their system model. Li *et al.* [5]

proposed a fluctuation-aware and predictive scheduling algorithm. They consider time-varying VM performance and employed an Auto-Regressive-Moving-Average (ARMA) model to predict the future performance of cloud infrastructures. Then they employed such prediction information in guiding workflow scheduling and used a genetic algorithm to yield scheduling plans. Abrishami *et al.* [1] extended the Partial-Critical-Paths (PCP) algorithm, developed for utility grid, into the Cloud-Partial-Critical-Paths (IC-PCP). IC-PCP aims at minimizing execution costs while meeting deadline constraints. Zhu *et al.* [24] proposed a evolutionary multi-objective workflow scheduling method, named EMS-C. They modeled the scheduling problem aiming at make-span and cost reduction and developed a meta-heuristic algorithm based on NSGA-II to solve it.

To sum up, static methods can generate optimal or near-optimal schedules featured by a global optimization view. However, due to the fact that the schedules are generated in advance and they are unrefinable at run time, they methods suffer from frequent violations to SLA especially when the supporting cloud infrastructures are under high stress, prone to performance fluctuations.

### 6.2 Reactive Workflow Scheduling

Reactive methods see the scheduling process as a gradual and evolvable one. They consider the decision-making to be reactive to real-time events, e.g., performance fluctuations and losses of resource connectivity. For example, Wu *et al.* [20] employed a machine-learning-based method to predict the future moving trajectory of mobile users, then they develop a heuristic to achieve a mobility-aware online workflow scheduling. Rodriguez *et al.* [15] presented an Execution-Time-Minimization-Budget-Driven method, BAGS, for clouds with finer-grained pricing schemes. The proposed method first partition DAGs into bags of tasks (BoTs), then distributes budgets into these BoTs, provisions cloud resources according to the planed budgets, and finally executes the tasks maintained in a scheduling queue in a first-in-last-out manner. Mao *et al.* [9] proposed a Scaling-Consolidation-Scheduling (SCS) algorithm for workflow scheduling for IaaS clouds. They developed an auto-scaling mechanism that allows VMs to be allocated and deallocated dynamically based on system status. Zhou *et al.* [23] considered both on-spot and on-demand instances to support the running of workflows. They first developed an A*-based algorithm for assigning a combination of spot and on-demand instances for every task in a workflow, then they proposed a heuristic with the feature of instance consolidation and reusing to refine the schedule at the run time. Poola *et al.* [13] presented a heuristic, dynamic, and fault-tolerant scheduling approach. They considered both on-demand and on-spot instances in the resource pool, and develop a fault-tolerance-guaranteed heuristic to achieve fault-tolerant by switching on-spot instance to on-demand one when the deadline constraint is likely to be violated. Malawski *et al.* [7] proposed a Dynamic-Provisioning-Dynamic-Scheduling (DPDS) algorithm, that is capable of dynamically scaling the VM pool and scheduling a group of inter-related workflows under budget and deadline constraints. It creates an initial

pool of homogeneous VMs with as many resources as allowed by the budget and updates it at runtime by monitoring resource utilization.

To sum up, reactive methods are able to make or refine scheduling decisions at the run time, they can thus achieve lower SLA-violation rate However, their limited task-level view of the problem restricts the possibility of designing an effective algorithm to get high-quality schedules as static ones. Fortunately, recent great progress made by deep-reinforcement-learning in complex sequential decision making brings us a new opportunity to this problem. In this paper, we propose a deep-reinforcement-learning-based method to get reactive schedules in real-time.

## 7 Conclusion and Future Study

This paper targets the reactive workflow scheduling problem over the IaaS cloud, where the performance of VMs fluctuates during the execution. We formulate the scheduling problem as an MDP problem and develop a deep-reinforcement-learning-based method, named DeepWS, to solve it. Experiments show that the proposed DeepWS can get lower running costs and SLA-violation rates.

The following issues should be well addressed as future work: 1) Some time-series prediction methods, e.g., ARIMA or LSTM neural networks can be used to predict the performance of VMs to achieve a further improvement; 2) this paper only consider on-demand VM instances as workers, resource pool constructed by both on-demand spot instances will be considered to further reduce cost.

**Acknowledgement.** This work is supported in part by the Graduate Scientific Research and Innovation Foundation of Chongqing, China (Grant No. CYB20062 and CYS20066), and the Fundamental Research Funds for the Central Universities (China) under Project 2019CDXYJSJ0022.

## References

1. Abrishami, S., Naghibzadeh, M., Epema, D.H.: Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. Future Gener. Comput. Syst. **29**(1), 158–169 (2013)
2. Belhajjame, K., Faci, N., Maamar, Z., Burégio, V., Soares, E., Barhamgi, M.: On privacy-aware eScience workflows. Computing 1–15 (2020)
3. Christophe, C., et al.: Downtime statistics of current cloud solutions. In: International Working Group on Cloud Computing Resiliency. Technical report (2014)
4. Irwin, D.E., Grit, L.E., Chase, J.S.: Balancing risk and reward in a market-based task service. In: Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing, pp. 160–169. IEEE (2004)
5. Li, W., Xia, Y., Zhou, M., Sun, X., Zhu, Q.: Fluctuation-aware and predictive workflow scheduling in cost-effective infrastructure-as-a-service clouds. IEEE Access (2018)
6. Li, X., Yu, W., Ruiz, R., Zhu, J.: Energy-aware cloud workflow applications scheduling with geo-distributed data. IEEE Trans. Serv. Comput. (2020)

7. Malawski, M., Juve, G., Deelman, E., Nabrzyski, J.: Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. In: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–11 (2012)
8. Malawski, M., Figiela, K., Bubak, M., Deelman, E., Nabrzyski, J.: Scheduling multilevel deadline-constrained scientific workflows on clouds based on cost optimization. Sci. Program. **2015**, 5 (2015)
9. Mao, M., Humphrey, M.: Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In: 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pp. 1–12. IEEE (2011)
10. Maurice, G., et al.: Downtime statistics of current cloud solutions. In: International Working Group on Cloud Computing Resiliency. Technical report (2012)
11. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
12. Papadimitriou, C.H., Tsitsiklis, J.N.: The complexity of Markov decision processes. Math. Oper. Res. **12**(3), 441–450 (1987)
13. Poola, D., Ramamohanarao, K., Buyya, R.: Fault-tolerant workflow scheduling using spot instances on clouds. Procedia Comput. Sci. **29**, 523–533 (2014)
14. Rana, O.F., Warnier, M., Quillinan, T.B., Brazier, F., Cojocarasu, D.: Managing violations in service level agreements. In: Rana, O.F., Warnier, M., Quillinan, T.B., Brazier, F., Cojocarasu, D. (eds.) Grid Middleware and Services, pp. 349–358. Springer, Boston (2008). https://doi.org/10.1007/978-0-387-78446-5_23
15. Rodriguez, M.A., Buyya, R.: Budget-driven scheduling of scientific workflows in IaaS clouds with fine-grained billing periods. ACM Trans. Auton. Adapt. Syst. (TAAS) **12**(2), 5 (2017)
16. Rodriguez, M.A., Buyya, R.: Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. IEEE Trans. Cloud Comput. **2**(2), 222–235 (2014)
17. Rodriguez, M.A., Buyya, R.: A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments. Concurr. Comput. Pract. Exp. **29**(8), e4041 (2017)
18. Schad, J., Dittrich, J., Quiané-Ruiz, J.A.: Runtime measurements in the cloud: observing, analyzing, and reducing variance. Proc. VLDB Endow. **3**(1–2), 460–471 (2010)
19. Vinyals, O., et al.: StarCraft II: a new challenge for reinforcement learning. arXiv preprint arXiv:1708.04782 (2017)
20. Wu, C., Peng, Q., Xia, Y., Lee, J.: Mobility-aware tasks offloading in mobile edge computing environment. In: 2019 Seventh International Symposium on Computing and Networking (CANDAR), pp. 204–210. IEEE (2019)
21. Wu, Q., Zhou, M., Zhu, Q., Xia, Y., Wen, J.: MOELS: multiobjective evolutionary list scheduling for cloud workflows. IEEE Trans. Autom. Sci. Eng. **17**(1), 166–176 (2019)
22. Yeo, C.S., Buyya, R.: Service level agreement based allocation of cluster resources: handling penalty to enhance utility. In: IEEE International Cluster Computing, pp. 1–10. IEEE (2005)
23. Zhou, A.C., He, B., Liu, C.: Monetary cost optimizations for hosting workflow-as-a-service in IaaS clouds. IEEE Trans. Cloud Comput. **4**(1), 34–48 (2016)
24. Zhu, Z., Zhang, G., Li, M., Liu, X.: Evolutionary multi-objective workflow scheduling in cloud. IEEE Trans. Parallel Distrib. Syst. **27**(5), 1344–1357 (2016)