Cost-Aware Dynamic Multi-Workflow Scheduling in Cloud Data Center Using Evolutionary Reinforcement Learning[*]

Victoria Huang[1], Chen Wang[1], Hui Ma[2], Gang Chen[2], and Kameron Christopher[1]

[1] National Institute of Water and Atmospheric Research, Wellington, New Zealand
{victoria.huang, chen.wang, kameron.christopher}@niwa.co.nz
[2] Victoria University of Wellington, Wellington, New Zealand
{hui.ma, aaron.chen}@ecs.vuw.ac.nz

Abstract. The Dynamic Multi-Workflow Scheduling (DMWS) problem aims to allocate highly complex tasks modeled as workflows to cloud resources while optimizing workflow brokers' interests. A workflow broker offers workflow execution services to end-users with agreed Service Level Agreements (SLA) while reducing its total VM rental fees in the meantime. Most existing DMWS-related research works focus on minimizing the workflow makespan by using either heuristics or hyper-heuristics techniques. However, these techniques were either designed for static workflow scheduling based on prior workflow information and/or the simplified cloud environment. In this paper, the DMWS problem is formulated to collectively minimize VM rental fees and SLA violation penalties. Moreover, we introduce a novel priority-based deep neural network scheduling policy that can flexibly adapt to a changing number of VMs and workflows. To train the new policy, a new Evolutionary Strategy based Reinforcement Learning (ES-RL) is developed and implemented. Different from gradient-based deep reinforcement learning algorithms, ES-RL has its advances in effectively training population-based and generally applicable policies in parallel as well as robustness to hyper-parameter settings. Our experiments with real-world datasets show that ES-RL can effectively train scheduling policies that can significantly reduce the costs by more than 90% compared to the state-of-the-art scheduling policies.

Keywords: Dynamic workflow scheduling · Cloud computing · Reinforcement learning · SLA violation · Evolutionary strategy.

1 Introduction

Large-scale and highly complex computational applications (e.g., weather forecasting and Tsunami prediction) are usually modeled as workflows in cloud [17,

[*] This work is in part supported by the NZ Government's Strategic Science Investment Fund (SSIF) and the New Zealand Marsden Fund with the contract number (VUW1510), administrated by the Royal Society of New Zealand.

18]. A workflow consists of a set of inter-dependent tasks connected by directed edges. These workflows are often outsourced to workflow brokers that offer workflow execution services to users [29,31]. A workflow broker usually uses computation resources, e.g., Virtual Machines (VMs), leased from cloud providers [10,28] to reduce the maintenance cost [28,29]. Service Level Agreements (SLAs) are often established between the users and the workflow brokers [28]. Brokers are highly motivated to comply with the commitments in SLAs, e.g., deadline constraints, in order to avoid paying SLA violation penalties [28, 30].

The process of *Workflow Scheduling* (WS) starts from users dynamically submitting workflows to brokers along with specified deadlines. Upon receiving a workflow, the broker makes scheduling decisions in real time which include the selection of VM resources (e.g., the VM number and types) and allocation from workflow tasks to VMs. Often, a broker needs to schedule multiple workflows for a customer. The goal of the broker is to maximize its profit by minimizing the VM rental fees and SLA violation penalties.

The WS problem is known to be NP-hard [9, 15] and has been widely investigated [9, 10]. For example, GRP-HEFT [9] schedules a given workflow to minimize the makespan under a VM rental budget. ProLiS [29] proportionally assigns a sub-deadline to each task and allocates tasks to VMs that can meet the deadline constraint as well as minimize the VM rental fees. However, existing methods [5, 9] were mostly designed for *static* WS where the workflow information (e.g., the arrival time, the number, and types of workflows) is known in advance. Moreover, many methods have been proposed with different goals and constraints, such as minimizing the workflow makespan with a budget constraint in GRP-HEFT [9] or minimizing the budget while satisfying the deadline constraint in ProLiS [29]. Different from these works, in this paper a workflow broker aims to strike a desirable trade-off between reducing the SLA penalty and the total VM rental fee such that the overall cost involving both of the two can be minimized. Moreover, existing algorithms focus on developing heuristics [3, 5, 9, 15, 29] or hyper-heuristics [1, 8, 16, 31] based on simplified cloud environment as well as prior information of all workflows to be scheduled. Some works only considered scheduling one workflow at a time [7].

In this paper, we consider a *Cost-aware Dynamic Multi-Workflow Scheduling (DMWS)* problem for workflow brokers, where different workflows are dynamically sent to brokers for execution. The broker needs to rent proper VMs and schedule workflows on the rented VMs in real-time to minimize VM rental fees and SLA violation penalties. Note that the number of VMs also needs to accommodate the workflow dynamics. Therefore, DMWS involves decisions of dynamically adjusting the number of rented VMs and allocating workflow tasks to VMs. Since WS decisions at a given time are affected by previous decisions and the currently available VM resources, this is a sequential decision problem. Therefore, Deep Reinforcement Learning (DRL) is a promising direction towards tackling this challenging problem. However, existing Q-learning or gradient-based DRL approaches for WS [7, 13, 20, 25] have certain limitations: (1) They do not guarantee high scalability due to the assumption that the number of VMs is predetermined and remains fixed [13, 20, 25]. (2) Their performance is sensitive to hyper-parameter settings while hyper-parameter search is difficult.

The aim of this paper is to propose an effective approach for training newly designed scheduling policies to handle a changing number of VMs and workflows to address the cost-aware DMWS. Specifically, we design a new Deep Neural Network (DNN) based scheduling policy to dynamically rent new VMs and allocate simultaneously any tasks ready for execution to the rented VMs. We show that the designed scheduling policies can be used to efficiently assign a priority value to each candidate VM for a given task. The VM with the highest priority will be selected for renting and/or task execution. In line with the new policy network design, a new training approach called Evolutionary Strategy based Reinforcement Learning (ES-RL) for DMWS is proposed. Our new training approach features the use of Evolutionary Strategy (ES), a deep neuroevolution algorithm, to achieve stable and effective training of our policy network. Meanwhile, ES-RL is not sensitive to hyper-parameter settings and offers significant performance gain and time reduction through its parallel training capabilities.

Specifically, the key contributions of this paper are listed as follows:

- A Priority-based Deep Neural Network (DNN) scheduling policy design is proposed to flexibly adapt to a changing number of VMs and workflows.
- An evolutionary DRL approach called Evolutionary Strategy based Reinforcement Learning (ES-RL) is proposed to achieve robust and time-efficient training of new policy networks that can solve the DMWS problem effectively.
- Extensive experiments with real-world datasets have been conducted to show that the scheduling policies trained by ES-RL can significantly reduce the costs by more than 90% compared to the state-of-the-art WS approaches.

2 Related Work

Problem formulation: Existing Workflow Scheduling (WS) studies can be divided into *static* and *dynamic* WS.

Static WS assumes the workflow information (e.g., arrival time, workflow type, and the number of workflows) is known in advance. Given the workflow information, the scheduling decisions are made offline and remained fixed during the workflow execution. Most of the existing works belong to this category [3,5,9, 29]. However, the assumption on prior knowledge of the workflow information in a cloud environment may not be practical because users can submit their workflows at any time and the workflows from one user can also vary in terms of structure and size from time to time [6,15]. Although an alternative way is to schedule the workflows periodically (e.g., batch scheduling) [2], deciding a suitable scheduling period is critical and challenging. For example, a short scheduling period can significantly increase VM rental fees due to low VM utilization (see GRP-HEFT performance in Sec. 6.2) while a long scheduling period introduces long workflow waiting time, potentially leading to high SLA violation penalties.

Dynamic WS makes scheduling decisions at run-time according to the current cloud environment. Unlike static WS, dynamic WS only received limited research interests [6,15]. For example, the existing studies [6,15] considered the workflow scheduling problem with the goal of minimizing the VM rental costs while treating the workflow deadline as a hard constraint. However, sometimes it can be more cost-efficient by paying the SLA penalties rather than renting additional/expensive VMs (see the comparison between ES-RL and ProLiS in Sec. 6.2). The trade-off between VM rental fees and SLA violation was not captured by the models proposed in existing works [9, 29]. Motivated by [11, 31], in this paper, we study the dynamic WS problem with the aim to optimize both VM rental fees and SLA violation penalties. That is, we consider the trade-off between rental fees and SLA violation fees to minimize the overall cost of brokers and therefore maximize their profits.

Algorithm design: Most existing works focus on developing heuristics or hyper-heuristics to generate approximate or near-optimal solutions for the NP-hard WS problem with constraints. For example, GRP-HEFT [9] was proposed which selected VMs using a greedy heuristic under a budget constraint and allocated tasks using a modified HEFT [26]. To solve the deadline constrained WS problem of a single workflow, ProLiS [29] distributed the user-assigned deadline to each task in the workflow and subsequently allocated the tasks to VMs in order to meet their sub-deadlines. Other heuristics can also be found in [3,5,15]. However, most of them rely on a simplified cloud environment and full knowledge of all workflows to be scheduled, which potentially limits their practical applicability. Apart from that, many of them [3, 5, 9] are designed for static workflow scheduling. Alternatively, meta-heuristic (e.g., Particle Swarm Optimization) [19, 21] and hyper-heuristic methods (e.g., Genetic Programming) [8, 31] have been applied to WS. However, these approaches either assume the workflow information is known in advance (i.e., static WS) or generate heuristics based on historical data.

Deep Reinforcement Learning (DRL) has been applied for WS due to its ability to optimize a solution via interacting with an unknown environment [7, 13, 20, 25, 27]. For example, a deep-Q-network based DRL algorithm was proposed [27] to optimize the workflow makespan and user's cost. However, existing works have certain limitations. First, they are usually designed with a given and fixed number of VMs. However, the given number of VMs may not be optimal to handle the changing workloads [13, 20, 25, 27]. Second, the performance of most DRL algorithms [23, 24] is sensitive to the hyper-parameter setting while hyper-parameter search is difficult [14].

To cope with these limitations, Evolutionary Strategy (ES) is leveraged in this paper to train the scheduling policy. ES is a population-based approach that evolves DNNs by simulating the process of natural selection. Existing studies [22] have shown that ES can achieve competitive performance with DRL algorithms. Moreover, ES is highly parallelizable and has fewer hyper-parameters needed to be tuned compared to DRL algorithms.

3 Problem Formulation

In this paper we study the cost-aware Dynamic Multi-Workflow Scheduling (DMWS) problem. This section presents a formal definition of the problem.

Cloud Environment: We consider a cloud data center equipped with a set of VM types. Thanks to the elasticity feature in the cloud, we assume that the number of VMs with each VM type for renting is "unlimited". A *VM* v with type $Type(v)$ can be described as:

$$v = \langle Type(v), Capa(v), Price(v) \rangle$$

where $Capa(v)$ is the VM processing capacity measured in Compute Units [9] and $Price(v)$ is the rental fee of each time unit depends on $Type(v)$. Following existing studies [9,31], we consider the rental time unit as one hour in this paper.

VM Rental Fee: For the DMWS problem, we consider a time interval $T = (t_s, t_e)$ where t_s and t_e are the starting and ending time. Within T, the same type of VM can be rented for multiple times. We denote the set of *rental periods* for a VM v with type $Type(v)$ within T as $RT(v,T)$:

$$RT(v,T) = \{(VMST(v,k,T), VMFT(v,k,T)) | k = 1, ...\}$$

where $(VMST(v,k,T), VMFT(v,k,T))$ is the k^{th} time pair for v within time period T. $VMST(v,k,T)$ is the rental start time which begins when a workflow task is allocated at v and $VMFT(v,k,T)$ is the corresponding rental finish time. The VM rental fees under a scheduling policy π can be calculated as follows:

$$RentFee(\pi, T) = \sum_{v \in I(\pi,T)} \left(Price(v) \times \sum_{(t_1,t_2) \in RT(v,T)} \left\lceil \frac{t_2 - t_1}{3600} \right\rceil \right)$$

where $I(\pi, T)$ is the set of VMs being rented within the time period T.

Workflow Model: A *workflow* w is represented as a Directed Acyclic Graph associated with its arrival time $ArrT(w)$ and a user-specified deadline $DL(w)$.

$$w = \langle DAG(w), ArrT(w), DL(w) \rangle$$

where $DAG(w)$ includes a set of tasks $\{Task(w,i) | i \in \{1, 2, ...\}\}$ and directed edges connecting the tasks to enforce their execution order. Note that $Task(w,i)$ is associated with execution time $RefT(Task(w,i))$ and can only be executed if all its predecessor tasks $Pre(Task(w,i))$ are completed. $Task(w,i)$ is an entry task if $Pre(Task(w,i)) = \emptyset$. Similarly, the successors of $Task(w,i)$ are denoted as $Suc(Task(w,i))$. A task with no successors is an exit task.

In this paper, we assume that the arrival time of any new workflows is not known in advance. To avoid SLA violation and flexibly utilize the VM resources, scheduling decisions are made in real time, e.g., whenever a task is ready. In particular, a task $Task(w,i)$ is defined as *ready* if it is either an entry task of a workflow (i.e., $Pre(Task(w,i)) = \emptyset$) or a task with all its predecessors $Pre(Task(w,i))$ completed. We define a set of candidate VMs as $CVM(t)$ which includes all leased VMs at time t and a set of VM options with all VM types that can be created. Whenever $Task(w,i)$ is ready at time t, π selects a VM v from $CVM(t)$ for $Task(w,i)$ allocation.

Following π, the start time for $Task(w,i)$ is $ST(Task(w,i),\pi)$ and the completion time is

$$CT(Task(w,i),\pi) = ST(Task(w,i),\pi) + \frac{RefT(Task(w,i))}{Capa(v(\pi))}$$

Thus, the completion time WCT of a workflow w is the maximum completion time among all tasks:

$$WCT(w,\pi) = \max_{Task(w,i)\in DAG(w)} \{CT(Task(w,i),\pi)\}$$

SLA Penalty: Following existing works [28, 31], the SLA violation penalty of workflow w can be defined as follows:

$$Penalty(w,\pi) = \begin{cases} 0, \text{ if } WCT(w,\pi) \leq DL(w) \\ \epsilon + \beta(w) \times (WCT(w,\pi) - DL(w)), \text{ otherwise} \end{cases}$$

where ϵ is a constant and $\beta(w)$ is the penalty rate for w.

The goal of the cost-aware DMWS problem is to find π to schedule a set of workflows $W(T) = \{w | ArrT(w) < t_e\}$ that arrive during T, so as to minimize SLA violation penalties and VM rental fees:

$$\operatorname*{argmin}_{\pi} \sum_{w\in W(T)} Penalty(w,\pi) + RentFee(\pi,T) \tag{1}$$

4 Priority-based DNN Policy Design

In the DMWS problem, the scheduling decision needs to be made in real time with minimum delay. Therefore, π must select a suitable VM quickly whenever a task is ready. Meanwhile, in order to tackle environment dynamics and capture the most recent information (e.g., how close is the workflow deadline), the scheduling decision is made as soon as the task is ready and before it is assigned to a VM.

In this paper, a design of priority-based Deep Neural Network (DNN) policy is proposed. As shown in Fig. 1, the policy π consists of three major components: the state extraction function O, the priority function f_θ parameterized by θ, and the mapping function Φ. Whenever a task is ready at time t, the policy π examines the VM status $z(v,t)$ extracted by O for $\forall v \in CVM(t)$. Then π assigns a priority value $p(v,t)$ to each VM v using f_θ. Based on the priorities, a VM is selected using Φ.

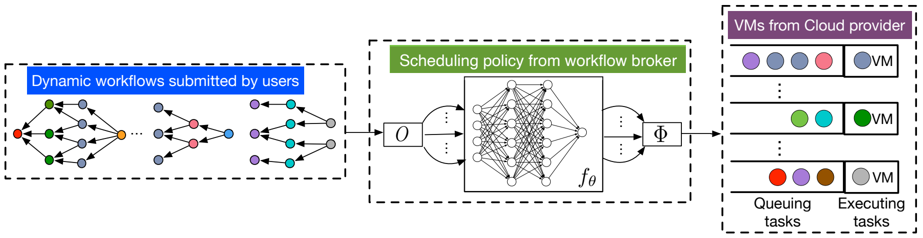


Fig. 1: The dynamic workflow scheduling system.

State Extraction: At time t, we use $S(t)$ to capture the state information of the current cloud environment including static information (e.g., the VM rental price) and dynamic information (e.g., VM rental period, availability, workflow processing information, etc). To allow the policy to be applied to a varying number of VMs, a state extraction function O is proposed to extract essential information regarding any given VM v from $CVM(t)$ and the ready task to be scheduled:

$$z(v,t) = O(S(t), v)$$

Whenever a task is ready, only information of one VM instead of all VM is fed into f_θ. Therefore, f_θ can be flexibly applied with a changing number of VMs.

Intuitively, the priority value of a VM depends on the ready task and the VM. Therefore, $z(v,t)$ includes both workflow-related and VM-related information. Given a ready task $Task(w,i)$ from a workflow w, we identify the following workflow-related information to estimate the workflow remaining processing time and predict future workload:

– the number of its successors $|Suc(Task(w,i))|$
– the workflow completion ratio[3]
– the estimated workflow arrival rate

For VM-related information, we estimate whether a VM is a good fit depending on if it will satisfy the deadline, introduce additional rental fees, and remain any rental time:

– a Boolean value indicating whether the VM can satisfy the task deadline[4]
– the potential cost of using the VM[5]
– the VM remaining rental time after allocating the ready task
– a Boolean value indicating whether the current VM is the one with the lowest cost and can satisfy the deadline.

The state extraction process can be formulated as follows:

$$Z(t) = [z(v,t)]_{v\in CVM(t)} = [O\left(S(t),v\right)]_{v\in CVM(t)}, t \in T \tag{2}$$

[3] The workflow completion ratio is the ratio of the number of completed tasks to the total number of tasks from the workflow.

[4] Motivated by ProLiS [29], we assign a deadline to a task based on the proportion of its computational time to the overall workflow computational time

[5] The potential cost is the sum of new VM rental fee and deadline violation penalty.

Priority Mapping: Using the extracted state features $z(v,t)$, the priority function f_θ with trainable parameters θ calculates a priority value $p(v,t)$ for every VM candidate v:

$$P(t) = [p(v,t)]_{v \in CVM(t)} = [f_\theta\left(z(v,t)\right)]_{v \in CVM(t)}$$

In this paper, DNN is adopted to implement the priority function.Meanwhile, neural engines and similar hardware technologies can quickly process our neural networks for priority mapping during practical use.

VM Selection: Given the priorities, a VM $a(t)$ with the highest priority value is selected by Φ:

$$a(t) = \Phi(P(t)), \text{ i.e., } a(t) = \underset{v \in CVM(t)}{\arg\max}\,(P(t))$$

The scheduling policy can be represented as follows:

$$a(t) = \pi(S(t)) = \Phi\left([f_\theta\left(O\left(S(t),v\right)\right)]_{v \in CVM(t)}\right) \tag{3}$$

5 Evolutionary Reinforcement Learning

Training a policy can be considered as a DRL task. However, as we discussed in Sec. 2, existing DRL-based approaches for WS assume the number of VMs is predetermined and fixed. Therefore, they cannot scale to a network with a different number of VMs. Meanwhile, their performance highly relies on hyper-parameter tuning.

To tackle these problems, we introduced a new Evolutionary Strategy based Reinforcement Learning (ES-RL) approach for DMWS. The pseudo-code of ES-RL is presented in Algorithm 1. In particular, ES-RL adopts the ES framework introduced by OpenAI in [22] for training scheduling policies. ES-RL is a population-based optimization method that runs iteratively, as shown in Fig. 2. At each iteration, given the current policy parameters $\hat{\theta}$, ES-RL samples a population of N individuals $[\theta_i]_{i=1,..,N}$ from an isotropic multi-variance Gaussian with mean $\hat{\theta}$ and fixed covariance $\sigma^2 I$, i.e., $\theta_i \sim \mathcal{N}(\hat{\theta}, \sigma^2 I)$, which is equivalent to

$$\theta_i = \hat{\theta} + \sigma\epsilon_i, \epsilon_i \sim \mathcal{N}(0, I)$$

The fitness value $F(\hat{\theta}+\sigma\epsilon_i)$ of each individual $\hat{\theta}+\sigma\epsilon_i$ is evaluated by applying the perturbed policy $\pi_{\hat{\theta}+\sigma\epsilon_i}$ in the cloud environment as discussed in Sec. 4. In line with our objective function in Eq. (1), we define the fitness function $F(\hat{\theta}+\sigma\epsilon_i)$ as the total cost incurred over T:

$$F(\hat{\theta}+\sigma\epsilon_i) = \sum_{w \in W(T)} Penalty(w,\pi) + RentFee(\pi, T) \tag{4}$$

The goal of ES-RL is to find θ that can minimize the total cost defined in Eq. (4) through minimizing the expected objective value over the population

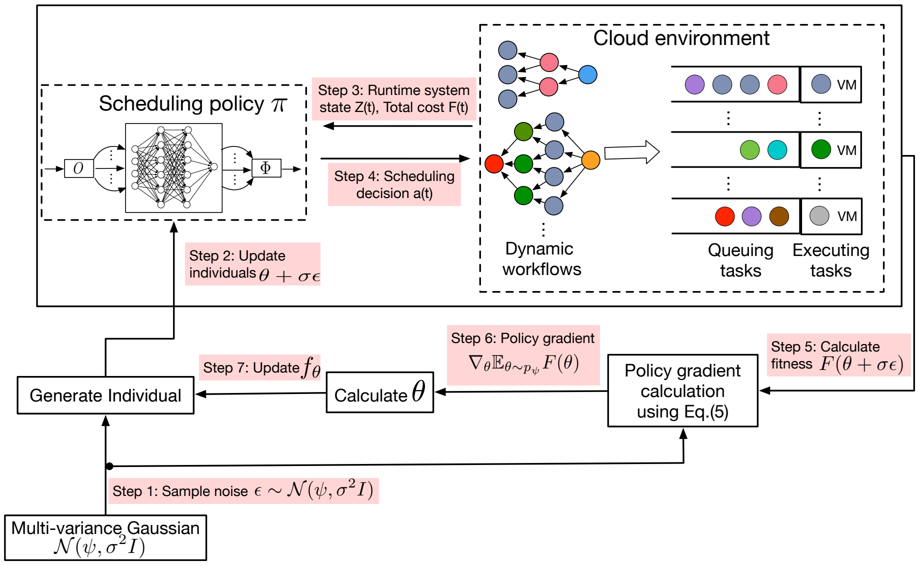

Cloud environment
Scheduling policy π
O
Φ
Step 3: Runtime system state Z(t), Total cost F(t)
Step 4: Scheduling decision a(t)
VM
Dynamic workflows
Queuing tasks
Executing tasks
Step 2: Update individuals $\theta + \sigma\epsilon$
Step 6: Policy gradient $\nabla_\theta \mathbb{E}_{\theta \sim p_\psi} F(\theta)$
Step 5: Calculate fitness $F(\theta + \sigma\epsilon)$
Step 7: Update f_θ
Generate Individual
Calculate θ
Policy gradient calculation using Eq.(5)
Step 1: Sample noise $\epsilon \sim \mathcal{N}(\psi, \sigma^2 I)$
Multi-variance Gaussian $\mathcal{N}(\psi, \sigma^2 I)$

Fig. 2: Scheduling policy training using ES-RL.

Algorithm 1 ES-RL for DMWS

```
1: Input: Population size $N$, initial policy parameter $\hat{\theta}$, learning rate $\alpha$; Gaussian
   noise standard deviation $\sigma$
2: Output: Scheduling policy
3: while the number of generations < the max number of generations: do
4:     for each individual $i = 1, ..., N$ do
5:         Sample $\epsilon_i \sim \mathcal{N}(0, I)$
6:         Update the WF scheduling policy $\pi_i$ using $\theta_i = \hat{\theta} + \sigma\epsilon_i$
7:         Evaluate fitness $F(\theta_i)$ in cloud environment using Eq. (4)
8:     end for
9:     Estimate policy gradient $\nabla_\theta \mathbb{E}_{\theta \sim p_\psi} F(\theta)$ using Eq. (5)
10:    Update $\hat{\theta} \leftarrow \hat{\theta} + \alpha \nabla_\theta \mathbb{E}_{\theta \sim p_\psi} F(\theta)$
11: end while
```

distribution, i.e., $\mathbb{E}_{\epsilon \sim \mathcal{N}(0,I)} F(\theta + \sigma\epsilon_i)$. To achieve this goal, ES updates θ using the following estimator:

$$\begin{aligned}\nabla_\theta \mathbb{E}_{\theta \sim p_\psi} F(\theta) = \nabla_\theta \mathbb{E}_{\epsilon \sim \mathcal{N}(0,I)} F(\theta + \sigma\epsilon) = \frac{1}{\sigma} \mathbb{E}_{\epsilon \sim \mathcal{N}(0,I)} \left[F(\theta + \sigma\epsilon)\epsilon\right] \\ \approx \frac{1}{N\sigma} \sum_{i=1}^{N} \left[F(\theta + \sigma\epsilon_i)\epsilon_i\right]\end{aligned} \tag{5}$$

Table 1: VM Setting based on Amazon EC2

Type	vCPU	Compute Unit	Memory(GB)	Cost($ per hour)
m5.large	2	10	8	0.096
m5.xlarge	4	16	16	0.192
m5.2xlarge	8	37	32	0.384
m5.4xlarge	16	70	64	0.768
m5.8xlarge	32	128	128	1.536
m5.12xlarge	48	168	192	2.304

6 Performance Evaluation

To evaluate the performance of our proposed ES-RL approach, we conduct experimental evaluations using a simulator based on real-world data from cloud data centers and benchmark workflows, comparing with two state-of-the-art methods.

6.1 Simulation Setting

Cloud Environment: We consider a cloud data center equipped with six different VM types. Following [3,9], VMs are configured based on the general-purpose on-demand VM group in the US East region provided by Amazon EC2. VM details are summarized in Tab. 1 which were collected in September 2020 from Amazon[6].

Workflows: Following existing studies [3, 9, 29], four classes of scientific workflows are used in our experiment which include CyberShake, Inspiral, Montage, and Sipht. All workflows are computational intensive and their communication time is trivial compared to the task execution time [9]. Detail analysis of these workflows can be found in [4,12]. Note that workflows in each class share similar structures but can differ from the numbers of their tasks (e.g., ranging from 25 to 50). During our training, we simulate small workflows from four classes where each of them contains 25 to 30 tasks. Nevertheless, our trained scheduling policy is generalized well and can be directly applied to large workflows with 50 tasks without retraining. Following [6, 15], we simulate the dynamic arrival of workflow applications by randomly sampling 30 workflows from four classes of workflows in each simulation. The arrival time of workflows follows a Poisson distribution with $\lambda = 0.01$ [15]. The penalty rate β is set to be \$0.24 per hour for each workflow [32].

To compare the algorithm performance under different deadlines, a deadline relaxation factor η is used. Similar to [3], we consider η changes from 1 to 2.25. Given η, the deadline of a workflow w is set as below:

$$DL(w) = ArrT(w) + \eta \times MinMakespan(w)$$

[6] https://aws.amazon.com/ec2/pricing/on-demand/

where $MinMakespan(w)$ is the shortest makespan of each workflow w as its execution time when its tasks are executed by the fastest VM v from Tab. 1 (i.e., m5.12xlarge).

Algorithm Implementation: We implement ES-RL based on the code released by OpenAI[7]. In terms of parameter settings, we set the Gaussian noise standard deviation $\sigma = 0.05$ and the learning rate $\alpha = 10^{-2}$. The population size N is 40 and each individual is evaluated for 1 episodes (i.e., N_E=1). The maximum generation number is 3000. For the priority function design, we follow the DNN architecture used by OpenAI baselines. Specifically, it is a fully connected multilayer feed-forward neural network with two hidden layers. Each hidden layer consists of 64 nodes with the tanh activation function.

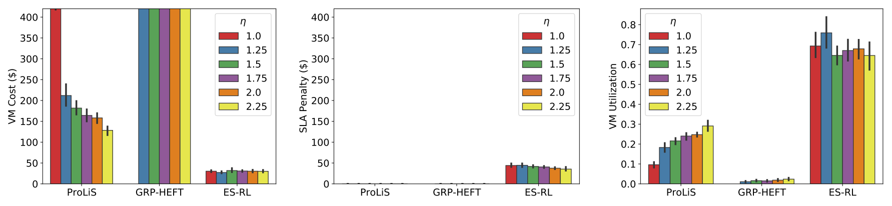
Baseline Algorithms: We compared ES-RL with two state-of-the-art scheduling algorithms (GRP-HEFT [9] and ProLiS [29]) which have similar objectives as (1). As we discussed in Sec. 2, both GRP-HEFT and ProLiS were designed for static WS of one single workflow. We need to adapt them to the DMWS problem. To enable GRP-HEFT to be comparable with ES-RL, we minimize the budget constraint for every newly arriving workflow by incrementally increasing the budget that is passed to GRP-HEFT as a constraint until the workflow satisfies its deadline. To enable ProLiS to be applicable for dynamic workflow arrival, ProLiS is triggered to assign deadlines to all tasks once a workflow arrives. Whenever a task is ready, the cheapest VM from $CVM(t)$ that can meet the task deadline will be selected.

6.2 Simulation Results

To demonstrate the effectiveness of ES-RL, we compare its performance (i.e., overall cost) with GRP-HEFT and ProLiS under different η (i.e., deadline relaxation factor) as shown in Tab. 2. Note that a smaller η implies a tighter deadline. We also analyze the testing performance among all algorithms with respect to VM cost (\$), SLA penalty (\$), and VM utilization (%), as shown in Fig. 3. Note that GRP-HEFT causes a high VM cost which ranges from \$830 to \$1988. Thus, to show the difference in VM rental cost in Fig. 3(a), we set the y-axis limit to an upper value (i.e., 420).

As shown in Tab. 2, we can observe that the overall cost of ProLiS and GRP-HEFT increases as η decreases (i.e., a tighter deadline). This is mainly because both ProLiS and GRP-HEFT consider the workflow deadline as a hard constraint. In other words, they always select VMs that can satisfy the workflow deadlines. This explanation also matches well with our observation in Fig. 3(b) where the SLA penalty remains 0 regardless of the η changes. With a tight deadline, more powerful VMs in terms of computational capacity are required. As a result, an increase in VM cost can be observed in Fig. 3(a) as η decreases.

[7] https://github.com/openai



(a) VM cost (b) SLA penalty (c) VM utilization

Fig. 3: Comparison of GRP-HEFT [9], ProLiS [29], and ES-RL on different factors with respect to different η with small workflows.

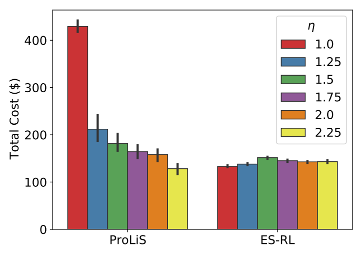
Table 2: The average fitness values (i.e., total cost) tested over multiple runs for GRP-HEFT [9] and ProLiS [29], and ES-RL with different η with small workflows. (Note: a lower value is better)

η	ProLiS	GRP-HEFT	ES-RL
1.00	395.5520 ± 11.633617	-	**74.490828 ± 7.060981**
1.25	108.5184 ± 15.815567	1775.9232 ± 161.077450	**71.817811 ± 7.664415**
1.50	91.7376 ± 12.662934	1171.8144 ± 75.565892	**73.737249 ± 6.339560**
1.75	83.4304 ± 10.828026	1258.9056 ± 90.980289	**71.436684 ± 5.725663**
2.00	76.9088 ± 10.752586	1164.7488 ± 76.973755	**68.036960 ± 5.993622**
2.25	68.3456 ± 10.332963	969.9840 ± 102.602041	**65.590513 ± 7.967546**

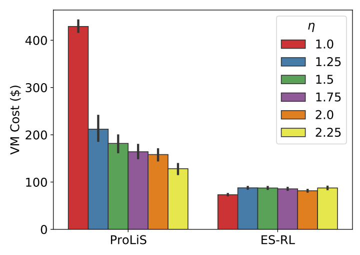
In comparison, ES-RL consistently outperforms both ProLiS and GRP-HEFT with the lowest overall cost as highlighted in Tab. 2. This is achieved by balancing the trade-off between VM rental fees and SLA penalties. As demonstrated in Fig. 3, ES-RL maintains the low overall costs by renting cost-effective VMs (see Fig. 3(a)) as well as utilizing existing VMs (see Fig. 3(c) for the high VM utilization). As a result, ES-RL can violate the workflow deadlines and therefore introduces SLA penalties (see Fig. 3(b)). Meanwhile, when η increases, the SLA penalty decreases because when the deadline becomes looser, VM selection has less impact on the SLA penalty.

Another interesting observation is that GRP-HEFT has the highest overall cost among the three approaches. This is mainly because GRP-HEFT is designed for static WS. In a scenario when two tasks from the same workflow are assigned to the same VM, the idle time slot between the two tasks cannot be utilized by a different workflow, leading to low VM utilization. This also matches our observation in Fig. 3(c) where GRP-HEFT presents the lowest VM utilization.

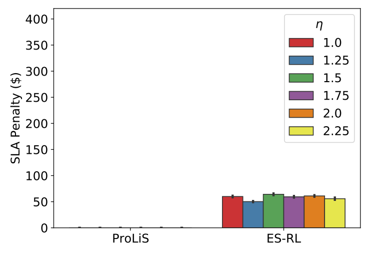
We also investigate the generalization capability of our trained scheduling policy. In particular, we define the generalization capability as the policy that was trained using small workflows can still be able to effectively schedule large workflows. The results are shown in Fig. 4. From the figures, we can see that ES-RL still managed to reduce the overall cost by balancing the trade-off between VM rental fees and SLA penalties. Meanwhile, compared to ProLiS, ES-RL can achieve significantly higher VM utilization. In general, our observations of ES-RL with large workflows are consistent with the results with small workflows shown in Fig. 3. Our results demonstrate that ES-RL can effectively train a generalized scheduling policy that can flexibly adapt to not only the changing number of VMs but also workflows with different sizes.



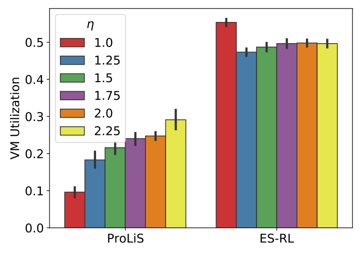
(a) Total cost



(b) VM cost



(c) SLA penalty



(d) VM utilization

Fig. 4: Comparison of ProLiS [29] and ES-RL on different factors with respect to different η with large workflows.

7 Conclusions

In this paper, we proposed an effective ES based approach for the cost-aware dynamic multi-workflow scheduling (DMWS) problem. In particular, we formulate a dynamic multi-workflow scheduling problem with the goal of minimizing both the VM rental cost and SLA violation penalties. To effectively solve this problem, we proposed a new scheduling policy design of a priority-based deep neural network that can be used in a dynamic environment with a changing number of VMs and workflows. Meanwhile, a new Evolutionary Strategy based RL (ES-RL) algorithm for DMWS is proposed to efficiently train a generally applicable scheduling policy in parallel. Our experiments with real-world datasets showed that the scheduling policies trained by ES-RL can effectively reduce the overall costs compared to two state-of-the-art algorithms.

References

1. Ahmad, S.G., Liew, C.S., Munir, E.U., Ang, T.F., Khan, S.U.: A hybrid genetic algorithm for optimization of scheduling workflow applications in heterogeneous computing systems. Journal of Parallel and Distributed Computing **87**, 80–90 (2016)
2. Alsurdeh, R., Calheiros, R.N., Matawie, K.M., Javadi, B.: Hybrid workflow scheduling on edge cloud computing systems. IEEE Access **9**, 134783–134799 (2021)
3. Arabnejad, V., Bubendorfer, K., Ng, B.: Budget and deadline aware e-science workflow scheduling in clouds. IEEE Transactions on Parallel and Distributed Systems **30**(1), 29–44 (2019)
4. Bharathi, S., Chervenak, A., Deelman, E., Mehta, G., Su, M., Vahi, K.: Characterization of scientific workflows. In: 2008 Third Workshop on Workflows in Support of Large-Scale Science. pp. 1–10 (2008)
5. Byun, E.K., Kee, Y.S., Kim, J.S., Deelman, E., Maeng, S.: Bts: Resource capacity estimate for time-targeted science workflows. Journal of Parallel and Distributed Computing **71**(6), 848–862 (2011)
6. Chen, H., Zhu, X., Liu, G., Pedrycz, W.: Uncertainty-aware online scheduling for real-time workflows in cloud service environment. IEEE Transactions on Services Computing pp. 1–1 (2018)
7. Dong, T., Xue, F., Xiao, C., Zhang, J.: Workflow scheduling based on deep reinforcement learning in the cloud environment. Journal of Ambient Intelligence and Humanized Computing **12**(12), 10823–10835 (2021)
8. Escott, K.R., Ma, H., Chen, G.: Genetic programming based hyper heuristic approach for dynamic workflow scheduling in the cloud. In: International Conference on Database and Expert Systems Applications. pp. 76–90. Springer (2020)
9. Faragardi, H.R., Saleh Sedghpour, M.R., Fazliahmadi, S., Fahringer, T., Rasouli, N.: GRP-HEFT: A Budget-Constrained Resource Provisioning Scheme for Workflow Scheduling in IaaS Clouds. IEEE Transactions on Parallel and Distributed Systems **31**(6), 1239–1254 (2020)
10. Genez, T.A.L., Bittencourt, L.F., Madeira, E.R.M.: Workflow scheduling for saas / paas cloud providers considering two sla levels. In: 2012 IEEE Network Operations and Management Symposium. pp. 906–912 (2012)
11. Hoseiny, F., Azizi, S., Shojafar, M., Tafazolli, R.: Joint qos-aware and cost-efficient task scheduling for fog-cloud resources in a volunteer computing system. ACM Transactions on Internet Technology (TOIT) **21**(4), 1–21 (2021)
12. Juve, G., Chervenak, A., Deelman, E., Bharathi, S., Mehta, G., Vahi, K.: Characterizing and profiling scientific workflows. Future Generation Computer Systems **29**(3), 682–692 (2013)
13. Li, H., Huang, J., Wang, B., Fan, Y.: Weighted double deep q-network based reinforcement learning for bi-objective multi-workflow scheduling in the cloud. Cluster Computing **25**(2), 751–768 (2022)
14. Liessner, R., Schmitt, J., Dietermann, A., Bäker, B.: Hyperparameter optimization for deep reinforcement learning in vehicle energy management. In: ICAART (2). pp. 134–144 (2019)
15. Liu, J., Ren, J., Dai, W., Zhang, D., Zhou, P., Zhang, Y., Min, G., Najjari, N.: Online multi-workflow scheduling under uncertain task execution time in iaas clouds. IEEE Transactions on Cloud Computing pp. 1–1 (2019)
16. Lopez-Garcia, P., Onieva, E., Osaba, E., Masegosa, A.D., Perallos, A.: Gace: A meta-heuristic based in the hybridization of genetic algorithms and cross entropy

methods for continuous optimization. Expert Systems with Applications **55**, 508–519 (2016)

17. Masdari, M., ValiKardan, S., Shahi, Z., Azar, S.I.: Towards workflow scheduling in cloud computing: a comprehensive analysis. Journal of Network and Computer Applications **66**, 64–82 (2016)
18. Oliver, H., Shin, M., Matthews, D., Sanders, O., Bartholomew, S., Clark, A., Fitzpatrick, B., van Haren, R., Hut, R., Drost, N.: Workflow automation for cycling systems. Computing in Science & Engineering **21**(4), 7–21 (2019)
19. Pandey, S., Wu, L., Guru, S.M., Buyya, R.: A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In: 2010 24th IEEE International Conference on Advanced Information Networking and Applications. pp. 400–407 (2010)
20. Qin, Y., Wang, H., Yi, S., Li, X., Zhai, L.: An energy-aware scheduling algorithm for budget-constrained scientific workflows based on multi-objective reinforcement learning. The Journal of Supercomputing **76**(1), 455–480 (2020)
21. Rodriguez, M.A., Buyya, R.: Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds. IEEE Transactions on Cloud Computing **2**(2), 222–235 (2014)
22. Salimans, T., Ho, J., Chen, X., Sidor, S., Sutskever, I.: Evolution strategies as a scalable alternative to reinforcement learning. arXiv preprint arXiv:1703.03864 (2017)
23. Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust region policy optimization. In: International Conference on Machine Learning (ICML). pp. 1889–1897 (2015)
24. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
25. Suresh Kumar, D., Jagadeesh Kannan, R.: Reinforcement learning-based controller for adaptive workflow scheduling in multi-tenant cloud computing. The International Journal of Electrical Engineering & Education p. 0020720919894199 (2020)
26. Topcuoglu, H., Hariri, S., Min-You Wu: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems **13**(3), 260–274 (2002)
27. Wang, Y., Liu, H., Zheng, W., Xia, Y., Li, Y., Chen, P., Guo, K., Xie, H.: Multi-objective workflow scheduling with deep-q-network-based multi-agent reinforcement learning. IEEE access **7**, 39974–39982 (2019)
28. Wu, L., Garg, S.K., Versteeg, S., Buyya, R.: Sla-based resource provisioning for hosted software-as-a-service applications in cloud computing environments. IEEE Transactions on Services Computing **7**(3), 465–485 (2014)
29. Wu, Q., Ishikawa, F., Zhu, Q., Xia, Y., Wen, J.: Deadline-constrained cost optimization approaches for workflow scheduling in clouds. IEEE Transactions on Parallel and Distributed Systems **28**(12), 3401–3412 (2017)
30. Xiaoyong, Y., Ying, L., Tong, J., Tiancheng, L., Zhonghai, W.: An analysis on availability commitment and penalty in cloud sla. In: 2015 IEEE 39th Annual Computer Software and Applications Conference. vol. 2, pp. 914–919 (2015)
31. Yang, Y., Chen, G., Ma, H., Zhang, M., Huang, V.: Budget and sla aware dynamic workflow scheduling in cloud computing with heterogeneous resources. In: 2021 IEEE Congress on Evolutionary Computation (CEC). pp. 2141–2148. IEEE (2021)
32. Youn, C.H., Chen, M., Dazzi, P.: Cloud Broker and Cloudlet for Workflow Scheduling. Springer (2017)