Batch Jobs Load Balancing Scheduling in Cloud Computing Using Distributional Reinforcement Learning

Tiangang Li, Shi Ying, Yishi Zhao, and Jianga Shang, *Member, IEEE*

***Abstract*—In cloud computing, how to reasonably allocate computing resources for batch jobs to ensure the load balance of dynamic clusters and meet user requests is an important and challenging task. Most existing studies are based on deep Q network, which utilizes neural networks to estimate the expected value of cumulative return in the scheduling process. The value-based DQN algorithms ignore the complete information contained in the value distribution and lack strong adaptability to time-varying batch jobs and dynamic cluster resources. Therefore, to capture the inherent stochasticity of the scheduling process caused by environmental stochasticity, we utilize Distributional Reinforcement Learning to model the value distribution of the cumulative return. Specifically, we formalize the load balancing scheduling as a multi-objective optimization problem and construct a Distributional Reinforcement Learning model. Then we introduce quantile regression to learn the value distribution of the cumulative return during scheduling and propose a dynamic load balancing scheduling algorithm based on Distributional Reinforcement Learning. In addition, we develop a cluster environment for real-time processing of batch jobs to simulate the arrival of batch jobs and train the Distributional Reinforcement Learning-based scheduling agent. We conduct empirical experiments and detailed analysis by using the real Alibaba Cluster cluster traces v2018 and v2020. The results show that compared to the baseline algorithms, the proposed algorithm performs better in terms of cluster load balancing, success rate of instance creation and average completion time of the tasks. The experimental results on different trace datasets also indicate that the propsoed algorithm exhibits excellent scalability.**

***Index Terms*—Batch jobs scheduling, cloud computing, distributional reinforcement learning, load balancing, service level agreement.**

Manuscript received 14 April 2023; revised 1 October 2023; accepted 15 November 2023. Date of publication 20 November 2023; date of current version 8 December 2023. This work was supported in part by the National Key Research and Development Plan Project of China under Grant 2022YFB3304300, and in part by the National Natural Science Foundation Project of China under Grants 62072342 and 61672392. Recommended for acceptance by J. Zola. *(Corresponding author: Shi Ying.)*

Tiangang Li and Shi Ying are with the School of Computer Science, Wuhan University, Wuhan 430072, China (e-mail: tiangangli@whu.edu.cn; yingshi@whu.edu.cn).

Yishi Zhao is with the School of Computer Science, China University of Geosciences, Wuhan 430072, China (e-mail: zhaoyishi@cug.edu.cn).

Jianga Shang is with the School of Computer Science and the National Engineering Research Center for Geographic Information System, China University of Geosciences, Wuhan 430072, China (e-mail: jgshang@cug.edu.cn).

Digital Object Identifier 10.1109/TPDS.2023.3334519

I. Introduction

CLOUD computing plays a supportive role in the development of information technology in various industries because of its resource sharing, on-demand access, and flexible deployment [1], [2]. As cloud computing services are widely used in different industries, batch jobs load from different types of users is gradually increasing [3], [4]. The resource usage of complexly composed batch jobs load during operation fluctuates significantly over time [5], [6], [7]. In cloud computing, load balancing scheduling plays a crucial role in maintaining the stability of the cloud computing platform and improving the Service of Quality (QoS) [8].

Therefore, how to effectively schedule batch jobs has become a key research issue in cloud computing, and is also regarded as a very challenging task. The specific description is as follows.

- *Complexity of time-varying batch jobs:* Batch jobs run for periods ranging from seconds to days, with random arrival times and significant fluctuations in resource usage over time [9]. Furthermore, batch jobs are combined according to different applications run by users, and each job contain multiple computing tasks with different resource requests [10], [11], [12]. The complexity of time-varying batch jobs makes scheduling more difficult. [13]
- *Dynamicity of cluster resources:* Due to the complexity and unpredictability of user requests, clusters need to dynamically adjust resources to meet user needs. Cloud computing cluster consists of a variety of machines with different resource configurations, providing multi-dimensional resources including CPU, memory, and storage [1], [9]. Therefore, load balancing scheduling in a dynamically changing multi-dimensional resource environment is a considerable challenge.
- *Constraints of Service Level Agreement (SLA):* Violations of SLA indicate that the service cannot be guaranteed, resulting in unnecessary consequences [14]. The goal of load balancing scheduling in cloud computing platform is to prevent various SLA violations, improve the quality of service as much as possible and increase the success rate of executing user requests [15], [16]. Under the constraints of SLA, load balancing scheduling becomes a complex multi-objective optimization problem [8].

Concerning load balancing scheduling in cloud computing, static algorithms represented by Round Robin and Random

Scheduling have been proposed [17]. But these algorithms are designed based on rules, which need to be adjusted according to scenarios in practical applications, and cannot adapt to a long-term dynamic changing environment. Some heuristics-based load balancing algorithms have also been widely studied [18], [19], [20]. These methods require a priori characteristics of the system to define the heuristics, and most of them only focus on job-related information, which also cannot effectively adapt to the long-running and constantly changing cloud computing environment. In addition, these algorithms cannot provide satisfactory solutions to the multi-objective optimization problem in cloud computing.

Recently, the method based on deep reinforcement learning (DRL) has been increasingly applied to load balancing scheduling in cloud computing and is regarded as a promising research direction. DRL introduces deep neural networks into reinforcement learning, which addresses the problem that traditional reinforcement learning algorithms cannot deal with high-dimensional space [21]. Therefore, in the load balancing scheduling, the method based on DRL is regarded as a promising method to deal with the changes in system state during the scheduling process caused by the complexity of batch jobs and the dynamicity of cluster resources. At the same time, DRL has become an effective method to solve multi-objective optimization problems in a dynamic environment due to its advantages of fast solution speed and strong generalization ability [22].

The DRL-based load balancing scheduling algorithm automatically acquires the characteristics of cluster and jobs in the cloud computing environment and make adaptive adjustments as the environment changes. The process of training DRL agent does not require prior features. Through interaction with the environment, the agent randomly explores different situations and makes autonomous decisions to take action to change the state of the environment. Then the agent receives the reward returned by the environment to accumulate experience for the next decision. By accumulating a large amount of experience, the agent can dynamically adjust the network parameters in response to changes in the environment to generate the optimal decision strategy, with the purpose of maximizing the future cumulative return.

Although the existing DRL-based methods have made some progress, most existing studies adopt the methods based on Deep Q-Network to make decision strategies by modeling the value of cumulative return, ignoring the complete information contained in the value distribution [8], [15], [16], [23]. However, the cluster resource environment is dynamic and changeable, with batch jobs arriving over time and complex composition. So it is difficult to converge to the optimal load balancing strategy by using the value-based DRL method. Compared to DQN-based algorithms, policy-based DRL algorithm (e.g., REINFORCE [24]) has the capability to directly learn stochastic policies. It updates policies by parameterizing policies and using the expected reward gradient over episodes. The objective of policy gradient algorithm is to maximize the expected reward. Policy-based algorithm is effective in handling large action spaces in cloud data center. However, Chen et al. [1] point out that policy-based algorithms might suffer from high variance when estimating the policy gradients, which can lead to reduced the convergence efficiency and stability of the algorithm.

The actor-critic framework combines the value-based and policy-based DRL algorithms [24], which can simultaneously utilize the actor network to learn the policy function for action output and the critic network to assess the value of actions based on the state. This approach helps mitigate the variance in policy gradient estimation, enhancing the stability of policy training. However, when learning the state-value function, the objective remains to maximize the value of the expected reward, and since the critic network initializes as a random network, it is challenging to provide accurate predictions of value function. This can result in inaccurate estimation of the policy gradient, affecting the convergence speed during training. In order to capture the inherent stochasticity of the scheduling process caused by the stochasticity of the environment. We utilize Distributional Reinforcement Learning [25] to model the distribution of cumulative returns, which can better cope with the complex and changeable load balancing scheduling environment.

In order to address the problems in the value-based DRL, we utilize quantile to describe the distribution of cumulative return during the scheduling process and introduce quantile regression to learn the quantile value corresponding to the distribution [26]. The proposed algorithm can better capture the stochasticity of batch jobs and cluster resources during scheduling. For a cloud computing environment with time-varying and complex batch jobs, dynamic resource changes, and SLA constraints, we develop a dynamic multi-objective load balancing scheduling algorithm based on Distributional Reinforcement Learning.

Specifically, the contributions of this paper are as follows.

- We design a system model for load balancing scheduling in cloud computing, including complex batch jobs that arrive over time, dynamically changing multidimensional cluster resources, and a scheduler that maintains load balancing. Throughout the whole scheduling process, we aim to improve the load balance of multi-dimensional resources and users' quality of service. We also model the scheduling process as a Markov decision process according to the system model and design the state space, action space, and reward function to be utilized in the proposed load balancing algorithm.
- We propose a dynamic load balancing scheduling algorithm based on Distributional Reinforcement Learning. From the perspective of value distribution, we model the distribution of cumulative return and capture the inherent stochasticity of the scheduling process due to the stochasticity of the cloud computing environment. The proposed algorithm better adapts to the dynamic cloud computing environment.
- We conduct extensive experiments using real production trace data from Alibaba clusters to evaluate the performance of the proposed algorithm in improving load balancing and users quality of service in cloud computing clusters. The experimental results demonstrate that the proposed algorithm achieves faster and better convergence and better performance in terms of load balancing and quality of service than two classical load balancing scheduling

algorithms and two improved DQN-based load balancing scheduling algorithms.

The rest of this paper is organized as follows. In Section II, we introduce the research work related to this paper. In Section III, we introduce the system model designed in this paper and the formalization of the research problem. In Section IV, we introduce the proposed Distributional Reinforcement Learning model and load balancing scheduling algorithm. In Section V, we carry out an experimental evaluation and analysis of the proposed method. Section VI summarizes the work of the full paper.

II. Related Work

Load balancing scheduling has long been a popular research direction in cloud computing. Many research works have played a certain role in promoting the development of this direction from different perspectives. In this section, we present research work related to traditional load balancing scheduling algorithms and DRL-based load balancing scheduling algorithms.

Traditional load balancing algorithms usually apply rule-based and heuristic methods for scheduling. Common methods include round-robin scheduling, random scheduling, Min-Min, Max-Min [17], [27]. These algorithms have been applied in practical production, and some studies have proposed improved methods based on them. Anselmi [28] combines Round Robin and size-interval task assignment (SITA) to design a scalable load balancing framework that can effectively balance the load in a distributed system. Xin et al. [29] proposed a method using a weighted random scheduling strategy to minimize the high load of devices by considering information related to the scheduling environment. Devaraj et al. [30] propose a load balancing algorithm that combines firefly and improved multi-objective particle swarm optimization to achieve effective load averaging in the cloud computing environment and achieve improvement in other metrics. Priya et al. [31] develop a scheduling method based on Fuzzy-based Multidimensional Resource Scheduling and Queuing Network in cloud data centers. The method can improve the load balancing and the average success rate. Yu et al. [32] propose three batch task management policies including a load balancing mechanism which can significantly improve the average resource utilization. In general, these research works to improve the load balancing performance and quality of service by introducing system-related jobs load or resource utilization information in specific scenarios and constructing corresponding rules. Moreover, it is difficult to respond to the dynamic and complex challenges brought by user requests and system resource changes in a timely and effective manner. To deal with the problems of traditional methods, DRL has been utilized for addressing dynamic and complex challenges of load balancing scheduling in cloud computing.

Therefore, many research works have been devoted to efficiently addressing the load balancing scheduling problem in cloud computing using DRL-based algorithms. Tong et al. [8] propose a DQN-based dynamic load balancing algorithm under the SLA constraints, which performed best in balancing load and reducing the task rejection rate compared to the baseline algorithm. Yi et al. [33] utilize an LSTM network to predict the system state of a cloud computing data center and then apply the prediction model to train a DQN-based compute-intensive job allocator. The evaluation results demonstrate that the algorithm can improve the overall system performance. Tong et al. [23] propose a DQN-based algorithm to manage tasks in dynamic online task scheduling in an attempt to address a bi-objective optimization problem including load balancing, and the experimental results demonstrate the superiority of their proposed method. Staffolani et al. [34] propose an adaptive workload allocation solution based on the Double DQN, Reinforcement Learning based Queues (RLQ), for distributed task queues. The evaluation results show that RLQ can adapt to varying workload frequencies without manual intervention.

In addition to these works utilizing value-based DRL algorithms, Mao et al. [35] propose a scheduling algorithm Decima that uses policy-based reinforcement learning to learn specific workloads. It introduces a novel representation method for job dependency graphs and designs an scalable RL model. Additionally, a RL training method is invented for handling continuous stochastic job arrivals. Bao et al. [3] present a REINFORCE-based scheduler for machine learning cluster. The scheduler addresses performance interference and minimizes average job completion time by efficient job placement. Recent literature has studied DRL algorithms based on the actor-critic framework, which can better combine value-based and policy-based DRL algorithms. Zhang et al. [5] propose a actor-critic based batch jobs scheduler, RLScheduler, which can automatically learn high-quality scheduling policies independently of expert knowledge. Zhang et al. [6] add a inspector based on actor-critic framework, namely SchedInspector, on top of the basic batch jobs scheduler. SchedInspector can intelligently enhance various batch jobs scheduling strategies. Yang et al. [36] propose an enhanced greedy optimization algorithm based on actor-critic framework considering scheduling problems in batch tasks, which improved the system gain by about 10% to 30%. A summary of the comparison of main related works about scheduling algorithms is shown in Table I.

Most of these works rely on the value-based DRL method for load balancing scheduling, but the ability to capture random changes in dynamic and complex system environments is insufficient. To address these challenges, we propose a multi-objective load balancing scheduling algorithm based on Distributional Reinforcement Learning. In contrast to value-based scheduling algorithms, the proposed algorithm starts from the value distribution of cumulative returns, which can well capture the stochasticity in the scheduling process caused by jobs arrival and resource usage, and better adapt to the dynamic complex environment.

III. System Model

In this section, we design a general batch jobs load balancing scheduling system model based on cloud computing. The continuously arriving batch jobs are reasonably allocated to clusters

TABLE I
SUMMARY OF THE COMPARISON OF MAIN RELATED WORKS ABOUT SCHEDULING ALGORITHMS.

References	Objective	Algorithm	Summarize
[8]	Reducing the load imbalance and task rejection rate	Deep Q Network (DQN)	A DQN-based dynamic load balancing algorithm under the SLA constraints
[33]	Reducing computing power consumption and processor temperature	Deep Q Network (DQN)	A DQN-based compute-intensive job allocator
[23]	Minimizing load balance and makespan in task scheduling	Deep Q Network (DQN)	A DQN-based algorithm to manage tasks in dynamic online task scheduling
[34]	Reducing the execution cost, the execution time and the waiting time	Double Deep Q Network (DDQN)	An adaptive workload allocation solution based on the Double DQN for distributed task queues
[35]	Minimizing average job completion time and masespan	Graph neural network, policy gradient REINFORCE algorithm	A scheduling algorithm Decima that uses policy based reinforcement learning to learn specific workloads
[3]	Minimizing interference and average job completion time	REINFORCE algorithm	An scheduler addressing performance interference and minimizing average job completion time by efficient job placement
[5]	Minimizing average waiting time, average response/turnaround time and average bounded slowdown, maximizing resource utilization	Proximal Policy Optimization (PPO)	An automated HPC batch job scheduler based on actor-critic reinforcement learning model
[6]	mMinimizing average bounded job slowdown, average waiting time, maximal bounded job slowdown	Proximal Policy Optimization (PPO)	ScheduleInspector can intelligently enhance various batch jobs scheduling strategies
[36]	Maximizing the total system gain defined as value minus costs	DRL based on the actor-critic framework, greedy algorithm	An enhanced greedy optimization algorithm based on DRL considering scheduling problems in batch tasks
Our work	Minimizing load balancing, success rate of instance creation (SRIC) and average completion time of the tasks (ACTT)	Distributional Reinforcement Learning	An enhanced greedy optimization algorithm based on DRL considering scheduling problems in batch tasks

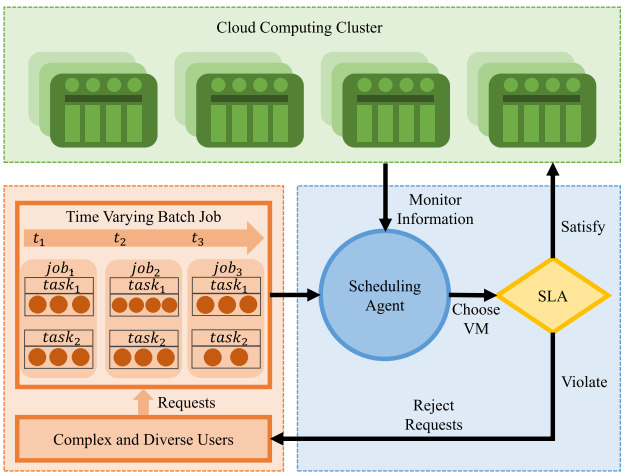


Fig. 1. System Model of Batch Jobs Load Balancing in cloud computing.

with dynamically changing resources by the scheduler. The goal is to improve the load balancing of cluster and user quality of service. Fig. 1 shows the framework diagram of the system model. The user's service requests arrive in the form of batch jobs. Load balancing scheduling includes an agent that selects the appropriate virtual machines and load balancing computing component. Cloud computing clustering includes a cluster of virtual machines and a monitoring component for collecting information from cluster resource utilization. The proposed scheduler is located in the platform layer of the cloud computing architecture, offering scheduling services for service requests submitted by users to the cloud computing cluster system. It automatically allocates jobs to computing nodes within the infrastructure layer based on their resource requirements and resource constraints.

In the proposed system model, service requests submitted by users arrive in the form of batch jobs $\{job_i, job_2, \ldots, job_N\}$ in chronological order. A job consists of multiple parallel tasks of different types, denoted as $job_j = \{task_{j,1}, task_{j,2}, task_{j,3}, \ldots, task_{j,O}\}$, where O is the number of tasks contained in job job_j. Instances of the same task require the same CPU and memory resources at the time of creation and can be scattered across different virtual machines for creation. The execution duration after successful creation is also the same. The most important property of a task is the number of instances that need to be created. In the proposed system, instances are the atomic execution units, so the task with a larger number of instances requires more scheduling times than the task with a smaller number of instances under the same conditions.

After the batch jobs arrive, the scheduling agent generates the load balancing scheduling strategy and outputs the scheduling signal according to the resource requirements of the current instance to be created and the resource utilization information of the current cluster. The scheduling signal is utilized to select the appropriate virtual machine in the cluster for instance creation. For scheduling signal, the system first determines whether the currently selected virtual machine meets the resource requirements of the current instance creation based on the SLA constraints. If the SLA is violated, the instance creation fails. The user is returned with a signal indicating the rejection of the request, and the agent is also returned with the information about the instance creation failure. If the SLA is satisfied, the scheduling signal is input into the cloud computing cluster, and the cluster selects the corresponding virtual machine to create an instance. The load balancing calculating component will calculate the current degree of load balancing based on the received cluster resource utilization information. Then the

current degree of load balancing is returned to the scheduling agent. Under the SLA constraints, the agent needs to meet (1) shown as follows to successfully schedule the instance.

$$InsSuccess = \begin{cases} insCpu_p \le VM_i^{cpuAvl} \\ insMem_p \le VM_i^{memAvl} \end{cases} \quad (1)$$

The above equation indicates that the CPU and memory allocated to the p th instance on the p th virtual machine is to be both less than or equal to the currently available CPU and memory resources on the virtual machine.

The main components of a cloud computing cluster are the virtual machines that create and run instances and the monitoring component that collects information about cluster resources. The cluster are represented as $VMs = \{VM_1, VM_2, \ldots, VM_M\}$. One of the most important attributes of the cluster is the CPU and memory resource capacity of the VMs. In this paper, the CPU and memory resource capacities differ between different types of VMs. Under the same conditions, virtual machines with larger resource capacity can create more instances than virtual machines with smaller resource capacity. The scheduling agent selects a virtual machine for the instance to be scheduled. If the SLA is satisfied by creating an instance on the virtual machine selected by the scheduling agent, the cloud computing cluster creates and runs the instance on the selected virtual machine. Then the load of the virtual machine increases, and the load balance of the cluster changes. Otherwise, the cloud computing cluster refuses to create the instance. After the instance is created and running, the cloud computing cluster returns the running result of the service request to the user. The monitoring component collects cluster resource utilization information and sends it to the load balancing calculating component for the current load balancing state of the cluster. At the same time, the cluster resource utilization information is also directly fed back to the agent for scheduling strategy making. The load changes for each virtual machine after an instance is created, and the essence of the load balancing is to make all virtual machines in the cluster have the same load.The load of VM_i is defined as follows.

$$lCpu_i = \frac{VM_i^{cpu} - VM_i^{cpuAvl}}{VM_i^{cpu}}, \quad (2)$$

$$lMem_i = \frac{VM_i^{mem} - VM_i^{memAvl}}{VM_i^{mem}}. \quad (3)$$

The average load of the cluster in terms of CPU and memory resources is defined as follows.

$$lCpu^{avg} = \frac{\sum_{i=1}^{M} lCpu_i}{M}, \quad (4)$$

$$lMem^{avg} = \frac{\sum_{i=1}^{M} lMem_i}{M}. \quad (5)$$

where $\sum_{i=1}^{M} lCpu_i$ and $\sum_{i=1}^{M} lMem_i$ are the sum of load of all virtual machines in the cluster. M is the number of virtual machines in the cluster. The degree of load balancing in the CPU and memory resource dimensions are defined as follows.

$$lbCpu = \sqrt{\frac{\sum_{i=1}^{M} (lCpu_i - lCpu^{avg})^2}{M}}, \quad (6)$$

TABLE II
NOTATIONS OF SYMBOLS IN SYSTEM MODEL

Notation	Definition
M	Total number of VMs in a cluster
N	Total number of jobs requested by users
O	Total number of tasks in the current job
P	Total number of instances in the current task
α	Index set of VMs, $\alpha = \{1, 2, \ldots, M\}$
β	Index set of jobs, $\beta = \{1, 2, \ldots, N\}$
γ	Index set of tasks in the job, $\gamma = \{1, 2, \ldots, O\}$
δ	Index set of instances in the task, $\delta = \{1, 2, \ldots, P\}$
VM_i^{cpu}	CPU resource limit of the VM_i, $i \in \alpha$
VM_i^{mem}	Memory resource limit of the VM_i, $i \in \alpha$
VM_i^{cpuAvl}	CPU resource available of the VM_i, $i \in \alpha$
VM_i^{memAvl}	Memory resource available of the VM_i, $i \in \alpha$
job_j	The j-th job in batch jobs, $j \in \beta$
$task_{j,o}$	The o-th task in j-th job, $o \in \gamma$
$insNum_{j,o}$	The number of instances to be created in $task_{j,o}$
$insSucc_{j,o}$	The number of instances created in $task_{j,o}$
$insCpu_p$	CPU resource demand of an instance, $p \in \delta$
$insMem_p$	Memory resource demand of an instance, $p \in \delta$

$$lbMem = \sqrt{\frac{\sum_{i=1}^{M} (lMem_i - lMem^{avg})^2}{M}}. \quad (7)$$

The definition of the degree of load balancing is obtained by calculating the standard deviation of the load of all virtual machines in the cluster. A larger degree of load balancing indicates a more unbalanced cluster. On the contrary, a smaller degree of load balancing indicates a more balanced cluster.

In the cloud computing system model designed in this paper, the load balancing and SRIC is utilized as a metric to evaluate the quality of service (QoS) provided by the cloud computing platform. The time when the users submit batch jobs is random, so the scheduling agent processes batch jobs without any prior knowledge about the arrival time of batch jobs. The scheduling agent considers the available resource capacity of the cluster and the resource request of the current instance and selects the best virtual machine for creating the current instance. In this paper, the goal of scheduling is to improve the load balance of clusters in multiple resource dimensionsthe and success rate of instance creation. The notations of symbols used in the proposed system model are shown in Table II.

We consider addressing the load balancing scheduling problem for cloud computing clusters under SLA constraints. According to the definition of load balancing, the goal of the scheduling agent is to make the load of virtual machines as close as possible to the average load of the cluster. Considering that there are two resource dimensions of CPU and memory in the system model we designed, the load balancing optimization objective of the system is expressed as the sum of the absolute values of the difference between the load of the virtual machines and the average load of the cluster on the dimensions of CPU and memory. The load balancing optimization objective f_1 is defined in (8).

$$f_1 = \min(|lCpu_i - lCpu^{avg}| + |lMem_i - lMem^{avg}|). \quad (8)$$

In the system model, the scheduling agent also needs to consider another optimization objective to improve users quality of service as much as possible. The goal of agent is achieved by

improving the load balancing and SRIC in the cluster. Hence the optimization objective of maximizing SRIC is as shown in (9).

$$f_2 = \max\left(\frac{\sum_{j=1}^{N}\sum_{o=1}^{O} insSucc_{j,o}}{\sum_{j=1}^{N}\sum_{o=1}^{O} insNum_{j,o}}\right). \tag{9}$$

Combining the above two optimization objectives, the multi-objective optimization problem to be addressed in this paper is formalized as the following (10).

$$f = [f_1, f_2], \text{ s.t. Eq. (1).} \tag{10}$$

IV. Distributional RL-Based Load Balance Scheduling Algorithm

In this section, we introduce the Distributional Reinforcement Learning-based load balancing scheduling algorithm for batch jobs, which is utilized to optimize the load balancing of cloud computing clusters and QoS under SLA constraints. Firstly, the Distributional Reinforcement Learning model construction is introduced. Then, the design details of the algorithm are presented. Finally, the training process of the proposed algorithm is introduced by a simple example.

A. Distributional Reinforcement Learning Model Construction

DQN is a classical value-based method that uses expectations to model cumulative returns. However, the load balancing scheduling environment is dynamic and changeable. The DQN-based method only models the value of cumulative returns, ignoring the information contained in its distribution, and cannot adapt well to the changing cluster and job arrival. In order to capture the inherent stochasticity of the scheduling process caused by the stochasticity of cluster and job arrival, we utilize distributed reinforcement learning to model the distribution of cumulative returns, which can adaptively cope with the dynamic and variable load balancing scheduling environment. Distributional reinforcement learning takes the cumulative return as a random variable and preserves distribution information by modeling the distribution of cumulative returns, rather than only modeling its expectations.

In reinforcement learning, the agent maximizes its action value function $Q(s,a)$(Sutton & Barto, 1998). Bellman's equation defined in (11) describes the value of $Q(s,a)$ in the terms of the expected return of the random transition $(s,a) \to (s',a')$.

$$Q(s,a) = \mathbb{E}R(s,a) + \gamma \mathbb{E}Q(s',a'). \tag{11}$$

Distributional reinforcement learning aims to exceed the notion of value and argue on the side of a distributional perspective on reinforcement learning. In particular, the main research object of Distributional Reinforcement Learning is the random return $Z(s,a)$ with distribution property, and the expectation of $Z(s,a)$ is $Q(s,a)$. The random return $Z(s,a)$ can be described by a recursive equation, which is defined as

$$Z(s,a) := R(s,a) + \gamma Z(s',a'). \tag{12}$$

In the distributional Bellman equation of the scheduling environment, the distribution of $Z(s,a)$ is presented by the interaction of three random variables: the reward $R(s,a)$, the next state-action (s',a'), and its random return $Z(s',a')$. This quantity is defined as the value distribution.

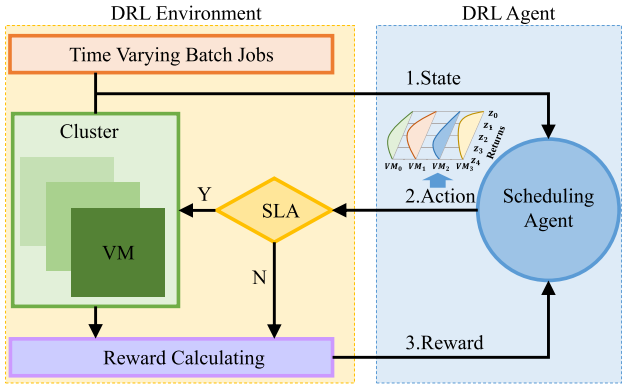


Fig. 2. Distributional reinforcement learning model for load balancing scheduling.

In this paper, we consider the load balancing scheduler of the cloud computing platform as a reinforcement learning agent. The cloud computing platform is regarded as an environment interacting with the agent. Each time an instance is scheduled, the agent obtains the current state of the scheduling environment. The environment state includes the resource request information of the instance to be scheduled and the load change information of the cluster. Then the agent makes an action according to the scheduling strategy to select the virtual machine suitable for creating the current instance to be scheduled.

In contrast to value-based DRL agents, when the agent makes decision actions based on distributional RL, the distribution $Z(s,a)$ of $Q(s,a)$ calculated based on action a and state s is modeled. For the determined action a_0 of selecting virtual machine VM_0, its return value is modeled as a distribution, representing the probabilities of all potential returns under the current (s,a_0). After the action is executed, the agent obtains the reward from the scheduling environment which is calculated according to the SLA constraints and the change of load balancing. In the proposed DRL model, an episode refers to the entire process from the arrival of the first job to the completion of scheduling by the agent for the last job. The interaction process between the scheduling environment and the DRL agent is shown in Fig. 2.

When Distributional Reinforcement Learning is utilized to solve the multi-objective optimization problem, the design of state space, action space, and reward function directly affects the effectiveness of the algorithm. For the batch jobs load balancing scheduling problem, state space, action space, and reward function in the proposed DRL model are defined as follows.

State space: The current state of the scheduling environment can be represented by a one-dimensional vector consisting of two parts. The first part of the vector is the cluster resource load information, which is defined in (13).

$$s_1 = [lCpu_1, lMem_1, \ldots, lCpu_N, lMem_N, lCpu^{avg}, lMem^{avg}], \tag{13}$$

where $lCpu_1, \ldots, lCpu_N$ indicates the current load in CPU dimension of all virtual machines in the cluster, $lMem_1, \ldots, lMem_N$ indicates the current load in memory dimension of all virtual machines in the cluster, $lCpu^{avg}$ and $lMem^{avg}$ indicates the average load of all virtual machines in the cluster. As the load of the cluster is large in magnitude, we normalize the load of machines to between 0 and 1 to improve the accuracy and effectiveness of the algorithm.

The second part of the vector is the resource requirements of a task instance of the current job, which is defined in (14).

$$s_2 = [taskType, insCpu_p, insMem_p, insNum_p] \tag{14}$$

where $taskType$ indicates the index of the current task, $insCpu_p$ and $insMem_p$ represents the CPU and memory requirements of an instance of the current task. As all instances of a task have the same resource requirement, the only other information required is the total number of instances to be created. The total number of instances is indicated as $insNum_p$. Hence the total current state of the scheduling environment can be represented as follows.

$$s = s_1 + s_2 \tag{15}$$

Action space: The scheduling agent make an action to select the appropriate virtual machine for instance creation. If there are M machines in the cluster, there are M discrete actions. Actions 1 to M specify the index of the machines selected for instance creation of the current job task. The action space can be expressed in the following form, where a indicates the choice of the virtual machine.

$$A = \{a | a \in \{1, 2, \ldots, M\}\}. \tag{16}$$

Reward function: The reward function r is of vital importance for reinforcement learning model training. The scheduling agent judges the accuracy of the action according to the reward value of the environment feedback and plays a guiding role in the next action selection. In the proposed reinforcement learning model, positive and negative values are respectively used to represent positive and negative reward values. Firstly, we consider the SLA constrains. If SLA constraints are violated, the virtual machine selected by the scheduling agent cannot satisfy the creation requirements of current instance. The agent receives a negative reward, $r = -1$. Then, we consider the load balancing variations in the cluster to decide the value of reward. We respectively calculate the difference value between the average cluster load and the load of the virtual machine that creates the currently scheduled instance. The difference values of load in cpu and memory dimensions are denoted respectively by $dvCpu$ and $dvMem$, which are defined as follows.

$$dvCpu = lCpu^{avg} - lCpu_i \tag{17}$$

$$dvMem = lMem^{avg} - lMem_i \tag{18}$$

The goal of load balancing is to keep the load of all virtual machines as close as possible to the average load of the current cluster. If the difference value is greater than 0, This indicates the extent to which the load of the virtual machine selected by the agent exceeds the average load of the cluster. For agent, the action utilized to select such a virtual machine is not encouraged. Therefore, when the SLA constrains are satisfied, the definition formula of reward function is as follows.

$$r = \begin{cases} 1, & dvCPU, dvMem \geq 0 \\ \frac{(dvCpu + dvMem)}{2}, & \text{otherwise.} \end{cases} \tag{19}$$

B. Design Details of the Algorithm

The proposed algorithm utilizes DQN with quantile regression based on the constructed DRL model to achieve outstanding load balancing and QoS in cloud computing clusters. QR-DQN approximates the distribution at each state by considering fixed probabilities but variable locations. It takes uniform quantile weights denoted by $q_i = \frac{1}{L}$ for $i = 1, \ldots, L$. The approximation aims to estimate quantiles of the target value distribution which is called a quantile distribution. The space of quantile distributions for fixed L is represented as $\mathcal{Z}_Q$. The cumulative probabilities of the quantile distribution is denoted by $\tau_1, \ldots, \tau_L$, and $\tau_i = \frac{i}{L}$ for $i = 0, 1, \ldots, L$. Here, a quantile distribution $Z_\theta \in \mathcal{Z}_Q$ is defined as follows.

$$Z_\theta(s, a) := \frac{1}{L} \sum_{i=1}^{L} \delta_{\theta_i}(s, a) \tag{20}$$

where δ_z denotes a Dirac at $z \in \mathcal{R}$. Hence, the state-action pair (s, a) in the load balancing scheduling can be mapped to a uniform probability distribution by Z_θ. The uniform probability distribution is supported on $\theta_i(s, a)$.

The projection of an value distribution $Z \in \mathcal{Z}$ onto the quantile distribution space $\mathcal{Z}_Q$ can be quantified as follows.

$$\Pi_{W_1} Z = \underset{Z_\theta \in \mathcal{Z}_Q}{\arg\min} W_1(Z, Z_\theta) \tag{21}$$

where W_1 denotes 1-Wasserstein metric.

For a distribution Z, and a given quantile τ, the quantile regression loss is defined as follows.

$$\mathcal{L}_{QR}^{\tau}(\theta) := \mathbb{E}_{\hat{Z} \sim Z}[\rho_\tau(\hat{Z} - \theta)] \tag{22}$$

where quantile loss function $\rho_\tau(u) = u(\tau - \delta_{u<0}), \forall u \in \mathbb{R}$.

Furthermore, in order to minimize values of $\{\theta_1, \ldots, \theta_L\}$ for $W_1(Z, Z_\theta)$, the loss defined as follows is given.

$$\sum_{i=1}^{L} \mathbb{E}_{\hat{Z} \sim Z}[\rho_{\hat{\tau}_i}(\hat{Z} - \theta_i)]$$

To solve the problem of lacking smoothness at zero point in the loss. The Huber loss is given as follows and $\mathcal{K}$ is a hyperparameter.

$$\mathcal{L}_{\mathcal{K}}(u) = \begin{cases} \frac{1}{2}u^2, & \text{if } |u| \leq \mathcal{K} \\ \mathcal{K}(|u| - \frac{1}{2}\mathcal{K}^2), & \text{otherwise.} \end{cases} \tag{23}$$

The quantile Huber loss can be seen as the asymmetric variant of the Huber loss and is formulated as follows.

$$\rho_\tau^{\mathcal{K}}(u) = |\tau - \delta_{\{u<0\}}| \mathcal{K}(u). \tag{24}$$

The deep neural network of the RL agent plays an important role in the proposed algorithms. To emphasize the innovation of learning value distribution in the proposed algorithm, the

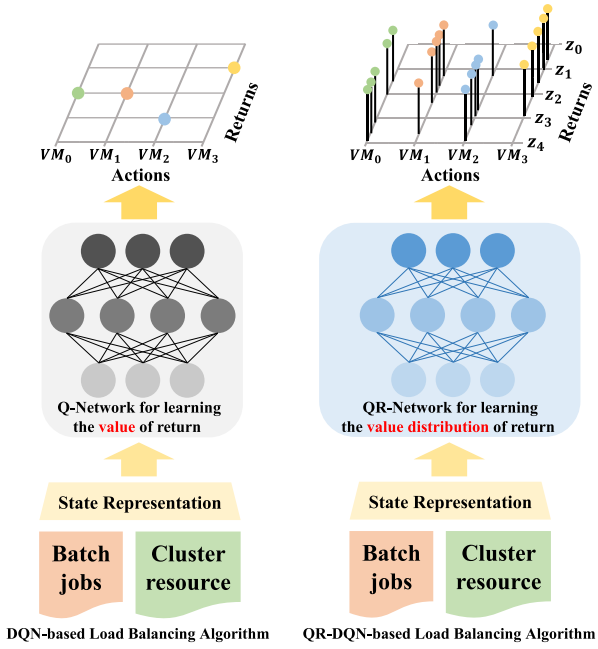


Fig. 3. Training example of the proposed scheduling agent.

Algorithm 1: The Distributional RL-Based Load Balancing Scheduling in Cloud Computing.

Input: Batch jobs $\{job_i, job_2, \ldots, job_N\}$ to be scheduled.
Output: VM_i used to create the scheduled instances.
for episode $e = 1$ to $MaxEpisodes$ **do**
  Initialize parameters, the agent observe the initial state s_0 from the cloud computing environment;
  for scheduling step $t = 1$ to $MaxSteps$ **do**
    Select an action a_t to schedule current instance of batch jobs;
    Execute action a_t in the environment, receive reward r_t and next state s_{t+1};
    $Q(s_{t+1}, a_{t+1}) := \sum_{j=1}^{Tau} q_j \theta_j(s_{t+1}, a_{t+1})$
    $a^* = \arg\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$
    $\mathcal{T}\theta_j = r + \gamma\theta_j(s_{t+1}, a^*)$
    Calculate Huber loss $Loss = \frac{1}{Tau}\sum_{i=1}^{Tau}\sum_{j=1}^{Tau}[\rho_\tau^1(\mathcal{T}\theta_j - \theta_i(s_t, a_t))]$
    Perform a gradient descent step on $Loss$ with respect to the network parameters θ
  end
end

difference between QR-Netwrok using quantile regression to learn value distribution and general Q-network based on value is shown in Fig. 3. The input of QR-Network is current environment state which represents batch jobs and cluster resources. The structure of QR-Network is a 3-layer multiple layer perceptron network. It's worth noting that the Q-Network's output is the value of the value function corresponding to each action. However, the QR-Network's output is the q-quantile of the value distribution of the value function corresponding to each action.

The q-quantile of the value distribution serves two purpose: 1) When optimizing QR-Network, optimizing the Huber loss function shown in (23) is equivalent to making the value of u approach to a specific value of a quantile τ. We employ the QR-Network to predict a series of u values, with each u value corresponding to a fixed quantile τ. Quantile regression is then applied to optimize the network for each u value. Where τ is a hyper-parameter; 2) When making action decisions, the q-quantiles are evenly divided on the cumulative distribution function, with each value having the same weight. Therefore, the expectation of the value distribution, which is the Q value, is the average of the q-quantiles.

The proposed dynamic load balancing scheduling algorithm based on Distributional Reinforcement Learning is concluded in Algorithm 1. We further analyze Algorithm 1 in terms of time and Spatial complexity.

Time Complexity Analysis: Agent training stage: the time complexity mainly depends on the training episodes E of the neural network and the number of scheduling steps S in each episode. The number of parameters of the neural network is $O(L \times n)$, where L denotes the number of layers and n denotes the number of neurons in each layer. In addition, when updating the neural network, the number of batch training samples B should also be considered. The time complexity is $O(E \times S \times L \times n \times B)$.

Spatial Complexity Analysis: 1) State observation stage: The spatial complexity of obtaining the cluster resource state is $O(M)$, and the spatial complexity of obtaining the current instance to be scheduled is $O(P)$; 2) Agent training stage: The spatial complexity of the neural network depends on the number of parameters of the neural network, which is $O(L \times n)$. The spatial complexity of the proposed algorithm is $O(M + P + L \times n)$, which mainly depends on the complexity of neural network training.

C. Training Example

A training example of the state, action and reward of the proposed scheduling agent is shown in Fig. 4. In the scheduling scenario of this example, the cluster has two machines with the following specifications: $VM_1 : \{cpu = 4, memory = 8\}$, $VM_2 : \{cpu = 8, memory = 16\}$. The red squares indicate the CPU, while the blue squares indicate Memory. Two jobs arrive in chronological order, with the following specifications:

job_1: $\{jobID = 1, taskID = 1, insCpu = 2, insMem = 2, insNum = 2, duration = 50\}$, $\{jobID = 1, taskID = 2, insCpu = 2, insMem = 4, insNum = 2, duration = 100\}$.

job_2: $\{jobID = 2, taskID = 1, insCpu = 4, insMem = 6, insNum = 1, duration = 100\}$, $\{jobID = 2, taskID = 2, insCpu = 4, insMem = 8, insNum = 2, duration = 150\}$.

According to the definition of environmental state, the initial state vector is [0,0,0,0,0,0,1,2,2,2]. The agent first successfully create all instances for the two tasks of Job_1.

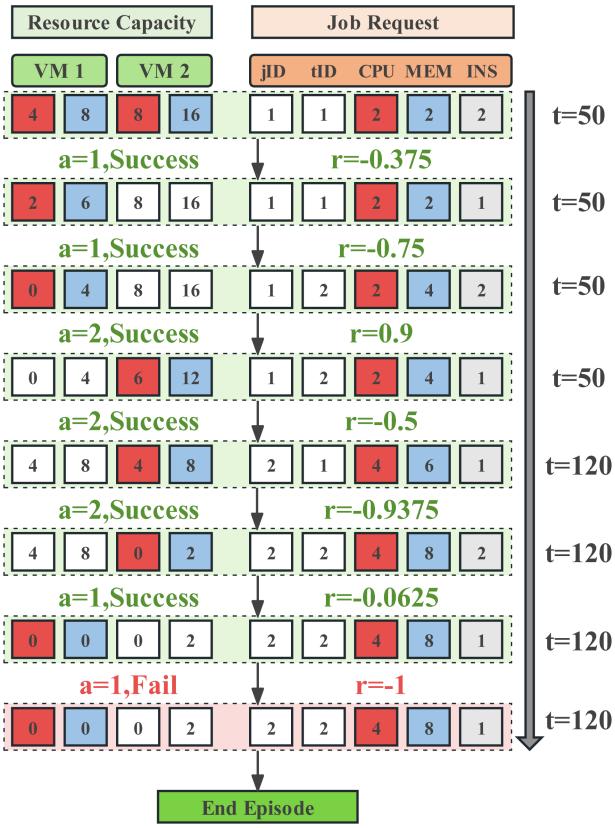


Fig. 4. Training example of the proposed scheduling agent.

After the scheduling of Job_1, the system state vector is [1,0.5,0.5,0.5,0.75,0.5,0,0,0,0]. At the time of $T = 120$, $task_1$ of Job_1 has finished running, so the occupied resources are released. The current state vector is [0,0,0.5,0.5,0.75,0.5,1,4,6,1]. Then the agent successfully create all instances for the first task of Job_2. When creating the second instance of the second task, the resources of the machine selected are limited and cannot satisfy the SLA, so this action is invalid. After all jobs are scheduled, the current episode finishes and the system state vector is [1,0.5,0.5,0.625,0.75,0.5625,0,0,0,0], the SRIC is 85.71%.

V. Experimental Analysis

In this section, we first introduce experimental setup which include cluster resource profiles, compared baselines, job arrival patterns, and evaluation patterns. Then, the convergence comparison of DRL algorithms is presented. Finally, experiments involving different numbers of jobs are carried out for real user workload tracing datasets.

A. Experimental Setup

In this paper, we run all experiments in the Python 3.6 environment and implement all algorithms by using PyTorch 1.5.1.

Cluster Resource Profiles: In the cloud computing environment for training, we simulate a cluster with 9 heterogeneous VMs. As shown in Table III, we have selected 3 types of VMs, and each type of VMs has different resource configuration information.

TABLE III
Three Types of Virtual Machines

Types	CPU cores	Memory capacity	VM numbers
1	8	16	3
2	16	32	3
3	32	64	3

Baselines: To demonstrate the advantages of the proposed algorithm for load balancing scheduling, we conduct comparative experiments. Four different algorithms are evaluated for comparison, including two improved DQN-based algorithms and two classical algorithms. The details are listed as follows.

- *Round Robin.* Instances are allocated fairly to each VM in a circular order. RR treats each VM in a balanced manner, regardless of the load of current cluster.
- *Random algorithm.* Instances are allocated to each VM by random order. The algorithm is suitable for simple industrial scenarios for its features of easily implementing
- *Double DQN-based algorithm.* Double DQN utilizes a double Q network to improve the DQN algorithm, which can alleviate the problem of high bias estimation of DQN [37].
- *Dueling DQN-based algorithm.* In Dueling DQN, the dueling network is proposed to improve the DQN algorithm, increase the state-dependent action advantage function to decouple the value estimation of DQN [38]. This improvement has achieved significant performance improvement.
- *REINFORCE-based algorithm.* REINFORCE is policy-based DRL algorithm which has the capability to directly learn stochastic policies. REINFORCE-based algorithm directly output actions with probability distribution is effective in handling large action spaces in cloud [3].
- *PPO-based algorithm.* PPO is biult based on actor-critic framework which combines the value-based and policy-based DRL algorithms. PPO-based algorithm can efficiently adapt to the changes in cloud computing [5], [6].

Job arrival patterns: The experiments are conducted based on Alibaba cluster trace v2018 and v2020. The cluster trace v2018 provides real production workload traces from about 4000 machines and 8-day running of offline computing tasks. We choose batch_task.csv from trace v2018 which includes information about tasks in the batch jobs. Compared to v2018, the cluster trace v2020 provides the more recent workload traces collected from the Alibaba Platform for Artificial Intelligence, spanning two months. In trace 2020, we choose pai_task_table.csv including the resource request information of tasks in the batch jobs. Additionally, we file out the DAG jobs and ensure that the time sequence of job arrival is not changed. Finally, we separately construct two job arrival patterns based on two different workload traces as shown below.

Job arrival patterns constructed based on Alibaba cluster trace v2018:

- *Normal pattern.* 1000 jobs, maximum number of instances per task: 10, maximum duration of each instance: 1000.

- *Intensive pattern.* 2000 jobs, maximum number of instances per task: 20, maximum duration of each instance: 500.

Job arrival patterns constructed based on Alibaba cluster trace v2020:

- *Normal pattern.* 2000 jobs, maximum number of instances per task: 5, maximum duration of each instance: 500.
- *Intensive pattern.* 4000 jobs, maximum number of instances per task: 10, maximum duration of each instance: 250.

Furthermore, for the trace v2018, we respectively select the next 1000 and 2000 jobs arriving in the normal and intensive patterns as the workloads for evaluation. Similarly, for the trace v2020, we select the next 2000 and 4000 jobs for evaluation. The jobs configuration information remains unchanged.

Evaluation metrics: The following three metrics are used as the basis of evaluating the comparison algorithms: degree of load balancing, success rate of instance creation and average completion time of the tasks. Further, the degree of load balancing in the scheduling process includes the cumulative load balancing and real-time load balancing according to the scheduling step.

1) DCLB (degree of cumulative load balancing): The degree of cumulative load balancing is the total degree of load balancing of all scheduling steps in the scheduling process. A smaller DCLB indicates better performance of the algorithm to balance the cluster load. After each instance scheduling, we collect the load of the current cluster and calculate the DCLB. At the scheduling step s, the degree of load balancing in cpu and memory dimensions $lbCpu_s$ and $lbMem_s$ are calculated respectively by (6) and (7). Therefore, DCLB is mathematically formulated as follows.

$$DCLB^{cpu} = \sum_{s=1}^{S} lbCpu_s, \tag{25}$$

$$DCLB^{mem} = \sum_{s=1}^{S} lbMem_s, \tag{26}$$

where S indicates the total number of scheduling steps which can be calculated as (27).

$$S = \sum_{j=1}^{N} \sum_{o=1}^{O} insNum_{j,o} \tag{27}$$

2) DRLB (degree of real-time load balancing): The degree of real-time load balancing indicates the degree of load balancing in the cluster at the current scheduling step. The sum of DRLB during the scheduling process is DCLB. Compared to DCLB, the change of DRLB during the scheduling process can dynamically represent the change of the algorithm's ability to balance the load in different scheduling steps.

After each scheduling step, the DRLB of the seven algorithms are ordered from smallest to largest, and the cumulative quantity of each algorithm at each rank is finally counted. A larger quantity of high ranks indicates a better performance of the algorithm in terms of load balancing.

3) SRIC (success rate of instance creation): When the batch jobs arrive at the cloud computing cluster, the cluster determines whether the task instance of the job can be successfully created according to SLA constraints. If SLA constraints are violated, the instance creation fails. The success rate of instance creation is the ratio of the number of successfully created instances to the total number of instances. The SRIC is defined in (28). As the success rate of creation increases, the QoS of cloud computing systems improves.

$$SRIC = \frac{\sum_{j=1}^{N} \sum_{o=1}^{O} insSucc_{j,o}}{\sum_{j=1}^{N} \sum_{o=1}^{O} insNum_{j,o}} \tag{28}$$

4) ACTT (average completion time of the tasks): In order to further explore the impact of the algorithms, we use the average completion time of the tasks to evaluate the performance of algorithms. The ACTT is defined in (29).

$$ACTT = \frac{Total\ completion\ time\ of\ the\ tasks}{Number\ of\ the\ tasks} \tag{29}$$

TABLE IV
KEY TRAINING AND EVALUATION PARAMETER VALUES FOR NORMAL AND INTENSIVE JOBS ARRIVAL PATTERNS

Parameter	Value
Batch Size	128
Training Episodes	10000
Evaluation Episodes	1
Learning Rate	0.001
Discount Factor	0.99
Initial Epsilon	1 (Normal) 0.9 (Intensive)
Epsilon Decay	0.995
Minimum Epsilon	0.1 (Normal) 0.05 (Intensive)
Hyper-Parameter $\mathcal{K}$	1
Number of Distribution Locations	51

B. Convergence of DRL Algorithms

Fig. 5 represents the convergence of the DRL algorithms training on Alibaba cluster trace v2018. We have trained the DRL agents for 10000 episodes for both the normal and intensive job arrival patterns. We calculate the average rewards every 10 episodes. For the normal and intensive job arrival pattern, we have both trained the agents with the fixed and different parameters. The key parameter values are shown in Table IV.

Fig. 5(a) and (b) represent average rewards accumulated by the DRL agent including DRL-based algorithms in training for the normal and intensive patterns, respectively. The average rewards consist of the immediate reward returned after each instance scheduling step during the scheduling process. Therefore, higher average rewards accumulated indicate that the DRL agent has learned a better policy to optimize the final objectives. In the initial stage of training, the DRL agents explore the state space randomly, and the curve representing the average reward gradually rises. Whether for the normal or intensive job arrival pattern, the proposed algorithm always achieves a smooth convergence, and the average rewards are higher than the baseline DRL algorithms. In addition, it's obvious that the convergence curves of baseline DRL algorithms fluctuate more violently than the proposed algorithm in Fig. 5(b). For intensive job arrival pattern, the value-based DRL algorithms do not have sufficient ability to capture the increased stochasticity in the process of scheduling. However, the proposed algorithm models

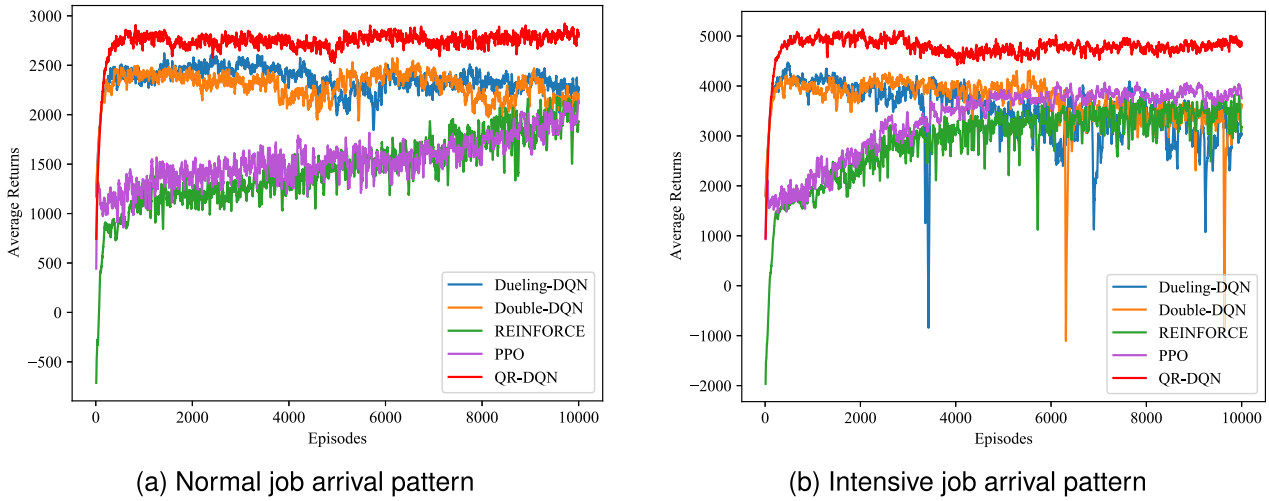


Fig. 5. Convergence of the DRL algorithms on Alibaba cluster trace v2018.

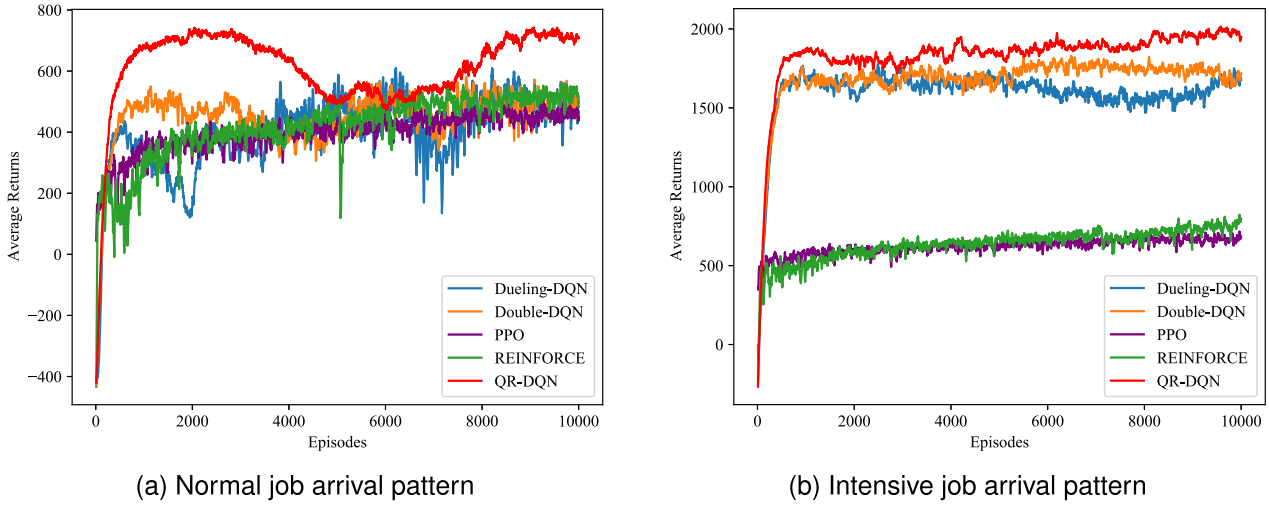


Fig. 6. Convergence of the DRL algorithms on Alibaba cluster trace v2020.

the cumulative returns during the scheduling process from a distribution perspective. The results verify that the proposed algorithm has strong adaptability to the time-varying batch jobs and dynamic cluster resources. The reasons for the instability of the compared algorithms in the context of load balancing include an increase in the number and types of jobs and faster changes in cluster resource states due to shorter job runtimes. This results in a larger state space and increased randomness in state representations, making it more challenging to fit value functions using neural networks. QR-DQN models the distribution of cumulative rewards, allowing for a more precise and comprehensive characterization of cumulative rewards. As a result, the training process becomes more stable.

The average rewards of DRL algorithms on Alibaba cluster trace v2020 are shown in Fig. 6. We notice that the convergence speed of REINFORCE and PPO is significantly slower than the proposed algorithm. Specifically, as shown in Fig. 6(b), the average rewards of REINFORCE and PPO remain around 500 after convergence, indicating that the algorithms have fallen into local optima during the training process. This is because policy-based methods can only use the samples obtained by interacting with the environment under the current policy when calculating the policy gradient, which leads to a decrease in sampling efficiency during training. Moreover, in the cloud computing environment studied in this paper, the stochasticity of the environment is relatively high. Sampling multiple trajectories under the same policy results in a wide range of different outcomes. Consequently, the high variance in estimated policy outcomes slows down the convergence speed and reduces training stability.

By comprehensively analyzing the convergence curves in Figs. 5 and 6, the proposed algorithm outperforms other algorithms in terms of convergence speed and final average

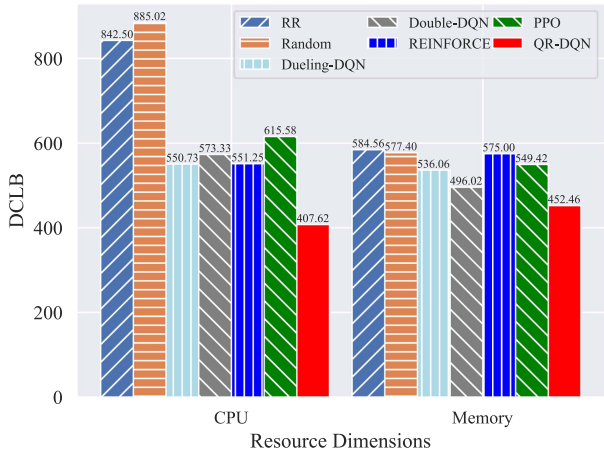
reward. However, in Fig. 6(a), the proposed algorithm shows a certain decrease after convergence, and then rises again to reach convergence. This is because the environmental rewards we set mainly consist of two parts: actions that successfully create task instances and actions that make load balancing better both receive positive rewards. The exploration of the latter action is obviously more challenging, and there is a possibility that the agent has the potential to avoid worse load balancing by creating failed instances. So, in the initial stage of training, the proposed algorithm can quickly achieve high rewards by correctly selecting the previous actions, and explore further actions to improve load balancing rewards by attempting to create failed instances, which are risky actions. During this process, the rewards obtained by the proposed algorithm may decrease to a certain extent. However, after completing the risky exploration, the rewards of the proposed algorithm will continue to rise and eventually converge.

C. Evaluation of Load Balancing

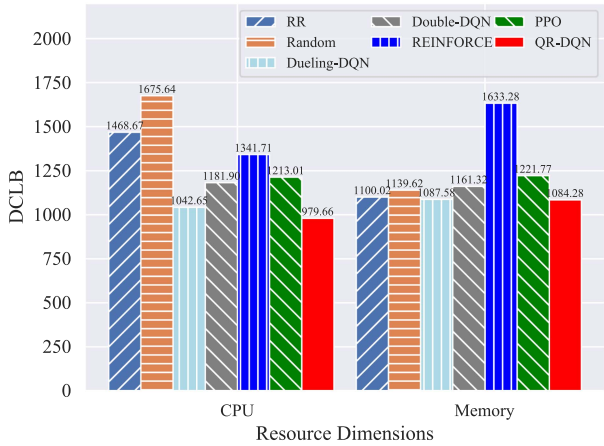
The evaluation includes cumulative load balancing and real-time load balancing. In this section, we first train the DRL agents 10000 episodes in normal and intensive job arrival patterns on the two datasets. Then we respectively select the next jobs from the real production workload traces according to the time sequence of the job arrival. Then we compare the performance of the proposed algorithm with the baseline algorithms. Fig. 7 shows the DCLB on CPU and memory dimensions in the two patterns on trace v2018. It can be seen that the proposed algorithm performs significantly better than the baseline algorithms.

Comparing Fig. 7(a) and (b), it can be observed that when the number of jobs increases to 2000, the load balancing performance gap between algorithms further narrows. The scheduling process takes a longer time and the state space is more complicated. It is worth noting that the process of the experiment in intensive arrival pattern is the same as that in normal arrival pattern. Further analysis of the results shown in Fig. 7(b) reveals that as job arrival patterns become more challenging, REINFORCE-based and PPO-based algorithms perform worse in load balancing in the memory dimension than in the CPU dimension. This also indicates that these two algorithms do not perform well in load balancing across multiple resource dimensions.

Through the results plotted in Fig. 8, the proposed algorithm still outperforms better than the baseline algorithms in load balancing, although the patterns of batch jobs arriving are more challenging. The results demonstrate that the excellence load balancing performance of the proposed algorithm in both job arrival patterns. Meanwhile, we can see in Fig. 8(a) that as the number of traces increases, the performance of REINFORCE-based and PPO-based algorithms is worse compared to other algorithms. This is consistent with our analysis in Section V-B, as the stochasticity in cloud computing environments increases, it is difficult to avoid the problems that arise when policy-based algorithms calculate policy gradients. Another point to note is that, as the results shown in Fig. 8(b), all algorithms exhibit significant differences in DCLB performance



(a) Normal job arrival pattern



(b) Intensive job arrival pattern

Fig. 7. CPU and Memory DCLB of the load balancing algorithms on Alibaba cluster trace v2018.

in CPU and memory dimensions, which differs greatly from the performance of algorithms in the other three patterns. This is because on trace v2018, the standard deviation of CPU changes for all jobs under intensive pattern is 170.205, and the standard deviation of memory changes is 75.477, with a ratio of 2.2255. However, on trace v2020, the standard deviation of CPU changes under intensive pattern is 298.08, and the standard deviation of memory changes is 11.842, with a ratio of 25.172. Under the intensive pattern on trace v2020, the number of jobs arriving is greater than other patterns, and the stochasticity of CPU changes is greater. In the case of jobs arriving sequentially, this means that there is greater uncertainty and unpredictability in the changes of jobs in the CPU dimension.

In order to further evaluate the performance of the proposed algorithm for load balancing scheduling, we record the DRLB of the evaluated algorithms after each scheduling. Then, the DRLB of the algorithms is ordered from smallest to largest, and the cumulative quantity of each algorithm at first rank is counted. The final statistical results are shown in Figs. 9 and 10. The comparison of results at first rank indicates that

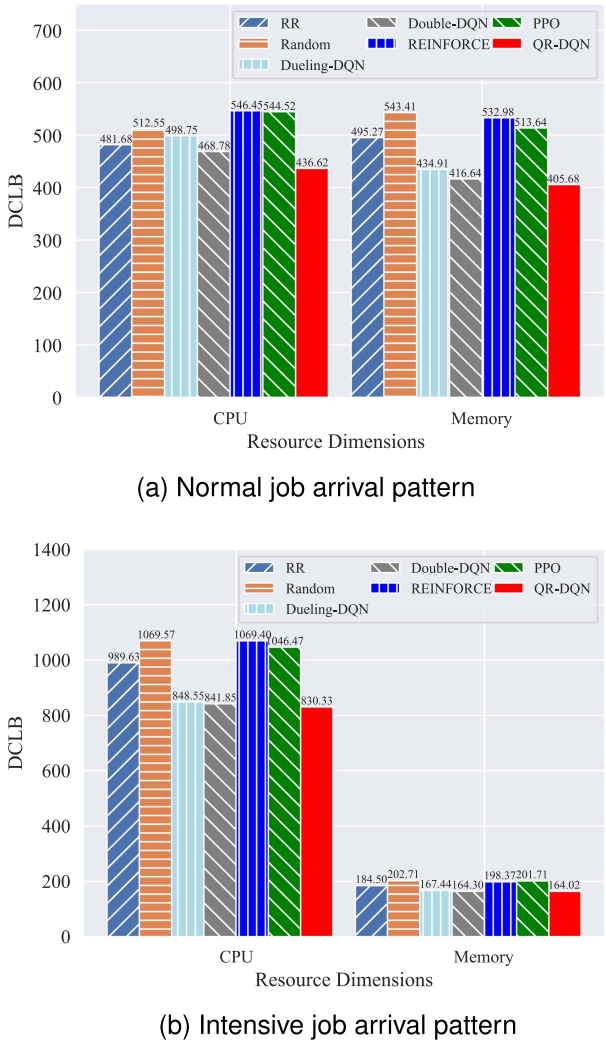


Fig. 8. CPU and Memory DCLB of the load balancing algorithms on Alibaba cluster trace v2020.

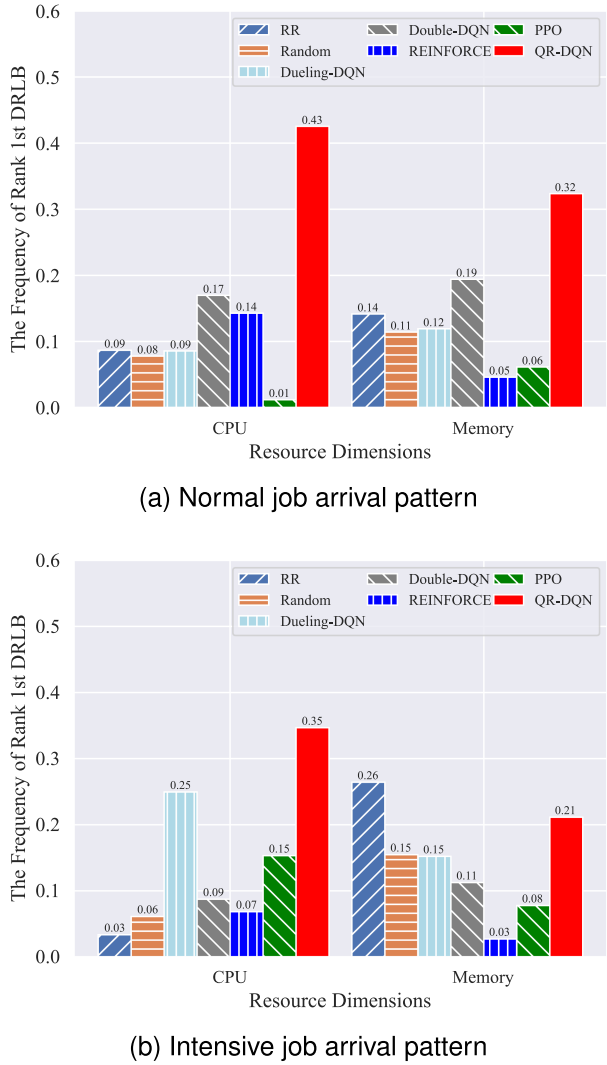


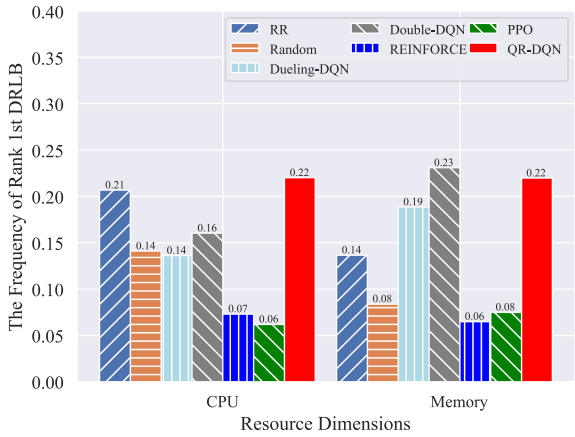
Fig. 9. CPU and Memory DRLB of the load balancing algorithms on Alibaba cluster trace v2018.

the proposed algorithm has more occurrences of the minimum DRLB in each step of the scheduling process than the baseline algorithms. Although the load balancing performance of the proposed algorithm is the best compared to baseline algorithms in both resource dimensions, it is obvious that the proposed algorithm has a better ability to balance the load in the CPU dimension. It is worth noting that in the memory dimension of Fig. 9(b), the performance of the proposed algorithm is worse than that of the RR. This is because in real cluster trace data, the changing characteristics of resource requests across different dimensions after combination are complex and difficult to predict. Moreover, this only represents the frequency comparison of the minimum DRLB, and in the overall load balancing comparison, the proposed algorithm is significantly superior to the RR algorithm. RR follows the principle of sequential rotation scheduling, which may exhibit ideal performance in a specific resource dimension in a specific pattern. However, the RR cannot demonstrate comprehensive and excellent performance.

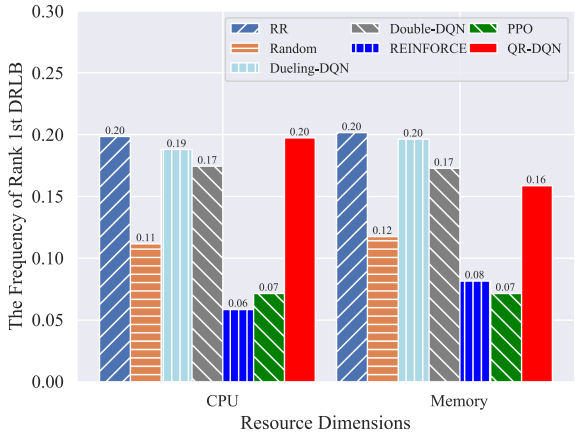
As shown in Fig. 10, in the intensive pattern, the performance of the proposed algorithm is consistent with that in the normal pattern. The results demonstrate that the proposed algorithm has sufficient capacity to cope with the arrival of more complex batch jobs. Comparing the experimental results of CPU and memory dimensions, it can be found that load balancing in the memory dimension is more challenging. Although the proposed algorithm still has a performance advantage compared to other algorithms, the advantage in the memory dimension is not significant compared to the advantage in the CPU dimension.

D. *Evaluation of SRIC*

Here, we analyze the performance of the algorithms in meeting SLA constraints in different batch job arrival patterns, in order to verify the impact of the proposed algorithm on improving QoS. They are evaluated by SRIC. Fig. 11 shows that the SRIC of all load balancing scheduling algorithms in normal and intensive jobs arrival patterns on Alibaba cluster trace v2018 and v2020. The proposed algorithm achieves the best performance on the both traces.

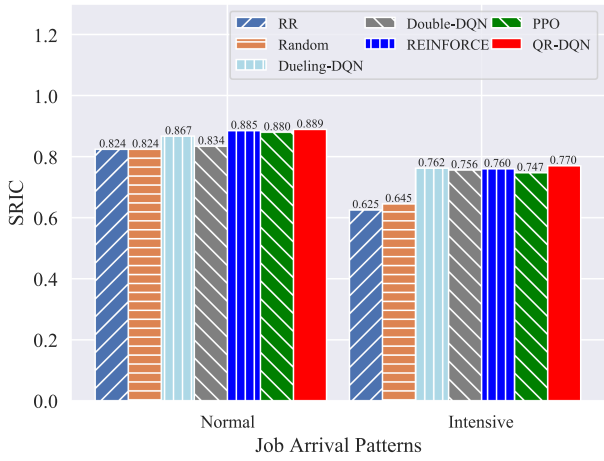


(a) Normal job arrival pattern

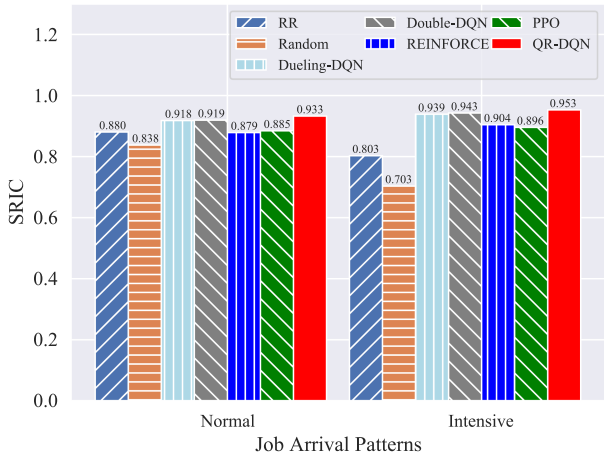


(b) Intensive job arrival pattern

Fig. 10. CPU and Memory DRLB of the load balancing algorithms on Alibaba cluster trace v2020.



(a) Alibaba cluster trace v2018



(b) Alibaba cluster trace v2020

Fig. 11. Success rate of instance creation in normal and intensive job arrival patterns on Alibaba cluster trace v2018 and v2020.

It is worth noting that the proposed algorithm and the two DQN-based algorithms perform significantly better than other algorithms in the intensive pattern. Moreover, it can be seen that the performance of REINFORCE-based and PPO-based algorithms is significantly worse than that of DQN-based algorithms as shown in Fig. 11(b). This is because the policy-based algorithms suffer from high variance when estimating the policy gradients. As the stochasticity of environment increase, sampling multiple trajectories under the same policy result in many different samples, which makes it difficult to calculate the policy gradient.

Another point worth further analysis is that on trace v2018 and v2020, the SRIC of DRL-based algorithms outperforms other algorithms, especially in both intensive patterns. This is because compared to load balancing, successfully creating task instances is easier. This also leads to a small gap in the SRIC of DRL-based algorithms, which can achieve ideal performance after training.

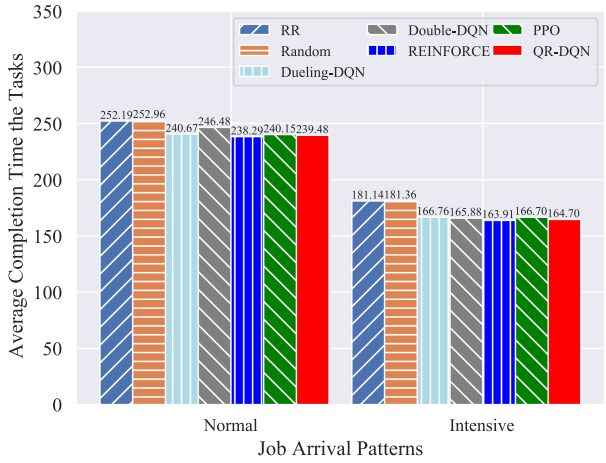
E. Evaluation of ACTT

It can be seen from Fig. 12(b) that the proposed algorithm performs best on trace v2020. However, in both patterns of trace v2018 as shown in Fig. 12(a), the performance of the proposed algorithm ranks second, both of which are inferior to the REINFORCE-based algorithm. This is because minimizing ACTT is easier to achieve good performance compared to load balancing. This also indicates that policy-based algorithms are easy to falling into local optima when dealing with minimizing load balancing during training.

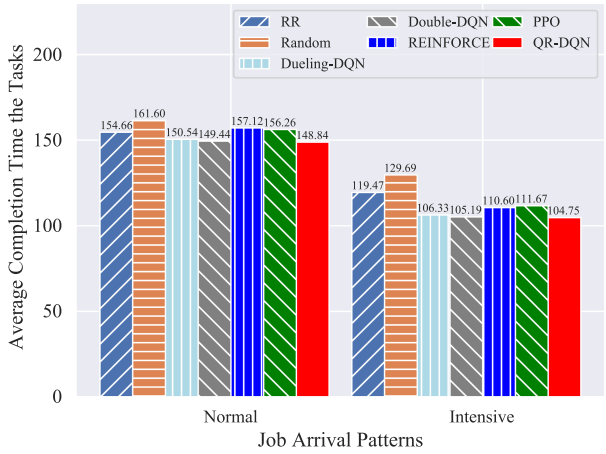
In addition, from the experimental results in Fig. 12(b), it can be seen that in the case of higher stochasticity of jobs arrival, the training difficulty of REINFORCE-based and PPO-based algorithms increases, and their performance in minimizing ACTT is significantly weaker compared to other DRL-based algorithms.

F. Computational Overhead and Throughput

We evaluate the computational overhead and throughput of different load balancing algorithms. The algorithms are executed on a computing server equipped with an Intel i5-9600 K CPU, 48 GB memory. The computations of the algorithms use the CPU. Tables V and VI shows the computational time and



(a) Alibaba cluster trace v2018



(b) Alibaba cluster trace v2020

Fig. 12. Average completion time of the tasks in normal and intensive job arrival patterns on Alibaba cluster trace v2018 and v2020.

TABLE V
COMPUTATIONAL OVERHEAD AND THROUGHPUT OF ALGORITHMS ON ALIBABA CLUSTER TRACE V2018

Algorithms	Computational time (s)	Throughput (tasks/s)
Round Robin	0.494 | 0.716	2023.187 | 2791.854
Random algorithm	0.599 | 0.957	1668.389 | 2090.076
Double DQN-based algorithm	0.606 | 1.025	1649.007 | 1951.190
Dueling DQN-based algorithm	0.675 | 1.048	1480.942 | 1908.984
REINFORCE-based algorithm	1.597 | 2.501	626.293 | 799.748
PPO-based algorithm	1.784 | 3,922	560.555 | 509.914
QR DQN-based algorithm	0.518 | 0.795	1931.964 | 2,517.184

TABLE VI
COMPUTATIONAL OVERHEAD AND THROUGHPUT OF ALGORITHMS ON ALIBABA CLUSTER TRACE V2020

Algorithms	Computational time (s)	Throughput (tasks/s)
Round Robin	0.255 | 0.561	7856.903 | 7133.426
Random algorithm	0.336 | 0.696	5957.176 | 5747.241
Double DQN-based algorithm	0.2842 | 0.756	7036.647 | 5290.625
Dueling DQN-based algorithm	0.2841 | 0.820	7038.983 | 4875.592
REINFORCE-based algorithm	2.647 | 5.805	755.547 | 689.051
PPO-based algorithm	3.198 | 6.180	625.474 | 647.253
QR DQN-based algorithm	0.625 | 0.993	3201.831 | 4028.158

throughput of different algorithms on the trace v2018 and v2020. Computational time indicates the total computational time of different algorithms in scheduling all jobs. In the Table V and VI, data before the vertical line represents the results under normal pattern, while data after the vertical line represents the results under intensive pattern.

We can see that PPO-based algorithm performs worst and the computational overhead and throughput of the proposed algorithm are acceptable as shown in Table V. However, on trace v2020, the performance of the proposed algorithm is not satisfactory. Apart from REINFORCE-based and PPO-based algorithms, which have high computational overhead due to policy gradient calculations, the proposed algorithm has larger computational overhead and lower throughput compared to other algorithms. This is because the proposed algorithm introduces additional overhead in modeling distributions, which limits its ability to handle tasks.

VI. CONCLUSION

In this paper, we first define the batch jobs load balancing scheduling problem in cloud computing as a deep reinforcement learning problem with dynamic system state and complex batch jobs. A prototype scheduling environment is developed to train DRL agents. Then, a multi-objective load balancing scheduling algorithm based on quantile regression deep Q network is proposed to improve the load balancing and QoS of the cloud computing platform. The proposed algorithm focuses more on the inherent stochasticity in the batch jobs load balancing scheduling process from the perspective of value distribution. Finally, extensive experiments are carried out by using real production workload traces from the Alibaba cluster trace v2018 and v2020. The results demonstrate that the proposed algorithm is effective in realizing dynamic adaptive load balancing scheduling and can better adapt to the changes in cloud computing compared to the baseline algorithms. Furthermore, when the batch jobs load increases, the proposed algorithm can still converge smoothly and obtain higher average rewards than the baseline algorithms.

In our future work, we plan to consider DAG (Directed Acyclic Graph) tasks with interdependencies, which will require us to account for task priority sequencing. We aim to extend the algorithms proposed in this paper to handle such scenarios. Specifically, we will need to preprocess DAG tasks, employing various sorting techniques that will significantly impact subsequent task scheduling. Therefore, we need to devise an efficient sorting algorithm that guarantees the performance of subsequent task scheduling algorithms. Furthermore, we will enhance the state space, action space, and reward functions of our reinforcement learning model. Additionally, we will address the dependencies among DAG tasks. Recent research has utilized graph neural networks to extract features related to these dependencies.

REFERENCES

[1] Z. Chen, J. Hu, G. Min, C. Luo, and T. El-Ghazawi, "Adaptive and efficient resource allocation in cloud datacenters using actor-critic deep reinforcement learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 8, pp. 1911–1923, Aug. 2022.

[2] C. Jiang et al., "Characterizing co-located workloads in Alibaba cloud datacenters," *IEEE Trans. Cloud Comput.*, vol. 10, no. 4, pp. 2381–2397, Oct.–Dec. 2022.

[3] Y. Bao, Y. Peng, and C. Wu, "Deep learning-based job placement in distributed machine learning clusters," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 505–513.

[4] Q. Liu and Z. Yu, "The elasticity and plasticity in semi-containerized co-locating cloud workload: A view from Alibaba trace," in *Proc. ACM Symp. Cloud Comput.*, 2018, pp. 347–360.

[5] D. Zhang, D. Dai, Y. He, F. S. Bao, and B. Xie, "Rlscheduler: An automated HPC batch job scheduler using reinforcement learning," in *Proc. SC20: Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2020, pp. 1–15.

[6] D. Zhang, D. Dai, and B. Xie, "Schedinspector: A batch job scheduling inspector using reinforcement learning," in *Proc. 31st Int. Symp. High- Perform. Parallel Distrib. Comput.*, 2022, pp. 97–109.

[7] Y. Fan et al., "Dras: Deep reinforcement learning for cluster scheduling in high performance computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 4903–4917, Dec. 2022.

[8] Z. Tong, X. Deng, H. Chen, and J. Mei, "DDMTS: A novel dynamic load balancing scheduling scheme under SLA constraints in cloud computing," *J. Parallel Distrib. Comput.*, vol. 149, pp. 138–148, 2021.

[9] S. S. Mondal, N. Sheoran, and S. Mitra, "Scheduling of time-varying workloads using reinforcement learning," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 9000–9008.

[10] J. Guo et al., "Who limits the resource efficiency of my datacenter: An analysis of Alibaba datacenter traces," in *Proc. Int. Symp. Qual. Serv.*, 2019, pp. 1–10.

[11] H. Tian, Y. Zheng, and W. Wang, "Characterizing and synthesizing task dependencies of data-parallel jobs in Alibaba cloud," in *Proc. ACM Symp. Cloud Comput.*, 2019, pp. 139–151.

[12] C. Mommessin et al., "Affinity-aware resource provisioning for long-running applications in shared clusters," *J. Parallel Distrib. Comput.*, vol. 177, pp. 1–16, 2023.

[13] W. Wei, H. Gu, K. Wang, J. Li, X. Zhang, and N. Wang, "Multi-dimensional resource allocation in distributed data centers using deep reinforcement learning," *IEEE Trans. Netw. Service Manage.*, vol. 20, no. 2, pp. 1817–1829, Jun. 2023.

[14] X. Zeng et al., "Detection of SLA violation for Big Data analytics applications in cloud," *IEEE Trans. Comput.*, vol. 70, no. 5, pp. 746–758, May 2021.

[15] M. T. Islam, S. Karunasekera, and R. Buyya, "Performance and cost-efficient spark job scheduling based on deep reinforcement learning in cloud computing environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 7, pp. 1695–1710, Jul. 2022.

[16] W. Guo, W. Tian, Y. Ye, L. Xu, and K. Wu, "Cloud resource scheduling with deep reinforcement learning and imitation learning," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3576–3586, Mar. 2021.

[17] P. Kumar and R. Kumar, "Issues and challenges of load balancing techniques in cloud computing: A survey," *ACM Comput. Surveys*, vol. 51, no. 6, pp. 1–35, Feb. 2019.

[18] D. Ardagna, M. Ciavotta, R. Lancellotti, and M. Guerriero, "A hierarchical receding horizon algorithm for QOS-driven control of multi-IAAS applications," *IEEE Trans. Cloud Comput.*, vol. 9, no. 2, pp. 418–434, Apr.–Jun. 2021.

[19] Z. Miao, P. Yong, Y. Mei, Y. Quanjun, and X. Xu, "A discrete PSO-based static load balancing algorithm for distributed simulations in a cloud environment," *Future Gener. Comput. Syst.*, vol. 115, pp. 497–516, 2021.

[20] M. Adhikari, T. Amgoth, and S.N. Srirama, "Multi-objective scheduling strategy for scientific workflows in cloud environment: A firefly-based approach," *Appl. Soft Comput.*, vol. 93, 2020, Art. no. 106411.

[21] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[22] Y. Tian, X. Li, H. Ma, X. Zhang, K. C. Tan, and Y. Jin, "Deep reinforcement learning based adaptive operator selection for evolutionary multi-objective optimization," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 7, no. 4, pp. 1051–1064, Aug. 2023.

[23] Z. Tong, H. Chen, X. Deng, K. Li, and K. Li, "A scheduling scheme in the cloud computing environment using deep Q-learning," *Inf. Sci.*, vol. 512, pp. 1170–1191, 2020.

[24] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[25] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 449–458.

[26] W. Dabney, M. Rowland, M. Bellemare, and R. Munos, "Distributional reinforcement learning with quantile regression," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 2892–2901.

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

[28] J. Anselmi, "Combining size-based load balancing with round-robin for scalable low latency," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 4, pp. 886–896, Apr. 2020.

[29] Y. Xin, Z.-Q. Xie, and J. Yang, "A load balance oriented cost efficient scheduling method for parallel tasks," *J. Netw. Comput. Appl.*, vol. 81, pp. 37–46, 2017.

[30] A. F. S. Devaraj, M. Elhoseny, S. Dhanasekaran, E. L. Lydia, and K. Shankar, "Hybridization of firefly and improved multi-objective particle swarm optimization algorithm for energy efficient load balancing in cloud computing environments," *J. Parallel Distrib. Comput.*, vol. 142, pp. 36–45, 2020.

[31] V. Priya, C. Sathiya Kumar, and R. Kannan, "Resource scheduling algorithm with load balancing for cloud service provisioning," *Appl. Soft Comput.*, vol. 76, pp. 416–424, 2019.

[32] J. Yu, W. Tong, P. Lv, and D. Feng, "Terms: Task management policies to achieve high performance for mixed workloads using surplus resources," *J. Parallel Distrib. Comput.*, vol. 170, pp. 74–85, 2022.

[33] D. Yi, X. Zhou, Y. Wen, and R. Tan, "Efficient compute-intensive job allocation in data centers via deep reinforcement learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1474–1485, Jun. 2020.

[34] A. Staffolani, V.-A. Darvariu, P. Bellavista, and M. Musolesi, "RLQ: Workload allocation with reinforcement learning in distributed queues," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 3, pp. 856–868, Mar. 2023.

[35] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proc. ACM Special Int. Group Data Commun.*, 2019, pp. 270–288.

[36] Y. Yang and H. Shen, "Deep reinforcement learning enhanced greedy optimization for online scheduling of batched tasks in cloud HPC systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 11, pp. 3003–3014, Nov. 2022.

[37] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.

[38] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 1995–2003.

Tiangang Li is currently working toward the PhD degree with the School of Computer, Wuhan University, China. His research interests include cloud computing, software engineering, and resource management and reinforcement learning.

Shi Ying is currently a professor with the School of Computer Science, Wuhan University, China. His main research interests include software engineering, intelligent operation and maintenance management, cloud computing and cloud service software, and artificial intelligence.

Yishi Zhao received the MSc degree with Distinction and the PhD degree in computing science from the Newcastle University in 2006 and 2011, respectively. He is currently a lecturer with the School of Computer Science, China University of Geosciences, Wuhan, China. His research interests include software performance engineering, intelligent decision-making, and applications of blockchain and digital twin.

Jianga Shang (Member, IEEE) received the PhD degree in computer system architecture from the Huazhong University of Science and Technology. He is currently a professor with the School of Computer Science and a senior researcher of the National Engineering Research Center for Geographic Information System, China University of Geosciences(CUG), Wuhan. He is also the director of Human-Cyber-Physical Software for Smart Spaces Lab (HCPS3Lab), the CUG and a member of the ACM, Special Committee on Software Engineering with China Computer Federation (CCF). His research interests include mobile and pervasive computing, software technology for human-cyber-physical systems, positioning and navigation of pedestrian and robotics for smart spaces.