# A Deep Reinforcement Learning-Based Preemptive Approach for Cost-Aware Cloud Job Scheduling

Cheng, L.; Wang, Y.; Cheng, F.; Liu, C.; Zhao, Z.; Wang, Y.

**DOI**
10.1109/TSUSC.2023.3303898

**Publication date**
2024

**Document Version**
Final published version

**Published in**
IEEE Transactions on Sustainable Computing

# A Deep Reinforcement Learning-Based Preemptive Approach for Cost-Aware Cloud Job Scheduling

Long Cheng, *Senior Member, IEEE*, Yue Wang, Feng Cheng, Cheng Liu, *Member, IEEE*, Zhiming Zhao, *Senior Member, IEEE*, and Ying Wang, *Member, IEEE*

***Abstract*—With some specific characteristics such as elastics and scalability, cloud computing has become the most promising technology for online business nowadays. However, how to efficiently perform real-time job scheduling in cloud still poses significant challenges. The reason is that those jobs are highly dynamic and complex, and it is always hard to allocate them to computing resources in an optimal way, such as to meet the requirements from both service providers and users. In recent years, various works demonstrate that deep reinforcement learning (DRL) can handle real-time cloud jobs well in scheduling. However, to our knowledge, none of them has ever considered extra optimization opportunities for the allocated jobs in their scheduling frameworks. Given this fact, in this work, we introduce a novel DRL-based preemptive method for further improve the performance of the current studies. Specifically, we try to improve the training of scheduling policy with effective job preemptive mechanisms, and on that basis to optimize job execution cost while meeting users' expected response time. We introduce the detailed design of our method, and our evaluations demonstrate that our approach can achieve better performance than other scheduling algorithms under different real-time workloads, including the DRL approach.**

***Index Terms*—Cloud computing, DRL, job scheduling, preemptive mechanism, optimization.**

## I. Introduction

CLOUD computing involves delivering hosted services, such as servers, networking, software, analytics, and intelligence, over the internet. Specifically, in a cloud, hardware and software resources are always organized as shared resource pools while enterprises can rent the resources from cloud service providers (e.g., Google) to run their business online and provide their services to end-users at anytime from anywhere. Moreover, moving to the cloud also offers numerous other benefits beyond just economic advantages. One such benefit is lower energy consumption, as highlighted by [1]. In addition, according to Internet Data Center (IDC), the adoption of cloud computing is projected to result in a reduction of one billion metric tons of carbon emissions in the next three years. Since cloud computing is flexible, reliable, scalable, on-demand and sustainability, using cloud can not only reduce the costs on IT infrastructures but can also greatly simplify their management and maintenance tasks. In such scenarios, more and more enterprises are choosing to deploy their applications and services on the cloud.

One of the core research problems in cloud computing is job scheduling, which is always associated with the performance of job executions and resource utilizations. Generally, a cloud job scheduling system aims to assign jobs submitted by end-users to appropriate virtual machine (VM) instances in a resource pool, and an effective and optimized job scheduling mechanism will not only improve execution efficiency but also reduce the cost [2]. Additionally, it can also optimize energy consumption at the same time if required [3]. Nevertheless, since job workloads and computing resources in cloud platforms are always dynamic and complex, achieving an optimal task scheduling is not easy [4]. In fact, even in a simple case that the number as well as the properties of jobs and VMs are given, the complexity of a scheduling problem could be NP-hard for a given optimization target [5].

To address the above problem, various effective cloud job scheduling algorithms have been introduced and evaluated in the past decades. They can be broadly classified into two types based on the types of processed job workloads: 1) algorithms for handling batch jobs, such as genetic algorithm [6], meta-heuristics algorithm and their variants [7], which are suitable for service scenarios such as scientific computing that end-users are not sensitive for job response time; and 2) algorithms for processing real-time job requests [8], which are mainly designed for transactional service scenarios such as online shopping, that even a short delay of processing would results in a violation of Service Level Agreements (SLAs) to provide high quality-of-service [9]. Obviously, compared to the design of the former approaches, the latter ones would meet more challenges, as the their optimized problems are more complex.

In fact, in a simple real-time scenario, we can use some straightforward algorithms such as random or round-robin to implement job scheduling in cloud systems. However, with the growing dynamics and scales of nowadays workloads and

Manuscript received 20 February 2023; revised 19 June 2023; accepted 5 August 2023. Date of publication 10 August 2023; date of current version 5 June 2024. This work was supported by the Fundamental Research Funds for the Central Universities under Grant 2023YQ002. Recommended for acceptance by Z. Xu, S. Ren, and O. Rana. *(Corresponding author: Feng Cheng.)*

Long Cheng and Yue Wang are with the School of Control and Computer Engineering, North China Electric Power University, Beijing 102206, China (e-mail: lcheng@ncepu.edu.cn; wangyue@ncepu.edu.cn).

Feng Cheng is with the School of Mathematics, Southwest Jiaotong University, Chengdu, Sichuan 610032, China (e-mail: chengfeng2013@swjtu.edu.cn).

Cheng Liu and Ying Wang are with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China (e-mail: liucheng@ict.ac.cn; wangying2009@ict.ac.cn).

Zhiming Zhao is with the Research Group of Multi-Scale Networked Systems, University of Amsterdam, 1012 WP Amsterdam, The Netherlands (e-mail: z.zhao@uva.nl).

Digital Object Identifier 10.1109/TSUSC.2023.3303898

resources in cloud, such algorithms start to meet perform issues in many application cases [10]. Therefore, we need a more intelligent and lightweight job scheduler to make optimal decisions in real-time as jobs arrive. Deep Reinforcement Learning (DRL) is one of such methods [11], in which the intelligent agent can learn scheduling policies offline and make decisions for incoming jobs online based on its own experience without any prior knowledge.

To date, some works have studied job scheduling algorithms in cloud based on DRL already [12]. However, to the best of our knowledge, none of them has ever considered the extra optimization opportunities for the allocated jobs in their scheduling frameworks. In detail, the approaches just focus on learning decisions on allocating which job to which VM, and ignore the fact that the job queue on each VM can be optimized to further improve the performance. In other words, existing DRL-based scheduling methods execute jobs in a first-decision first-served way for each VM. More advanced strategies, such as job preemptive mechanism [13], have not been studied in a DRL-based framework yet, which could result in a suboptimal scheduling policy for DRL agent in a global view.

To improve the scheduling capability of existing DRL-based approaches, in this work, we introduce a new method for real-time cloud job scheduling by combining DRL and preemptive strategies together in a seamless way. More specifically, we try to improve the training of DRL agent with effective preemptive mechanisms, and on that basis to optimize job execution cost while meeting users' expected response time as much when possible. We present the design as well as implementation of our method, and our experiments show that our approach can outperform existing methods under different job workloads. Generally, the main contributions of this work are summarized as follows:

- We propose a DRL-based preemptive job scheduling framework for cloud systems. As far as we know, this is the first work on integrating preemptive strategies into DRL to optimize real-time cloud job scheduling.
- With the targets to minimize execution cost of VMs and guarantee users' expected response time can be maximumly achieved, we introduce a cost-aware optimization model and give the details of our implementation using Deep Q-Network.
- Compare to existing scheduling algorithms including the current DRL-based approach, our evaluations show that the proposed method can perform better under different real-time workloads.

The structure of this paper is organized as follows. In Section II, we report the related work in detail. We introduce the system scheduling model and the proposed DRL-based preemptive method in Section III and Section IV, respectively. We present our experimental evaluations in Section V and conclude this work in Section VI.

## II. Related Works

Job scheduling problems have been extensively studied in the domain of distributed systems [14]. Specially, in the field of cloud computing, an optimized scheduling scheme can not only increase performance for job execution, reduce cost on resource utilization and improve quality of service for end-users [15], but also reduce power consumption, and save energy [16]. In such scenarios, it is necessary to design an advanced job scheduling method cloud platform, which is actually still a challenging problem in the current research.

With the increasing diversity of job requirements submitted by end-users, optimizing job scheduling procedures in a reasonable manner remains a challenging problem. The cloud computing scheduling problem is also considered to be as difficult as the non-deterministic polynomial optimization problem, i.e., an NP problem [17]. In the past decades, many scholars have applied mathematical theories such as queuing theory [18], cybernetics [19], and game theory [20] to the cloud computing scheduling process in order to obtain optimal solutions. In addition, to solve the job scheduling optimization problems in a quick way, heuristic algorithms and solutions such as genetic algorithms and meta-heuristics approaches have been introduced into the domain of cloud computing. For instance, the work [2] applies the latest whale optimization algorithm to improve the cloud system performance with computing resource constraints.

Although the above approaches have demonstrated their powerful capability in optimizing cloud job scheduling, they are not able to handle the real-time scheduling problem to be studied in this paper. For example, the theoretical analysis based on queueing theory as proposed in the work [21] is designed for processing relatively stable cloud environments, due to the limitations of the proposed mathematical model. Moreover, job scheduling algorithms over meta-heuristics and their variants focus on scheduling batch jobs rather than real-time jobs. To address this problem, we focus on how to schedule jobs in real-time in cloud with the goal to design an adaptive job scheduler that is able to capture changes in the cloud environment and make allocation decisions in real-time as jobs arrive. In detail, we are aiming to propose a DRL-based scheduling algorithm, which can schedule jobs in real-time in cloud.

Generally, DRL is an advanced machine learning technique with the basis of applying deep neural networks to reinforcement learning. The approach has been widely used in solving real-time control decision problems in different domains, such as IoT [22], edge computing [23], wireless sensor networks [24], and domain optimization problems [25]. In a dynamical cloud environment, the main challenge for real-time scheduling is that retrieving prior knowledge of incoming jobs for decision making is not easy. Nevertheless, the DRL-based approach can remedy this issue effectively [26]. This is because the DRL agent can gain experience through continuous interaction with environments and use existing experiences to make appropriate scheduling decisions when new jobs arrive, based on the state of the whole system. Currently, DRL has been applied in cloud computing for real-time job scheduling. For example, the work [4] presents a DRL-based algorithm focusing on improving quality of services (QoS) to the end-users while reducing the executed cost in VM instances. Besides, the work [5] introduces a DRL-based framework targeting for maximizing the success rate of jobs along with reducing job response time. Moreover, the work [27] proposes an energy-aware task scheduling method with DRL,

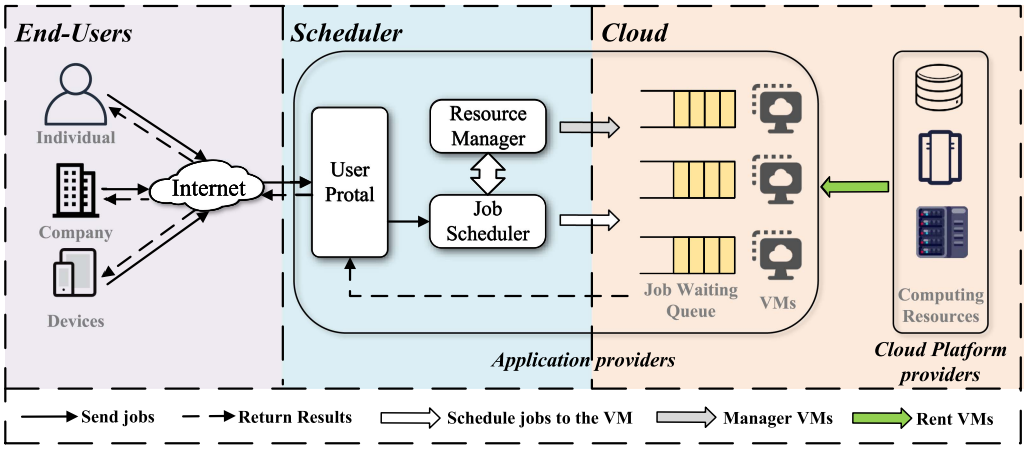



Fig. 1. General job scheduling framework in cloud computing.

which effectively reduces system energy consumption and ensures the quality of service.

In fact, many other relevant works have also shown that DRL can indeed optimize real-time cloud job scheduling in different dimensions, such as energy efficiency, response time, QoS, and job success rates [12], [28], [29]. In the meantime, some other works like [30] and [31] have tried to optimize the DRL method by employing more efficient state information extraction strategies, such as heterogeneous graph neural networks and attention mechanisms. These approaches can assist the agent in making better quality decisions. Furthermore, a small number of researchers have considered the cost optimization of real-time job scheduling [32], [33]. Nevertheless, none of them has ever considered the further improvement for the allocated jobs after each decision in their scheduling frameworks. This means that there are still a lot of room for optimization and are well worth studying. For example, in existing works, no matter how high the priority of a job is, the job must wait in a queue of a VM after the allocation, which could result in bad performance in many application scenarios. Specifically, in e-commerce services, users always expect that their requests can be responsible in a quick time, and high-priority jobs should be prioritized to meet the needs of important end-users when all the underlying VMs are in a busy mode.

With the above understanding, we propose integrating a preemption mechanism into the existing DRL-based job scheduling framework. Preemptive scheduling, commonly employed alongside priority scheduling, allows for the suspension of ongoing processes before completion. This approach facilitates the allocation of CPU computing resources to other high-priority processes, thereby enhancing system response time. Additionally, the preemption mechanism helps prevent scenarios where certain processes monopolize CPU time, reducing the risk of system crashes. As a result, preemptive scheduling finds extensive application in real-time operating systems and multitasking operating systems. By incorporating the preemption mechanism, the state space of the entire scheduling system will expand beyond what existing works have considered. Consequently, the DRL agent can learn a more efficient scheduling policy for cloud scheduling.

## III. System Architecture and Problem Statement

In this section, we introduce our system architecture on cloud job scheduling and present the model for the cost-aware optimization problem as we study in this work.

### A. System Architecture

In this paper, we focus on the scheduling of jobs within the application-level context, which are being studied in some recent works like [5]. Specifically, we consider a common cloud system as shown in Fig. 1. There, application providers (i.e., tenants) rent heterogeneous VM instances from cloud providers. Their programs are deployed on these VM instances, which are used to continuously receive various job requests from end-users. The received jobs are handed over to the job scheduler for assignment. Namely, the scheduler dynamically assigns the jobs to the appropriate VMs for execution based on the state of jobs and VMs. Without losing generality, we assume that all the jobs as study in this work are independent from each other. Then, the job scheduling process (based on a DRL agent) can be mainly divided into four steps as follows.

1) When a new job arrives, the cloud system extracts the characteristics of the job (e.g., arrival time, QoS, type) and the state of all VMs as the state of the system.
2) Based on the extracted state, the agent makes an allocation decision and assigns the job to a suitable VM instances for executions.
3) The pre-designed preemptive mechanism is triggered once the job is assigned. If the preemption condition is satisfied, the VM will stop the execution of current job immediately and start to perform another job picked by the preemption algorithm.
4) After the job is executed, the execution result is feedback to the user.

### B. Problem Statement

Cloud job scheduling can be performed with different optimization objectives, among which the execution cost of VM instances is one of the major concerns for current business. For

TABLE I
TABLE OF NOTATIONS

| Notation | Meaning |
|---|---|
| $ID_j$ | the id for job $j$ |
| $aT_j$ | the arrival time point for job $j$ |
| $reqCom_j$ | required number of instructions to process job $j$ |
| $Type_j$ | type of job $j$ (either I/O or computing intensive) |
| $QoS_j$ | QoS requirement of job $j$ (expected response time by users) |
| $T_j^{res}$ | response time point of job $j$ |
| $T_j^{exe}$ | execution time of job $j$ |
| $T_j^{wait}$ | waiting time of job $j$ |
| $VID_i$ | the id for VM $i$ |
| $VCom_i$ | processing speed of VM $i$ (instructions per time unit) |
| $VType_i$ | type VM $i$ (either I/O or computing intensive) |
| $VC_i$ | execution cost per unit time of VM $i$ |
| $V_j^{it}$ | the available time point of VM $i$ |
| $T_i^{switch}$ | the context switch time for preemption on VM $i$ |
| $T^{current}$ | the current time of the system |

example, the works [34], [35], [36] propose various cost-aware scheduling approaches with the targets to reduce the monetary cost on using cloud. Similar to them, we mainly focus on the optimization of execution cost on the basis of a scheduling which can meet the expected response time from each user. From this basis, we present the mathematical model of the DRL-based cloud computing scheduling problem in detail as follows. The meaning of the main parameters we have used in this paper are presented in Table I.

*1) Workload and Cloud Resources:* In this paper, each job submitted by the user is considered as an independent job and can arrive at any time. Jobs are mainly classified into two types, CPU intensive and I/O intensive jobs (or high CPU and high I/O in this work). To further capture the service requirements from users, we define each job as:

$$\mathcal{J}ob_j = \{ID_j, aT_j, reqCom_j, Type_j, QoS_j\} \tag{1}$$

in which $ID_j$ is the id of job $j$, $aT_j$ is the arrival time point of job $j$, $reqCom_j$ is quantified by the number of required computing instructions for job $j$, $Type_j$ is the type of job and $QoS_j$ means the response time expected by the user who submits the job. All these information can be recorded by the cloud system at the time point when the job arrives.

Following the model of workloads, we also categorize the VM instance resources into CPU intensive ones and I/O intensive ones, and define each VM as:

$$\mathcal{V}M_i = \{VID_i, VCom_i, VType_i, VC_i\} \tag{2}$$

where $VID_i$ is the ID of the VM, $VCom_i$ is the computing capacity of the VM, which can be presented by the number of instructions the VM can perform per second. Moreover, $VType_i$ is the type of the VM, and $VC_i$ is the execution cost per unit time of the VM.

*2) Execution Mechanism:* In the cloud platform, we assume that only one job can be processed at a point in time on a VM and other jobs must wait until the VM is available. In real-world scenarios, there is a continuous influx of job requests, potentially reaching hundreds or more per second. This leads to a situation where jobs are assigned to a VM and then wait for execution. To address this, we implement a job waiting queue that keeps track of the assigned jobs for each VM. Whenever a new job arrives, a preemption mechanism is triggered to determine if preempting the VM for priority execution is necessary. For the purpose of this paper, we use the shortest remaining time in our preemptive strategies to manage the jobs on each VM. To better capture the core problem in cloud job scheduling, we assume that each job can be preempted at most once. The reason for limiting the number of preemptions is to avoid excessive context switches, which can negatively impact performance by introducing performance degradation and increased latency. In such scenarios, the response time of each job in our system can be calculated by its waiting time, execution time, and context switching time in the preemption process. Namely, we have:

$$T_j^{res} = T_j^{exe} + T_j^{\text{wait}} + k * T_i^{\text{switch}} \tag{3}$$

here, $T_j^{res}$ means the response time of job $j$, and $T_j^{exe}$ and $T_j^{\text{wait}}$ mean the execution time and waiting time respectively. Moreover, $T_i^{\text{switch}}$ is the time it takes for a context switch in the VM. For simplicity, we can set it to a constant. Specifically, we set the value to 0.02 ms in our final evaluations. Moreover, $k$ in the above equation refers to the number of times that job $j$ has been preempted. Obviously, in our mode, the value of $k$ will be either 0 or 1, i.e., a job can be preempted once as the maximum.

In a scheduling process, a job could be allocated to a VM, the type of which is not the same as the job type. In this condition, we will have to use a function to describe the relationships between types and computing capacity, and on that basis to calculate the execution time of the job. The function can be different for different cloud systems and that users can customize the function based on their own cases. Here, we employ a discount fact to describe the relationship as follows.

$$T_j^{exe} = \begin{cases} \frac{reqCom_j}{VCom_i} & \text{if } \mathcal{T}ype_j = \mathcal{V}Type_i \\ \frac{2reqCom_j}{VCom_i} & \text{else} \end{cases} \tag{4}$$

where $T_j^{exe}$ is the execution time of job $j$ on VM $i$, which is calcualated based on the division of $reqCom_j$ and $VCom_i$, and we double the value when the type of a job and an assigned VM does not match.

In relation to the waiting time $T_j^{\text{wait}}$ in the job responsible time, let $V_j^{it}$ be the available time point at which the $VM_i$ can start executing job $j$, then $T_j^{\text{wait}}$ can be expressed as:

$$T_j^{\text{wait}} = \begin{cases} 0 & \text{if queue is empty} \\ V_j^{it} - aT_j & \text{if no preemption} \\ T^{\text{current}} - aT_j + T_i^{\text{switch}} & \text{else} \end{cases} \tag{5}$$

in which $T^{\text{current}}$ is the current time of the scheduling system, $T_i^{\text{switch}}$ is the time cost of a context switch in a VM. In particular, for a newly arrived job $j$, the available time point that the VM $i$ can start the execution of job $j$ is:

$$V_j^{it} = T_{j'}^{\text{wait}} + T_{j'}^{exe} + aT_{j'} \tag{6}$$

where $j'$ means id of the job recorded in the end of the waiting queue (before putting job $j$ in the queue), and $T_{j'}^{\text{wait}}$, $T_{j'}^{exe}$, $aT_{j'}$ mean the is the waiting time, the execution time, and the arrival time point of job $j'$, respectively.

The time spent on execution jobs will involve certain monetary cost in relation to the underlying computing infrastructures.

We are aiming to reduce the cost in our job execution, and define the execution cost of a job as:

$$cost = T_j^{exe} * VC_i \tag{7}$$

where $VC_i$ is the monetary cost per time unit of a VM which executes the job. Furthermore, we also try to ensure that the response time of each job can meet the expected time assigned by each user as much as possible. In the case that a user gets the job results within the expected time, we assume that the job execution is successful, otherwise the job is considered as failed. To present the case, we use $success$ to indicate the execution of job is successful or not as follows:

$$success = \begin{cases} 1, & \text{if } T_j^{res} < QoS_j \\ 0, & \text{else} \end{cases} \tag{8}$$

With the above model, we can calculate the response time of jobs as well as the execution cost of VM instances in a formal way. From this basis, we give the detailed design of our scheduling framework in the following section with targets to reduce the execution cost and increase the success rate of job executions.

## IV. The Proposed Approach

In this section, we introduce the detailed design and implementation our method with some background about DRL.

### A. Deep Reinforcement Learning

*1) Reinforcement Learning:* Reinforcement learning (RL) is a well-established machine learning technique, and it has been widely used to solve various decision problems [37]. Generally, a RL framework consists of a system environment with an agent, a state space, an action space, and a reward function. It works in a way that, for a given optimization problem, the environment will give a reward signal to the agent (to evaluate whether the behavior is good or bad) after the agent chooses a decision action. In the meantime, the whole system will move to a new state. This process will continue in the training process of the agent, and will be terminated until the agent learns enough good decision policy for the optimization problems, which can be achieved by tracking the reward values in the training process.

The most common model used for RL is the Markov decision process (MDP), which can be presented by a five-tuple $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ [38]. Here $\mathcal{S}$ is the state of the system, specifically in cloud job scheduling, it includes the state of VM instances and jobs. $\mathcal{A}$ is the action space which is the set of the ID number of each VM. It means the assignment of the job to which VM for execution. $\mathcal{R}$ is a reward function which is used to evaluate the goodness of the action. $\mathcal{P}$ is a state transfer function that represents the probability of moving from one state to another, and $\gamma$ is the discount factor. As it can be seen that the whole job scheduling process can be expressed as $(s_0, a_0, r_0; \cdots; s_n, a_n, r_n)$, and the updates between different states can be represented by $p(s_{t+1}) = \sum_{a_t} p(s_{t+1}|s_t, a_t)p(a_t|s_t)$. Here, we have $a_t \in \mathcal{A}$, $p(a_t|s_t) = \pi(s_t, a_t)$, and $\pi$ is the scheduling policy of the agent. In the long term, the final cumulative reward received by the agent is $\mathscr{R} = \sum_{t=0}^{n} r_t \gamma^t$.

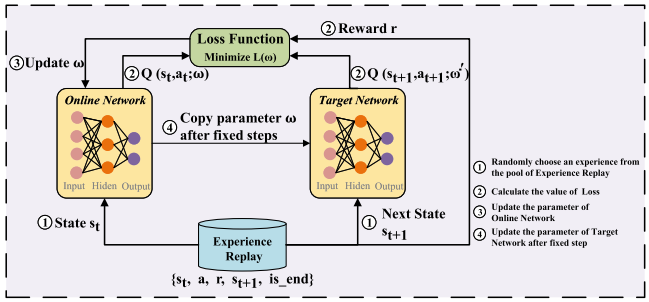



Fig. 2. Detail implementation of the DQN algorithm.

*2) Deep Reinforcement Learning:* Among many DRL algorithms, we have used the most popular Deep Q-learning algorithm as the basis of our implementation. Here, the Q-Learning algorithm is one of the value learning methods with the main idea of learning the value function $\mathcal{Q}^*$, which can represent the maximum long-term cumulative reward that can still be obtained for an action. The Q-Learning algorithm creates a Q-table and continuously update the values in the table till it converges to $\mathcal{Q}*$. However, Q-tables are more suitable for low-dimensional data. Inspired by the use of neural networks for processing high-dimensional data, Deep Q-Learning network (DQN) has been introduced to solve this problem [39]. It usually uses a DNN network to represent the value function, which is denoted as $Q(s, a; \omega)$, where $\omega$ represents the parameters of the neural network. DQN is to continuously learn the parameter $\omega$ through experience, and the prediction value of its neural network is as close to $Q^*$ as possible. Based on the established experience, we set a $Q_{\text{target}}(s_t, a_t; \omega')$ to continuously update $Q$. $Q_{\text{target}}$ and $Q$ have the same network parameters, and the function value of $Q_{\text{target}}$ remains unchanged in the initial state. The parameter value in $Q$ will be updated in real-time at each step, and $Q_{\text{target}}$ will only copy the parameter value in $Q$ after a set of operation steps. In the process of network training, we define the loss function as: $\sum_{i=1}^{|B|}(r_{t_i} + \gamma \text{max}_{\hat{a}\in\mathcal{A}} Q_{\text{target}}(s_{t_i+1}, \hat{a}; \omega') - Q(s_{t_i}, a_{t_i}; \omega))^2$, where $B$ is the mini-batch of training. The goal of the training is to minimize the value of the loss function, and the detail of the process is given in Fig. 2

### B. Design of DRL-Based Preemptive Scheduling

Different from the traditional job scheduling methods, the DRL-based approach can deal with the real-time scheduling problem [12], [4]. In addition, we introduce a preemption mechanism based on DRL-based scheduling to further improve the success rate of job executions. The overall framework of the proposed job scheduling approach is shown in Fig. 3. There, when a new job arrives, the agent will assign the job to a specified VM. After that, the preemption mechanism is triggered to determine whether preemption should be performed. If it is the case, then the VM will run the preemption immediately, otherwise the job will be allocated in the waiting queue of the VM as usual.

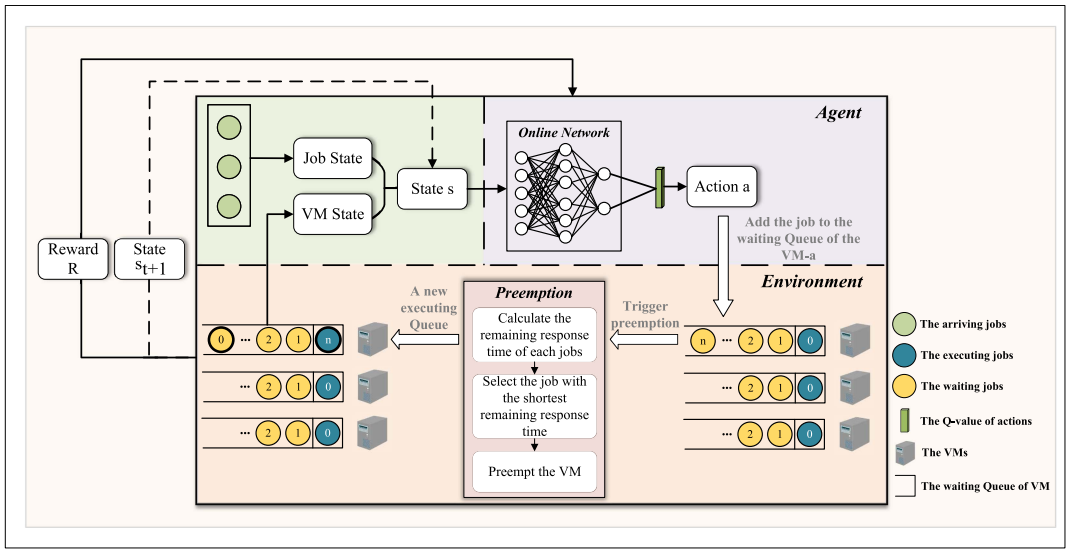



Fig. 3. Proposed DRL-based preemptive scheduling approach for real-time cloud jobs.

The following describes in detail the DRL-based job scheduling method and the design of the preemption mechanism.

*1) Preemption Mechanism:* Usually, there are two types of job scheduling methods: preemptive and non-preemptive, and various works [13], [40] have shown that the preemptive job scheduling has great advantage over the non-preemptive ones such as on successful rate of job executions. In this work, we use the preemption method with limited minimum remaining response time to determine whether a job needs to preempt a VM instance to enter execution. Here, the preemption mechanism is triggered when an agent assigns a job to a VM, and we always choose the job with the shortest remaining response time in the waiting queue to preempt the VM and start execution. Obviously, the remaining response time $T_j^{\text{minres}}$ of a job in the VM waiting queue can be calculated as:

$$T_j^{\text{minres}} = aT_j + QoS_j - T^{\text{current}} \tag{9}$$

*2) Job Scheduling Algorithm:* The main advantage of using DRL for our scheduling problem is that the agent can assign jobs to the appropriate VMs in real-time after the interactions with the system. Moreover, each assignment will be used as experience of the agent, which will help the agent to make better decisions over time. We will train the neural network parameters in the DRL-based job scheduling method using the steps given in Algorithm 1. The algorithm consists of two parts, one is for online decision making and one is for offline training, the details of which are given as follows.

*Online decision:* As mentioned previously, we have implemented the agent with DQN for the decision making in our scheduling system. Generally, when a new job arrives, the agent will get the current state of the system environment, which includes the states of VMs and allocated jobs. Then, the agent uses a DNN to estimate the Q-value of all the possible actions, i.e., which VM the job to be allocated. Afterwards, the job will be allocated to an appropriate VM according to a defined policy. In the meantime, the agent will get a reward from the environment as a feedback for its action. Following this process, we can mathematically model our DRL-based job scheduling problem by the state space, action space and reward function as below.

**Algorithm 1:** The Proposed DRL-Based Preemptive Algorithm.

1: Initialize parameters: $\epsilon$, $\alpha$, $\gamma$, learning frequency $f$, minibatch $S_\Delta$, replay period $\eta$
2: Initialize the size of replay memory $\Delta$ as $N_\Delta$
3: Initialize the parameters of evaluation value function $Q$ with $\omega$, and target value function $\hat{Q}$ with $\omega'$
4: **for** a job $j$ arrives **do**
5: Randomly choose an action (VMs) with the probability $\epsilon$; or choose by $A_j = argmax_A Q(S_j; A; \omega)$
6: Assign the job $j$ to the VM
7: Trigger preemption mechanism
8: Obtain reward $R_j$ based on the action and and change to a new state $S_{j+1}$ at the next time $t_{j+1}$
9: Store the experience $(S_j, A_{j+1}, R_{j+1}, S_{j+1})$ in $\Delta$
10: **if** $j \geqslant 0$ *and* $j \ mod \ f \equiv 0$ **then**
11: Update $\hat{Q} = Q$
12: **end if**
13: Choose experimental samples $S_\Delta$ randomly from $N_\Delta$
14: **for** each experience $(S_j, A_{j+1}, R_{j+1}, S_{j+1})$ in $S_\Delta$ **do**
15: $target_t = r_{t_i} + \gamma \text{max}_{\hat{a} \in \mathcal{A}} Q_{\text{target}}(s_{t_i+1}, \hat{a}; \omega')$
16: Calculate the loss function with $\sum_{i=1}^{|B|}(r_{t_i} + \gamma \text{max}_{\hat{a} \in \mathcal{A}} Q_{\text{target}}(s_{t_i+1}, \hat{a}; \omega') - Q(s_{t_i}, a_{t_i}; \omega))^2$
17: Update the parameters $\omega$
18: **end for**
19: Gradually cut down $\epsilon$ until the last state
20: **end for**

*State space:* In our scheduling system, the state of the whole system is formulated as $S = (S_j, S_{VM}^{tj})$. Here, $S_j$ is the state of the incoming jobs, and $S_{VM}^{tj}$ is the state of all VM instances.

Obviously, the former state depends on the properties of the job itself. In the meantime, we can use the available time point of a VM as the feature the VM, which can be calculated based on the jobs in the waiting queues and ones being executed on the VM. In this case, we can formulate the state space as:

$$S_{tj} = \{reqCom_j, QoS_j, T_{j1}^{\text{wait}}, \ldots, T_{jN}^{\text{wait}}\} \tag{10}$$

where $T_{jN}^{\text{wait}}$ is the waiting time of the current job $j$ waiting on the $N$-th VM.

*Action space:* The action space $A$ is composed by all the possible actions that the agent can perform in a scheduling process. In our cloud job scheduling scenario, the agent is responsible to assign a job to a specified VM. Therefore, the action space can be defined as:

$$A = \{VID_1, VID2 \ldots, VID_N\} \tag{11}$$

*Reward function:* In this work, our goal is to reduce the execution cost of VMs as much as possible. Moreover, to provide better services to users, we are also aiming to ensure that the job response time can meet users' expected response time as much as possible, i.e., improving the successful rate of job executions through the DRL-based preemption approach. To model this mechanism, for a job allocation, we can set the value of the reward given by the environment being inversely proportional to the execution cost and also the job response time. Namely, the lower the cost and the shorter the response time are, the more reward the agent will get. Users can customize their own reward functions following this principle. Specifically, we define our reward function as follows.

$$R = \begin{cases} \frac{1}{T_j^{res}} + \frac{1}{cost}, & \text{if success = 1} \\ 0, & \text{success = 0} \end{cases} \tag{12}$$

*Offline training:* We use a common used experience replay strategy in the offline training phase. In detail, each decision made by the agent is stored as experience in the memory pool shown in Fig. 2, and such historical experiences collected in the job scheduling process are used to train DNN to secure accurate $Q$ values for actions.

Generally, at the beginning of the training, the agent takes a $\epsilon$-greedy method to select actions randomly according to the possibility $\epsilon$, which decays as the number of learning of the agent increases. Moreover, each performed action of the agent is stored in a memory pool $\Delta$ of capacity $N_\Delta$ in this form $(s_t, a_t, r_t, s_{t+1})$, and the parameter training samples of the DNN neural network will be selected from the $\Delta$.

In the training process, as shown in Fig. 2, we set up two deep neural networks, and keep their structure as the same. Among them, one is the online network $\omega$, and one is the target network $\omega'$. The only difference of two networks is their mechanism on parameter updates. Usually, the online network updates its parameters in the case that the DRL agent takes an action. In comparison, the target network performs updates by copying the values of $\theta$ after a round of decisions. This kind of design can effectively eliminate the divergences and oscillations of DNNs.

## V. Evaluation

In this section, we experimentally compare the performance of our proposed approach with existing methods.

### A. Experiment Setup

In our evaluations, we considered an application service provider that leases 10 VM instances from a cloud platform. These VMs are utilized by the application service provider to offer services to its users. To evaluate the performance of our approach, we generated a dataset comprising 8000 jobs, which served as the foundational test data for our experiments.

In order to simulate transactional workloads on the cloud in real scenarios, we have generated workloads with high complexity. These workloads take into account various attributes such as job arrival rates, job length, job type, and the number of jobs. These attributes collectively provide a comprehensive representation of the workload and enable us to assess the performance of our approach under diverse conditions. In our experiments, we assume that the arrival of jobs follows a Poisson distribution. To vary the workload intensity, we set the mean arrival rate to different values selected from the set {10, 15, 20, 25, 30}. The job arrival rate serves as an indicator of job complexity, reflecting the number of job requests within a given time period. Higher job arrival rates correspond to increased possibilities of system congestion, imposing greater demands on the scheduler to make optimal scheduling decisions. By considering different job arrival rates, we can evaluate the robustness and effectiveness of the scheduler across varying workload intensities, allowing us to assess its performance under diverse levels of system congestion. In the meantime, we let the job length follow a normal distribution with setting the mean value to 200 and variance value to 40 respectively. In relation to the job types in workloads, we vary the proportion of I/O intensive jobs to values {0.1, 0.3, 0.5, 0.7, 0.9}, where the I/O intensive jobs refer to tasks that frequently require input/output operations. These types of jobs typically involve frequent interactions with storage devices or external systems, resulting in substantial I/O processing compared to computational tasks. Similar to that, the proportion of VMs with high I/O property is selected from the set of {0.1, 0.3, 0.5, 0.7, 0.9}, where the I/O intensive VMs mean VMs which offer high-performance, low-latency disk support for I/O intensive workloads.

In our experiments, we have chosen four typical scheduling algorithms for real-time cloud jobs as the baselines, i.e., random algorithm, Round-Robin algorithm, Earliest algorithm, and the DQN approach, which are proposed in [4] and [5]. Among them, the Random algorithm allocates jobs to VMs for execution randomly, without any specific ordering or criteria. The Round-Robin algorithm ensures fair job scheduling by selecting VMs in a round-robin order. Each VM is assigned incoming jobs in a cyclic manner, distributing the workload evenly across the available resources. The Earliest algorithm prioritizes newly arrived jobs and schedules them to the earliest available resources. This strategy aims to minimize waiting times by assigning jobs to the VMs that become available first. Our approach is

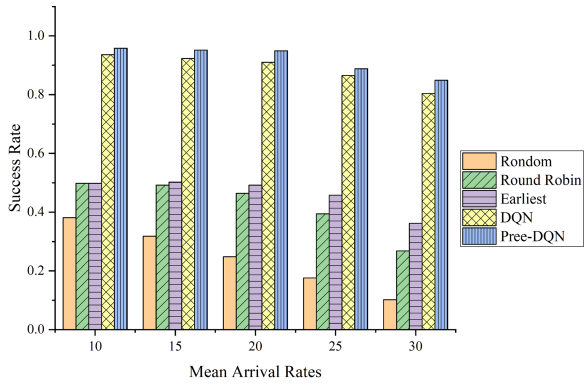
implemented based on DQN with a preemption mechanism, thus we refer to our method as Pree-DQN here. Besides, we use three different metrics to evaluate the performance of the five algorithms, which include *Success Rate*, *Cost*, and *Average Response Time*. Specifically, the *Success Rate* means the number of jobs which are executed successfully, i.e., the response time of a job is not larger than its expected response time. The *Cost* is used to measure the execution cost of the VMs and the *Average Response Time* is for the average time on handling a job.

Following our cost-aware scheduling algorithm, a neural network which includes hidden layers with 20 neurons is used as the DRL agent. Similar to other works [32], we conducted experiments involving various parameter combinations. By comparing the results obtained from these experiments, we were able to identify the optimal parameters that yielded the best performance and results for our approach. This process of parameter exploration and selection allowed us to fine-tune our method and enhance its effectiveness. Specifically, we set the values of replay memory $N_\Delta$ and minibatch $S_\Delta$ to 800 and 30, respectively. Moreover, we set the learning rate to 0.01, the target iteration to 50 decisions per episode, other parameters as: $\gamma = 0.9$, $f = 1$, $\tau = 500$, and the value of $\epsilon$ decreases from 0.9 by 0.002 in each learning iteration.

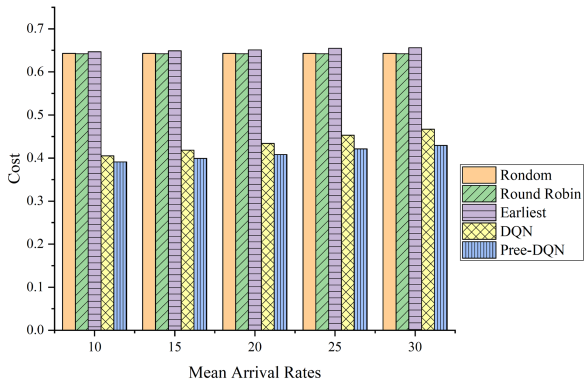
### B. Performance Comparison

*1) Varying Mean Arrival Rates:* First, we compare the performance of the five methods under different workloads by varying the job arrival rates. In this part, the proportion of both the type of jobs and the type of VMs is set to 0.5 and we increase the average arrival rate from 10 to 30 by 5 for each test. The relevant results are reported in Fig. 4. Overall, as observed, with an increase in the arrival rate, there is a slight decrease in success rates, an increase in cost, and an increase in average response time. However, it is noteworthy that the Pree-DQN method consistently outperforms other methods and demonstrates its robustness in the presence of complex workloads. The changes in Pree-DQN's performance are relatively minor compared to other methods, indicating its ability to handle the varying workload conditions effectively. Specifically, from Fig. 4(a), we can see our proposed method achieves a significantly higher success rate than the random, Round-Robin, and earliest algorithms, and also performs better than the DQN method. Moreover, from the execution cost as presented in the Fig. 4(b), it can be seen that the random, Round-Robin and earliest algorithms perform almost the same, and our proposed method still performs the best, the cost of which is about 50% smaller than the three approaches. Similarly, as reported in Fig. 4(c), DQN perform much better than the conventional three algorithms, but worse than the proposed Pree-DQN. Specifically, when jobs arrive at the highest rate, our method and DQN present much better performance than the other approaches.

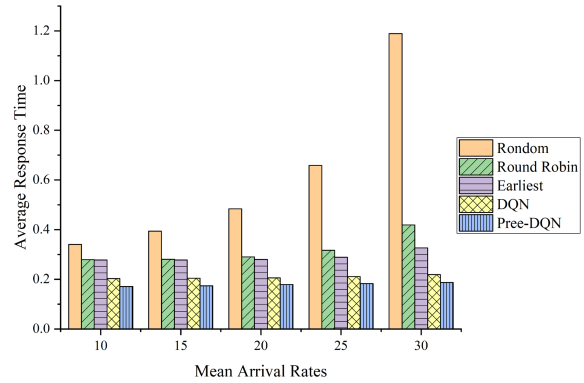
*2) Varying Job Type Proportions:* Next, we evaluate the performance of the five approaches by varying the proportions of job types with setting the value to 0.1, 0.3, 0.5, 0.7, and 0.9, respectively. Here, 0.1 means that 10% of all jobs to be scheduled




(a) Success Rate.



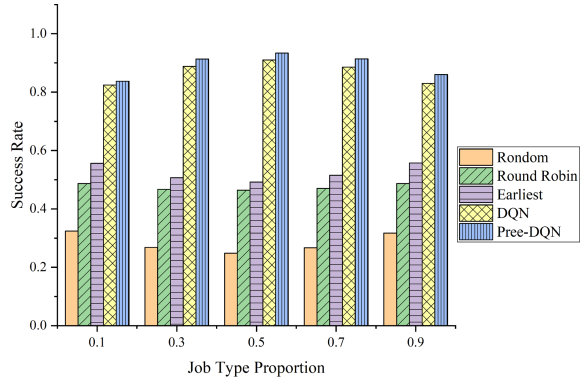

(b) Cost.



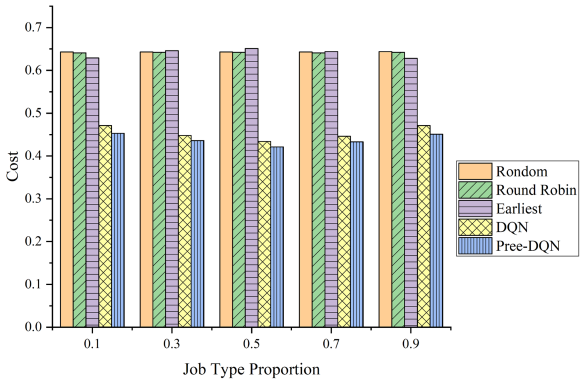

(c) Average Response Time.

Fig. 4. Comparison by varying the mean arrival rates of jobs.

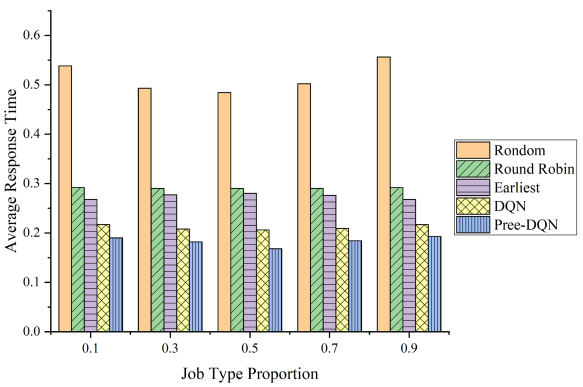
are I/O sensitive while others are computing intensive. In this case, we set the value of the mean arrival rate to 20 and the proportion of the VM types to 0.5. The relevant results are presented in Fig. 5. From Fig. 5(a) that our propose Pree-DQN still perform much better than other four algorithms including DQN on the success rate on job executions. Moreover, from Fig. 5(b), although DQN has has much lower cost (i.e., about 20%) than the conventional algorithms, our Pree-DQN still can always perform




(a) Success Rate.

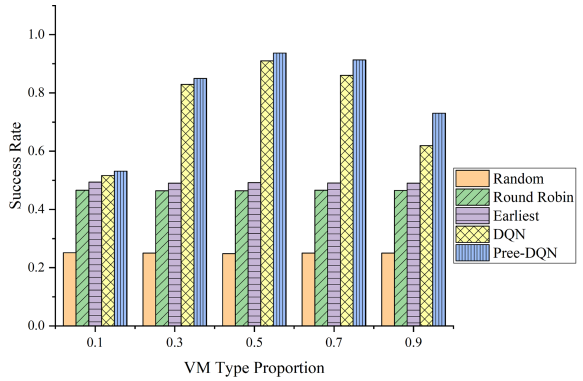



(b) Cost.

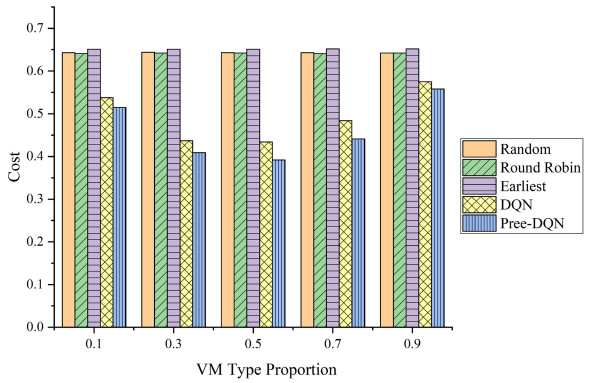



(c) Average Response Time.

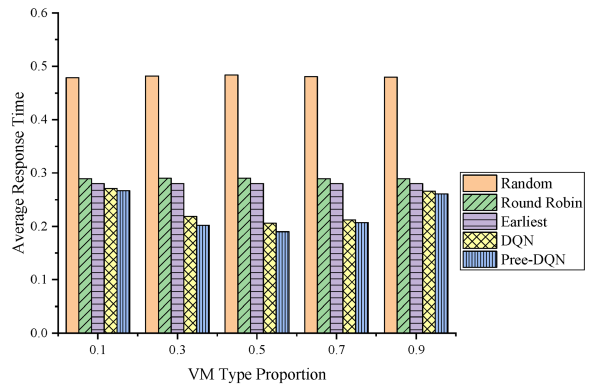
Fig. 5. Comparison by varying the proportions of job types.




(a) Success Rate.




(b) Cost.




(c) Average Response Time.

Fig. 6. Comparison by varying the proportions of VM types.

slightly better than DQN with a lower execution cost. In addition, our method also outperforms other four methods on average response time as reported Fig. 5(c).

*3) Varying VM Type Proportions:* Last, we evaluate the performance by varying the value of proportion of VMs from 0.1 to 0.9, while 0.1 means that 10% of all the VMs are I/O intensive and the rest ones are computing intensive. Besides, we fix the value of the mean arrival rate to 20 and the proportion of job types to 0.5. The results about success rate, cost and average responsible time for different job proportions are given in Fig. 6. We can observe that Pree-DQN still outperforms the other four algorithms in all the cases. For example, about the average responsible time, although the earliest algorithm has considered the time cost, it cannot handle the dynamical cloud environment well. In contrast, DQN and our Pree-DQN can adjust their scheduling strategies based on the states of the cloud environments, and thus can perform better in a global view. In addition, with these results, we can generally conclude that our method can achieve robust performance with regarding to the configuration of VM types.

## VI. Conclusion

Although DRL has been used to optimize cloud job scheduling in current studies, all of them focus on learning decisions on allocating which job to which VM, and ignore the fact that the job queue on each VM can be optimized during the training of DRL agents. To further extend the performance optimization space for real-time job executions, in this work, as the first time, we have proposed a DRL-based preemptive scheduling approach for cloud jobs. Specifically, we focus on reducing execution cost of VMs and guaranteeing users' expected response time can be maximumly achieved. We have given the detailed design of our method and compared its performance with other scheduling algorithms. Our experimental results have shown that the proposed approach can outperform existing scheduling algorithms in terms of reducing execution cost of VMs and improving the success rate of job executions.

Our approach expands the optimization space of the scheduling problem by incorporating preemptive strategies. However, it does have limitations when dealing with complex scenarios. First, it is specifically designed for independent jobs and may not effectively handle situations involving real-time workflows with interrelated jobs and shared performance goals. Second, in larger-scale systems with a significant number of virtual machines, a single agent may encounter performance issues. To overcome these limitations, our future work will involve exploring more advanced strategies such as multi-agent reinforcement learning, to address the challenges posed by complex scenarios and large-scale systems in cloud job scheduling.

## References


[1] D. C. Marinescu, *Cloud Computing: Theory and Practice*. San Mateo, CA, USA: Morgan Kaufmann, 2022.

[2] X. Chen et al., "A WOA-based optimization approach for task scheduling in cloud computing systems," *IEEE Syst. J.*, vol. 14, no. 3, pp. 3117–3128, Sep. 2020.

[3] J. Yan et al., "Energy-aware systems for real-time job scheduling in cloud data centers: A deep reinforcement learning approach," *Comput. Elect. Eng.*, vol. 99, 2022, Art. no. 107688.

[4] F. Cheng, Y. Huang, B. Tanpure, P. Sawalani, L. Cheng, and C. Liu, "Cost-aware job scheduling for cloud instances using deep reinforcement learning," *Cluster Comput.*, vol. 25, no. 1, pp. 619–631, 2022.

[5] Y. Huang et al., "Deep adversarial imitation reinforcement learning for QoS-aware cloud job scheduling," *IEEE Syst. J.*, vol. 16, no. 3, pp. 4232–4242, Sep. 2022.

[6] S. H. Sahraei, M. M. R. Kashani, J. Rezazadeh, and R. Farahbakhsh, "Efficient job scheduling in cloud computing based on genetic algorithm," *Int. J. Commun. Netw. Distrib. Syst.*, vol. 22, no. 4, pp. 447–467, 2019.

[7] E. H. Houssein, A. G. Gad, Y. M. Wazery, and P. N. Suganthan, "Task scheduling in cloud computing based on meta-heuristics: Review, taxonomy, open challenges, and future trends," *Swarm Evol. Comput.*, vol. 62, 2021, Art. no. 100841.

[8] S. Meng et al., "Security-aware dynamic scheduling for real-time optimization in cloud-based industrial applications," *IEEE Trans. Ind. Inform.*, vol. 17, no. 6, pp. 4219–4228, Jun. 2021.

[9] J. Liu et al., "A low-cost multi-failure resilient replication scheme for high-data availability in cloud storage," *IEEE/ACM Trans. Netw.*, vol. 29, no. 4, pp. 1436–1451, Aug. 2021.

[10] L. Wang and E. Gelenbe, "Adaptive dispatching of tasks in the cloud," *IEEE Trans. Cloud Comput.*, vol. 6, no. 1, pp. 33–45, First Quarter 2018.

[11] N. Liu et al., "A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 372–382.

[12] M. Cheng, J. Li, and S. Nazarian, "DRL-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers," in *Proc. 23rd Asia South pacific Des. Automat. Conf.*, 2018, pp. 129–134.

[13] G. Kaur and S. Midha, "A preemptive priority based job scheduling algorithm in green cloud computing," in *Proc. 6th Int. Conf.-Cloud Syst. Big Data Eng.*, 2016, pp. 152–156.

[14] M. Kumar, S. C. Sharma, A. Goel, and S. P. Singh, "A comprehensive survey for scheduling techniques in cloud computing," *J. Netw. Comput. Appl.*, vol. 143, pp. 1–33, 2019.

[15] J.-W. Lin, C.-H. Chen, and C.-Y. Lin, "Integrating QoS awareness with virtualization in cloud computing systems for delay-sensitive applications," *Future Gener. Comput. Syst.*, vol. 37, pp. 478–487, 2014.

[16] J. Zeng, D. Ding, K. Kang, H. Xie, and Q. Yin, "Adaptive DRL-based virtual machine consolidation in energy-efficient cloud data center," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 11, pp. 2991–3002, Nov. 2022.

[17] A. Senthil Kumar and M. Venkatesan, "Multi-objective task scheduling using hybrid genetic-ant colony optimization algorithm in cloud environment," *Wirel. Pers. Commun.*, vol. 107, no. 4, pp. 1835–1848, 2019.

[18] H. Yuan, J. Bi, W. Tan, M. Zhou, B. H. Li, and J. Li, "TTSA: An effective scheduling approach for delay bounded tasks in hybrid clouds," *IEEE Trans. Cybern.*, vol. 47, no. 11, pp. 3658–3668, Nov. 2017.

[19] M. Shojafar, S. Javanmardi, S. Abolfazli, and N. Cordeschi, "FUGE: A joint meta-heuristic approach to cloud job scheduling algorithm using fuzzy theory and a genetic method," *Cluster Comput.*, vol. 18, no. 2, pp. 829–844, 2015.

[20] M.-R. Shie, C.-Y. Liu, Y.-F. Lee, Y.-C. Lin, and K.-C. Lai, "Distributed scheduling approach based on game theory in the federated cloud," in *Proc. Int. Conf. Inf. Sci. Appl.*, 2014, pp. 1–4.

[21] J. Vilaplana, F. Solsona, I. Teixidó, J. Mateo, F. Abella, and J. Rius, "A queuing theory model for cloud computing," *J. Supercomput.*, vol. 69, no. 1, pp. 492–507, 2014.

[22] Q. Liu, L. Cheng, T. Ozcelebi, J. Murphy, and J. Lukkien, "Deep reinforcement learning for IoT network dynamic clustering in edge computing," in *Proc. 19th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2019, pp. 600–603.

[23] H. Peng and X. Shen, "Deep reinforcement learning based resource management for multi-access edge computing in vehicular networks," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 4, pp. 2416–2428, Fourth Quarter 2020.

[24] Q. Liu, L. Cheng, A. L. Jia, and C. Liu, "Deep reinforcement learning for communication flow control in wireless mesh networks," *IEEE Netw.*, vol. 35, no. 2, pp. 112–119, Mar./Apr. 2021.

[25] Z. Zhang, H. Liu, M. Zhou, and J. Wang, "Solving dynamic traveling salesman problems with deep reinforcement learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 4, pp. 2119–2132, Apr. 2023.

[26] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38, Nov. 2017.

[27] J. Li, X. Zhang, Z. Wei, J. Wei, and Z. Ji, "Energy-aware task scheduling optimization with deep reinforcement learning for large-scale heterogeneous systems," *CCF Trans. High Perform. Comput.*, vol. 3, pp. 383–392, 2021.

[28] K. X. Kang, D. Ding, H. M. Xie, Q. Yin, and J. Zeng, "Adaptive DRL-based task scheduling for energy-efficient cloud computing," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 4, pp. 4948–4961, Dec. 2022.

[29] Q. Liu, T. Xia, L. Cheng, M. Van Eijk, T. Ozcelebi, and Y. Mao, "Deep reinforcement learning for load-balancing aware network control in IoT edge systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 6, pp. 1491–1502, Jun. 2022.

[30] W. Song, X. Chen, Q. Li, and Z. Cao, "Flexible job-shop scheduling via graph neural network and deep reinforcement learning," *IEEE Trans. Ind. Inform.*, vol. 19, no. 2, pp. 1600–1610, Feb. 2023.

[31] Z. Chen, L. Zhang, X. Wang, and K. Wang, "Cloud–edge collaboration task scheduling in cloud manufacturing: An attention-based deep reinforcement learning approach," *Comput. Ind. Eng.*, vol. 177, 2023, Art. no. 109053.

[32] L. Cheng et al., "Cost-aware real-time job scheduling for hybrid cloud using deep reinforcement learning," *Neural Comput. Appl.*, vol. 34, no. 21, pp. 18 579–18 593, 2022.

[33] M. S. Mekala et al., "A DRL-based service offloading approach using DAG for edge computational orchestration," *IEEE Trans. Computat. Social Syst.*, early access, Apr. 12, 2022, doi: 10.1109/TCSS.2022.3161627.

[34] A. Chung, J. W. Park, and G. R. Ganger, "Stratus: Cost-aware container scheduling in the public cloud," in *Proc. ACM Symp. Cloud Comput.*, 2018, pp. 121–134.

[35] Z. Li et al., "A security and cost aware scheduling algorithm for heterogeneous tasks of scientific workflow in clouds," *Future Gener. Comput. Syst.*, vol. 65, pp. 140–152, 2016.
[36] Z. Liu, S. Wang, Q. Sun, H. Zou, and F. Yang, "Cost-aware cloud service request scheduling for SaaS providers," *Comput. J.*, vol. 57, no. 2, pp. 291–301, 2014.
[37] L. Buşoniu, T. de Bruin, D. Tolić, J. Kober, and I. Palunko, "Reinforcement learning for control: Performance, stability, and deep approximators," *Annu. Rev. Control*, vol. 46, pp. 8–28, 2018.
[38] M. V. Otterlo, "Solving relational and first-order logical Markov decision processes: A survey," in *Reinforcement Learning*. Berlin, Germany: Springer, 2012, pp. 253–292.
[39] J. Fan, Z. Wang, Y. Xie, and Z. Yang, "A theoretical analysis of deep Q-learning," in *Proc. Learn. Dyn. Control*, PMLR, 2020, pp. 486–489.
[40] A. Arafa, R. D. Yates, and H. V. Poor, "Timely cloud computing: Preemption and waiting," in *Proc. IEEE 57th Annu. Allerton Conf. Commun. Control Comput.*, 2019, pp. 528–535.

**Long Cheng** (Senior Member, IEEE) received the BE degree from the Harbin Institute of Technology, China in 2007, the MSc degree from the University of Duisburg-Essen, Germany in 2010 and the PhD degree from the National University of Ireland Maynooth in 2014. He is a full professor with the School of Control and Computer Engineering, North China Electric Power University in Beijing. He was an assistant professor at Dublin City University, and a Marie Curie fellow at University College Dublin. He also has worked in organizations such as Huawei Technologies Germany, IBM Research Dublin, TU Dresden, and TU Eindhoven. He has published more than 80 papers in journals and conferences like *IEEE Transactions on Parallel and Distributed Systems*, *IEEE/ACM Transactions on Networking*, *IEEE Transactions on Computers*, *IEEE Transactions on Services Computing*, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *IEEE Transactions on Cloud Computing*, HPCA, ICPP and Euro-Par, etc. His research focuses on distributed systems, deep learning, cloud computing, and process mining. He is an associate editor of *IEEE Transactions on Consumer Electronics*, and a co-chair of *Journal of Cloud Computing*.

**Yue Wang** received the BEng degree from ShanXi University in 2021. She is currently working toward the master's degree with North China Electric Power University (NCEPU) in Beijing. Her research mainly focuses on cloud computing, machine learning, and reinforcement learning.

**Feng Cheng** received the BS and MS degrees in mathematics from Southwest Jiaotong University in 2006 and 2009 respectively, and the PhD degree in statistics and management from Tongji University, Shanghai, China, in 2013. She is an assistant professor with the Southwest Jiaotong University, Chengdu, China. Her research interests include stochastic operations and algorithms.

**Cheng Liu** (Member, IEEE) received the BE and ME degrees in microelectronic engineering from the Harbin Institute of Technology in 2009 and the PhD degree in computer engineering from The University of Hong Kong in 2016. He is an associate professor of Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing. His research focuses on FPGA based reconfigurable computing and domain-specific computing systems.

**Zhiming Zhao** (Senior Member, IEEE) received the PhD degree in computer science from the University of Amsterdam (UvA), Amsterdam, The Netherlands, in 2004. He is an associate professor with the Group of Multi-Scale Networked Systems, System and Networking Laboratory, UvA. He leads the Research Team of "quality critical distributed computing," and coordinates the UvA contributions to several EU projects, including SWITCH, ENVRIPLUS, ENVRIFAIR, ARTICONF, VRE4EIC, BlueCloud, and CARIFY. His research interests include cloud computing, software-defined infrastructure, DevOps, Big Data management, and blockchain.

**Ying Wang** (Member, IEEE) received the BE and MS degrees from the Harbin Institute of Technology, in 2007 and 2009 respectively, and the PhD degree from ICT in 2014. He is an associate professor with the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS). His research interests includes computer architecture and VLSI design, specifically memory system, on-chip interconnects, resilient and energy-efficient architecture, machine learning accelerators, and parallel data systems.