TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 13/20 pp806–817
DOI: 10.26599/TST.2023.9010058
Volume 29, Number 3, June 2024

Improved Double Deep Q Network-Based Task Scheduling Algorithm in Edge Computing for Makespan Optimization

Lei Zeng†, Qi Liu†, Shigen Shen*, and Xiaodong Liu

Abstract: Edge computing nodes undertake an increasing number of tasks with the rise of business density. Therefore, how to efficiently allocate large-scale and dynamic workloads to edge computing resources has become a critical challenge. This study proposes an edge task scheduling approach based on an improved Double Deep Q Network (DQN), which is adopted to separate the calculations of target Q values and the selection of the action in two networks. A new reward function is designed, and a control unit is added to the experience replay unit of the agent. The management of experience data are also modified to fully utilize its value and improve learning efficiency. Reinforcement learning agents usually learn from an ignorant state, which is inefficient. As such, this study proposes a novel particle swarm optimization algorithm with an improved fitness function, which can generate optimal solutions for task scheduling. These optimized solutions are provided for the agent to pre-train network parameters to obtain a better cognition level. The proposed algorithm is compared with six other methods in simulation experiments. Results show that the proposed algorithm outperforms other benchmark methods regarding makespan.

Key words: edge computing; task scheduling; reinforcement learning; makespan; Double Deep Q Network (DQN)

1 Introduction

With the development and application of the Internet of Things (IoT) technology, traditional cloud-based data center service capabilities have become limited due to distance. Thus, edge computing now plays an increasingly important role[1]. In various industries, data are growing exponentially. For example, a smart city has a large amount of data and massive service scheduling requirements per day[2]. In meteorological and electric power fields, the service data generated by IoT devices in remote places especially rely on edge computing processing[3]. Similar to Moore's law, the number of users and the amount of data they generate are expected to increase exponentially in certain periods[4]. The rapid growth of data has put forward new requirements for edge computing. However, edge nodes are usually limited in computing resources, and the demand for task processing is increasing day by day. As such, efficient and reasonable task scheduling is an important part of improving edge resource utilization and service quality.

Task scheduling in the edge environment is an NP-hard problem because of its complexity[5]. In such an environment, task scheduling optimization strategies

• Lei Zeng is with School of Computer Science, Nanjing University of Information Science and Technology, Nanjing 210044, China. E-mail: 20201220002@nuist.edu.cn.
• Qi Liu is with School of Software, Nanjing University of Information Science and Technology, Nanjing 210044, China. E-mail: qi.liu@nuist.edu.cn.
• Shigen Shen is with the School of Information Engineering, Huzhou University, Huzhou 313000, China. E-mail: shigens@zjhu.edu.cn.
• Xiaodong Liu is with School of Computing, Edinburgh Napier University, Edinburgh, EH10 5DT, UK. E-mail: x.liu@napier.ac.uk.
† Lei Zeng and Qi Liu contribute equally to this paper.
∗ To whom correspondence should be addressed.
Manuscript received: 2023-02-10; revised: 2023-04-25; accepted: 2023-06-03

are usually divided into static and dynamic. Based on heuristic ideas, the static scheduling strategy has certain apparent defects. For example, static algorithms only consider the feasible schemes of resource allocation without considering the cluster state. Therefore, in resource allocation, fragmented resources are easily generated, and the scheduling efficiency is low[6].

As the complexity of the edge environment and task requirements increases, the need for dynamic scheduling strategies for edge computing becomes apparent. In recent years, researchers have therefore paid extensive attention to artificial intelligence, such as reinforcement and deep learning methods. The generalization learning ability of these methods and the self-learning of interacting with the environment provide new research ideas for task scheduling in edge environments. Reinforcement learning shows good performance on decision-making problems by learning and adjusting actions based on rewards from the environment[7]. At each state, the agent chooses an action to execute and gains a corresponding reward value[8]. The agent’s goal is to maximize the cumulative reward by choosing a sequence of actions. The corresponding relationship between all these behaviors and states is the strategy learned by the agent[9]. This strategy is formed by constantly trying new actions and making adjustments. This idea of learning allows reinforcement learning to be a very effective edge task scheduling approach[10, 11].

In general, no predefined model describes the dynamic and complex task scheduling in edge environments. The task scheduling based on the reinforcement learning model does not need to consider the environment model in advance. The agent explores the environment through different actions and gradually adapts to the dynamic changes of the external environment. The state of the virtual machine changes with the execution of actions, as shown in Fig. 1. The states of the virtual machines refer to the indicators that require optimization, such as the maximum completion time (makespan) and task response time. The agent adjusts actions to optimize scheduling goals by observing changes in the state of the virtual machines. Therefore, reinforcement learning is more suitable for task scheduling problems compared with other machine learning algorithms[12–14].

Yang et al.[15] proposed a multi-objective task scheduling strategy based on Q-learning, which sorts

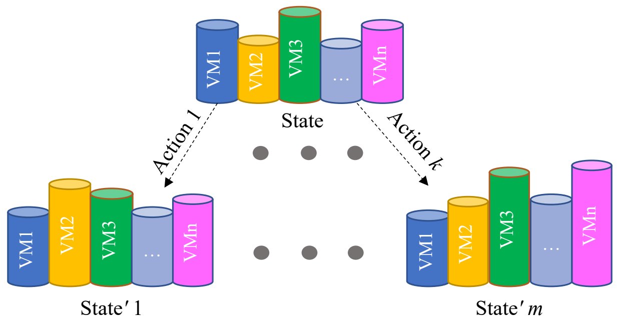

Fig. 1 State changes of the virtual machines.

the tasks first and then assigns these to the appropriate virtual machines. Ding et al.[16] used M/M/S queuing model to assign users to each server in the cloud, sorted tasks based on weighted attributes, and finally used the Q-learning algorithm to assign tasks to different virtual machines. However, the Q-learning method based on table storage also faces a fatal problem, which is state explosion. When the dimensional space of the state–action is high, the storage for states not only occupies a large space but also causes a time-consuming search. Therefore, the traditional Q-table approaches of storage and search have become the bottleneck restricting the scheduling efficiency. A Deep Q Network (DQN) is combined with the value function approximation and neural network, and the target network and experience replay method are used to train the network[17]. Che et al.[18] proposed a task scheduling algorithm based on a deep reinforcement learning model, which minimizes task execution time through dynamic scheduling using a priority relationship with edge servers.

Reinforcement learning methods can serve as a general framework for problems in decision making and control; however, its cognition of new things requires cultivation, which requires learning under a large range of data. The agent’s learning process is usually slow because of the lack of valid data in the initial state and a large number of inefficient or even invalid behavior attempts. As such, the major contributions of this study are concluded as follows:

•A Double DQN-based task scheduling approach is proposed to avoid the problem of overestimation in DQN. The calculation of target Q values and the selection of corresponding actions are placed in two networks. Furthermore, a new reward function is designed for the agent.

• A control unit is added to the agent’s experience replay unit, where the way of managing and deleting experience data is modified to fully utilize its value and improve learning efficiency.

• An improved Particle Swarm Optimization (PSO) algorithm with a new fitness function is proposed to generate optimal solutions for task scheduling. These optimized solutions are provided as empirical knowledge learning, which can improve the cognition ability of the agent regarding the environment.

The rest of this paper is organized as follows. Section 2 reviews related work on task scheduling. Section 3 describes the system model. The proposed approach is designed in Section 4. The performance evaluation of the proposed approach is presented in Section 5. Section 6 presents the conclusion and possible future works.

2 Related Work

Task scheduling plays a crucial role in improving the resource utilization efficiency and service quality of edge computing, and thus has received much research attention[19–22]. The edge environment is dynamic, heterogeneous, and has a large number of tasks. Traditional heuristic algorithms such as First Come, First Service (FCFS) and Round-Robin cannot adapt to the dynamic context in edge and cloud environments. The meta-heuristic approach combines stochastic algorithms and local search and thus obtains the ability to dynamically explore the optimal solution. For example, Qi[23] proposed a resource scheduling method based on an improved PSO to optimize the service quality and execution time of tasks. Huang et al.[24] proposed a task allocation algorithm for mobile ad-hoc networks based on simulated annealing and PSO. However, meta-heuristic methods also lack good performance in dealing with dynamic and complex edge and cloud environments.

With its emergence and superiority over humans in handling decision-making problems, reinforcement learning has been used to solve dynamic task scheduling problems in edge and cloud environments. For example, the Google research team successfully reduced the cooling power consumption of their data center by 40% by using reinforcement learning technology and historical data[25]. Dab et al.[26] proposed a task allocation algorithm named QLJoint based on Q-learning to minimize task delay. Zhao et al.[27] proposed a low-load Distributed Intrusion Detection System (DIDS) task scheduling method based on Q-learning. The load of DIDS can be reduced by adjusting the scheduling when the network changes in the edge environment. Guevara et al.[28] considered the QoS requirements of applications and proposed a task scheduler based on reinforcement learning to optimize the makespan and processing cost of workflows. These are value-based methods that stores the Q value in a table and then searches for the most appropriate one when making a decision. However, when the state–action dimension increases in size, the Q table undergoes difficulties in terms of storage and search problems.

The deep reinforcement learning approach has just made up for the shortcomings of Q-learning and has achieved certain progress. Wei et al.[29] proposed a QoS-aware job scheduling approach, which realizes the online scheduling of single tasks by using a deep Q-learning algorithm. Dong et al.[30] proposed a task scheduling method that combines deep learning and Q-learning. Li et al.[31] proposed a generative confrontation reinforcement learning task scheduling algorithm based on expert experience, which can use the optimal policy in the expert experience pool to guide the agent to learn dynamic task scheduling. Gazori et al.[32] proposed a double deep Q-learning scheduling algorithm to schedule tasks submitted by users and minimize the average service delay through the analysis of hyperparameters. Zhang et al.[33] proposed an edge scheduling approach based on DVFS and double deep Q-learning, wherein the Double DQN is used to generate the Q value for the DVFS. Swarup et al.[34] proposed a scheduling algorithm based on deep reinforcement learning, which aims to reduce the cost and service delay of scheduling IoT tasks in fog-based environments.

The combination of the decision-making ability of reinforcement learning and the perceived ability of deep learning considerably enhances the cognitive ability of the agent. However, the deep reinforcement learning approach also faces new problems.

• Deep learning methods require large amounts of labeled data for training. However, deep reinforcement learning methods do not have enough data in the initial state. Therefore, in the early stage of task scheduling, its cognition of the environment is poor.

• Reinforcement learning is based on scalar rewards, which are often sparse, noisy, and delayed. Therefore, the experience data cannot be fully utilized to improve learning efficiency.

3 System Model

Reinforcement learning in task scheduling can be explained in two parts, namely, the edge environment and the agent. Their relationship is shown in Fig. 2.

The edge environment contains machine clusters and tasks submitted by users, as well as task and machine feature descriptions. These features contain task length, task arrival time, and remaining machine resources, which are the conditions used as the basis for scheduling operations. Based on reinforcement learning, the agent is a scheduler that selects actions and schedules tasks to corresponding machines based on the current states and then receives rewards from environmental feedback. The agent can thus obtain the optimal scheduling sequence by maximizing the value of reward accumulation.

The main part of reinforcement learning is the agent, which interacts with the environment. Observing a certain state S_t at time t, the agent chooses an action A_t according to a certain preset strategy then a certain state transition probability $p(S_{t+1},R_{t+1}|S_t,A_t)$ from the current state S_t to the next state S_{t+1}. Meanwhile, the environment also gives the agent a reward R_{t+1}. This process produces the following sequence:

$$S_0,\ A_0,\ R_1,\ S_1,\ A_1,\ R_2,\ S_2,\ \ldots,\ S_{T-1},\ A_{T-1},\ R_T,\ S_T.$$

Markov Decision Processes (MDPs) are adopted to model reinforcement learning problems. Through continuous learning, the agent can make better decisions when interacting with the environment. MDP is a kind of stochastic process, and its original model is the Markov chain. The environment can be simplified by Markov characteristics as in Eq. (1):

$$p(S_{t+1}|S_t) = p(S_{t+1}|S_1,S_2,\ldots,S_t) \tag{1}$$

The edge task scheduling is in line with this characteristic. The next state after each scheduling only depends on the current state; that is, as shown in Fig. 3, State T_1 is only related to State T_0, while State T_2 is only related to State T_1 but not to State T_0.

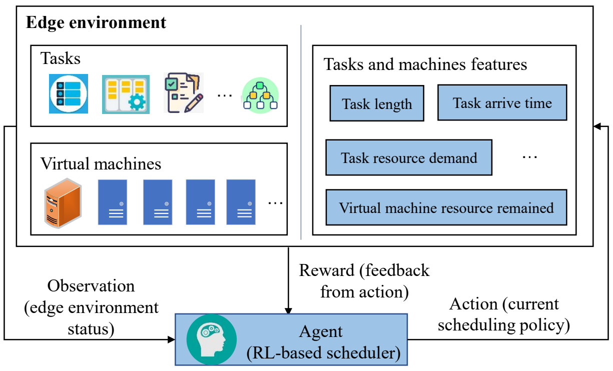
Edge environment
Tasks
...
Virtual machines
...
Tasks and machines features
Task length
Task arrive time
Task resource demand
...
Virtual machine resource remained
Observation (edge environment status)
Reward (feedback from action)
Action (current scheduling policy)
Agent (RL-based scheduler)

Fig. 2 Task scheduling framework for edge environment based on reinforcement learning.

Systems can be modeled based on this property. Edge computing resources can be normalized to be composed of N physical servers. Each of the physical servers can create multiple virtual machines through virtualization technology to perform tasks.

The following steps show the modeling of resources, tasks, and goals in task scheduling. The scheduler receives n independent task requests. The task list can be defined as $T=\{t_1,t_2,t_3,\ldots,t_n\}$, where $t_i\ (i\in[1,n])$ denotes the i-th task, and n is the total number of tasks in the task list T. The task t_i can be represented by a vector $t_i=(l_i,m_i,p_i,d_i)$, where l_i denotes the duration of task t_i expressed in Million Instructions Per Second (MIPS) while m_i, p_i, and d_i represent the memory space required by task t_i, number of processors, and the deadline, respectively. The cluster contains m heterogeneous virtual machine resources, which can be defined as $R=\{r_1,r_2,r_3,\ldots,r_m\}$. The configurations of these resources, such as CPU frequency and memory, vary. Meanwhile, $r_j=\left(\text{mips}_j,\text{core}_j,\text{ram}_j,\text{hyper}_j\right)$ can be used to denote the virtual resource r_j, where $j\in[1,n]$. The scheduler is responsible for assigning the received tasks to the appropriate machines. Execution Time (ET) is defined as the time from start to finish of task execution of a specific virtual machine, which can be calculated by using Eq. (2). mips_j represents the MIPS value of the virtual machine r_j.

$$\text{ET}_i = \frac{l_i}{\text{mips}_j} \tag{2}$$

In edge task scheduling, the max value of the completion time among all the virtual machines is defined as makespan, which is used to evaluate the efficiency and quality of task scheduling algorithms. Under limited resources, a lower makespan means less time for completing tasks and higher utilization of the machine. In task scheduling, makespan is calculated by using Eq. (3):

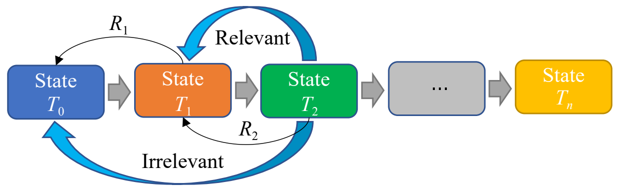

Fig. 3 Markov process in edge task scheduling.

$$\text{MS} = \max(\{\text{CT}_0, \text{CT}_1, .., \text{CT}_m\}) \tag{3}$$

where $\text{CT}_j = \gamma_j - \text{start}(r_j)$, $j = 0, 1, \ldots, m$, is the completion time of the j-th virtual machine r_j and γ_j indicates the time when the j-th virtual machine completes all of its assigned tasks. $\text{start}(r_j)$ is the time when tasks are begun. MS is the maximum of all machine completion time, that is, makespan. The average completion time CT_{ave} and standard deviation of completion time of all machines CT_{var} are shown in Eqs. (4) and (5), respectively.

$$\text{CT}_{\text{ave}} = \frac{\sum_{j=1}^{m} \text{CT}_j}{m} \tag{4}$$

$$\text{CT}_{\text{var}} = \frac{\sum_{j=1}^{m} \left(\text{CT}_j - \text{CT}_{\text{ave}}\right)^2}{m} \tag{5}$$

An agent for reinforcement learning consists of three important components: action space, state space, and reward function. These three parts depend on and promote each other, ensuring continuous and stable agent learning. On the basis of the state of the environment, the agent chooses and executes an action. The reward function is used to motivate the agent to adjust actions to obtain the maximum reward value.

The state space reflects the allocation and usage of tasks and virtual machines. For the state of resources of each virtual machine, the completion time CT_j of the currently assigned task can be calculated. The global maximum completion time is also recorded, $V_{\text{current}} = \{v_1, v_2, v_3, \ldots, v_m, \text{MS}\}$ can be used to represent the machine status. For the task state, the agent only needs to observe the task queue to be scheduled, which can be represented by $T_{\text{wait}} = \{t_1, t_2, t_3, \ldots\}$. The state space is described by $S = \{S_1, S_2, \ldots\}$, where $S_i = (V_{\text{current}}, T_{\text{wait}})$.

The action space reflects the behavior of the scheduler in assigning tasks to virtual machines. With m virtual machines, the action space for each task scheduling is the set of serial numbers of m virtual machines. Thus, $A = \{1, 2, 3, \ldots, m\}$ can be used to represent the set of action spaces. The criterion for the agent to choose the needed actions is to minimize the makespan. Therefore, when the scheduling is not finished, the agent can consider scheduling tasks to machine r_i with the smallest completion time. Then a reward value and update of the state of the environment can be obtained. A deep learning neural network generates actions that lead to obtaining the greatest cumulative rewards.

The reward function is the feedback given by the environment corresponding to the actions chosen by the agent, which is crucial to improve the performance of the reinforcement learning algorithm. The agent adjusts its actions according to the reward feedback obtained to better adapt to the environment. The goal of this study is to obtain the optimal makespan, and a new reward function is designed, as shown in Eq. (6).

$$R = -(\text{MS}(S') - \text{MS}(S))\rho \tag{6}$$

where $\text{MS}(S')$ is the makespan that enters the state S' after executing an action in state S during task scheduling, and ρ is a constant used to adjust the reward to a suitable range. The reward is given by slowing the makespan growth. To maximize the reward value, the slowest growth of the makespan is necessary. Therefore, the scheduler becomes more inclined to schedule tasks to virtual machines with light loads and low completion time.

4 Proposed Approach

4.1 Design of Double DQN model for task scheduling

Despite its contributions to the Q value to approach the optimization goal quickly, the greedy method is also prone to overfitting. In the DQN model, the target network adopts the action selection strategy of maximizing the Q value by $\max_{a'} Q(s_{t+1}, a', \theta^-)$. Therefore, the algorithm model obtained by DQN probably has a large deviation, which results in overestimation. To tackle the overestimation problem, we need to change the training method of the network self-fitting. Therefore, this study proposes a learning model based on Double DQN to modify the learning method for task scheduling. By placing the calculation and action selection of the target Q value in two networks, the transfer of the maximum deviation is cut off.

A target network with the same structure as the original network but with different parameters is constructed. The $Q(s, a, \theta)$ is called an evaluation network, and the target network is $Q(s, a, \theta^-)$, where $\theta^- \neq \theta$. The roles of the two networks differ. The evaluation network is responsible for controlling the agent and collecting experience, and its parameters θ

are up-to-date. The target network $Q(s,a,\theta^-)$ is used to compute the target value.

Figure 4 shows the algorithm learning procedure of scheduling in an edge environment. The agent receives the state of virtual machines and tasks from the edge scheduling environment. Then appropriate actions, the task-to-virtual machine mappings, are generated by the evaluation network. Then the agent assesses whether all the tasks have been scheduled. If scheduling is complete, then the target value is r. If not, the target network is used to calculate the target value. The gradient descent algorithm is then performed to update the evaluation network.

Action selection: based on state s_{t+1}, the evaluation Q-network is adopted to seek an action that maximizes the output of Double DQN. In this network, a^* is the action that can produce the largest Q value and is defined as in Eq. (7):

$$a^* = \arg\max_{a\in A} Q(s_{t+1},a,\theta) \tag{7}$$

Target value calculation: Target Q-network is adopted to calculate (s_{t+1},a^*), that is, the Q value of action a^* in the target network. y_t is calculated by using Eq. (8).

$$y_t = r_t + \gamma Q(s_{t+1},a^*,\theta^-) \tag{8}$$

Given that the target and the evaluation Q-networks are independent of each other, a^* is not necessarily the $\arg\max Q$ under the target Q-network θ^- parameter, which can be described as the following inequality in Formula (9). Thus, the error can be effectively reduced, and the problem of network overestimation can be avoided.

$$Q(s_{t+1},a^*,\theta^-) \leqslant \max_{a\in A} Q(s_{t+1},a,\theta^-) \tag{9}$$

Then, the target Q value can be calculated by Eq. (10) in Double DQN.

$$y_t = r_t + \gamma Q\left(s_{t+1}, \arg\max_{a\in A} Q(s_{t+1},a,\theta),\theta^-\right) \tag{10}$$

The evaluate Q-network is trained by minimizing the mean squared error (MSE) between the target and the estimated values. The minimized MSE loss function is shown in Eq. (11).

$$L(\theta) = E_{s,a}\left[(y_t - Q(s,a,\theta))^2\right] \tag{11}$$

Gradient descent is then performed to update the network. The gradient of parameter θ is shown in Eq. (12).

$$\nabla\theta = E[y_t - Q(s,a,\theta)\nabla Q(s,a,\theta)] \tag{12}$$

Evaluation Q-network parameters $\theta = \theta + \nabla\theta$ are then updated. After a certain step size C, we update the Temporal Difference (TD) target network parameters to set $\theta^- = \theta$. In the training of Double DQN, the reinforcement learning and the deep learning gradient descent algorithms are carried out together. The empirical sample data obtained by the agent is sent to the network for training. Thus, the network continuously approaches the action value function, and the agent learns the optimal strategy.

The parameter update intervals of the target network are improved. Given that the gradient descent method is used in the network training, the gradient value $\nabla\theta$ can be used to evaluate the network and assist in judging whether to update the target network or not. In the initial stage, the value of the target network update

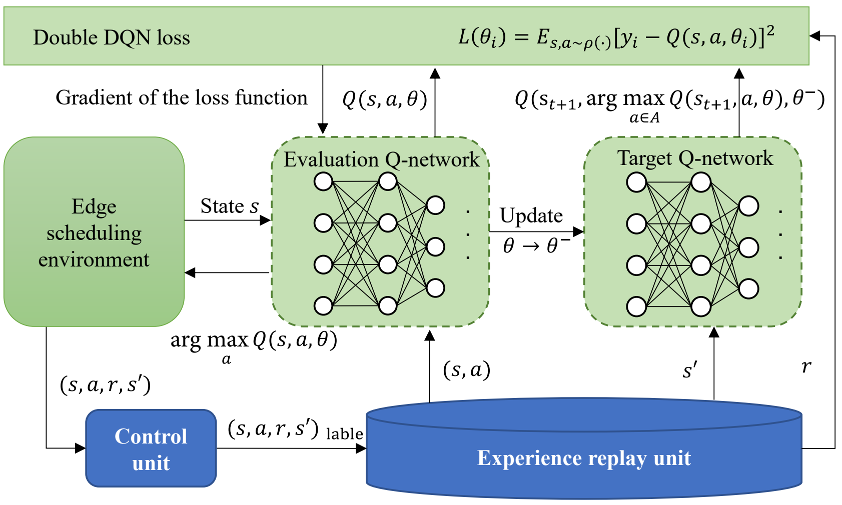
Double DQN loss
$L(\theta_i) = E_{s,a\sim\rho(\cdot)}[y_i - Q(s,a,\theta_i)]^2$
Gradient of the loss function
$Q(s,a,\theta)$
$Q(s_{t+1}, \arg\max_{a\in A} Q(s_{t+1},a,\theta),\theta^-)$
Edge scheduling environment
State s
Evaluation Q-network
Update $\theta \to \theta^-$
Target Q-network
$\arg\max_a Q(s,a,\theta)$
(s,a,r,s')
(s,a)
s'
r
Control unit
(s,a,r,s') lable
Experience replay unit

Fig. 4 Learning process of the Double DQN algorithm.

interval C is defined. Then, the gradient value during network training is observed. If the gradient value is lower than the threshold, then the network tends to converge, and the number of steps in the update interval can be reduced. If the update interval C is reached but the gradient remains large, then the former can increase appropriately. Thus, the dynamic adjustment of the update interval of the target network can be realized.

4.2 Control unit for experience replay

The agent learns based on the reward, which is a scalar value. However, given that the reward is usually sparse and delayed in the learning process, this study proposes a novel experience replay. This learning method is used in deep reinforcement learning. Double DQN has a storage unit for storing previous learning experiences, given that Q-learning can learn not only from current and past experiences but also from others. Therefore, the trained data are stored in the replay buffer, and the previous experience can be randomly extracted for learning, thereby breaking the correlation among data.

The original Double DQN experience replay mechanism is simple, saving queue structures and deleting old data when the queue is full. For the selection of empirical data, uniform distribution is used to randomly select from the memory unit. However, among the experiences data (s,a,r,s'), a few samples can be used to help the network improve the learning quality and speed up the convergence; the others are not important. This simplistic approach to data management does not fully leverage the value of experience data. Therefore, the experience replay unit in Double DQN is redesigned to better serve the network training. Before the experience replay unit, a control unit is added to label the experience data and a new data management method is designed.

The data (s,a,r,s') input into the experience replay unit carry two pieces of information that can be used to mark their values. One is the TD error $\delta = y_t - Q(s_t,a,\theta)$. In the Double DQN model, The TD error is the difference between the two values of real and estimated Qs, which are calculated by the two networks, respectively. The goal of training is to achieve the smallest possible expectation of δ^2. Therefore, δ can mark the importance of the sample to a certain extent. However, δ is not completely accurate because of the random environment and thus cannot fully reflect the state. The other information is the reward. The principle of reinforcement learning to choose an action is to obtain the maximum reward. When the state S_t enters into the next state S_{t+1} by taking an action A_t, a reward r is obtained. A high reward r indicates a high value of the action.

Based on these two pieces of information, an experience replay unit with a control gate is designed. Before the data (s,a,r,s') are stored in the memory unit, a flag bit is added by the control unit. This flag bit limits the lifespan of the sample data stored in the memory unit to adjust the probability of different data being sampled and learned. Instead of being simply and passively managed and deleted in the memory by using time and space criteria, the data are replaced by novel management in an active way. Thus, the flag bit τ_{age} turns the data into $(s,a,r,s',\tau_{\text{age}})$. The value of τ_{age} can be calculated by Eq. (13):

$$\tau_{\text{age}} = L_{\text{ini}} + r\alpha_1 - |\delta|\alpha_2 \tag{13}$$

where L_{ini} is the initial lifetime set for each datum. Reward r and TD error δ are used to adjust the value of τ_{age}. α_1 and α_2 are constants used to adjust r and δ to a suitable range. The larger the reward, the slower the growth of the makespan, indicating that the actions performed in this state increase the balance of the cluster load. According to the reward calculation formula, the reward is a value less than zero. The smaller the absolute value of the TD error, the more accurate the neural network prediction.

The management of experience data is modified. The τ_{age} indicates the length of life of the sample data in the memory unit, and the management of life is realized through τ_{age}. Every time data are sampled and learned, the value of τ_{age} is reduced by one, and the corresponding sample is deleted from the memory unit until τ_{age} is zero. Experience data with a smaller τ_{age} value are deleted first when the memory unit space is full. Thus, a kind of priority management can be achieved indirectly. The more valuable the data, the longer life they will have in the memory unit, and the greater their probability of being sampled and learned. By contrast, the probability of sampling and learning is lower for data without much value.

4.3 Pretraining for evaluation network

Deep learning requires a large amount of labeled data for training to achieve better performance. However, reinforcement learning, which acquires experience by interacting with the environment, lacks data in the early stages of the method. Therefore, this paper proposes a pretraining-based reinforcement learning task scheduling method to improve the scheduling performance of the algorithm. An improved PSO algorithm is proposed to obtain the optimal scheduling sequence. Then a neural network of deep reinforcement learning is used to fit these optimized solutions to enable the agent to obtain prior knowledge. Given that the task data are independent and identically distributed, the pretraining can provide effective experiences.

Calculations of the completion time of each machine in the cluster show great variations in edge resources. For example, the percentage difference between maximum and minimum completion time is as high as 584% in FCFS, 192% in Shortest Job First (SJF), and 379% in PSO. These results indicate a load imbalance in the task scheduling process. Therefore, based on the load balancing mechanism, a new fitness function is designed for PSO, which enables particles to obtain better solutions in task scheduling. The new fitness function is formulated as Eq. (14):

$$f = \sigma \mathrm{MS} + (1-\sigma)\mathrm{CT}_{\mathrm{var}} \tag{14}$$

where MS is the optimization target makespan; $\mathrm{CT}_{\mathrm{var}}$ is the variance of the completion time of all virtual machines; σ is a constant between 0 and 1, which is used to adjust the optimization ratio between MS and $\mathrm{CT}_{\mathrm{var}}$; and f is the fitness function, which is used to evaluate the optimization goal. The objective is optimized by minimizing the value of the fitness function. The optimization constraints for cluster load balancing are added to the new fitness function. The makespan can further decrease by reducing the load difference across the cluster machines. This action can also reduce the load on several machines.

The improved PSO algorithm is used to generate multiple sets of optimized solutions, which are the mapping sequence from tasks to edge resources. Then these optimized solutions can be used to pre-train the evaluation Q-network θ of Double DQN to obtain a better cognition of the environment. Thus, the agent can obtain a lower makespan in task scheduling through the learning of prior knowledge.

5 Performance Evaluation

5.1 Experiment setup

In this study, the proposed approach is tested and compared with other methods in a simulation environment. The experimental setup is as follows. An edge node containing 14 physical machines is constructed, with the physical machines having 3000 MIPS values, four cores, and 16 GB memory. Then three types of virtual machines are created with different resource configurations, as shown in Table 1.

The task dataset from the Alibaba data center is used as task input. Experiments are carried out under different workload scales and machine numbers to test the scheduling performance of the proposed approach in different environments. The algorithms used for comparison include FCFS, SJF, PSO, Simulated Annealing PSO (SA-PSO), DQN, and Double DQN task scheduling. For the convenience of drawing icons, the Double DQN is abbreviated as DDQN, and the method proposed in this paper is referred to as G-DDQN.

5.2 Experimental result

The dataset cloud_trace_2017 released by the Alibaba data center contains a large number of task datasets in real production application environments. Figure 5 shows the length distribution of tasks.

The task duration in this dataset is almost within 1000 s, most of which are within 200 s, accounting for 88.7%. The tasks within 200 s are relatively evenly distributed in different time periods, as follows: 1–20 s account for 30.9%, 20–50 s account for 20.9%, 50–100 s account for 22.5%, and 100–200 s account for 25.6%. The proportion of longer tasks of more than 200 s continuously decreases.

Table 1 Virtual machine configuration.

VM type	MIPS	Number of cores	Memory (GB)
Type-1	1000	1	1
Type-2	1500	1	2
Type-3	2000	1	3

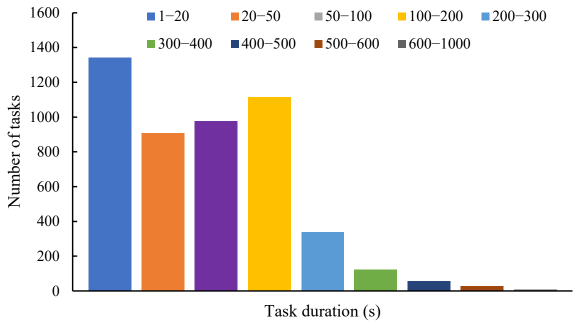

Fig. 5 Task duration distribution.

A heterogeneous cluster of three types of virtual machines is created, as follows: 12 Type-1, 8 Type-2, and 6 Type-3. The influence of different numbers of hidden layers and the number of neurons on the Double DQN task scheduling algorithm model is tested. Multiple sets of Double DQN models are constructed with different hidden layers, and the average value of multiple training sessions is taken for each model. The results are shown in Fig. 6. The experimental results show that the effect of two hidden layers is better than that of one. However, the number of hidden layers is not always better. Specifically, with three hidden layers, the performance of the algorithm will no longer be significantly improved. The result shows that the two hidden layers can already achieve a better fitting effect. Moreover, the more hidden layers, the longer the training time of the algorithm.

Under the two hidden layers, the effect of different numbers of neurons is tested. Figure 7 shows that better scheduling results can be obtained after increasing the number of neurons. However, similar to the number of hidden layers, the makespan does not always decrease linearly as the number of neurons continues to increase. When the number of neurons reaches 12 or more, the scheduling results gradually stabilize. However, the optimal number of neurons is not an absolute value and also needs to be modified as the task schedule scales to accommodate changes in fitting complexity.

The number of virtual machines is fixed to test the performance of the scheduling algorithm under different workloads. Multiple sets of experiments are carried out by varying the number of tasks loads with increases of 50. The proposed method is compared with six other methods. Five sets of experiments are carried out under different numbers of tasks. The results of makespan and the variances of completion time are shown in Figs. 8 and 9.

As shown in Fig. 8, the following results can be obtained. First, the value of the makespan of the FCFS algorithm is always the largest and increases the fastest as the number of tasks increases. G-DDQN improves the scheduling performance by an average of 46.7% under different loads compared with FCFS. Second, compared with the SJF algorithm, the percentage improvement of G-DDQN increases from approximately 12% to 21% as the number of tasks

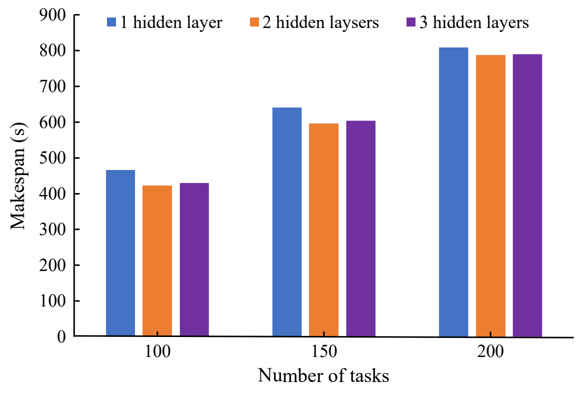

Fig. 6 Experimental results with different number of hidden layers.

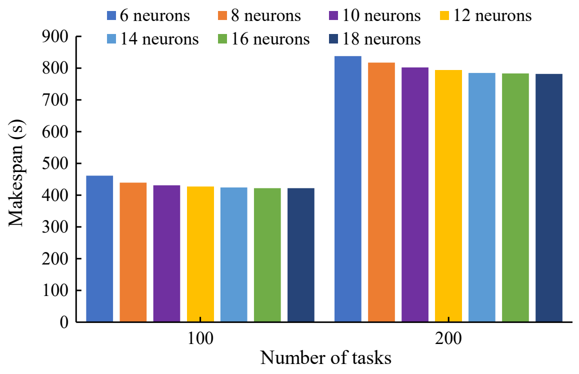

Fig. 7 Experimental results under different number of neurons.

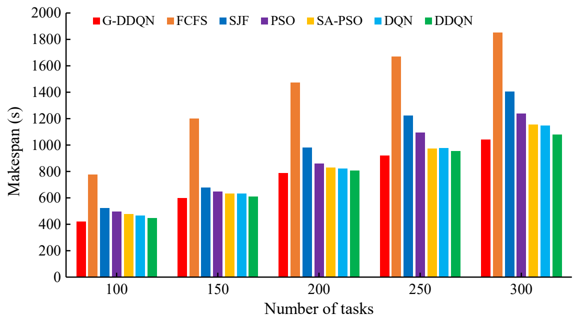

Fig. 8 Makespan under different number of tasks.

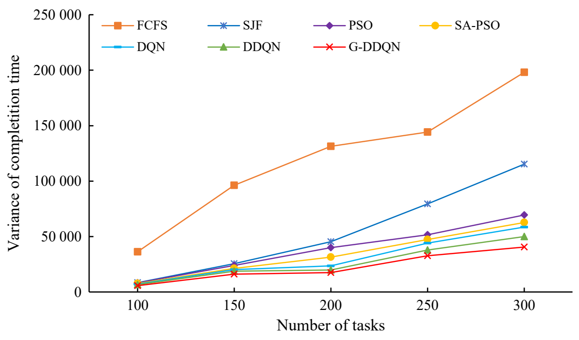

Fig. 9 Variance of completion time under different number of tasks.

increases. This result indicates that the scheduling performance of SJF tends to degrade when faced with an increase in the number of tasks. As the number of tasks increases, more long tasks need to be scheduled, and the scheduling performance of SJF decreases. G-DDQN increases prior knowledge training and improves the management and utilization of empirical data, and therefore can achieve the optimal scheduling effect. Compared with PSO, SA-PSO, DQN, and DDQN under different loads, the G-DDQN method improves performance by 9.9%, 6.1%, 5.8%, and 2.6% on average in terms of makespan, respectively.

Figure 9 reflects the load balancing of the cluster. The larger of the variance of completion time, the larger of the load difference for the cluster. Therefore, the makespan value of the FCFS method is poor because of its large cluster load difference during the scheduling. For SJF, the decrease in performance when the number of tasks increases is also due to the deterioration of the system load balance. When the number of tasks exceeds 200, the value of the variance of the machine completion time in SJF scheduling rapidly begins to increase. Compared with these heuristic methods, the PSO, SA-PSO, DQN, DDQN, and G-DDQN methods have smaller variance increases in machine completion time. Among them, G-DDQN can be kept at the lowest level, and thus the system load balance in the scheduling process is the best. As the number of tasks increases, G-DDQN can maintain better system load balance than other algorithms.

When the cluster has a heavy workload, the number of virtual machine resources must be increased to relieve system pressure. Therefore, the experiment also tests the scheduling performance of the proposed method under elastic changes in the number of virtual machines. In this study, a new cluster is created with multiple Type-2 virtual machines. The workload is set to 300, and then the number of virtual machine resources is dynamically changed for experimentation. Five sets of experiments with different numbers of virtual machines are carried out, and the following results are obtained.

Figure 10 shows the result of the makespan. The performance of the FCFS scheduling algorithm is unstable because its makespan sometimes even increases as the number of virtual machines increases. Under different numbers of virtual machines, the makespan of G-DDQN is 28.7% lower than that of FCFS on average. As the number of virtual machines

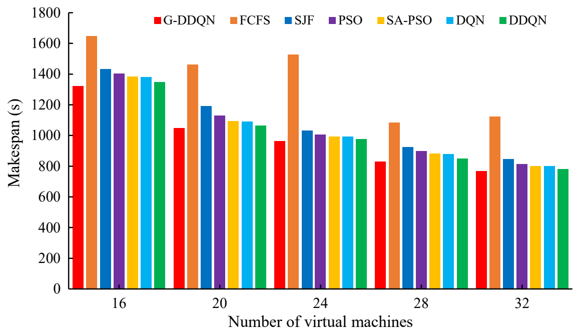

Fig. 10 Makespan under different number of virtual machines.

decreases, G-DDQN can obtain the lowest makespan among the algorithms in task scheduling. Compared with SJF, PSO, SA-PSO, DQN, and DDQN, the scheduling performance of G-DDQN is improved by 9.2%, 5.3%, 4.1%, 4%, and 1.8%, respectively.

The data in Fig. 11 reflect the load balance of the system. As can be seen from the variance values of machine completion time, the results of makespan in FCFS are consistent with the system load differences. Under a steady workload, the variance of completion time for the other algorithms decreases slightly as the number of machines increases but remains stable overall. Among them, the G-DDQN method can maintain the lowest system load difference. Thus, the proposed scheduling method can also maintain stable performance when dealing with elastic changes in the number of cluster virtual machines.

6 Conclusion

This study proposes a task scheduling algorithm based on improved Double DQN to optimize makespan in the edge environment. First, the selection of action and the calculation of the Q value are decoupled, and a new reward function is designed. Second, the design of experience replay method and the management of

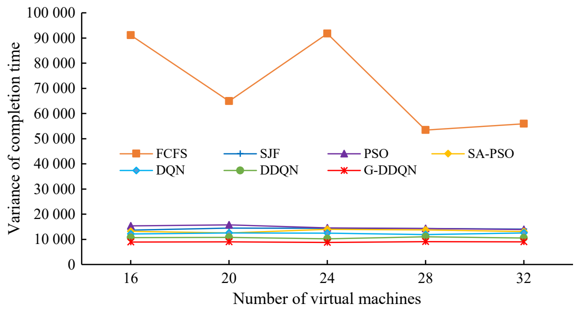

Fig. 11 Variance of completion time under different number of virtual machines.

experience data are modified to fully tap its different values. Then, prior knowledge is provided for the reinforcement learning agent to pre-train the network parameters to improve the agent's cognition level of the environment. Finally, the proposed approach is tested and compared in different environments. Comparisons with other methods show that the proposed scheduling can improve cluster load balancing and reduce makespan. In future works, we will further study workflow scheduling.

Acknowledgment

This work was supported by the National Key Research and Development Program of China (No. 2021YFE0116900), National Natural Science Foundation of China (Nos. 42275157, 62002276, and 41975142), and Major Program of the National Social Science Fund of China (No. 17ZDA092).

References

[1] X. Xu, H. Li, W. Xu, Z. Liu, L. Yao, and F. Dai, Artificial intelligence for edge service optimization in Internet of vehicles: A survey, *Tsinghua Science and Technology*, vol. 27, no. 2, pp. 270–287, 2021.

[2] M. Laroui, B. Nour, H. Moungla, M. A. Cherif, H. Afifi, and M. Guizani, Edge and fog computing for IoT: A survey on current research activities & future directions, *Comput. Commun.*, vol. 180, pp. 210–231, 2021.

[3] X. Xu, H. Tian, X. Zhang, L. Qi, Q. He, and W. Dou, DisCOV: Distributed COVID-19 detection on X-ray images with edge-cloud collaboration, *IEEE Trans. Serv. Comput.*, vol. 15, no. 3, pp. 1206–1219, 2022.

[4] S. B. Slama, Prosumer in smart grids based on intelligent edge computing: A review on artificial intelligence scheduling techniques, *Ain Shams Eng. J.*, vol. 13, no. 1, p. 101504, 2022.

[5] H. Wang, L. Cai, X. Hao, J. Ren, and Y. Ma, ETS-TEE: An energy-efficient task scheduling strategy in a mobile trusted computing environment, *Tsinghua Science and Technology*, vol. 28, no. 1, pp. 105–116, 2022.

[6] M. S. U. Islam, A. Kumar, and Y. C. Hu, Context-aware scheduling in fog computing: A survey, taxonomy, challenges and future directions, *J. Netw. Comput. Appl.*, vol. 180, p. 103008, 2021.

[7] X. Xu, Q. Jiang, P. Zhang, X. Cao, M. R. Khosravi, L. T. Alex, L. Qi, and W. Dou, Game theory for distributed IoV task offloading with fuzzy neural network in edge computing, *IEEE Trans. Fuzzy Syst.*, vol. 30, no. 11, pp. 4593–4604, 2022.

[8] X. Gao, D. Peng, G. Kui, J. Pan, X. Zuo, and F. Li. Reinforcement learning based optimization algorithm for maintenance tasks scheduling in coalbed methane gas field, *Comput. Chem. Eng.*, vol. 170, p. 108131, 2023.

[9] H. Tian, X. Xu, T. Lin, Y. Cheng, C. Qian, L. Ren, and M. Bilal, DIMA: Distributed cooperative microservice caching for Internet of Things in edge computing by deep reinforcement learning, *World Wide Web*, vol. 25, no. 5, pp. 1769–1792, 2022.

[10] A. Jayanetti, S. Halgamuge, and R. Buyya, Deep reinforcement learning for energy and time optimized scheduling of precedence-constrained tasks in edge-cloud computing environments, *Future Gener. Comput. Syst.*, vol. 137, pp. 14–30, 2022.

[11] Z. Li, X. Xu, X. Cao, W. Liu, Y. Zhang, D. Chen, and H. Dai, Integrated CNN and federated learning for COVID-19 detection on chest X-ray images, *IEEE/ACM Trans. Comput. Biol. Bioinform.*, doi: 10.1109/TCBB.2022.3184319.

[12] S. Vemireddy and R. R. Rout, Fuzzy reinforcement learning for energy efficient task offloading in vehicular fog computing, *Comput. Netw.*, vol. 199, p. 108463, 2021.

[13] Z. Tang, W. Jia, X. Zhou, W. Yang, and Y. You, Representation and reinforcement learning for task scheduling in edge computing, *IEEE Trans. Big Data*, vol. 8, no. 3, pp. 795–808, 2022.

[14] M. N. Tran and Y. Kim, A cloud QoS-driven scheduler based on deep reinforcement learning, in *Proc. 2021 Int. Conf. Information and Communication Technology Convergence* (*ICTC*), Jeju Island, Republic of Korea, 2021, pp. 1823–1825.

[15] Z. Yang, M. Xiao, and Y. Ge, Dynamic resource scheduling of cloud-based automatic test system using reinforcement learning, in *Proc. 2017 13th IEEE Int. Conf. Electronic Measurement & Instruments* (*ICEMI*), Yangzhou, China, 2017, pp. 159–165.

[16] D. Ding, X. Fan, Y. Zhao, K. Kang, Q. Yin, and J. Zeng, Q-learning based dynamic task scheduling for energy-efficient cloud computing, *Future Gener. Comput. Syst.*, vol. 108, pp. 361–371, 2020.

[17] H. Tian, X. Xu, L. Qi, X. Zhang, W. Dou, S. Yu, and Q. Ni, CoPace: Edge computation offloading and caching for self-driving with deep reinforcement learning, *IEEE Trans. Veh. Technol.*, vol. 70, no. 12, pp. 13281–13293, 2021.

[18] H. Che, Z. Bai, R. Zuo, and H. Li, A deep reinforcement learning approach to the optimization of data center task scheduling, *Complexity*, vol. 2020, p. 3046769, 2020.

[19] Q. Liu, R. Mo, X. Xu, and X. Ma, Multi-objective resource allocation in mobile edge computing using PAES for Internet of Things, *Wirel. Netw.*, doi: https://doi.org/10.1007/s11276-020-02409-w.

[20] H. Xu, J. Zhou, W. Wei, and B. Cheng, Multiuser computation offloading for long-term sequential tasks in mobile edge computing environments, *Tsinghua Science and Technology*, vol. 28, no. 1, pp. 93–104, 2022.

[21] T. Choudhari, M. Moh, and T. S. Moh, Prioritized task scheduling in fog computing, in *Proc. ACMSE 2018 Conference*, Richmond, Kentucky, 2018, pp. 1–8.

[22] Q. Liu, X. Wu, X. Liu, Y. Zhang, and Y. Hu, Near-data prediction based speculative optimization in a distribution environment, *Mob. Netw. Appl.*, vol. 27, no. 6, pp. 2339–2347, 2022.

[23] W. Qi, Optimization of cloud computing task execution

time and user QoS utility by improved particle swarm optimization, *Microprocess. Microsyst.*, vol. 80, p. 103529, 2021.

[24] B. Huang, W. Xia, Y. Zhang, J. Zhang, Q. Zou, F. Yan, and L. Shen, A task assignment algorithm based on particle swarm optimization and simulated annealing in ad-hoc mobile cloud, in *Proc. 2017 9th Int. Conf. Wireless Communications and Signal Processing* (*WCSP*), Nanjing, China, 2017, pp. 1–6.

[25] J. Gao, Machine learning applications for data center optimization, Google White Paper, https://research.google/pubs/pub42542/, 2014.

[26] B. Dab, N. Aitsaadi, and R. Langar, Q-learning algorithm for joint computation offloading and resource allocation in edge cloud, in *Proc. 2019 IFIP/IEEE Symp. on Integrated Network and Service Management* (*IM*), Arlington, VA, USA, 2019, pp. 45–52.

[27] X. Zhao, G. Huang, L. Gao, M. Li, and Q. Gao, Low load DIDS task scheduling based on Q-learning in edge computing environment, *J. Netw. Comput. Appl.*, vol. 188, p. 103095, 2021.

[28] J. C. Guevara, R. da S Torres, L. F. Bittencourt, and N. L. S. da Fonseca, QoS-aware task scheduling based on reinforcement learning for the cloud-fog continuum, in *Proc. GLOBECOM 2022-2022 IEEE Global Communications Conference*, Rio de Janeiro, Brazil, 2022, pp. 2328–2333.

[29] Y. Wei, L. Pan, S. Liu, L. Wu, and X. Meng, DRL-scheduling: An intelligent QoS-aware job scheduling framework for applications in clouds, *IEEE Access*, vol. 6, pp. 55112–55125, 2018.

[30] T. Dong, F. Xue, C. Xiao, and J. Li, Task scheduling based on deep reinforcement learning in a cloud manufacturing environment, *Concurr. Comput. Pract. Exp.*, vol. 32, no. 11, p. e5654, 2020.

[31] J. Li, X. Zhang, J. Wei, Z. Ji, and Z. Wei, GARLSched: Generative adversarial deep reinforcement learning task scheduling optimization for large-scale high performance computing systems, *Future Gener. Comput. Syst.*, vol. 135, pp. 259–269, 2022.

[32] P. Gazori, D. Rahbari, and M. Nickray, Saving time and cost on the scheduling of fog-based IoT applications using deep reinforcement learning approach, *Future Gener. Comput. Syst.*, vol. 110, pp. 1098–1115, 2020.

[33] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, S. U. Khan, and P. Li, A double deep Q-learning model for energy-efficient edge scheduling, *IEEE Trans. Serv. Comput.*, vol. 12, no. 5, pp. 739–749, 2019.

[34] S. Swarup, E. M. Shakshuki, and A. Yasar, Energy efficient task scheduling in fog environment using deep reinforcement learning approach, *Procedia Comput. Sci.*, vol. 191, pp. 65–75, 2021.

Shigen Shen received the BS degree in fundamental mathematics from Zhejiang Normal University, Jinhua, China in 1995, the MS degree in computer science and technology from Zhejiang University, Hangzhou, China in 2005, and the PhD degree in pattern recognition and intelligent systems from Donghua University, Shanghai, China in 2013. He is a professor at School of Information Engineering, Huzhou University, Huzhou, China. His current research interests include the Internet of Things, cyber security, edge computing, and game theory. He has published more than 100 technical papers in many respected journals. He is currently serving as a member of the editorial boards of *CMC-Computers, Materials & Continua* and *Intelligent Automation & Soft Computing*, as well as the editorial review board of the *Journal of Organizational and End User Computing*. He is a senior member of IEEE.

Qi Liu received the BS degree in computer science and technology from Zhuzhou Institute of Technology, China in 2003, and the MS and PhD degrees in data telecommunications and networks from University of Salford, UK in 2006 and 2010, respectively. He is a professor at School of Software, Nanjing University of Information Science and Technology, Nanjing, China. His research interests include context awareness, data communication in MANET and WSN, and smart grid. His recent research work focuses on intelligent agriculture and meteorological observation systems based on WSN.

Lei Zeng received the BS degree in IoT engineering from Nanjing University of Information Science and Technology, China in 2018. He is currently pursuing the MS degree at School of Computer Science, Nanjing University of Information Science and Technology, China. His research interests include edge and cloud computing, task scheduling, and big data.

Xiaodong Liu received the PhD degree in computer science from De Montfort University, UK, and joined Napier in 1999. He is a professor at School of Computing, Edinburgh Napier University, UK. He is the director of the Centre for Information & Software Systems. He is an active researcher in software engineering with an internationally excellent reputation and leading expertise in context-aware adaptive services, service evolution, mobile clouds, pervasive computing, software reuse, and green software engineering.