Hindawi
Complexity
Volume 2020, Article ID 3046769, 12 pages
https://doi.org/10.1155/2020/3046769

Research Article

A Deep Reinforcement Learning Approach to the Optimization of Data Center Task Scheduling

Haiying Che,[1] Zixing Bai,[1] Rong Zuo,[1] and Honglei Li[2]

[1]*Beijing Institute of Technology, Zhongguancun South Street No. 5, Beijing 100081, China*
[2]*Liaoning Normal University, Huanghelu 850, Dalian 116029, China*

Correspondence should be addressed to Honglei Li; lhl@lnnu.edu.cn

Received 25 May 2020; Revised 19 July 2020; Accepted 13 August 2020; Published 31 August 2020

Academic Editor: Shuping He

With more businesses are running online, the scale of data centers is increasing dramatically. The task-scheduling operation with traditional heuristic algorithms is facing the challenges of uncertainty and complexity of the data center environment. It is urgent to use new technology to optimize the task scheduling to ensure the efficient task execution. This study aimed at building a new scheduling model with deep reinforcement learning algorithm, which integrated the task scheduling with resource-utilization optimization. The proposed scheduling model was trained, tested, and compared with classical scheduling algorithms on real data center datasets in experiments to show the effectiveness and efficiency. The experiment report showed that the proposed algorithm worked better than the compared classical algorithms in the key performance metrics: average delay time of tasks, task distribution in different delay time levels, and task congestion degree.

1. Introduction

For the data centers with a huge number of servers, even a little bit improvement of the operation can save millions of dollars. Good task scheduling has been proved to be a practical way to bring benefits without extra hardware investment. Today's data center scheduling systems mostly use heuristic algorithms such as Fair Scheduling (Fair), First-Come-First-Service (FCFS) scheduling, and Shortest-Job-First (SJF) scheduling. These algorithms are easy to understand and implement but only take effects in some certain situations due to the limitation of the complicated production environment. The challenges of data center scheduling in the real world are as follows:

(1) The data center environment is complex and dynamically changing. To achieve efficient and effective scheduling, traditional heuristic algorithms mostly rely on precise environment modeling. In other words, if the environment cannot be accurately modeled, the reasonable and effective scheduling algorithm will not be successfully applied. Therefore, most data center scheduling algorithms still use basic and simple heuristic algorithms, such as Fair, FCFS, and SJF. Practically, it is too hard to model the environment precisely due to the uncertainty of the coming tasks and the dynamic environment. For example, the execution time of a task is affected by network bandwidth, the processing performance of different machines, the disk speed, and the location of the required resources to support the task execution.

(2) Data center scheduling is usually performed without sufficient information support. There are no patterns to predict the task arriving way. That is, the number and size of tasks coming next are unknown. So, the algorithm has to schedule the tasks at once without any prior experience and prepared information.

(3) Resource requirements change dynamically. For data center services or tasks, the demand for resources varies according to different time period, environmental conditions, and so on. The scheduling algorithm needs automatically optimize resource utilization based on the changing demand.

In order to solve the above problems, a lot of related studies are carried out. Most of them focus on specific scheduling scenarios or rely on the acquired details of the coming tasks in advance. In addition, most of the previous studies are single objective optimization-oriented.

In recent years, the deep reinforcement learning [1, 2] which achieves outstanding performance in complex control fields strongly shows its superiority of decision making in complex and unknown environments. Mao et al. tried to translate the problem of packing tasks with multiple resource demands into a learning problem. Their work shows that deep reinforcement learning performs comparably to state-of-the-art heuristics, adapts to different conditions, converges quickly, and learns strategies that are sensible in hindsight [3]. Inspired by these research results, we believe that deep reinforcement learning is suitable for task scheduling of data centers in complex production environments. This paper proposed a method based on deep reinforcement learning to improve task scheduling performance and resource utilization of data centers. With the neural network model trained through deep reinforcement learning, tasks scheduling and resource utilization improvement were achieved.

In this study, we used deep reinforcement learning and aimed at two objectives: minimizing the average task completion time and improving resource utilization efficiency without any prior knowledge of the coming tasks. The experiments and verification were performed using real production data from the Alibaba Cluster Trace Program sponsored by Alibaba Group [4]. The results showed that compared with the traditional heuristic algorithms, the proposed method achieved better performance.

This paper is organized as follows. In Section 2, related studies were discussed. Reinforcement learning-based scheduling as the prevailing technology used in task scheduling of the data center was introduced. In Section 3, the technical architecture and related definitions of the scheduling system proposed in this paper were illustrated. In Section 4, the reinforcement learning algorithm for scheduling optimization was introduced in detail. In Section 5, experiments were performed on Alibaba real production dataset to show the advantage of the proposed scheduling model. In the last section, the conclusion and future work were discussed.

2. Related Studies

For data center task scheduling, many studies have been launched in the past decade. Heuristic algorithms and reinforcement learning are popular in this domain.

2.1. Heuristic Algorithm-Based Studies. The traditional methods are mainly based on heuristic algorithms. Typically, Delimitrou and Kozyrakis proposed the ARQ algorithm, a multiclass admission control protocol that constrains application waiting time and limits application latency to achieve QoS [5]. They evaluated the algorithm with a wide range of workload scenarios, on both small- and large-scale systems and found that it enforces performance guarantees for 91 percent of applications, while the utilization is improved. Perry et al. proposed a Fastpass algorithm to improve data center transmission efficiency [6]. Fastpass incorporates two fast algorithms: the first determines the time at which each packet should be transmitted, while the second determines the path used for that packet. They deployed and evaluated Fastpass in a portion of Facebook's data center network, which shows that Fastpass achieves better efficiency in transmission. Tseng et al. argued that previous studies have typically focused on modifying the original TCP or increasing additional switch hardware costs, and rarely focused on the existing data center network (DCN) environments. So, they proposed a cross-layer flow schedule with a dynamic grouping (CLFS-DG) algorithm to reduce the effect of TCP incast in DCNs [7]. Yuan et al. solved the cost optimization problem under CDCs (cloud data centers) from two aspects. Firstly, a revenue-based workload admission control method is proposed to selectively accept requests. Then, a cost-aware workload scheduling method is proposed to allocate requests among multiple Internet service providers connected to the distributed CDCs. Finally, intelligent scheduling requests are realized, which can achieve lower cost and higher throughput for CDC providers [8]. Yuan et al. proposed the profit maximization algorithm (PMA) for the profit maximization challenge in the hybrid cloud scenario. The algorithm uses the hybrid heuristic optimization algorithm to simulate annealing particle swarm optimization (SAPSO) to improve the throughput and profit of private cloud [9]. Bi et al. proposed a new dynamic hybrid meta heuristic algorithm based on simulated annealing and particle swarm optimization (PSO) for minimizing energy cost and maximizing revenue of various applications running in virtualized cloud data centers [10]. Yuan et al. proposed a heuristic time scheduling algorithm (TTSA) to minimize the cost of private cloud data centers in hybrid cloud, which can effectively improve the throughput of private cloud data centers [11]. Yuan et al. proposed a biological target differential evolution algorithm (SBDE) based on simulated annealing, aiming at the challenges of maximizing profit and minimizing the probability of average task loss in distributed green data centers scenario. Compared with several existing scheduling algorithms, SBDE achieves greater benefits [12].

With the development of data centers, reasonable prediction is very important to improve the efficiency of task scheduling. Although prediction is not involved in the experiment, it is necessary to predict the execution time when the model is transferred to the actual scene. Zhang et al. (2018) proposed an integrated forecasting method equipped with noise filtering and data frequency representation, named Savitzky–Golay and wavelet-supported stochastic configuration networks (SGW-SCNs) [13]. Bi et al. also proposed an integrated forecasting method that combines Savitzky–Golay filtering and wavelet decomposition with stochastic configuration networks to get the

workload forecast in the next period [14]. Bi et al. proposed an integrated prediction method that combines the Savitzky–Golay filter and wavelet decomposition with stochastic configuration networks to predict workload at the next time slot [15].

Although the previous research results are quite abundant, Luo et al. argued that not only because the scheduling problem is theoretically NP-hard, but also because it is tough to perform practical flow scheduling in large-scale DCNs. It is quite challenging to minimize the task completion time in today's DCNs [16]. That means because of the scalability, complexity, and variability of data center production environment, heuristic algorithms cannot achieve the expected performance even with the deliberated design and exhausting tuning work in real production environment.

2.2. Reinforcement Learning-Based Studies. Reinforcement learning [1, 17], as the prevailing machine learning technology, dramatically becomes a new way to the task scheduling of data centers in recent years. Unlike supervised learning which requires amount of manpower and time to prepare the labeled data, reinforcement learning can work with unlabeled data. This so-called model-free mode allows users to start the modeling without the preparation of accurate server environmental data from the scratch. Therefore, at present, more researchers turn to try reinforcement learning to solve task scheduling in complex data center environments. For example, for the purpose of energy saving, Yuan et al. used the Q-learning algorithm to reduce the data center energy consumption. They tested the algorithm in the CloudSim, a cloud computing simulation framework issued by cloud computing, and distributed system laboratory of the University of Melbourne. The result shows that it can reduce about 40% of the energy consumption of the non-power-aware data center and reduce 1.7% energy consumption of the greedy scheduling algorithm in data center scheduling area [18]. Lin et al. used TD-error reinforcement learning to reduce the energy consumption of data centers, which does not rely on any given stationary assumptions of the job arrival and job service processes. The effectiveness of the proposed reinforcement learning-based data center power management framework was verified with real Google cluster data traces [19]. Li et al. also proposed an end-to-end cooling control algorithm (CCA) that is based on the deep deterministic policy gradient algorithm (DDPG) to optimize the control of cooling system in the data center. The result shows that CCA can achieve about 11% cooling cost saving on the simulation platform compared with a manually configured baseline control algorithm [20]. Shaw et al. proposed an advanced reinforcement learning consolidation agent (ARLCA) based on the Sarsa algorithm to reduce cloud energy consumption [21]. Their work proved that the ARLCA makes a significant improvement in energy saving while the number of service violations is reduced.

Scholars also apply reinforcement learning to solve the problems on other purposes of the data center task scheduling. Basu et al. applied reinforcement learning to build cost-models on standard online transaction processing datasets [22]. They modeled the execution of queries and updates as a Markov decision process whose states are database configurations, actions are configuration changes, and rewards are functions of the cost of configuration change and query and update evaluation. The approach was empirically and comparatively evaluated on a standard OLTP dataset. The result shows that the approach is competitive with state-of-the-art adaptive index tuning, which is dependent on a cost model. Peng et al. tried Gaussian process regression with reinforcement learning based on the Q-learning algorithm to solve the problem of state-action space incomplete exploration of reinforcement in cloud data centers [23]. The computational results demonstrated that the schema can balance the exploration and exploitation in the learning process and accelerate the convergence to a certain extent. Ruffy et al. presented a new emulator, Iroko, to support different network topologies, congestion control algorithms, and deployment scenarios. Iroko interfaces with the OpenAI Gym toolkit, which allows for fast and fair evaluation of different reinforcement learning and traditional congestion control algorithms under the same conditions [24]. By the way, some scholars try to combine neural network with heuristic algorithm and have achieved valuable research results. For example, He et al. proposed a new strategy iteration method for online H ∞ optimal control law design based on neural network for nonlinear systems. Numerical simulation is carried out to verify the feasibility and applicability of the algorithm [25]. In the next year, a PI algorithm based on neural network was proposed to solve the problem of online adaptive optimal control of nonlinear systems. Two examples were given to illustrate the effectiveness and applicability of the method [26].

The above studies show that reinforcement learning can help us to deal with the scheduling problems of data centers in many domains. However, most of them aim at single objective or are based on the ideal hypothesis that the information about the coming tasks and environment are accurate and adequate in advance, which limits the application of the related studies in real production environment.

Furthermore, for most of today's data centers, resources in fixed scale are usually allocated in advance for most business, which is apparently a low-efficiency mode. In fact, the amount of resources required by business may change with time. If we cannot customize the resource amount according to the changing requirement, when more resources are needed, it will cause serious task delay and affect the user experience badly. Or if fewer resources are needed, more unused resources will be wasted. In this article, we proposed reinforcement learning to optimize scheduling efficiency and to improve resource utilization simultaneously. This is a two-objective optimization work which addresses the primary demand of data center operations.

Table 1: Notations in the scheduling system model.

Notation	Memo	Type
T	The duration of period of task scheduling	Model parameter
$\widehat{T}$	The duration of period of resource optimization	Model parameter
$p(i)$	The priority function to estimate the priority of task i	Function
t	The start time of task scheduling which also represents the ID of period of task scheduling (the period is also called time slot)	Variable
$\widehat{t}$	The start time of resource optimization which also represents the ID of period of resource optimization	Variable
(s_1, a_1, r_1)	The state, action, and reward vector for task scheduling agent	Variable
(s_2, a_2, r_2)	The state, action, and reward vector for resource optimization agent	Variable
μ, η	Calibration parameters to adjust the influence of average task priority and active virtual machine proportion	Model parameter
$\widehat{\mu}$, $\widehat{\eta}$	Calibration parameters to tune the proportion of the active virtual machine and the proportion of idle virtual machines	Model parameter
$e\_t_{\widehat{t}}$,$e\_t'_{\widehat{t}}$	The sum of the execution time of tasks arriving in period $\widehat{t}$ and the sum of the execution time of tasks not executed in period $\widehat{t}$	Variable
$n\_t_{\widehat{t}}$,$n\_t'_{\widehat{t}}$	The number of tasks arriving in period $\widehat{t}$ and the number of tasks not executed in period $\widehat{t}$	Variable
M	The number of virtual machines in cloud server	Model parameter
K	The ratio of $\widehat{T}$ to T	Model parameter
α,β,H,γ	The hyperparameter of A2C algorithm	Hyperparameter

3. The Reinforcement Learning-Based Scheduling Model

In this session, all key parts of the scheduling model as well as the related definitions and equations were illustrated. The important notations are listed in Table 1 for better understanding.

3.1. The Model of Scheduling System. The reinforcement learning-based scheduling system consisted of two parts: environment and scheduling agents. As shown in Figure 1, the environment contained task queue, virtual machine cluster, and scheduler. The task queue was the pool to collect the unimplemented tasks in the data center. The virtual machine cluster was the container of virtual machine handlers. The scheduler was the dispatcher to execute the actions from scheduling agents. In this scheduling system, it was assumed that the number of virtual machines was fixed. The configuration and performance of all virtual machines were the same. Tasks in task queue can be scheduled to any idle virtual machine. The virtual machine cluster was defined as $\text{VMs} = \text{vm}_1, \text{vm}_2, \text{vm}_3, \ldots, \text{vm}_m$ where vm_i was the handler of virtual machine i.

The agent part included two scheduling agents: Agent1 and Agent2. Each agent was responsible for its optimization objective. Agent1 was for task scheduling, and Agent2 was for resource utilization.

In the scheduling model, time is an important factor in task scheduling and optimization of resource utilization. The time in the scheduling model was divided into two types, time t and time $\widehat{t}$ shown in Figure 2. t was the start time of task scheduling, and $\widehat{t}$ was the start time of optimization of resource utilization. Time t and $\widehat{t}$ were defined in the relative time period for ease of calculation. The initial value of t and $\widehat{t}$ was 0. In the scheduling model, each task scheduling last Tseconds and each optimization of resource utilization last $\widehat{T}$ seconds. $\widehat{T} = K * T\,(K > 0)$, where K was a parameter in the scheduling model configuration, which hinted the optimization of resource utilization spends more time than task scheduling. In this paper, we defined the interval between t and $t + 1$ as time slot t. So, T was the duration of time slot t. The same definitions were for time slot $\widehat{t}$.

At time t, Agent1 made the decision on whether to execute each task in the task queue. The scheduling action decision was stored in a_{1t}. After $\widehat{T}$ seconds, that is, at time $\widehat{t}$, the system started to optimize resource utilization. The virtual machine cluster would turn on or off a certain number of virtual machines according to the optimization decision action $a_{2\widehat{t}}$ output by Agent2. The procedure was illustrated in detail as follows.

Whenever time t came, the system would receive the tasks arriving from time $t$and store the tasks in the task queue. Then, the following actions were performed in sequence:

(i) The system selected tasks according to the priority calculated by $p(i)$, the task priority function defined in equation (1), and input the current state of the environment s_{1t} to Agent1

(ii) Agent1 output action a_{1t} and returned it to the environment

(iii) The environment executed a_{1t} and returned the reward r_{1t} to Agent1 when $a_{1t}$was finished

When all tasks in the queue were arranged to execute according to a_{1t}, the system entered the next task scheduling time slot $t + 1$.

When the system reached time $\widehat{t}$, the system would start the optimization of resource utilization. At time $\widehat{t}$, the corresponding actions were performed in the following steps:

(i) The system input the current environment status $s_{2\widehat{t}}$ to Agent2

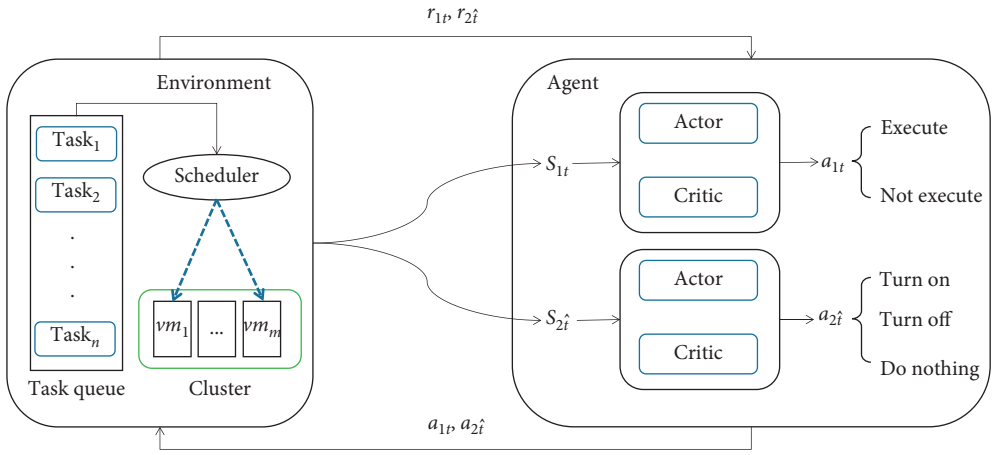

Figure 1: Scheduling system architecture.

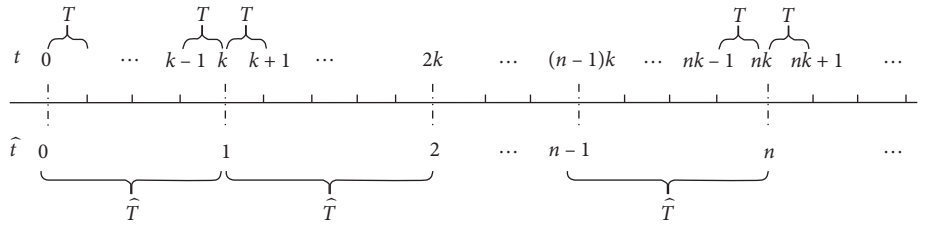

Figure 2: Model of scheduling time.

(ii) Agent2 output action decision $a_{2\hat{t}}$ and returned it to the environment

(iii) The environment executed action decision $a_{2\hat{t}}$ to shut down or start up a certain number of virtual machines and returned the reward r_{2t} to Agent2; then, the system entered the next time slot $\hat{t}+1$

3.2. Related Definitions

3.2.1. Task Priority. Tasks were the jobs running on virtual machine cluster. Tasks arriving in time slot t were added to the task queue waiting for virtual machine allocation. As mentioned above, the environment had little information about the exact number and size of tasks in advance, so the task priority could hardly be simply calculated by waiting time or task execution time. We proposed function $p(i)$ to estimate the priority of task i. In this function, e_i was the execution time of the task i and w_i was the waiting time of the task i. $p(i)$ is defined in the following equation:

$$p(i) = \frac{(e_i + w_i)}{e_i}. \tag{1}$$

After the priorities were calculated, all tasks were scheduled according to their priorities.

3.2.2. Action1. Action1 was the action space of all actions in task schedule. The element of Action1, a_{1i}, indicated whether a certain virtual machine was allocated to the task i. $a_{1i} \in [0,1]$. The values of a_{1i} were determined in the following equation:

$$a_{1i} = \begin{cases} 1, & \text{if task } i \text{ gets a virtual machine,} \\ 0, & \text{if task } i \text{ does not get a irtual machine.} \end{cases} \tag{2}$$

3.2.3. State1. State1 was the status space of environment for task scheduling. s_{1t}, the instance of State1, was defined as the vector $(e_{1t}, p_t, m_{1t}, n_{1t})$, where e_{1t} was the execution time of the task that was allocated a virtual machine, p_t was the priority of the task that was allocated a virtual machine, m_{1t} was the average priority of all the tasks in the task queue (see equation (3)), n_{1t} was the proportion of the active virtual machines in virtual machine cluster (see equation (4)), and Nt was the number of tasks in time slot t:

$$m_{1t} = \frac{1}{Nt}\sum_{i=1}^{Nt} p(i), \tag{3}$$

$$n_{1t} = \frac{n_{\text{active\_vm}}}{M}. \tag{4}$$

3.2.4. Reward1. The reward value represented the feedback value after the action was performed. The reward value of time slot t is defined in the following equation:

$$r_{1t} = \mu * m_{1t} + \eta * n_{1t}, \tag{5}$$

where μ and η were calibration parameters, which were used to adjust the influence of average task priority m_{1t} and active virtual machine proportion n_{1t}. The values of μ and η were between -1 and 1.

3.2.5. Action2. Action2 represented the number of virtual machines turned on or off in time slot $\widehat{t}$. The instance of Action2 was defined as $a_{2\widehat{t}} \in [-M, M]$. When $a_{2\widehat{t}}$ was greater than 0, it meant there were $a_{2\widehat{t}}$ virtual machines turned on. When $a_{2\widehat{t}}$ was equal to 0, it meant that no change occurred. When $a_{2\widehat{t}}$ was less than 0, it meant that there were $a_{2\widehat{t}}$ virtual machines shut down.

3.2.6. State2. State2 was the state space for Agent2. $s_{2\widehat{t}}$, the instance of State2, was defined as $(e_{2\widehat{t}}, l_{\widehat{t}}, m_{2\widehat{t}}, n_{2\widehat{t}})$, where $e_{2\widehat{t}}$ was defined as the logarithm of the sum of e_$t_{\widehat{t}}$ and e_$t'_{\widehat{t}-1}$, where e_$t_{\widehat{t}}$ was the sum of the task execution time of the tasks arrived in time slot $\widehat{t}$, and e_$t'_{\widehat{t}-1}$ was the sum of the task execution time of the tasks not executed at previous time slot $\widehat{t}-1$. $l_{\widehat{t}}$ was the logarithm of the sum of $n\_t_{\widehat{t}}$ and $n\_t'_{\widehat{t}-1}$, where $n\_t_{\widehat{t}}$ was the number of tasks arrived in time slot $\widehat{t}$ and $n\_t'_{\widehat{t}-1}$ was the number of tasks not executed in the previous time slot $\widehat{t}-1$. $m_{2\widehat{t}}$ was the average value of m_{1t} in time slot $\widehat{t}$. $n_{2\widehat{t}}$ was the average proportion of idle virtual machines in time slot $\widehat{t}$. `

$$\begin{aligned} e_{2\widehat{t}} &= \log\left(e\_t_{\widehat{t}} + e\_t'_{\widehat{t}-1}\right), \quad \widehat{t} = 0, 1, 2, 3, \ldots, \\ l_{\widehat{t}} &= \log\left(n\_t_{\widehat{t}} + n\_t'_{\widehat{t}-1}\right), \quad \widehat{t} = 0, 1, 2, 3, \ldots, \\ m_{2\widehat{t}} &= \frac{1}{K} \sum_{i=K^*(\widehat{t}-1)}^{K*\widehat{t}} m_{1i}, \quad \widehat{t} = 0, 1, 2, 3, \ldots; K = \frac{\widehat{T}}{T}, \\ n_{2\widehat{t}} &= \frac{1}{K} \sum_{i=k*(\widehat{t}-1)}^{k*\widehat{t}} 1 - n_{1i}, \quad \widehat{t} = 0, 1, 2, 3, \ldots; K = \frac{\widehat{T}}{T}. \end{aligned} \tag{6}$$

3.2.7. Reward2. Reward2 was the value of reward function for Agent2. It was determined in equation (7), where $\widehat{\mu}$ and $\widehat{\eta}$ were calibration parameters. We adjusted the value of $\widehat{\mu}$ and $\widehat{\eta}$ according to the actual situation to tune the proportion of the active virtual machine and the proportion of idle virtual machines in time slot $\widehat{t}$:

$$r_{2\widehat{t}} = \widehat{\mu} n_{2\widehat{t}} - \widehat{\eta} m_{2\widehat{t}}. \tag{7}$$

4. Reinforcement Learning Algorithm for Scheduling Optimization

In this section, the actor-critic deep reinforcement learning algorithm [21, 27, 28] was applied to create the model for scheduling optimization of data centers. The actor-critic algorithm is a hybrid algorithm based on Q-learning and policy gradient which are two classic algorithms of reinforcement learning. The actor-critic algorithm shows outstanding performance in complicated machine learning missions.

A2C was selected in this study. The structure based on A2C is shown in Figure 3. In A2C, the actor network is used for action selection and the critic network is used to evaluate the action.

As mentioned in Section 3, Agent1 acted as the optimization model for task scheduling and Agent2 as the optimization model for resource utilization. (State1, Action1, Reward1) and (State2, Action2, Reward2) were used to describe the state space, action space, and reward function of Agent1 and Agent2, respectively. Hence, (s_{1t}, a_{1t}, r_{1t}) and $(s_{2\widehat{t}}, a_{2\widehat{t}}, r_{2\widehat{t}})$ separately represented one instance in the state spaces, action space, and the reward functions of Agent1 and Agent2 at time slot t and time slot $\widehat{t}$. The data entry (s_t, a_t, r_t, s_{t+1}) was recorded as a sample for the training with the A2C algorithm.

The parameter of the actor network is updated by advantage function $A(s_t, a_t)$ (see equation (8)). θ_a (see equation (9)) and θ_c (see equation (10)) are the parameters of the actor network and critic network, respectively:

$$A(s_t, a_t) = r_t + \gamma V^{\pi_\theta}(s_{t+1}; \theta_c) - V^{\pi_\theta}(s_{t+1}; \theta_c), \tag{8}$$

$$\theta_a \longleftarrow \theta_a + \alpha \sum_t \nabla \log \pi_{\theta_a}(s_t, a_t) A(s_t, a_t) + \beta \nabla_{\theta_a} H\left(\pi_\theta\left(\cdot \mid s_t\right)\right), \tag{9}$$

$$\theta_c \longleftarrow \theta_c - \alpha' \sum_t \nabla_{\theta_c} \left(A(s_t, a_t)\right)^2, \tag{10}$$

where α is the learning rate of the actor network; β is a hyperparameter, and H is the entropy of the policy.

In this study, the full-connection layers were used to build the network, in which Agent1 used the six-layer full-connection network, and Agent2 used the four-layer full-connection network. The size of the hidden layer in both agents was 1024. In the training phase, in order to solve the cold start problem of reinforcement learning and accelerate the convergence of the model, the First-Come-First-Service tactic was applied in the early stage for the allocation of virtual machines in the virtual machine cluster and experiences were collected from the results to achieve a better initial status. In the experiences, running_steps, agent1_batch_size, and agent2_batch_size were the control parameters of the training algorithm. The flowchart of the training algorithm is shown in Figure 4.

In this study, floating point operations per second (FLOPS) was used to evaluate the complexity of the proposed scheduling algorithm. According to the structure of the full-connection network and the input data shown in Figure 3, the complexity of the proposed algorithm was evaluated by

$$\text{Time} \sim O\left(L * I^2 * K * N\right), \tag{11}$$

where K was the ratio of duration of resource utilization $\widehat{T}$ to the duration of task scheduling T defined in the model of scheduling time in Section 3.1; N was the maximum number

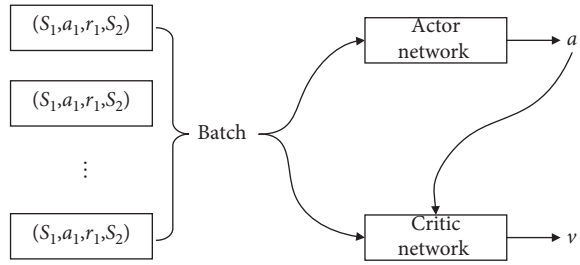

Figure 3: Advantage actor-critic structure.

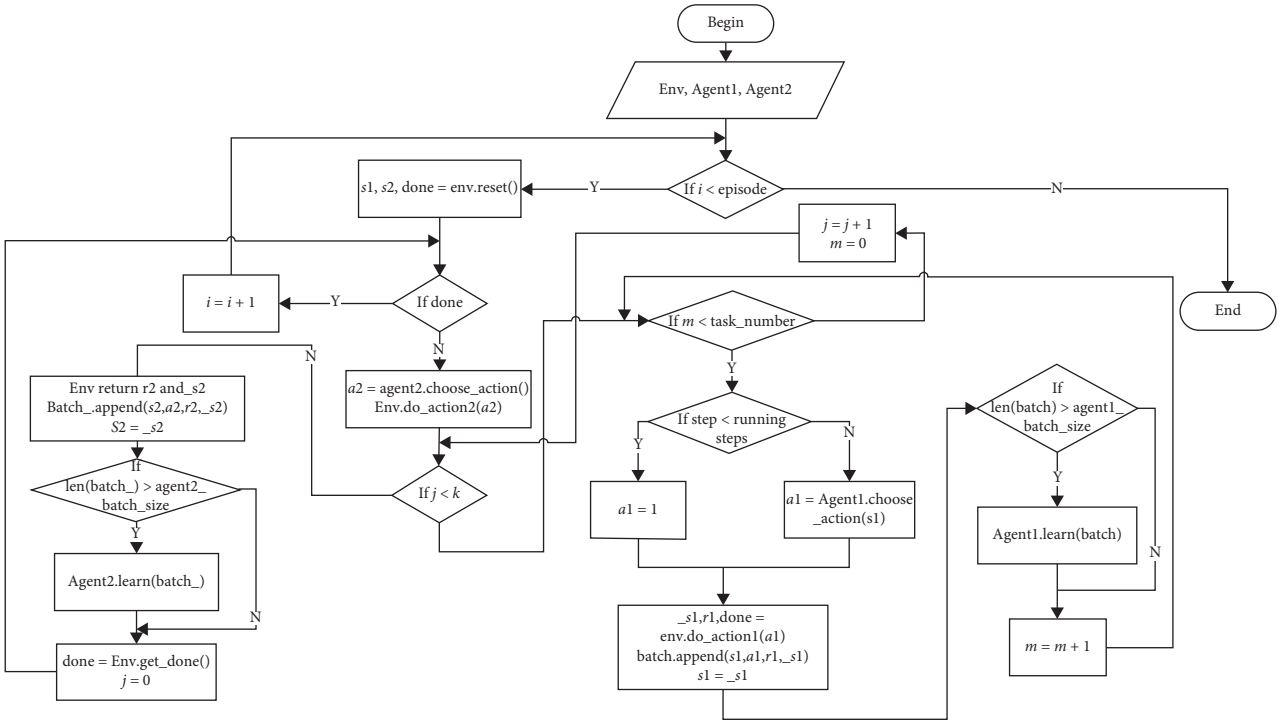

Figure 4: The flowchart of model training process.

of tasks in time slot t; I was the number of nodes in the hidden layer, and L was the number of hidden layers of the network of Agent1 and Agent2. It hinted that given the A2C-based model above, the performance of the proposed algorithm was highly influenced by K and N. In this study, the complexity of the proposed algorithm with the model trained above was about $O(6 * 2^{20} * K * N)$.

5. Experiment Study

5.1. Dataset. Most previous studies used self-generated datasets in their training work, which was not suitable to validate the applicability of the algorithm in the real production environment. In order to verify the effectiveness of the proposed algorithm in the real production environment, cluster-trace-v2017, the real production dataset published by the Alibaba Cluster Trace Program was used in the experiment study. The data are about the cluster trace from real production environment, which helps the researchers to get better understanding of the characteristics of modern Internet data centers (IDCs) and the workloads [4]. The trace dataset includes the collocation of online services and batch workloads about 1300 machines in a period of 12 hours. The trace dataset contains six kinds of collections: machine_meta.csv, machine_usage.csv, container_meta.csv, container_usage.csv, batch_instance.csv, and batch_task.csv. The task information used in the experiment was from batch_instance.csv collection. In this collection, task id, start time, end time, and other attributes are provided.

5.2. Experiment Settings. Here, it was assumed that one virtual machine could not perform multiple tasks at the same time. The virtual machine was equipped with i7-8700k CPU, gtx2080 graphics card, and 32G RAM. The parameters of scheduling system were set as shown in Table 2.

T_{and} $\widetilde{T}$ were set to the empirical value in scheduling. M was the number of virtual machines in different clouding configurations for comparison. α,α',γ, and γ' were set to the empirical value in the A2C algorithm. μ and η were used to control the average priority of tasks and the proportion of active virtual machines. The default value of them was −1. If

Table 2: System parameter setting of scheduling experiment.

Parameter	Value	Parameters	Value
T	3 sec.	M	300~500
$\widehat{T}$	300 sec.	α	1e–4
μ	−1.0	α'	1e–4
η	−0.9	γ	0.9
$\widehat{\mu}$	−1.0	γ'	0.9
$\widehat{\eta}$	1.0		

the goal was to reduce the priority, the ratio of μ/η was increased appropriately. If the goal was to control the cost and reduce the proportion of idle virtual machines, the ratio of μ/η was decreased. The setting of $\widehat{\mu}$ and $\widehat{\eta}$ was the same.

5.3. Model Training. With the trace data from the real production environment, we trained the model with the algorithm introduced in Section 4 and recorded the loss value at each training step. The reward values were represented by the average reward of each episode.

Figure 5 shows the trend of the loss and the reward in training process. Figure 5(a) shows loss trend graph, in which x-axis is the number of training steps and y-axis is the value of loss. It can be seen from the graph that with the increase of training steps, loss gradually decreases until convergence. Figure 5(b) is the reward trend graph, in which the y-axis is reward value and the x-axis is the episode. It shows that with the increase of episode, reward gradually increases and eventually converges at a higher value. It hinted that the performance of the model trained by the algorithm was satisfactory.

5.4. Comparison with Traditional Scheduling Methods. We compared the proposed A2C scheduling algorithm with classical First-Come-First-Service (FCFS), Shortest-Job-First (SJF), and Fair algorithms in the following two experiments.

5.4.1. Experiment1. The fixed number of virtual machines was set to 300, 350, 400, 450, and 500 in the cluster and ran the different algorithms on the dataset. For the proposed algorithm of this paper, only the task scheduling agent (Agent1) worked in experiment1. The result of experiment 1 shown in Figure 6 shows that the average task delay time and the average task priority of the proposed A2C algorithm are less than those of other algorithms with different size of clusters. The results implied that the proposed algorithm worked better in task scheduling than others with different fixed numbers of resources.

5.4.2. Experiment2. In this experiment, Agent 2 worked with Agent1 to schedule the task with dynamic resource allocation. The performance of the proposed algorithm was compared with other algorithms in three dimensions: average delay time of tasks, tasks distribution in different delay time levels, and task congestion degree.

The initial size of virtual machine cluster (M) was set the same for FCFS, SJF, and Fair algorithms, and a dynamic size (M') was set for the proposed algorithm. In order to ensure the fair resources supporting for all algorithms, the maximum value of M' was set up to $1.1 \times M$.

(1) Comparison of Average Delay Time of Tasks. The experiment results on different size of virtual machine clusters are shown in Table 3.

It shows that the proposed algorithm automatically expands the cluster size when the cluster scale is smaller than 400. When the size is set to 300, the cluster size increases by 4% with the task delay decreases by at least 22% compared with those of other algorithms. When the size is set to 350, the cluster size increases by 2.8% with the task delay decreases by at least 28% compared with others. When the cluster size is at a larger level over 400, the proposed algorithm can automatically reduce the cluster size significantly, while the task delay is also considerably smaller than that of Fair, FCFS, and SJF algorithms. In order to show the performance of the proposed algorithm clearly, we defined the relative delay time τ in equation (12). If $\tau = 1$, it means the performance of the compared algorithm is as good as the proposed algorithm. If $\tau > 1$, it means the performance of the compared algorithm is worse than the proposed algorithm; otherwise, it means the performance of the algorithm is better than the proposed algorithm:

$$\tau_m^i = \frac{t_m^i}{t_m'} * \frac{m}{m'}, \tag{12}$$

where $i \in$ (Fair, FCFS, SJF):

$$\begin{aligned} M &\in (300, 350, 400, 450, 500), \\ M' &\in (312, 360, 387, 438, 464), \end{aligned} \tag{13}$$

t_m^i was the average delay time of algorithm i with the cluster size m. t_m' was the average delay time of the proposed algorithm with the cluster size M'.

According to equation (12), the relative delay time was converted from the data in Table 2 and is shown in Table 4.

The results are all greater than 1, which indicates that the task scheduling performance of other compared algorithms with resource utilization is not as good as the proposed algorithm.

(2) *Comparison of Task Distribution in Different Delay Time Levels.* In this section, the tasks in different delay time intervals were considered. The unit of the delay time interval was T, the duration of task scheduling. Table 5 shows the statistical result.

The data in Table 5 are about the percentage of tasks whose delay time is less than the delay time interval. The statistical result clearly shows that the proposed algorithm has higher percentage than other algorithms in all delay time intervals. It means, compared with other algorithms, the less the delay time interval, the higher the percentage of undelayed tasks in this delay time interval. Especially, in the

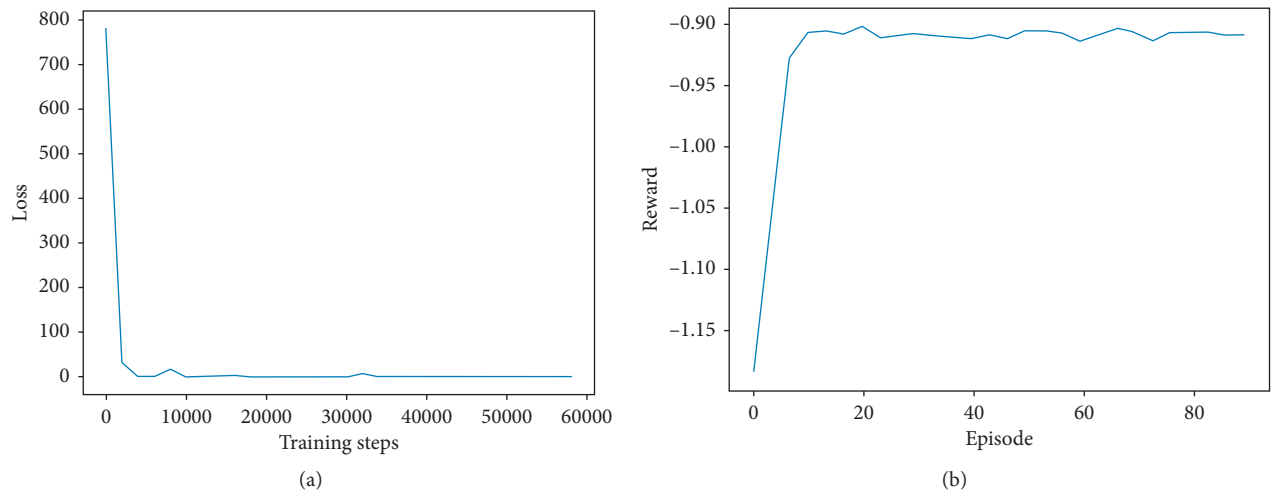

Figure 5: Trend of the loss and the trend of the reward. (a) Loss trend. (b) Reward trend.

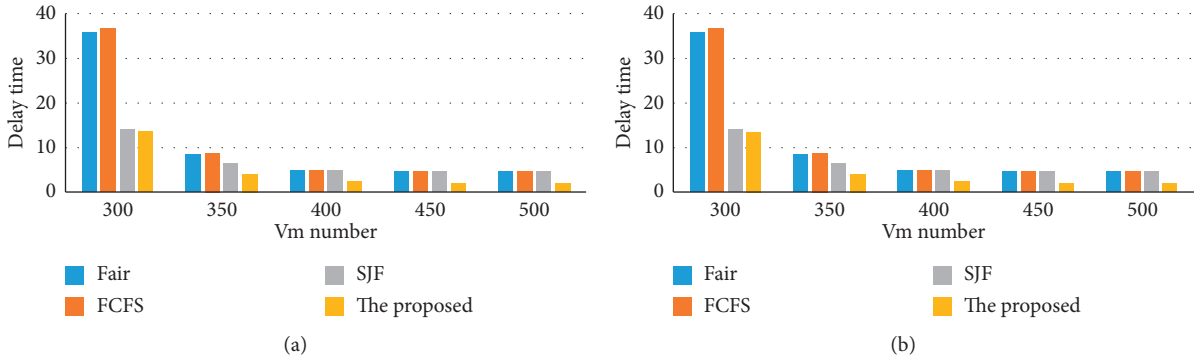

Figure 6: Results of experiment 1. (a) Comparison of average task delay. (b) Comparison of average task priority.

Table 3: Average delay time of different algorithms on different size of virtual machine clusters (in seconds).

Cluster size	Fair	FCFS	SJF	Proposed algorithm (dynamic cluster size)
300	35.9267	36.8301	14.1581	10.5032 (312)
350	8.5735	8.7036	6.3110	2.86432 (360)
400	4.9019	4.9312	4.8311	2.284914 (387)
450	4.6944	4.6953	4.6936	2.0014 (438)
500	4.6931	4.6931	4.6931	1.9975 (464)

Table 4: The relative delay time of the compared algorithms.

Cluster size	Fair	FCFS	SJF
300	3.371	3.371	1.296
350	2.954	2.954	2.142
400	2.231	2.231	2.185
450	2.410	2.410	2.409
500	2.531	2.531	2.532

1T level, the percentage of tasks out of all is almost twice as much as that of others.

(3) Comparison of Task Congestion Degree. Task congestion degree reflected the number of tasks waiting for execution in the task queue at the end of each time slot. It was measured by the percentage of time slots in which the number of congested tasks was less than a certain benchmark number. Therefore, with a certain benchmark number, the less the congestion degree was, the better the scheduling algorithm worked. The statistical result is illustrated in Table 6 and Figure 7.

It can be seen from the data in first row of Table 6 that the proposed algorithm makes all tasks executed in over 82% time slots and for other algorithms, all tasks are executed only in about 30% of all time slots. The change of task congestion degree of all algorithms with more benchmark numbers is visualized by the plot chart in Figure 7, which shows that the congestion degree with the proposed algorithm remains relatively stable and better than other algorithms with all benchmark numbers of congested tasks.

Table 5: The task distribution in different delay time intervals.

Task delay time interval (T)	SJF (%)	FCFS (%)	Fair (%)	The proposed algorithm (%)
1	36.675	13.045	25.688	51.781
2	66.734	46.833	51.021	81.115
3	77.775	65.546	63.935	84.691
4	83.226	71.036	70.797	87.173
5	86.451	73.118	74.603	88.783
10	92.610	76.438	80.914	92.954
15	94.807	78.642	83.803	94.905
20	95.948	80.761	86.011	96.017

Table 6: Task congestion degree.

Number of congested tasks	SJF (%)	FCFS (%)	Fair (%)	The proposed algorithm (%)
0	33.739	31.778	31.802	82.277
5	65.444	62.005	61.990	83.320
10	75.056	70.965	71.107	84.335
15	79.332	74.847	74.999	85.325
20	81.584	76.622	76.867	86.248
25	83.204	77.809	77.952	87.044
30	84.366	78.685	78.643	87.835
...	...	...	...	...

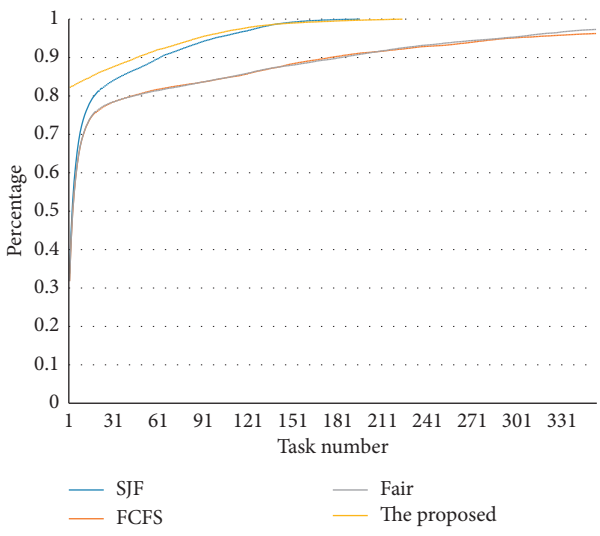

Figure 7: Task congestion degree chart.

6. Conclusions and Further Study

With the expansion of online business in data center, task scheduling and resource utilization optimization become more and more pivotal. Large data center operation is facing the proliferation of uncertain factors, which leads to the geometric increase of environmental complexity. The traditional heuristic algorithms are difficult to cope with today's complex and constantly changing data center environment. However, most of the previous studies focused on one aspect of data center optimization and most of them did not verify their algorithms on the real data center dataset.

Based on the previous studies, this paper designed a reasonable deep reinforcement learning model, optimized the task scheduling and resource utilization. Experiments were also performed with the real production dataset to validate the performance of the proposed algorithm. Average delay time of tasks, task distribution in different delay time levels, and task congestion degree as performance indicators were used to measure the scheduling performance of all algorithms applied in the experiments. The experiment results showed that the proposed algorithm worked significantly better than the compared algorithms in all indexes listed in experiments, ensured the efficient task scheduling and dynamically optimized the resource utilization of clusters.

It should be noted that reinforcement learning is a kind of time-consuming work. In this study, it took 12 hours to train the model with the sample data from the real production environment containing 1300 virtual machines. We also used other production data (cluster-trace-v2018) from the Alibaba Cluster Trace Program to train the scheduling model. The dataset is about 4000 virtual machines in an 8-day period, which is much bigger than the former dataset. As the dataset's scale increased, the training time became considerably longer and underfit occurred if the number of layers of the machine learning model was not increased. As shown in equation (11), the performance of the proposed algorithm is determined by the complexity of the full-connection network, time slot ratio, and the number of tasks in a time slot. Considering the trade-off of the training time cost and the performance, it is strongly recommended to prepare the sample dataset to a reasonable scale with sampling technology to reduce the complexity of the scheduling model.

In addition, we did not consider other task attributes and constraints of data center environment. Fox example, in this study, it was assumed that the number of virtual machines was not dynamically changed, and the ability of all virtual

machines were the same. It was also assumed that one virtual machine cannot perform multiple tasks at the same time. Therefore, we do not imply that the proposed model is available for all kinds of clusters. Notwithstanding its limitation, in the future study, the proposed model should be improved to optimize the task scheduling in the heterogeneous environments of the data center clusters through taking more constraints and attributes of real production environment into account.

Data Availability

The dataset used to support the study is available at the Alibaba Cluster Trace Program (https://github.com/alibaba/clusterdata).

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

The research for this article was sponsored under the project "Intelligent Management Technology and Platform of Data-Driven Cloud Data Center", National Key R&D Program of China, no. 2018YFB1003700.

References

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

[2] Z. Wang, Z. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proceedings of the 33rd International Conference on Machine Learning*, pp. 1995–2003, New York, NY, USA, June 2016.

[3] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pp. 50–56, Atlanta, GA, USA, November 2016.

[4] Alibaba Group, "Alibaba cluster trace program," 2019, https://github.com/alibaba/clusterdata.

[5] C. Delimitrou and C. Kozyrakis, "QoS-Aware scheduling in heterogeneous datacenters with paragon," *ACM Transactions on Computer Systems*, vol. 31, no. 4, pp. 1–34, 2013.

[6] J. Perry, A. Ousterhout, H. Balakrishnan, and D. Shah, "Fastpass: a centralized zero-queue datacenter network," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 307–318, 2014.

[7] H. W. Tseng, W. C. Chang, I. H. Peng, and P. S. Chen, "A cross-layer flow schedule with dynamical grouping for avoiding TCP Incast problem in data center networks," in *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, pp. 91–96, Odense, Denmark, October 2016.

[8] H. Yuan, J. Bi, W. Tan, and B. H. Li, "CAWSAC: cost-aware workload scheduling and admission control for distributed cloud data centers," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 976–985, 2016.

[9] H. Yuan, J. Bi, W. Tan, and B. H. Li, "Temporal task scheduling with constrained service delay for profit maximization in hybrid clouds," *IEEE Transactions on Automation Science and Engineering*, vol. 14, no. 1, pp. 337–348, 2017.

[10] J. Bi, H. Yuan, W. Tan et al., "Application-aware dynamic fine-grained resource provisioning in a virtualized cloud data center," *IEEE Transactions on Automation Science and Engineering*, vol. 14, no. 2, pp. 1172–1183, 2017.

[11] 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 Transactions on Cybernetics*, vol. 47, no. 11, pp. 3658–3668, 2017.

[12] H. Yuan, H. Liu, J. Bi, and M. Zhou, "Revenue and energy cost-optimized biobjective task scheduling for green cloud data centers," *IEEE Transactions on Automation Science and Engineering*, vol. 17, pp. 1–14, 2020.

[13] L. Zhang, J. Bi, and H. Yuan, "Workload forecasting with hybrid stochastic configuration networks in clouds," in *Proceedings of the 2018 5th IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS)*, pp. 112–116, Nanjing, China, November 2018.

[14] J. Bi, H. Yuan, L. Zhang, and J. Zhang, "SGW-SCN: an integrated machine learning approach for workload forecasting in geo-distributed cloud data centers," *Information Sciences*, vol. 481, pp. 57–68, 2019.

[15] J. Bi, H. Yuan, and M. Zhou, "Temporal prediction of multiapplication consolidated workloads in distributed clouds," *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 4, pp. 1763–1773, 2019.

[16] S. Luo, H. Yu, Y. Zhao, S. Wang, S. Yu, and L. Li, "Towards practical and near-optimal coflow scheduling for data center networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 11, pp. 3366–3380, 2016.

[17] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, USA, 1998.

[18] J. Yuan, X. Jiang, L. Zhong, and H. Yu, "Energy aware resource scheduling algorithm for data center using reinforcement learning," in *Proceedings of 2012 Fifth International Conference on Intelligent Computation Technology and Automation*, pp. 435–438, Hunan, China, January 2012.

[19] X. Lin, Y. Wang, and M. Pedram, "A reinforcement learning-based power management framework for green computing data centers," in *Proceedings of 2016 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 135–138, Berlin, Germany, April 2016.

[20] Y. Li, Y. Wen, D. Tao, and K. Guan, "Transforming cooling optimization for green data center via deep reinforcement learning," *IEEE Transactions on Cybernetics*, vol. 50, no. 5, pp. 2002–2013, 2020.

[21] R. Shaw, E. Howley, and E. Barrett, "An advanced reinforcement learning approach for energy-aware virtual machine consolidation in cloud data centers," in *Proceedings of 2017 12th International Conference for Internet Technology and Secured Transactions (ICITST)*, pp. 61–66, Cambridge, UK, December 2017.

[22] D. Basu, Q. Lin, W. Chen et al., "Regularized cost-model oblivious database tuning with reinforcement learning," *Lecture Notes in Computer Science*, vol. 9940, pp. 96–132, 2016.

[23] Z. Peng, D. Cui, J. Xiong, B. Xu, Y. Ma, and W. Lin, "Cloud job access control scheme based on Gaussian process regression and reinforcement learning," in *Proceedings of 2016 IEEE 4th*

International Conference on Future Internet of Things and Cloud (FiCloud), pp. 276–284, Vienna, Austria, August 2016.

[24] F. Ruffy, M. Przystupa, I. Beschastnikh, and " Iroko, "A framework to prototype reinforcement learning for data center traffic control," 2018, http://arxiv.org/abs/1812.09975.

[25] S. He, H. Fang, M. Zhang, F. Liu, X. Luan, and Z. Ding, "Online policy iterative-based H∞ optimization algorithm for a class of nonlinear systems," *Information Sciences*, vol. 495, pp. 1–13, 2019.

[26] S. He, H. Fang, M. Zhang, F. Liu, and Z. Ding, "Adaptive optimal control for a class of nonlinear systems: the online policy iteration approach," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 2, pp. 549–558, 2020.

[27] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in Neural Information Processing Systems*, vol. 12, pp. 1057–1063, MIT Press, Cambridge, MA, USA, 2000.

[28] Y. Wu, E. Mansimov, S. Liao, A. Radford, and J. Schulman, "OpenAI baselines: ACKTR & A2C," 2017, https://openai.com/blog/baselines-acktr-a2c.