Received August 15, 2018, accepted September 17, 2018, date of publication September 28, 2018, date of current version October 19, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2872674

DRL-Scheduling: An Intelligent QoS-Aware Job Scheduling Framework for Applications in Clouds

YI WEI, LI PAN, SHIJUN LIU, LEI WU, AND XIANGXU MENG
School of Software, Shandong University, Jinan 250101, China

Corresponding authors: Li Pan (panli@sdu.edu.cn) and Shijun Liu (lsj@sdu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61872222, in part by the National Key Research and Development Program of China under Grant 2017YFA0700601, in part by the Key Research and Development Program of Shandong Province under Grant 2017CXGC0605, Grant 2017CXGC0604, and Grant 2018GGX101019, in part by Shandong University through the Young Scholars Program, and in part by the Special Funds of Taishan Scholar Construction Project.

ABSTRACT As an increasing number of traditional applications migrated to the cloud, achieving resource management and performance optimization in such a dynamic and uncertain environment becomes a big challenge for cloud-based application providers. In particular, job scheduling is a non-trivial task, which is responsible for allocating massive job requests submitted by users to the most suitable resources and satisfying user QoS requirements as much as possible. Inspired by recent success of using deep reinforcement learning techniques to solve AI control problems, in this paper, we propose an intelligent QoS-aware job scheduling framework for application providers. A deep reinforcement learning-based job scheduler is the key component of the framework. It is able to learn to make appropriate online job-to-VM decisions for continuous job requests directly from its experiences without any prior knowledge. Experimental results using synthetic workloads and real-world NASA workload traces show that compared with other baseline solutions, our proposed job scheduling approach can efficiently reduce average job response time (e.g., reduced by 40.4% compared with the best baseline for NASA traces), guarantee the QoS at a high level (e.g., job success rate is higher than 93% for all simulated changing workload scenarios), and adapt to different workload conditions.

INDEX TERMS Cloud computing, deep Q-Learning, job scheduling, QoS, reinforcement learning.

I. INTRODUCTION

With the rapid development of technologies such as virtualization and service-oriented architectures, cloud computing has emerged as one of the most popular computing paradigms in today's computer industry. IaaS (Infrastructure as a Service) vendors such as Amazon EC2 and Microsoft Azure take the responsibility of providing powerful computing and storage resources for their users. From the perspective of application providers, they are able to rent appropriate virtualized resources from IaaS vendors to easily construct their IT infrastructure with a lower cost. Therefore, an increasing number of traditional public or enterprise applications have been migrated to the cloud.

In the cloud environment, resource management of cloud-based applications is an important issue which has attracted great attention. In particular, achieving efficient job scheduling is one of the most essential goals. Cloud-based applications continuously receive jobs from users. They need an intelligent job dispatcher to make online decisions to assign massive jobs to suitable resources. Compared with job scheduling in grid computing and traditional distributed systems, the characteristics of clouds make job scheduling for cloud-based applications more challenging. Firstly, IaaS vendors normally provide multiple VM instance types rather than one suit-for-all instance type for users to choose [1]. For example, application providers can rent different types of VM instances such as high-CPU VM instances or high-memory VM instances from Amazon EC2 based on their actual needs. Furthermore, the performance of VM instances is constantly fluctuating. Even for a particular VM instance, its processing speed is often changed under the influence of loads, network or some other factors of datacenters. In addition, the workloads of diverse cloud-based applications differ widely and may vary significantly over time [2]. Taking an e-commerce application for example, the proportions of browsing, bidding, and purchasing requests always change in a day. More importantly, cloud-based applications are expected to not only execute jobs submitted by users but also guarantee varied users' QoS requirements [3].

Workloads of applications can be classified into two types: transactional workloads and non-interactive workloads [4]. Workloads of transactional applications such as web applications always vary with time. As user jobs are independent and arriving continuously, the job scheduler is required to efficiently dispatch jobs to resources in time and achieve desirable QoS levels. In general, average response time is an important QoS requirement for transactional applications. By contrast, workloads generated by bag-of-task applications and workflow applications belong to non-interactive workloads. For these applications, user jobs arrive in batches. Offline scheduling is allowed because real-time response is not a primary requirement. The scheduling goal of non-interactive workloads is to minimize total cost or makespan of all batch jobs. Since most existing job assignment approaches are only suitable for offline scheduling rather than real-time scenarios [5], in this paper, we focus on the study of online job scheduling for transactional applications in clouds.

Dispatching continuous job requests of transactional applications in a dynamic and uncertain cloud environment is a difficult online scheduling task. Job routing decisions are typically made each time when a new job arrives. Application providers only know the information of their rented VM instances and the new job. They have no knowledge of future jobs in terms of arrival time, resource demands, and other properties. In our work, we try to investigate whether cloud-based applications can learn to make wise job dispatching decisions on their own. That is to say, cloud-based applications are expected to have the capacity of automatically assigning the incoming jobs to appropriate cloud resources and ensuring a high user satisfaction level.

In order to fulfill the above goal, we propose an intelligent QoS-aware job scheduling framework using deep reinforcement learning (DRL). Reinforcement learning (RL) [6] is a popular machine learning technique for solving automatic control problems. An outstanding feature of RL-based methods is they do not require any prior knowledge of the underlying system. Thus RL seems possible to help cloud-based applications learn the long-term optimal job scheduling policy online. Moreover, to avoid the curse of dimensionality of traditional RL techniques, RL combined with deep neural network (i.e. DRL) is adopted in our work. We have evaluated our proposed approach in three workload scenarios: stable workload scenarios, changing workload scenarios, and real-world workload scenarios. Compared with five baseline solutions, our proposed job scheduling approach performs much better in terms of the average response time, resource utilization rate, and job success rate.

The remainder of the paper is organized as follows. Related work is discussed in section II. In section III, we define the system model and formulate the job scheduling problem of cloud-based applications. Section IV introduces relevant theories of reinforcement learning and elaborates the implementation of our proposed intelligent job scheduler. In section V, we present an evaluation in three scenarios and discuss experimental results. Finally, we draw some conclusions and give hints for future work in section VI.

II. RELATED WORK

In the research area of distributed systems, job scheduling is an essential but challenging problem which has been studied for decades. Particularly in cloud computing, similar to its predecessors such as grid computing, various job assignment algorithms and approaches have been proposed [7], [8].

Theoretical research is a common method which typically takes advantage of some statistical theories, such as queuing theory [9]–[11], control theory [12], [13] or game theory [14], [15], to build a mathematical model of the underlying system. Based on these models, theoretical analysis of job scheduling schemes is conducted. For example, the cloud center is modeled as a M/G/m/m+r queuing system in [9], thereby the request response time obtained easily. Reference [12] proposes a hybrid job scheduling approach which uses fuzzy theory to improve the performance of basic genetic algorithm. Reference [14] presents a distributed scheduling scheme which uses Nash equilibrium to optimize the social welfare. In addition, many studies regard the job scheduling task as an NP-hard problem whose goal is to optimize the job-to-resource assignment in a known and stable environment [16]. Thus heuristics techniques such as genetic algorithm (GA) [17], ant colony optimization (ACO) [18], and particle swarm optimization (PSO) [19] are widely used to design job scheduling approaches.

Although theoretical research and heuristics techniques have successfully solved job scheduling problems to some extent, these methods are not applicable to our topic. Theoretical analytical approaches are more suitable for relatively stable environments. They do not consider the dynamic nature and diversity of clouds. Heuristics based algorithms can be used to solve offline job scheduling problems, such as dispatching given workflow jobs under resources and SLA constraints. However, they are unable to achieve efficient online job scheduling for transactional applications in clouds.

In the domain of artificial intelligence, machine learning is an active methodology which enables programs to automatically learn patterns or models of the underlying system [20]. In particular, reinforcement learning (RL) [6] is a popular machine learning method which is widely used to solve challenging decision making problems. RL agents without any prior knowledge are able to learn how to behave properly to obtain long-term rewards from experience interacting with the unknown environment. Nowadays, reinforcement learning has been successfully applied to solve problems in cloud computing systems. For instance, in [21], the reinforcement learning approach is applied to learn the optimal bidding strategy of application providers. References [22]–[24] propose RL-based resource provisioning and power management approaches for cloud datacenters. References [25]–[28] make use of RL framework to realize resource horizontal or vertical

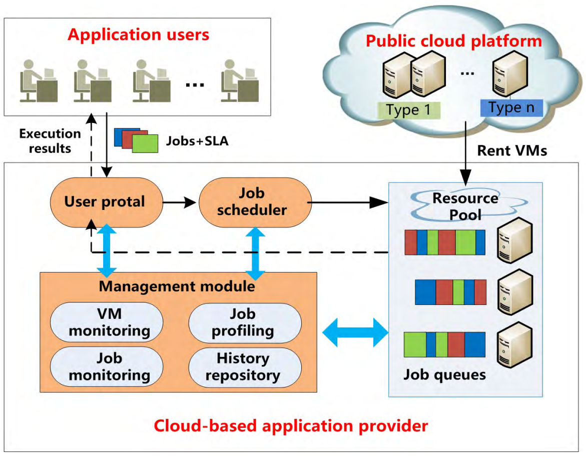


FIGURE 1. Job scheduling framework in the cloud.

auto-scaling along with the change of workload. However, there are less studies focus on how to utilize finite cloud resources rationally. Reference [29] tries to deal with this problem but it simplifies the RL state space in order to avoid the curse of dimensionality. In fact, the emerging deep reinforcement learning (DRL) which combines reinforcement learning with neural network can solve complicated control problems with large state and action spaces [30], [31]. Therefore, in this paper, we design a DRL-based online job scheduling framework which can obtain better performance.

III. SYSTEM ARCHITECTURE AND PROBLEM STATEMENT

A. SYSTEM MODEL

We consider a common three-tier cloud environment [32] which involves three roles: IaaS vendors, application providers, and application end users. IaaS vendors offer various cloud resources on their public cloud platforms. Cloud-based application providers rent several VM instances from IaaS vendors to construct their resource pools. Their applications are deployed on these VM instances and continually receive diverse job requests submitted by application end users. End users are allowed to claim their QoS requirements such as the expected job response time. From the perspective of application providers, how to realize efficient job scheduling is a non-trivial task.

Fig.1 illustrates the job scheduling framework of cloud-based applications. It consists of three key components: user portal, job scheduler, and management module. User portal is the bridge connecting the application and its end users. It is responsible for receiving incoming job requests from application users. Besides, execution results of all jobs will be delivered to specific users through the portal. Job scheduler is the core component which makes job distributed routing decisions. A certain VM instance will be selected to execute the new job request immediately or later. Management module involves several functions which can help user portal and job scheduler work better. Specifically, VM monitor continuously observes the load and performance of each VM instance while job monitor keeps track of all jobs' distribution and execution. Job profiling is able to identify job types or other characteristics of jobs. History repository stores the historical records of job processing and VM performance. All above-mentioned information can be used to improve the job scheduling mechanism for cloud-based applications.

B. PROBLEM FORMULATION AND ASSUMPTIONS

1) APPLICATION WORKLOADS

In our model, the cloud-based application workload is considered as non-dependent jobs submitted by numerous end users. Job requests arrive to the application in an online fashion. Application providers do not have knowledge about the incoming workload in advance. User jobs can be classified into several different types, such as computing intensive jobs and I/O intensive jobs. Application users are allowed to declare their personalized QoS requirements when submitting jobs. Based on the above assumptions, each job can be described as:

$$job_i = \{jID_i, arrivalT_i, jType_i, jSize_i, QoS_i\} \tag{1}$$

where jID_i, $arrivalT_i$, $jType_i$, $jSize_i$ and QoS_i represent the job's id, arriving time, type, length (i.e. the number of instructions), and QoS requirements respectively.

2) CLOUD RESOURCES

In cloud computing market, IaaS vendors offer diverse VM instances with different types for application providers to choose. Task Amazon EC2 for example, common types of VM instances include standard type, high-CPU type and high-memory type. Because user jobs also belong to different types, a given job may take different execution time on different types of VM instances. For example, a computing intensive job runs faster on high-CPU VM instances than on similar configured high-memory VM instances. Moreover, application providers can rent VM instances under different charging modes, e.g. on-demand pattern, reserved pattern, and spot pattern [33]. In our model, we simplify the rented VM instances of application providers. We suppose there are limited reserved VM instances in the application's resource pool. The total number of rented VM instances is M and each instance is defined as:

$$VM_j = \left\{VMid_j, VMtype_j, vCom_j, vIO_j\right\} \tag{2}$$

where $VMid_j$ identifies the VM, $VMtype_j$ denotes its instance type, $vCom_j$ and vIO_j are the average processing speeds for computing intensive jobs and I/O jobs separately.

3) JOB SCHEDULING AND EXECUTION MECHANISM

Job scheduler is in charge of allocating user jobs to rented reserved VM instances. We assume as long as a new job arriving, it will be dispatched immediately. A job can be executed on any VM instance in the resource pool. When a job is assigned to a VM instance, it will enter the corresponding job waiting queue of that instance. Supposing all waiting queues are large enough to temporarily hold all unexecuted jobs. Jobs in waiting queues are served in FCFS (first come first service) manner. Each VM instance can only execute a single job at any time. Preemption is not allowed which means any job cannot be interrupted by other jobs during its execution. If a new job is allocated to a VM instance whose job waiting queue is empty, this job will be processed immediately. Otherwise, this job cannot begin to execute until all earlier arrival jobs in this queue have been finished.

Job response time T_i is defined as the total amount of time that job_i stays in the application. It can be calculated by the following equation:

$$T_i = T_i^{exe} + T_i^{wait} \tag{3}$$

where T_i^{exe} represents the amount of time that the specified VM instance needs to process job_i, T_i^{wait} indicates the duration of job_i spending in the waiting queue. Supposing the job scheduler assigns job_i to VM_j, then the expected execution time of job_i is

$$T_{ij}^{exe'} = \begin{cases} jSize_i/vCom_j, & \textit{if } job_i \textit{ is computing intensive} \\ jSize_i/vIO_j, & \textit{if } job_i \textit{ is IO intensive} \end{cases} \tag{4}$$

As we mentioned before, the performance of VM instances are not constant in clouds, $vCom_j$ and vIO_j are the average processing speeds of VM_j, thus

$$T_i^{exe} \approx T_{ij}^{exe'} \tag{5}$$

Supposing there are $queueL_j^i$ jobs waiting in the queue of VM_j when job_i arriving. $job_{i'}$ denotes the last job dispatched to VM_j before job_i. The latency time of job_i can be calculated as

$$T_{ij}^{wait} = \begin{cases} 0, & \textit{if } queueL_j^i = 0 \\ VM\_T_{ij}^{idle} - arrivalT_i, & \textit{else} \end{cases} \tag{6}$$

in which $VM\_T_{ij}^{idle}$ is the available time of VM_j for current job job_i, and it can be obtained by

$$VM\_T_{ij}^{idle} = \begin{cases} VM\_T_{i'j}^{idle} + T_{i'}^{exe}, & \textit{if } VM\_T_{i'j}^{idle} > arrivalT_{i'} \\ arrivalT_{i'} + T_{i'}^{exe}, & \textit{else} \end{cases} \tag{7}$$

where $VM\_T_{i'j}^{idle}$ is the available time of VM_j for last job $job_{i'}$, $T_{i'}^{exe}$ and $arrivalT_{i'}$ are the execution time and arrival time of $job_{i'}$ respectively.

When VM_j has finished processing job_i, execution results of job_i will be delivered back to the relevant user through user portal. The completion time of job_i can be expressed as

$$T_i^{leave} = arrivalT_i + T_i \tag{8}$$

4) QOS-AWARE JOB SCHEDULING PROBLEM

Most cloud-based applications allow end users to define QoS requirements when submitting job requests. In our research, we assume the cloud-based application workload is transactional workload. Hence job response time is an important QoS metric for both cloud-based applications and end users. For each job, if QoS_i represents user's acceptable maximum response time, we use the following formula to judge whether a job-to-VM assignment is success.

$$success(job_i, VM_j) = \begin{cases} 1, & \textit{if } T_i \leqslant QoS_i \\ 0, & \textit{else} \end{cases} \tag{9}$$

In order to attempt to satisfy users' demands, a fast online job scheduler is required to help applications appropriately dispatch jobs to limited VM instances.

IV. INTELLIGENT JOB SCHEDULER WITH DEEP REINFORCEMENT LEARNING

As we mentioned in section III.A, the job scheduler plays a significant role in the job scheduling framework of cloud-based applications. The performance of job schedulers largely depends on what underlying algorithm is adopted. In this section, we will present an intelligent job scheduling algorithm which is able to make appropriate job dispatching decisions under the constraints of VM instances and QoS requirements. Because the theoretical foundation of our proposed job scheduling algorithm is reinforcement learning, we will introduce the relevant basic theories first, and then describe our proposed intelligent job scheduling algorithm in detail.

A. BACKGROUND OF REINFORCEMENT LEARNING AND DRL

Reinforcement learning (RL) [6] is an important area of machine learning which is inspired by the inherent learning style of living beings. Considering the general learning model of RL, it consists of an agent, an environment, a finite state space S, a set of available actions A for the agent, and a reward function: $S \times A \to R$. The basic idea of RL is to let the agent learn to make wise decisions by trial-and-error interactions with the environment. At each decision time t, the agent takes an action a_t based on the observation of current state s_t in the environment. Once the action has been performed, the state of the environment will move to a new state s_{t+1}. At the same time, the agent will receive a reward r_t which reflects the value of state transition. Such a kind of agent-environment interaction is a continual process. The RL agent is a long-sighted decision maker thus its goal is to maximize its expected cumulative rewards over time: $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$, where $\gamma \in (0, 1]$ is a factor discounting future rewards. Its goal can be achieved by learning a policy which guides the agent how to choose proper actions at different states.

Q-learning [34] is one of the most popular model-free RL algorithms. The Q-learning agent is not required to have any prior knowledge about the system, such as the state transition probabilities. It is able to learn how to make rational decisions from experiences. The agent keeps a value function $Q(s, a)$ for each state-action pair, which represents the expected long-term reward when taking action a at state s. According to the value function, the agent can know the estimated q-value for each action at current state. Based on these q-values, the agent

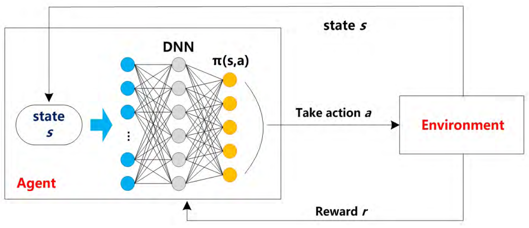


FIGURE 2. Reinforcement learning with DNN.

decides which actions should be taken to achieve the maximum cumulative rewards in the long run. Each time when an interaction happens, the value function will be updated iteratively by the following expression:

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{old\ value} + \alpha * [\overbrace{\underbrace{r(s_t, a_t)}_{reward} + \gamma * \underbrace{max_a Q(s_{t+1}, a))}_{max\ future\ value}}^{expected\ discounted\ reward} - \overbrace{Q(s_t, a_t)}^{old\ value}] \tag{10}$$

where $\alpha \in (0, 1]$ is the learning rate, $r(s_t, a_t)$ is the reward observed by taking action a_t in state s_t, $\gamma \in (0, 1]$ is the discount factor.

Traditional Q-learning algorithm describes the value function in tabular form, i.e. recording all q-values in a table named Q-table. However, it is impossible to handle with complex control problems with numerous states and actions. The high dimension of state and action spaces seriously affects the convergence speed of Q-learning. In order to overcome this drawback, function approximators can be used to approximately represent the real value function. In particular, the emerging Deep Reinforcement Learning (DRL) which combines RL and deep learning techniques has achieved great success in applications such as cooling datacenters [35] and playing video games [30]. DRL utilizes a deep neural network (DNN) to build the correlation between each state-action pair and its associated value function. In addition, based on the underlying DNN, a Deep Q-learning framework is used for online decisions, i.e. suggest optional actions for the agent. Fig.2 illustrates how DRL works in the agent-environment interaction system.

B. QOS-AWARE JOB SCHEDULING ALGORITHM

As cloud-based application workloads are unpredictable and time-varying, it is desirable to perform job scheduling in an online adaptive manner. We believe RL-based methods are especially well-suited for this scenario, because they do not need to know user workload, state transition and other information of the underlying system in advance. The optimal job dispatching policy can be learned in an online fashion as the system runs. Specifically, the RL agent keeps making job-to-VM assignment decisions over time thus generating enormous training data for itself to learn how to make better decisions in the future.

DRL can be regarded as an enhanced version of traditional RL which is able to solve complex decision-making problems with large state space. However, countless possible actions will give rise to unacceptable long training duration of DNN. To avoid this problem, we adopt an event-driven decision framework which means each scheduling decision made once a new job request arrives. Moreover, job execution order on each VM instance follows the FCFS manner. In this way, the action at each decision time is only the target VM instance for the current job, thereby reducing the number of available actions.

The QoS-aware job scheduling algorithm we proposed involves two collaborative phases: online decision phase and offline training phase. In the following, we will introduce them separately and present the entire algorithm at last (see Algorithm 1).

Algorithm 1 QoS-Aware Job Scheduling Algorithm

```
1:  Input: initial $\epsilon$, $\alpha$, $\gamma$,learning frequency $f$, start learning
    time $\tau$, minibatch $S_\Delta$, replay period $\eta$
2:  Initialize replay memory $\Delta$ with capacity $N_\Delta$
3:  Initialize evaluation action-value function $Q$ with random
    parameters $\theta$
4:  Initialize target action-value function $\hat{Q}$ with random
    parameters $\theta'$
5:  for each new job $j$ arrives at $t_j$ do
6:    with probability $\epsilon$ randomly choose an action;
      otherwise $a_j = argmax_a Q(s_j, a; \theta)$
7:    Schedule job $j$ according to action $a_j$, receive reward
      $r_j$,and observe state transition at next decision time $t_{j+1}$
      with a new state $s_{j+1}$
8:    Store transition $(s_j, a_j, r_j, s_{j+1})$ in $\Delta$
9:    if $j \geqslant \tau$ and $j \equiv 0$ mod $f$ then
10:     if $j \equiv 0$ mod $\eta$ then
11:       Reset $\hat{Q} = Q$
12:     end if
13:     randomly select samples $S_\Delta$ from $\Delta$
14:     for each transition $(s_k, a_k, r_k, s_{k+1})$ in $S_\Delta$ do
15:       $target_k = r_k + \gamma max_{a'} \hat{Q}(s_{k+1}, a'; \theta')$
16:       update DNN parameters $\theta$ with a loss function of
          $target_k - Q(s_k, a_k; \theta)^2$
17:     end for
18:     Gradually decrease $\epsilon$ until to the lower bound
19:   end if
20: end for
```

1) ONLINE DECISION PHASE

The online decision phase adopts deep Q-learning techniques. It is responsible for choosing proper VM instance for the new coming job. At each decision time, the DRL agent observes the current environment state and then uses DNN to estimate q-values for all available VM instances at current state. According to a specific policy, a VM instance will be chosen to execute this job. Besides, DRL agent will get an immediate reward relevant to this job-to-VM scheduling decision.

Now, we will describe the RL model of the online job scheduling problem.

a: STATE SPACE AND ACTION SPACE

As we mentioned before, we use the event-driven decision framework to keep the action space small. In our model, it is assumed that the cloud-based application provider rents a fixed number of reserved VM instances. Each VM instance holds a job waiting queue with no length limit and is able to execute any job request. Therefore, the action space can be represented as a collection of all VM instances $a = [VMid_1, VMid_2, \ldots, VMid_M]$, where M is the total number of rented reserved VM instances of the cloud-based application provider.

For a new arrived job j of the cloud-based application, we define its arrival time is t_j. At the moment, the state s^{t_j} can be described using two components, job j's state s_j and all VM instances' current state $s^{t_j}_{vm}$, i.e. $s^{t_j} = s_j \cup s^{t_j}_{vm}$. More specifically, job j's state s_j consists of three elements: job j's type, length, and QoS requirements. All VM instances' state $s^{t_j}_{vm}$ denotes the expected latency time of job j on each candidate VM instance. Hence, the entire state vector of job j can be expressed as $s^{t_j} = [jType_j, jSize_j, QoS_j, T^{wait}_{j1}, T^{wait}_{j2}, \ldots, T^{wait}_{jM}]$. The state space contains all possible state thus has a high dimension.

b: REWARD FUNCTION

The reward signal is used to help the agent make wise decisions for the objective of the job scheduling framework, which is efficiently dispatching job requests to limited VM instances under users' QoS requirements. In our model we consider average job response time as the primary QoS metric. Thus for each job j, its reward is given by $r_j = (T^{exe}_j)$âĂ"T_j, which means the immediate reward decreases with the increase of job slowdown (i.e. the ratio of actual job response time to ideal job response time).

c: ACTION SELECTION AND STATE TRANSITION

At a decision time, the DRL agent needs to choose a proper action based on current state and estimated q-values deriving from the DNN. Here we apply ϵ-greedy policy to achieve this goal, which means a random VM instance will be chosen to execute the incoming job with probability ϵ, otherwise the VM instance with the highest estimated q-value will be chosen with probability $(1 - \epsilon)$. ϵ is a variable which starts with a large value and gradually decreases to a minimum value over time. In this way, the agent always randomly explores possible actions in the beginning, but after accumulating a great deal of decision-making experience, the agent prefers to exploit existing knowledge to choose appropriate actions with higher rewards.

Once the job scheduler performs a specific action a_j for job j and the next job $j + 1$ of job j is coming, the state will transfer from s^{t_j} to $s^{t_{j+1}}$, where:$s^{t_{j+1}} = [jType_{j+1}, jSize_{j+1}, QoS_{j+1}, T^{wait}_{(j+1)1}, \ldots, T^{wait}_{(j+1)M}]$.

2) OFFLINE TRAINING PHASE

In the offline training phase, historical job scheduling decisions and results are used to train the underlying DNN to obtain more accurate value functions for state-action pairs. Experience replay and Fixed Q-targets are two key techniques to achieve this goal.

a: EXPERIENCE REPLAY [30]

For each decision time t_j, the DRL agent performs an action a_j based on current state s_j. After receiving the immediate reward r_j and observing the next state s_{j+1}, the transition profiles (s_j, a_j, r_j, s_{j+1}) will be stored into a replay memory Δ with capacity N_Δ. Each time when Q-learning update happens, parameters θ of the DNN will be updated using minibatch which contains a fixed number of random samples S_Δ from the replay memory Δ. In order to avoid time complexity, such an update could happen every U decision episodes ($U \geqslant 1$). As the experience replay strategy makes the agent learn from random transition samples rather than sequential experiences, the correlation among training data will be broken thereby reducing the variance of updated parameters. In addition, the utilization of training data is high because each sample is potentially chosen more than once to update parameters.

b: FIXED Q-TARGETS [31]

In order to further eliminate the divergence and oscillations of DNN parameters during training processes, two neural networks named target network and evaluation network are adopted together. Two networks have the same structure but different parameters. Target network is used to generate target q-values when updating Q-learning. One significant characteristic of target network is that it is a temporarily frozen network. Its parameters are periodically copied from evaluation network. By contrast, evaluation network holds the latest parameters and it is used to estimate q-values.

V. PERFORMANCE EVALUATION

In this section, we will evaluate our proposed QoS-aware job scheduling algorithm through a series of experiments. Firstly, we introduce the experiment setup which involves the basic simulated cloud environment and five other job scheduling approaches. Then we apply our algorithm and five baselines in different workload scenarios (i.e. stable workloads, changing workloads, and real-world workloads) to compare and evaluate their performance. All experiments are conducted in Python 3 environment with TensorFlow on a MacBook Pro with 2.7 GHz Intel Core i5 processor and 8GB RAM.

A. EXPERIMENT SETUP

1) SIMULATED ENVIRONMENT

In our experiments, we consider a cloud-based application provider who rents 10 long-term reserved VM instances from IaaS vendors to provide service for its end users. These VM instances consist of 5 high-CPU VM instances and

TABLE 1. Average processing capacities of VM instances.

	Computing intensive job	I/0 intensive job
High-CPU VM	AVG 1000 MIPS STD 100 MIPS	AVG 500 MIPS STD 50 MIPS
High-I/O VM	AVG 500 MIPS STD 50 MIPS	AVG 1000 MIPS STD 100 MIPS

5 high-memory VM instances. Application workload is in the form of continuous computing intensive job requests and I/O intensive job requests from numerous end users. By default, the lengths of jobs obey the normal distribution with mean 200 MI and standard deviation 20 MI. We summarize the average processing capacities of rented VM instances in Table.1.

2) DNN CONSTRUCTION AND RL PARAMETERS

In our implementation of the proposed QoS-aware job scheduling algorithm, we construct the underlying DNN by using a feed-forward neural network that has a single fully connected hidden layer with 20 neurons. Here we set the capacity of replay memory $N_{\Delta} = 800$ and the size of minibatch $S_{\Delta} = 30$. RMSProp algorithm is adopted to update the evaluation network parameters with the learning rate of 0.01. Parameters of target network are cloned from evaluation network every 50 decision episodes. The DNN starts to be trained after accumulating enough transition samples in replay memory. We set $\tau = 500, f = 1, \gamma = 0.9$, and ϵ is decreased from 0.9 by 0.002 in each learning iteration.

3) BASELINE SOLUTIONS

In order to evaluate the performance of our proposed QoS-aware job scheduling algorithm (denoted by 'DRL'), we compare it with five baseline approaches: random scheduling approach, round-robin scheduling approach, earliest scheduling approach, best-fit scheduling approach, and sensible scheduling approach.

Random scheduling approach (denoted by 'random') is a very simple method which selects a random VM instance for each job. Round-robin scheduling approach (denoted by 'round-robin') mainly focuses on how to fairly dispatch jobs to VM instances. Thus VM instances are chosen in circular order to execute incoming jobs. Earliest scheduling approach (denoted by 'earliest') is a time greedy strategy, where the new arrived job is scheduled to the earliest idle VM instances. Best-fit scheduling approach (denoted by 'best-fit') is also a greedy strategy. Compared with earliest scheduling approach, best-fit scheduling approach not only takes time factor into account, but also considers whether the type of the chosen VM instance matches the new arrived job' type. As we mentioned before, assigning a give job to a VM instance whose type is suitable for this job can reduce the execution time. Therefore, best-fit scheduling approach allocates the new job to the earliest idle one among all VM instances whose type is suitable for this job. Sensible scheduling approach (denoted by 'sensible') [36] is an adaptive algorithm which uses randomized routing policy based on the expected QoS, i.e. average job response time in this paper. With a higher probability it assigns jobs to the VM instance that provides a lower average response time over a period of time. Sensible scheduling approach needs two parameters: observation duration D and discount factor α. We set $D = 5s$ and $\alpha = 0.7$ for experiments in subsection B, $D = 0.2s$ and $\alpha = 0.7$ for experiments in subsection C and D.

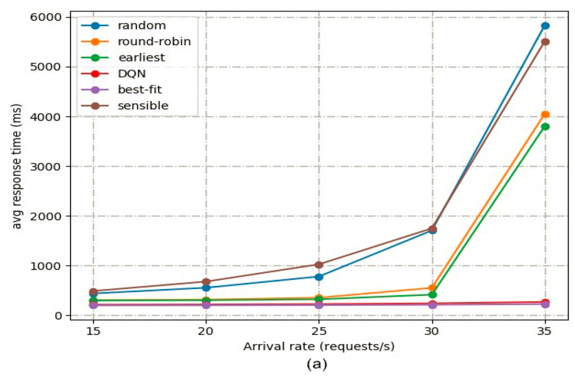
B. EXPERIMENTS UNDER STABLE WORKLOADS

We first consider an ideal and simple scenario: job scheduling for cloud-based applications with stable workloads. Stable workloads mean job requests arrive into the application at a relatively steady speed. Besides, the probability distribution of the types of jobs is fixed. In this part, two sets of experiments are conducted to compare the performance of our QoS-aware job scheduling algorithm and baselines.

1) COMPARISON OF WORKLOADS WITH DIFFERENT JOB ARRIVAL RATES

Job arrival rate is an important indicator which reflects the amount of workload of the cloud-based application. A higher arrival rate signifies more job requests arrive into the system within a shorter period of time. In order to know the influence of job arrival rates on the performance of different job scheduling approaches, we record the corresponding average job response time and VM utilization rate in Fig.3. In our experiments, job requests arrive according to a Poisson distribution. We set different job arrival rates ranging from 15 requests/s to 35 requests/s. For each job arrival rate, six job scheduling approaches are applied to schedule a workload consisting of 5000 stochastic jobs. The ratio of computing intensive jobs and I/O intensive jobs is 1:1. Each experiment is repeated 20 times and the average results are presented.

Fig.3(a) demonstrates the average job response time of different scheduling approaches when job arrival rate varies from 15 requests/s to 35 requests/s. We can generalize three conclusions: (1) the average job response time of each scheduling approach increases with the increase of job arrival rate. The reason for this is obvious. As the increase of the loads in the application, the limited capacity of rented VM instances cannot execute each job immediately, which may result in a longer waiting time for jobs. (2) the increases in average response time produced by different approaches are various. It can be seen that our proposed approach and best-fit scheduling approach keep a low response time all the time. On the contrary, the growth rates of other baselines are rather high. It means that our approach and best-fit scheduling approach are more adaptive in terms of response time than other approaches. (3) under the same arrival rate setting, the proposed QoS-aware job scheduling algorithm obtains a much shorter average response time than all baselines except the best-fit scheduling approach. Besides,



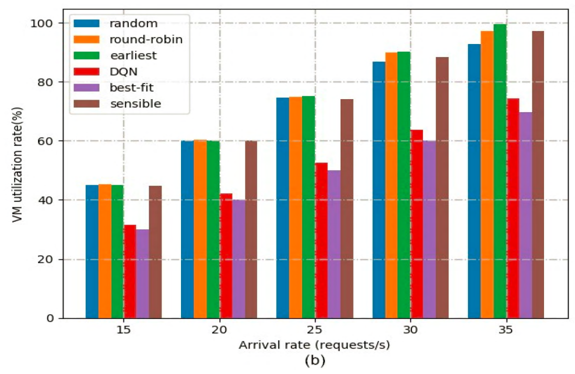


FIGURE 3. **Performance comparisons of six job scheduling approaches under different arrival rate settings. Simulation results in terms of average job response time and VM utilization rate are shown in Fig.3(a) and Fig.3(b) respectively.**

from Fig.3(b), it is obvious that the VM utilization rates of our approach and the best-fit scheduling approach are much lower than other baselines. These observations indicate that our approach and the best-fit scheduling approach can efficiently reduce job response time and make full use of resources to serve more jobs. In particular, when the job arrival time reaches 35 requests/s, if baselines except the best-fit scheduling approach are adopted, capacity provided by limited VM instances is hardly enough to serve such a workload. We can find that VM utilization rates of these baselines are all over 90% and corresponding average response time are rather long. In conclusion, our proposed QoS-aware job scheduling algorithm and best-fit scheduling approach surpass other baselines in such a scenario.

2) COMPARISON OF WORKLOADS WITH DIFFERENT JOB TYPE PROBABILITY DISTRIBUTIONS

The probability distribution of the types of jobs is another important characteristic of workloads. It describes how many computing intensive jobs and I/O intensive jobs are contained in the workload. In this set of experiments, we study the performance of our proposed algorithm and five baselines applied to schedule workloads with different job type probability distributions. As for the experimental settings, the job arrival rate of workload is fixed to 35 requests/s. The proportion of computing intensive jobs varies from 0.1 to 0.9 with an increment of 0.2. Other settings are similar to those mentioned in last experiment.

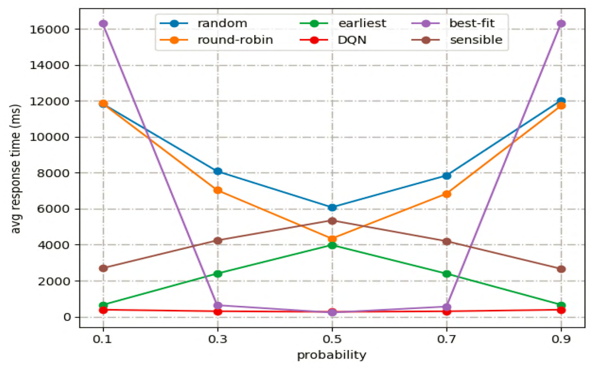


FIGURE 4. **Comparisons of average response time of six job scheduling approaches dealing with workloads with different job type probability distributions.**

Fig.4 illustrates the average job response time of different scheduling approaches when the probability of computing intensive job arrivals goes from 0.1 to 0.9. It is clear that our proposed QoS-aware job scheduling algorithm performs much better than all baselines. No matter how the probability changes, the average job response time of our approach is relatively stable and kept in a low level. By comparison, the performance of other approaches fluctuates largely. When the probability changes from 0.1 to 0.5, the average job response time of earliest scheduling approach and sensible scheduling approach gets longer, whereas the average job response time of the other three baselines becomes shorter. In order to investigate the reasons behind the performance observations, we also present the proportion of jobs assigned to each type of VM instances and the utilization rates of different types of VM instances in Fig. 5. Noting that in Fig.5 we only present the VM performance comparisons when the probability is no more than 0.5. This is because the performance of the probability less than 0.5 and more than 0.5 are symmetric.

From Fig.5(a), Fig.5(c), and Fig.5(e), it can be seen that random scheduling approach and round-robin scheduling approach fairly dispatch jobs to high-CPU VM instances and high-I/O VM instances all the time. With the increase of the probability of computing intensive job arrivals (i.e. increasing from 0.1 to 0.5), more jobs are potential to be assigned to the type-matched VM instances, thereby reducing average job execution time and latency time. Therefore, the average job response time of random scheduling approach and round-robin scheduling approach gradually decline. Round-robin scheduling approach performs better than random scheduling approach because the former approach tends to dispatch jobs to VM instances with lower loads.

Earliest scheduling approach provides rather shorter average job response time when the proportion of computing intensive jobs is 0.1. In this case, most jobs are of the same type thus earliest policy is the near-optimal scheme to allocate jobs to limited resources. However, the performance

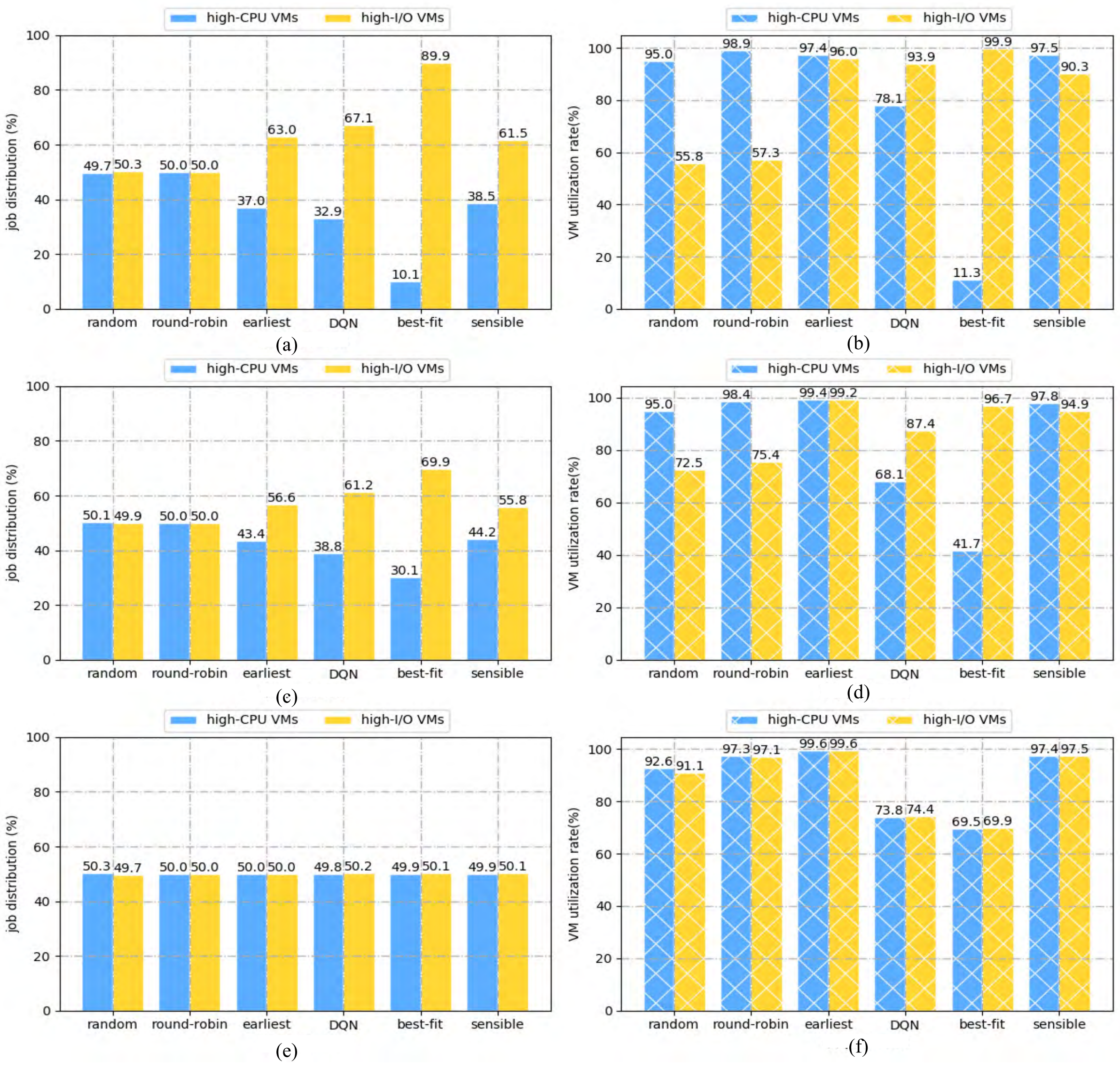


FIGURE 5. **VM performance comparisons of six job scheduling approaches dealing with workloads with different job type probability distributions. Fig.5(a),Fig.5(c), and Fig.5(e) describe the proportions of jobs assigned to high-CPU VM instances and high-I/O VM instances when workloads contain 10%, 30%, and 50% computing intensive jobs respectively. Fig.5(b), Fig.5(d), and Fig.5(f) show the utilization rates of different types of VM instances when workloads contain 10%, 30%, and 50% computing intensive jobs respectively.**

gets worse with the probability growing from 0.1 to 0.5. According to Fig.5(a), Fig.5(c) and Fig.5(e), the gap between jobs scheduled to high-I/O VM instances and high-CPU VM instances is getting smaller. This is because assigning each job to the earliest idle VM instance can cut down job latency time, but may incur a longer execution time if the types of the VM instance and the job are not matched. Hence dispatching jobs to type-matched VM instances could be a better policy with the increase of probability. Since sensible scheduling approach is also a time-greedy policy, its performance is similar to the performance of earliest scheduling approach (as shown in Fig.4 and Fig.5).

As we mentioned before, best-fit scheduling approach is able to assign jobs to the type-matched VM instances. From Fig.5(a), Fig.5(c), and Fig.5(e), it can be seen that the ratio of jobs assigned to high-CPU VM instances and high-I/O VM instances is closed to the expected ratio of computing intensive jobs and I/O intensive jobs contained in the workload. When the proportion of computing intensive jobs is similar to that of I/O intensive jobs (e.g. the probability equaling to 0.3, 0.5, and 0.7), best-fit scheduling approach is able to keep the average job response time in a low level. However, when the workloads consist of many computing intensive jobs or I/O intensive jobs (i.e. the probability is 0.1 or 0.9), the average job response time generated by best-fit scheduling approach is the longest among all approaches. As shown in Fig.5(b), best-fit scheduling approach does not make good use of the limited resources. The occupation of

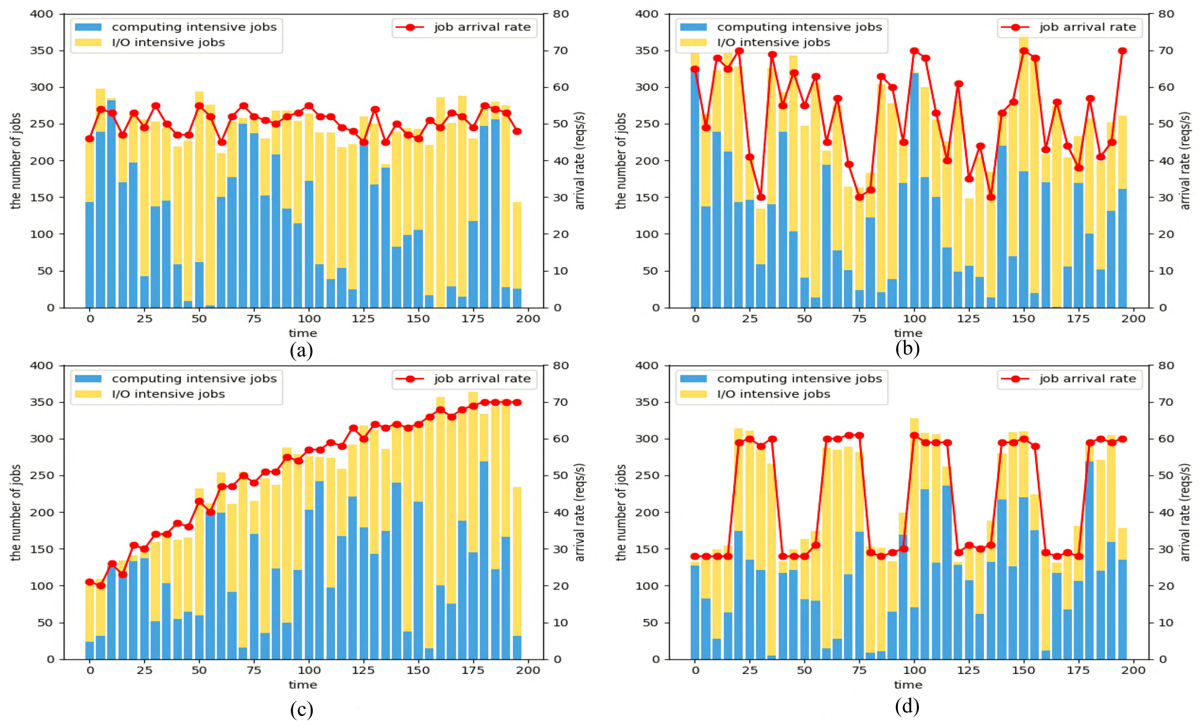


FIGURE 6. Examples of four types of workload patterns. For (a) stable pattern, job arrival rate varies between 45 requests/s and 55 requests/s. For (b) unpredicted pattern, each job arrival rate is selected from [30 requests/s, 70 requests/s]. For (c) growing pattern, job arrival rate increases from 20 requests/s to 70 requests/s. For (d) on-and-off pattern, the value ranges of the lower job arrival rate and the higher job arrival rate are [28 requests/s, 32 requests/s] and [58 requests/s, 62 requests/s] respectively.

high-CPU VM instances is only 11.3%, whereas the high-I/O VM instances are busy all the time.

Facing with the same workload and VM instances constraints, our proposed QoS-aware job scheduling algorithm is more intelligent than all baselines. It can adaptively dispatch jobs to different VM instances according to the state of the environment, thereby offering a lower average response time.

C. EXPERIMENTS UNDER CHANGING WORKLOADS

In this part, we consider a more realistic scenario: job scheduling for cloud-based applications with varying workloads. Job arrival rate and job type probability distribution of the workload always change over time. According to [37], we simulate four types of common workload patterns to describe job arrival rates, which are stable pattern, unpredicted pattern, growing pattern, and on-and-off pattern. Stable pattern means workloads have relative stable job arrival rates and no big fluctuations. Unpredicted pattern represents highly dynamic workloads that have sudden increases or decreases in the job arrival rate. Growing pattern represents workloads with rising trends. On-and-off pattern depicts workloads that have spikes at regular intervals. In our experiments, the lengths of jobs obey the normal distribution with mean 100 MI and standard deviation 20 MI. The QoS requirement (i.e. acceptable maximum response time) of each job is selected uniformly between 250 milliseconds and 350 milliseconds. The probability that the new arrived job belongs to computing intensive type is chosen uniformly between 0 and 1. Job arrival rate and job type probability distribution change every 5 seconds. For each workload pattern, we generate 20 stochastic workloads and each workload lasts 200 seconds.

Fig.6 displays four workload examples used in our experiments. Each diagram represents one type of workload patterns. Line charts describe the job arrival rates and bar charts demonstrate the actual number of computing intensive jobs and I/O intensive jobs. Based on these synthetic workloads, the relevant performance results of different scheduling approaches are shown in Fig.7. In these experiments, we focus on two metrics: job success rate and average job response time. Since our proposed QoS-aware job scheduling algorithm needs to be trained at the beginning, it can be seen that in the first 30 seconds our approach dose not perform well but its performance is getting better. After a short period of learning, no matter facing with what kind of workload, our approach is able to efficiently schedule jobs to appropriate VM instances thereby improving job success rate and reducing average job response time. Table.2 summaries the average results of job success rate and average job response time for different workload patterns. It is obvious that our proposed QoS-aware job scheduling algorithm is suitable for all workload patterns and surpasses all baselines. It keeps the job success rates are above 90% and VM utilizations are lower than most baselines.

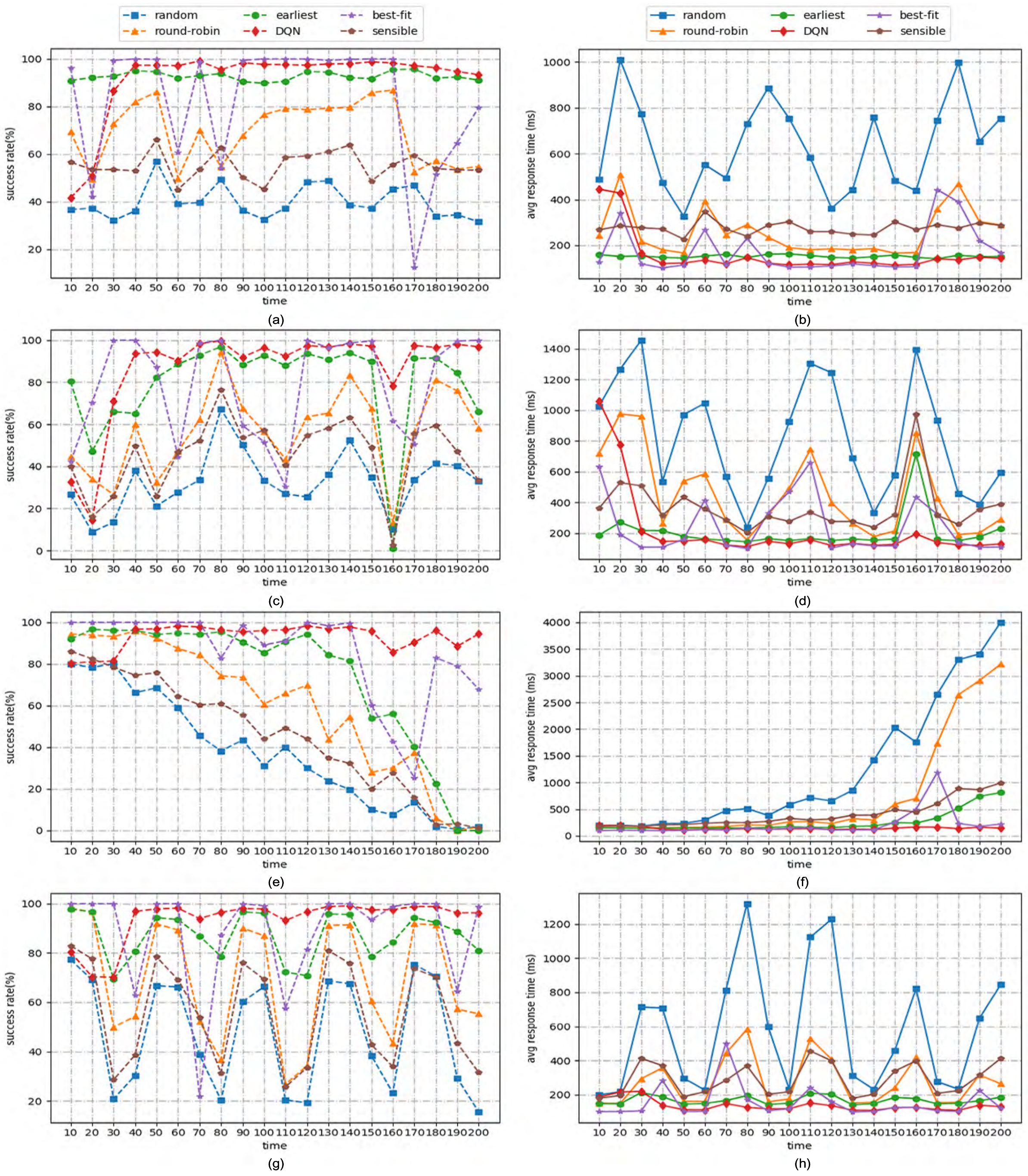


FIGURE 7. Performance comparisons of applying different scheduling approaches to handle workloads showed in Fig.6. Job success rates and average response time for the stable pattern workload (i.e. Fig.6(a)), the unpredicted pattern workload (i.e. Fig.6(b)), the growing pattern workload (i.e. Fig.6(c)), and the on-and-off pattern workload (i.e. Fig.6(d)) are displayed in Fig.7(a)-(b), Fig.7(c)-(d), Fig.7(e)-(f), and Fig.7(g)-(h) respectively.

D. EXPERIMENTS UNDER REAL-WORLD WORKLOADS

Similar to experiments conducted in [38] and [25], we use a well-known real-world workload trace from NASA [39] to further evaluate our approach. NASA workload trace contains all HTTP requests to the NASA Kennedy Space Center WWW server in Florida, from August 1 to 31, 1995. Because of Hurricane Erin, there were no accesses recorded from 01/Aug/1995:14:52:01 until 03/Aug/1995:04:36:13. Hence, we get rid of the first three days' records and use the other 28 days' data (including more than 1 million requests) in our experiments.

As shown in Fig.8, the entire workload trace follows the time-of-day and day-of-week patterns. Thus, it could represent numerous common cloud-based applications which are busy during working hours in day time but receive fewer requests at night and at weekends. Fig.8(b) and Fig.8(c)

TABLE 2. Average results for different workload patterns *.

Metrics		Policy					
		DQN	Random	Round-robin	Earliest	Best-fit	Sensible
Stable pattern	Success rate	97.0%	40.7%	72.0%	92.4%	90.9%	55.2%
	Response time	126 ms	518 ms	222 ms	154 ms	143 ms	275 ms
	VM utilization	54.4%	74.6%	75.0%	72.9%	50.0%	73.2%
Unpredicted pattern	Success rate	94.5%	35.3%	60.1%	80.6%	80.3%	47.3%
	Response time	138 ms	721 ms	378 ms	187 ms	229 ms	334 ms
	VM utilization	55.1%	74.9%	75.2%	73.3%	50.3%	73.7%
Growing pattern	Success rate	93.4%	28.1%	46.0%	71.1%	71.9%	38.9%
	Response time	141 ms	1432 ms	882 ms	236 ms	328 ms	433 ms
	VM utilization	57.0%	74.1%	75.3%	74.8%	51.4%	74.9%
On-and-off pattern	Success rate	95.5%	40.1%	61.8%	86.4%	79.3%	51.3%
	Response time	132 ms	675 ms	368 ms	170 ms	214 ms	309 ms
	VM utilization	48.7%	66.0%	66.5%	64.9%	44.5%	65.1%

* Results excepting the first 30s training period.

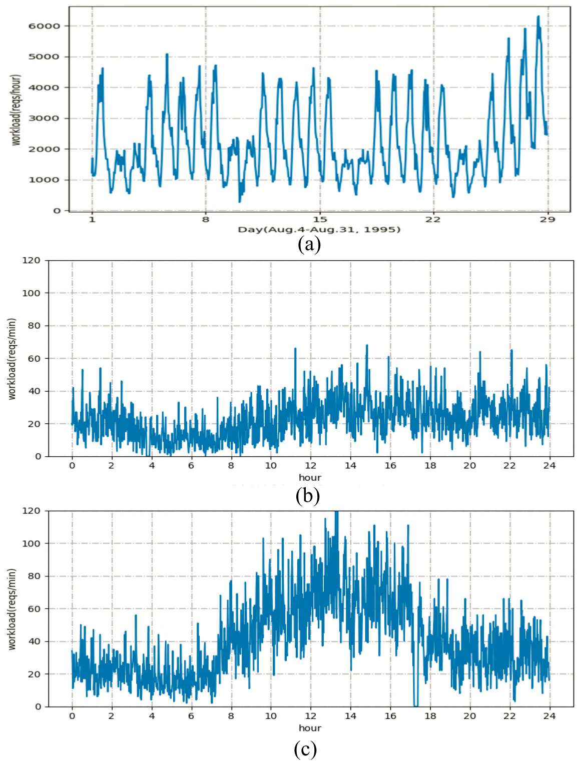


FIGURE 8. Workload traces of NASA. Fig.8(a) describes the entire workload between August 4 and August 31, 1995. Fig.8(b) and Fig.8(c) display the number of requests per minute on August 5, 1995 and August 7, 1995 respectively.

depict workloads (denoted by the number of requests per minute) of a Saturday and a Monday respectively. It is obvious that the former one is relatively stable, whereas the latter one changes largely. In our experiments, we set the maximum processing capacity of VM instances is 1200 MI. Other parameters are the same as those used in section V.C.

We apply our proposed QoS-aware job scheduling algorithm and other five baselines to schedule jobs contained in the four-week workload. Performance comparisons are shown in Table.3. Our approach provides the highest job success rate, which means it is able to satisfy the QoS requirements of most users. Moreover, the average job response time of random scheduling approach is around 7 times that of our approach. Even compared with the best baseline, i.e. the best-fit scheduling approach, our approach could reduce the average job response time by 40.4%

TABLE 3. Average results for scheduling NASA workload.

	Success rate	Response time
DQN	98.3%	158 ms
Random	81.7%	1340 ms
Round-robin	85.6%	1110 ms
Earliest	91.8%	545 ms
Best-fit	94.5%	265 ms
Sensible	89.8%	792 ms

VI. CONCLUSION AND FUTURE WORK

The problem of efficient job scheduling is a challenging task for cloud-based application providers. In this paper, we investigated this problem and presented an online job scheduling framework. In order to implement the job scheduler of the framework, we proposed a deep reinforcement learning based algorithm to help application providers dispatch jobs to limited resources under QoS requirement constraints. Because of the characteristics of reinforcement learning, our proposed QoS-aware job scheduling algorithm could dynamically adapt to the uncertainties and fluctuations of workloads. We evaluated the performance of our approach in three workload scenarios: stable workloads, changing workloads, and real-world workloads. Compared with several baseline solutions, our proposed approach decreases the average job response time, and ensures the job success rate at a high level. In the future, we plan to extend our proposed DRL-scheduling framework to achieve efficient job scheduling in a more complex cloud environment. Several factors such as elastic resource provisioning, VM failures, and relationships between jobs will be considered. Moreover, we will integrate our proposed approach with some strategies, such as admission control, to further improve its performance.

REFERENCES

[1] M. Mao, J. Li, and M. Humphrey, "Cloud auto-scaling with deadline and budget constraints," in *Proc. IEEE/ACM Int. Conf. Grid Comput.*, Oct. 2011, pp. 41–48.

[2] A. Khan, X. Yan, S. Tao, and N. Anerousis, "Workload characterization and prediction in the cloud: A multiple time series approach," in *Proc. Netw. Oper. Manage. Symp.*, 2012, pp. 1287–1294.

[3] S. K. Garg, A. N. Toosi, S. K. Gopalaiyengar, and R. Buyya, "SLA-based virtual machine management for heterogeneous workloads in a cloud datacenter," *J. Netw. Comput. Appl.*, vol. 45, pp. 108–120, Oct. 2014.

[4] S. K. Garg, S. K. Gopalaiyengar, and R. Buyya, "SLA-based resource provisioning for heterogeneous workloads in a virtualized cloud datacenter," in *Algorithms and Architectures for Parallel Processing*, vol. 7016. Berlin, Germany: Springer, 2011, pp. 371–384.

[5] A. Tchernykh, U. Schwiegelsohn, V. Alexandrov, and E. G. Talbi, "Towards understanding uncertainty in cloud computing resource provisioning," *Procedia Comput. Sci.*, vol. 51, pp. 1772–1781, 2015.

[6] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *IEEE Trans. Neural Netw.*, vol. 9, no. 5, p. 1054, Sep. 1998.

[7] S. Singh and I. Chana, "A survey on resource scheduling in cloud computing: Issues and challenges," *J. Grid Comput.*, vol. 14, no. 2, pp. 217–264, 2016.

[8] T. Mathew, K. C. Sekaran, and J. Jose, "Study and analysis of various task scheduling algorithms in the cloud computing environment," in *Proc. Int. Conf. Adv. Comput., Commun. Informat.*, 2014, pp. 658–664.

[9] H. Khazaei, J. Mišić, and V. B. Mišić, "Performance analysis of cloud computing centers using M/G/m/m+r queuing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 5, pp. 936–943, May 2012.

[10] X. Liu, W. Tong, X. Zhi, Z. Fu, and W. Liao, "Performance analysis of cloud computing services considering resources sharing among virtual machines," *J. Supercomput.*, vol. 69, no. 1, pp. 357–374, 2014.

[11] H. Yuan, B. Jing, T. Wei, M. C. Zhou, H. L. Bo, 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.

[12] 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.

[13] A. Abraham, H. Liu, W. Zhang, and T. G. Chang, "Scheduling jobs on computational grids using fuzzy particle swarm algorithm," *Future Gener. Comput. Syst.*, vol. 26, no. 8, pp. 1336–1343, 2010.

[14] F. Palmieri, L. Buonanno, S. Venticinque, R. Aversa, and B. D. Martino, "A distributed scheduling framework based on selfish autonomous agents for federated cloud environments," *Future Gener. Comput. Syst.*, vol. 29, no. 6, pp. 1461–1472, 2013.

[15] 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.

[16] M. Kalra and S. Singh, "A review of metaheuristic scheduling techniques in cloud computing," *Egyptian Informat. J.*, vol. 16, no. 3, pp. 275–295, 2015.

[17] S. H. Jang, T. Y. Kim, J. K. Kim, and J. S. Lee, "The study of genetic algorithm-based task scheduling for cloud computing," *Int. J. Control Autom.*, vol. 5, no. 4, pp. 157–162, 2012.

[18] M. A. Tawfeek, A. El-Sisi, A. E. Keshk, and F. A. Torkey, "Cloud task scheduling based on ant colony optimization," in *Proc. Int. Conf. Comput. Eng. Syst.*, pp. 64–69, 2014.

[19] B. Gomathi and K. Krishnasamy, "Task scheduling algorithm based on hybrid particle swarm optimization in cloud computing environment," *J. Theor. Appl. Inf. Technol.*, vol. 7, no. 1, p. 575, 2013.

[20] D. L. Poole and A. K. Mackworth, *Artificial Intelligence: Foundations of Computational Agents*. Cambridge, U.K.: Cambridge Univ. Press, 2010.

[21] M. Abundo, V. D. Valerio, V. Cardellini, and F. L. Presti, "QoS-aware bidding strategies for VM spot instances: A reinforcement learning approach applied to periodic long running jobs," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage.*, May 2015, pp. 53–61.

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

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

[24] F. Farahnakian, P. Liljeberg, and J. Plosila, "Energy-efficient virtual machines consolidation in cloud data centers using reinforcement learning," in *Proc. Euromicro Int. Conf. Parallel, Distrib. Netw.-Based Process.*, 2014, pp. 500–507.

[25] J. Liu, Y. Zhang, Y. Zhou, D. Zhang, and H. Liu, "Aggressive resource provisioning for ensuring QoS in virtualized environments," *IEEE Trans. Cloud Comput.*, vol. 3, no. 2, pp. 119–131, Apr./Jun. 2015.

[26] X. Bu, J. Rao, and C. Z. Xu, "Coordinated self-configuration of virtual machines and appliances using a model-free learning approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 4, pp. 681–690, Apr. 2013.

[27] H. Arabnejad, C. Pahl, P. Jamshidi, and G. Estrada, "A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling," in *Proc. IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, May 2017, pp. 64–73.

[28] C. Mera-Gómez, F. Ramírez, R. Bahsoon, and R. Buyya, "A debt-aware learning approach for resource adaptations in cloud elasticity management," in *Service-Oriented Computing*. Cham, Switzerland: Springer, 2017, pp. 367–382.

[29] Z. Peng, D. Cui, J. Zuo, Q. Li, B. Xu, and W. Lin, "Random task scheduling scheme based on reinforcement learning in cloud computing," *Cluster Comput.*, vol. 18, no. 4, pp. 1595–1607, 2015.

[30] V. Mnih *et al.* (2013). "Playing atari with deep reinforcement learning." [Online]. Available: https://arxiv.org/abs/1312.5602

[31] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[32] L. Wu, S. K. Garg, and R. Buyya, "SLA-based admission control for a software-as-a-service provider in cloud computing environments," *J. Comput. Syst. Sci.*, vol. 78, no. 5, pp. 1280–1299, 2012.

[33] M. Al-Roomi, S. Al-Ebrahim, S. Buqrais, and I. Ahmad, "Cloud computing pricing models: A survey," *Int. J. Grid Distrib. Comput.*, vol. 6, no. 5, pp. 93–106, 2013.

[34] C. J. C. H. Watkins and P. Dayan, "*Q*-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[35] R. Evans and J. Gao, "Deepmind ai reduces google data centre cooling bill by 40%," DeepMind, London, U.K., Tech. Rep. 7, 2016. [Online]. Available: https://deepmind.com/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-40/

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

[37] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proc. High Perform. Comput., Netw., Storage Anal.*, 2011, pp. 1–12.

[38] M. Ghobaei-Arani, S. Jabbehdari, and M. A. Pourmina, "An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach," *Future Gener. Comput. Syst.*, vol. 78, pp. 191–210, Jan. 2017.

[39] *NASA-HTTP Workload Traces*. Accessed: Jun. 20, 2018. [Online]. Available: http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html

YI WEI received the B.S. degree in software engineering from Shandong University, Jinan, China, in 2013, where she is currently pursuing the Ph.D. degree in computer science with the School of Computer Science and Technology. Her research interest includes cloud computing and reinforcement learning.

LI PAN received the Ph.D. degree from the Research Center of Human-Computer Interaction and Virtual Reality, Shandong University, in 2011. She is currently a Lecturer with the School of Software, Shandong University, and a member of the HCI&VR Group. Her main research interests are service computing and cloud computing.

SHIJUN LIU received the B.S. degree in oceanography from the Ocean University of China and the M.S. and Ph.D. degrees in computer science from Shandong University, China. He is currently a Professor with Shandong University. His current research interests include services computing, enterprise services computing, and services system for manufacturing.

LEI WU was born in Feicheng, Shandong, China, in 1980. She received the B.S degree from the Qingdao University of Science and Technology, China, in 2002, and the Ph.D. degree from Shandong University, China, in 2008. From 2008 to 2013, she was a Lecturer with Shandong University. Since 2013, she has been an Assistant Professor with the School of Software, Shandong University. Her interests include manufacturing cloud, service computing, and enterprise computing.

XIANGXU MENG received the Ph.D. degree from the Institute of Computing Technology, Chinese Academy of Science, in 1998. He is currently a Professor and a Doctoral Supervisor with the School of Software, Shandong University. His main research interests include human–computer interaction, computer graphics theory and methods, virtual reality and virtual prototyping, grid computing, service computing, and manufacturing of information technology.