Research Article

Resource Allocation Strategy Using Deep Reinforcement Learning in Cloud-Edge Collaborative Computing Environment

Junjie Cen and Yongbo Li

1College of Computer Science and Technology, Henan Institute of Technology, Xinxiang, Henan 453002, China
2College of Computer and Information Engineering, Henan Normal University, Xinxiang, Henan 453002, China

Correspondence should be addressed to Junjie Cen; cen@hait.edu.cn

Received 26 April 2022; Revised 9 June 2022; Accepted 25 June 2022; Published 12 July 2022

Copyright © 2022 Junjie Cen and Yongbo Li. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

With the development of technologies such as IoT and 5G, the exponential explosion in the amount of new data has put more stringent requirements on ultra-reliable and low-delay communication of services. To better meet these requirements, a resource allocation strategy using deep reinforcement learning in a cloud-edge collaborative computing environment is proposed. First, a collaborative mobile edge computing (MEC) system model, which combines the core cloud center with MEC to improve the network interaction ability, is constructed. The communication model and computation model of the system are considered at the same time. Then, the goal of minimizing system delay is modeled as a Markov decision process, and it is solved by using the deep Q network (DQN) which is improved by hindsight experience replay (HER), so as to realize the resource allocation with the minimum system delay. Finally, the proposed method is analyzed based on the simulation platform. The results show that when the number of user terminals is 80, the maximum user delay is 1150 ms, which is better than other comparison strategies and can effectively reduce the system delay in complex environment.

1. Introduction

In recent years, mobile devices have become essential tools in our daily life, such as communication, socializing, and entertainment. The demand for mobile computing continues to escalate, resulting in an explosion in the number of mobile devices [1]. However, battery capacity and computational resources are not sufficient to meet users' needs, so cloud computing, which allows the computation tasks to be transmitted to servers with more computing capacity from mobile devices, has been developed greatly to solve the challenge of limited resources for mobile devices [2]. Mobile cloud computing (MCC) combines the advantages of both mobile computing and cloud computing to further address this problem [3]. Meanwhile, [4] MCC enables the use of data from the Internet to extend the capabilities of mobile devices lacking in computing, communication, and caching and to extend effective working time within the limit of battery life.

Nowadays, the main implementation method is to offload tasks such as computing and storage to a remote public cloud platform [5]. However, the traditional architecture of MCC is also facing new challenges. On the one hand, users have to interact with data centers when using mobile applications, and network delay has a significant impact on some delay-sensitive applications depending on the relative distance between the user and the cloud data center. On the other hand, because all data interactions generated by applications must be carried out through the core network, there will be great pressure on the core network during network peak hours [6]. In order to solve the problem of high delay and alleviate the network pressure, mobile edge computing (MEC) technology came into being. The core idea of MEC is to decentralize part of the computing and storage capacity of the data center in MCC to the edge of the network, i.e., close to the user. Hence, the data-processing requirements generated by mobile applications can be executed by the MEC server which is located at the edge of...
local network, and results can be returned without going through the core network and data center [7].

A resource allocation strategy using deep reinforcement learning in a cloud-edge collaborative computing environment is proposed to address the limitation of hardware resources such as server storage and computing as well as the nonuniformity of user distribution, which leads to problems such as higher network delay. Compared with traditional MEC system strategies, the main contributions of this paper are as follows:

1. In order to make full use of the limited capacity of MEC servers while reducing the delay of user-service interaction, a collaborative MEC system model is proposed, which not only greatly relieves the pressure on the core network but also reduces the network delay of applications through the allocation of the cloud computing center and MEC.

2. To address the reward sparse problem caused by complex environments, a special experience replay method, which is named as hindsight experience replay (HER), is introduced to give certain rewards to actions that do not reach the target state as well, so as to accelerate the learning efficiency of agents and guide them to the correct learning direction in order to achieve a global load-optimal strategy.

The remaining sections of this paper are arranged as follows: Section 2 introduces the related research in this field. Section 3 introduces the system model and optimization objectives. Section 4 introduces the resource allocation strategy based on deep reinforcement learning. In Section 5, experiments are designed to verify the performance of the proposed strategy. Section 6 is the conclusion.

2. Related Works

Computing offloading is an essential part in MEC, and the key point of offloading decisions is how they are designed. A flexible and effective network computational resource allocation strategy is an important guarantee for the efficient operation of MEC [8]. In this regard, scholars at home and abroad have already conducted some researches. In [9], a migration strategy of cloud collaborative computing was designed. Simulations verified the feasibility of selecting the optimal migration strategy based on task division. However, the overall computational efficiency still needs to be improved. Gao et al. proposed a hierarchical multi-agent optimization algorithm that combines the genetic algorithm (GA) and multi-agent optimization [10]. This algorithm implements an improved GA by maximizing resource utilization to find a set of service nodes for deploying the requested tasks so as to maximize resource utilization and bandwidth cost in cloud computing. The algorithm pays more attention to resource utilization of cloud computing, and cloud-edge collaboration is not fully considered. In [11], in order to realize effective edge learning on heterogeneous edges with constraints on resources, an online learning strategy was proposed based on the budget-constrained multiarmed slot machine model, which improves the computational efficiency in cloud computing environments. But the performance of bandwidth optimization should be enhanced during resource allocation. A resource allocation strategy for synchronous wireless information and power transmission relay systems under general interference was studied in [12]. A resource allocation coordinate system was established using the $h_2$ method to divide the energy harvesting area and DF area. The optimal value was obtained by applying the golden division method, but the overall communication delay still needs to be improved. A stochastic mixed integer nonlinear programming model was formulated in [13] to jointly optimize task offloading decisions, flexible computational resource scheduling, and radio resource allocation. The proposed problem was decomposed into four separate subproblems using Lyapunov's optimization theory and solved by convex decomposition method and matching game. A low probability of intercept based collaborative power and bandwidth allocation strategy for multi-target tracking was proposed in [14]. But the algorithm is biased toward the optimal allocation of bandwidth resources, and further improvement is required for computational resource allocation. By defining a reward function for each user with consideration of aggregation effects of the other users, Liu et al. developed a multi-agent reinforcement learning framework based on independent learners to solve the problem about resource allocation in the cloud computing environment [15]. However, this algorithm has yet to be further studied for collaborative resource optimization at the cloud edge in some complex situations.

3. System Model and Optimization Objective

3.1. System Scenario. Traditional cloud computing, whether the distributed storage computing is used or not, is still centralized at a macro level, and all service requests still need to be transmitted to the cloud for execution. However, the cloud data center will decide to execute tasks according to the optimization strategy, which makes it difficult for traditional cloud computing to adapt to the changes in users’ requests over time, space, and other factors. And MEC can better deal with this problem.

The core idea of MEC is to offload part of computing and storage capacity of the data center in MCC to the edge of the network, i.e., close to the user. Data-processing requirements generated by mobile applications can be executed by the MEC server, and then results can be directly returned at the edge of their local network without going through the core network and data centers, which not only greatly relieves network pressure on the core network, but also significantly reduces network delay for applications [16]. In addition, as MEC servers usually only serve the area for which they are responsible, they can be more integrated with local information. A typical collaborative MEC scenario is shown in Figure 1, consisting of a core cloud data center and $M$ sub-MEC servers.

As shown in Figure 1, $K$ services are running in the core cloud data center, while each MEC server is split into one area for task requests. The backhaul network connects each area to the core cloud data center, and users connect to the
core cloud data center to access services via the backhaul network. To better serve the users in each area and to reduce the interaction delay and the pressure on the backhaul network, services in the core cloud data center are offloaded to the MEC servers within their capacity, so that users in that area only need to connect to the MEC servers for services. In other words, if the requested service of a user in a certain area is in its local MEC server, the switching delay is produced by the device when connecting to the local MEC server [18]. Otherwise, besides communicating with the local MEC server, it also needs to connect to the MEC server that has recently been deployed with service to obtain it via the local MEC server. Therefore, the delay is correspondingly higher [19].

3.2. Communication Model. In the MEC system model, each user terminal can offload its computation tasks to the MEC server or execute them locally. Each user is associated with the nearest base station, and the MEC server which is connected to that base station can be referred to as the user’s local MEC server. When a user terminal offloads a task to the MEC server, the user needs to transmit the task to the base station associated with it via a wireless link.

Define $U_m$ as the set of user terminals associated with the base station $m$, and define $B_m$ as the bandwidth of the base station $m$. Multiple user terminals choose to offload computation tasks at the same time, and then the wireless bandwidth is evenly allocated to the offloading user terminals to upload data. The uplink transmission rate $v_{nm}$ of the wireless link of the user terminal $n$ can be expressed as follows:

$$v_{nm} = \frac{B_m}{U_m} \log_2 \left(1 + \frac{p_n h_{n,m}}{B_m (U_m) \delta_n} \right),$$  

where $p_n$ is the transmission power of the user terminal $n$ to upload data through the wireless link, $h_{n,m}$ is the channel gain between the user terminal $n$ and the base station $m$, and $\delta_n$ is the power spectral density of white noise.

3.3. Computation Model. For each user terminal, there are three types of execution of computation tasks, which are local execution at the user terminal, execution at the user’s local MEC server, and execution at the user’s neighboring MEC server.

3.3.1. Local Execution Model. When a user terminal $n$ chooses to execute a computation task $\phi_n$ locally, and $f_n^L$ represents the local computing capacity of the user terminal $n$, which is the CPU frequency, then the time delay $T_n^L$ for executing the computation task $\phi_n$ locally can be calculated as follows:

$$T_n^L = \frac{C_n}{f_n^L}$$  

where $C_n$ represents the number of CPU cycles required to execute the computation task $\phi_n$.

3.3.2. Local MEC Server Computing Model. When a user terminal $n$ chooses to execute a computation task $\phi_n$ at the local MEC server, the user $n$ first needs to offload the computation task $\phi_n$ to its associated base station $m$ via the wireless link, and the corresponding transmission delay is generated when uploading the computation task. According to the communication model, the uplink transmission delay $T_{nm}^r$ of user $n$ can be calculated as follows:

$$T_{nm}^r = \frac{D_n}{v_{nm}}$$  

where $D_n$ represents the amount of the input data for the computation task $\phi_n$.

Next, the MEC server in the base station allocates the computational resources to execute the task, which results in the processing delay of the computation task. Let $f_{nm}$ denote the computational resources allocated by the base station $m$ to the user terminal $n$; then, the processing delay $T_{nm}^c$ for the MEC server to execute this task can be written as follows:

$$T_{nm}^c = \frac{C_n}{f_{nm}}$$  

Finally, the results are returned to the user terminal $n$ from the MEC server. Generally, the amount of data returned by the computation task after processing is small, and the data rate of the downlink is high, so the delay due to returning the results can be ignored. From the above analysis, it can be inferred that the total delay when the computation task $\phi_n$ of user terminal $n$ is executed at the local MEC server can be formulated as follows:

$$T_n^\Omega = T_{nm}^r + T_{nm}^c$$  

3.3.3. Neighborhood MEC Server Computing Model. When a user terminal $n$ chooses to execute a computation task $\phi_n$ at a neighboring MEC server $g$, it first needs to offload the computation task $\phi_n$ to its associated base station...
m via a wireless link. Next, the base station m transmits the task to the base station g via a wired link, and then the MEC server in the base station g allocates the corresponding computational resources to execute the task. Finally, the MEC server returns the results to the user terminal n.

Based on the above process, the total delay for executing the computation task at the neighboring MEC server g can be defined as \( T^{\text{n}}_{\text{n,g}} \) which mainly consists of the uplink transmission delay \( T^{t}_{\text{n,m,g}} \), the transmission delay \( T^{t}_{\text{m,g}} \) between base stations, and the processing delay \( T^{\text{c}}_{\text{n,g}} \) in the MEC server. Like the local MEC server computing model, the delay incurred by the return of results to the user can be ignored. As the uplink transmission delay \( T^{t}_{\text{n,m,g}} \) can be calculated in the same way as the local MEC server, it is not written in detail here. Let \( f_{n,g} \) denote the computational resources allocated to the user terminal n by the base station g, and let \( v_{m,g} \) denote the transmission rate between the base station m and the base station g; the processing delay \( T^{\text{c}}_{\text{n,g}} \) in the MEC server can be formulated as follows:

\[
T^{\text{c}}_{\text{n,g}} = \frac{C_{n}}{f_{n,g}}.
\]

The transmission delay \( T^{t}_{\text{m,g}} \) between base stations can be calculated as follows:

\[
T^{t}_{\text{m,g}} = \frac{D_{n}}{v_{m,g}}.
\]

The objective of this work is to optimize the maximum task delay among all users of the system within the computational resource constraints of the MEC server, thus ensuring fairness among users. Therefore, it can be formulated as a joint optimization problem of offloading strategy \( \mathcal{F} = \{ \alpha, \beta, \gamma \} \) and computational resource allocation \( f = \{ f_{n,m}, n \in N, m \in M \} \) for all user terminals, which can be shown as follows:

\[
\begin{align*}
\min & \sum_{n \in N} \gamma_{n,g}, \quad n \in N, g \in M, \quad C1: \alpha + \beta + \sum_{g \in M} \gamma_{n,g} = 1, \quad n \in N, C2: \sum_{m \in M} f_{n,m} \leq F_{n}, \quad m \in M, \quad C4: f_{n,m} \geq 0, \quad n \in N, m \in M,
\end{align*}
\]

where \( M_{n} = (M/m) \) represents the set of adjacent base stations of user terminal n. Constraints C1 and C2 are restrictions on the offloading strategy of each user terminal. Constraint C3 specifies that the sum of computational resources allocated to the user cannot exceed the computing capacity of the MEC server. Constraint C4 ensures that the computational resources allocated cannot be negative.

As there are both discrete and continuous variables in this optimization problem and the objective function is a min–max model, the optimization problem can be regarded as mixed integer nonlinear programming (MINLP) problem, which is NP-hard and non-convex. The solving complexity of such problems is very high, and they cannot be solved in polynomial time.

4. Resource Allocation Strategy Based on Deep Reinforcement Learning

4.1. Markov Decision Process. Markov Decision Progress (MDP) is a mathematical model of sequential decision making and is a combination of Markov process and deterministic dynamic programming, which can be used to model the policy and reward that the agent can achieve in a Markovian environment. The Markov property requires that the next state in the system is only related to the current state and has no relationship to all earlier states. Therefore, a state can be defined as having the Markov property when it does not have historical information and the current state can determine the next or future state. If each state in a random state sequence has the Markov property, then the random sequence can be regarded as a Markov process. And MDP is a Markov process that takes actions and returns into account [20, 21].

MDP is constructed according to the environment and the agent, which can be represented by a quadruplet \( (S, A, P, R) \). S is the state space, representing the set of states of the environment. A is the action space, representing the set of actions that can be performed by agents. \( P \) is the state transfer probability, representing the distribution probability of the current state becoming the next different state after selecting a different action. \( R \) is the reward function, which indicates the reward value obtained by the agent when selecting an action to transfer to next state, and the reward value indicates whether it is suitable to select this action.
under the current state. Additionally, the discount factor \( \lambda \) \((\lambda \in [0, 1])\) is introduced, which is one of the parameters for calculating the cumulative reward. When \( \lambda \) tends to be 0, it means that the future reward is not important and only the current reward needs to be considered. When \( \lambda \) tends to be 1, it means that the future reward is very important, so it is necessary to take both future reward and current reward into consideration.

Mathematically, the interaction between agents with environment over a discrete time series \( T \) is usually described as a Markov process. At each moment \( t \) in \( T \), the agent is in a state and needs to select an action, constituting the set of actions \( A = \{a_0, a_1, \ldots\} \). And a reward value will be obtained when an action is selected, which constitutes the set of reward values \( R = \{r_0, r_1, \ldots\} \).

In MDP, a policy is a mapping between the probabilities of selecting each action in the action space under the current state, indicating the action that the agent should perform based on the current state. There are two main types of policies. The first is the deterministic policy, which means that the agent can only select one action to perform in a certain state. The second is a stochastic policy, which indicates that the agent may perform one or more actions in a certain state. Hence, the policy \( \pi \) can be defined as the probability that the agent selects action \( a \) under state \( s \), at the moment \( t \), which can be written as follows:

\[
\pi(a|s_t) = \Pr[A = a|S = s_t],
\]

where \( \pi(a|s_t) \geq 0 \) and \( \sum \pi(a|s_t) = 1 \).

The state-value function is the expectation of the cumulative reward in the current state \( s_t \), i.e., the expectation of the cumulative reward \( R_t \) after taking the policy \( \pi \) in the current state \( s_t \). The state-value function is mainly used to evaluate each state of a given policy \( \pi \). Therefore, the state-value function is associated with the policy and can be defined as \( Q_\pi(s_t) \), which is the sum of the future reward values when all the actions selected according to policy \( \pi \) in current state \( s_t \) are multiplied by the discount factor \( \lambda \). \( Q_\pi(s_t) \) can be calculated as follows:

\[
Q_\pi(s_t) = E_\pi \left[ \sum_{i=0}^{T} \lambda^i r_{t+i+1} | S = s_t \right].
\]

Correspondingly, the state-action value function represents the expectation of the cumulative reward \( R_t \) when the action \( a_t \) is selected based on policy \( \pi \) under the current state \( s_t \). \( Q_\pi(s_t, a_t) \) represents the sum of future reward values when the action \( a_t \) is selected with the discount factor \( \lambda \) based on policy \( \pi \) in current state \( s_t \), which can be formulated as follows:

\[
Q_\pi(s_t, a_t) = E_\pi \left[ \sum_{i=0}^{T} \lambda^i r_{t+i+1} | S = s_t, A = a_t \right].
\]

Therefore, MDP can be used in the scenario of finding the optimal policy. By means of random sampling and dynamic programming, MDP can be used to solve the problem of maximizing the cumulative reward value. And the proposed method applies deep Q network (DQN) to find the optimal strategy.

### 4.2. DQN-Based Offloading Strategy

The DQN uses experience replay to disrupt the correlation between the sample data, as there is a link between successive action states. But the neural network is a nonlinear model that requires the samples to be independently and homogeneously distributed. Thus, by random uniform sampling in the experience replay buffer, the data distribution is averaged, and the training process is smoothed. Because the off policy learning method is adopted, the parameters of the network generating data samples are different from those of the trained network, which is the Fixed Q-targets network. The two networks have the same structure but different parameters. One of the networks is used to obtain the estimated Q value, and the other network is used to obtain the realistic Q value. The second network will periodically replicate the parameters of the first network, thus reducing the correlation between two Q values to a certain extent and improving the network stability. However, deep learning can only be adapted to a specific environment and model, and these adapted hyperparameters will not be effective due to the large number of training episodes needed to adapt to the new environment. Meanwhile, due to the excessive complexity of the problem and sparse rewards in real-life scenarios, the reward function is required to obtain valid rewards from invalid actions to guide the agent to make better decisions [22].

To deal with the problems mentioned above, HER is introduced, which is a good solution to the problem of sparse rewards and dichotomous rewards, as it allows rewards even if the final goal is not achieved and accelerates the learning process of the agent. It is assumed that the learning trajectory of the agent starts from the initial state \( s_0 \) and actually reaches the state \( \tilde{s} \) while the final goal is state \( s_\infty \). Then, the real learning process of the agent can be written as follows:

\[
\{(s_0, s_\infty, a_0, r_0, s_1), (s_1, s_\infty, a_1, r_1, s_2), \ldots, (s_n, s_\infty, a_n, r_n, \tilde{s})\},
\]

where \( s_0 \) represents the state at time 0 and \( a_0 \) represents the action taken by the agent at time 0. Similarly, \( a_t \) is the action at time \( t \). And \( r_0 \) represents the reward received at time 0. If the target state \( s_\infty \) is replaced by \( \tilde{s} \), then even though the agent does not reach the target state \( s_\infty \), it will still have feedback when it arrives at \( \tilde{s} \). Hence, the learning process can be written as follows:

\[
\{(s_0, \tilde{s}, a_0, r_0, s_1), (s_1, \tilde{s}, a_1, r_1, s_2), \ldots, (s_n, \tilde{s}, a_n, r_n, \tilde{s})\}.
\]

The action taken by an agent at time \( t \) is determined by the current state \( s_t \) jointly with the target state \( s_\infty \), which can be written as follows:

\[
a_t = \pi(\tilde{s}, s_t).
\]

The corresponding rewards can be shown as follows:
Then, each experience from the learning process is placed into HER, including the current state \( s_t \), the action taken \( a_t \), the immediate reward obtained \( r_t \), the next state \( s_{t+1} \), and the final goal \( s_{\infty} \). A hypothetical goal \( \tilde{s} \) is generated by taking a policy, and the experience \( r' \) is calculated jointly with the current state and action and deposited into HER. Here, \( r' \) can be calculated as follows:

\[
r' = R(s_t, a_t, \tilde{s}).
\]

The policy used to generate the hypothetical goal is “future.” Specifically, there are \( L \) number of states which can be observed in the randomly selected episode. These states are used as target states to compute new experience values that will guide the agent in further learning. HER offers a new direction in solving the problem of sparse reward; that is, it is not necessary to achieve specific goals to learn useful experience [24]. It should be noted that the goal which is selected or achieved in the so-called “failed experience” should be related to the final goal in some way. And the learning performance will be very poor if the similarity between two goals is very low [25, 26]. The pseudocode for a deep reinforcement learning algorithm based on multiple objectives and experience replay is shown in Algorithm 1.

The proposed DQN structure is shown in Figure 2.

The structure of the proposed DQN is the same as a normal DQN, where the agent first reaches the state \( s_t \), then inputs data into the neural network to calculate the estimated Q value and select the action with the largest Q value, executes this action to get the reward with the new state, puts the experience into the replay pool, randomly selects the minibatch when there is enough experience in the replay pool, and finally calculates the loss function between the Q value of the main network and Q which is calculated based on the target network.

\[
L(\theta) = E\left( (Q - \tilde{Q}(s_t, a_t; \theta))^2 \right),
\]

\[
Q = r_{t+1} + \gamma \max_a \tilde{Q}(s_{t+1}, a; \theta^\prime).
\]

The parameters in the network are updated by gradient descent, and the target network is periodically updated with parameters [27]. However, it should be noted that the replay pool here uses HER that randomly samples the target as a new hypothetical target in the same episode and then computes a new experience, overcoming the problem of sparse rewards and making the agent learn faster.

5. Experiments and Analysis

Consider a \( 520 \times 520 \) m area with 5 uniformly deployed base stations and \( N \) number of users, where each base station is equipped with a MEC server with 8 CPU cores and the path loss model is \( h = 127 + 30 \log (d[\text{km}]) \) (operating system: Windows 10, CPU: Intel® Core™ i7 3.2 GHz, memory: 8 GB DDR3, 1600 MHz). Define the average rate at which users offload tasks to the MEC server at moment \( t \) of task as

\[
\tau_{nnm} = (1/t)\sum_{\tau=0}^{t-1} v_{nnm}(\tau).
\]

Considering that the number of tasks arriving in each time slot differs in magnitude from the power consumed by the users and the MEC, the parameters are set as follows. The revenue per task is set to be \( 1 \times 10^{-3} \) units/bits, and the power cost per unit is set to be 0.2 units/W. Other simulation parameters are set as shown in Table 1.

5.1. Convergence of Markov Decision Algorithm. The convergence of Markov decision algorithm in different environments is first investigated, and the convergence performance of Markov decision algorithm is shown in Figure 3 when the number of tasks is 20.

As can be seen in Figure 3, the Markov decision algorithm converges quickly to the optimal value after 100 iterations, with a total system utility of approximately 0.2. Its computational complexity is much lower than the number of enumerations in an exhaustive search.

5.2. Comparison with Other Algorithms

5.2.1. Comparison of the Average End-to-End Delay of Different Algorithms. According to Little’s law, the average waiting delay is proportional to the average queuing time, so the average waiting delay can be defined as the sum of the waiting time in the task queues of users and MEC servers. Then, the relationship between task arrival rate and end-to-end delay in the four algorithms is shown in Figure 4.

As illustrated in Figure 4, the algorithm proposed in [15] has the highest average end-to-end delay compared to the other algorithms due to its excessive reliance on cloud computing, while the other three algorithms consider local execution, which reduces the delay to some extent. For the algorithm developed in [12], the average end-to-end delay is higher because it does not consider the queuing threshold, which results in the increase of waiting delay in the queue, and the golden partition method causes more transmission delay. In addition, it can also be found in Figure 4 that when the task arrival rate is small, the performance of [9] is close to that of the proposed algorithm and the difference between these two algorithms broadens gradually when the arrival rate increases. This is due to the fact that the proposed algorithm optimizes the bandwidth resources while considering the power resource allocation, thus reducing the transmission delay of the task to a certain extent. And when the task arrival rate is 4 kbits/slot, the average end-to-end delay is about 450 ms.

5.2.2. Effect of Different Numbers of MEC Servers on Average Delay. As the main optimization objective of the proposed algorithm, the average network delay has an intuitive effect on the performance of the algorithm, and the average network delay indicates the time required for each service request to access the required service. The average network delay for different algorithms is illustrated in Figure 5 when the number of MEC servers is 10, 55, and 100.

It is indicated in Figure 5 that the average communication delay is decreasing as the number of MEC servers
increases, due to the fact that the increase of MEC speeds up the communication. Meanwhile, the proposed algorithm has the shortest average communication delay, which is below 300 m when the number of MEC servers is 100. Due to its cloud-edge collaboration approach, users do not need to offload tasks to a remote cloud computing center when the number of MEC servers is high, and the DQN strategy can get the best resource allocation scheme to reduce the communication delay. In contrast, [15] focuses on cloud computing environment, so the increase of MEC servers is not significant for the improvement of performance. Reference [12] has a higher overall delay which exceeds 600 ms due to the lack of a collaborative cloud-edge allocation model and a high-performance optimization algorithm.

5.2.3. Relationship between the Number of Terminals and the Maximum User Delay. The proposed algorithm is evaluated by the performance metric of maximum user delay, and the relationship between the number of user terminals and maximum user delay for different algorithms is shown in Figure 6.

It can be seen from Figure 6 that the maximum user delay increases with the number of user terminals, and the proposed algorithm has a smaller maximum user delay. The

---

**Algorithm 1:** Pseudocode of DQN algorithm based on multi-objective and experience replay.

```
Initialization:
Initialize experience playback memory;
Initialize behavior value function Q with random weight \( \theta \);
Initialize the target behavior value function \( \tilde{Q} \) with weight \( \tilde{\theta} = \theta \).

Begin
(1) For episode \( i = 1, 2, \ldots, I \)
(2) do The initial observation \( s_1 \) is received and the preprocessing \( s_1 \) is taken as the start state \( x_1 \)
(3) For \( t = 1, 2, \ldots, T \)
(4) do Select behavior \( a_t \) randomly with random probability \( \epsilon \);
(5) Otherwise, select behavior: \( a_t = \arg \max Q(x, a; \theta) \);
(6) Execute actions \( a_t \) in the system to obtain reward \( r_t \) and observe \( s_{t+1} \) at the next moment, and update \( s_{t+1} \) to \( x_{t+1} \);
(7) Store experience to experience playback memory;
(8) Obtain samples in random small batches from playback memory;
(9) Calculate the target Q value of the target DQN;
(10) Update the main DQN by minimizing the loss function \( L(\theta) \);
(11) For network parameter \( \theta \), gradient descent is performed on \( L(\theta) \);
(12) Update target network Q value.
(13) End For
(14) End For
(15) End
```

**Figure 2:** DQN structure.
proposed strategy maximizes the processing efficiency of computation tasks and reduces system delay by combining the advantages of cloud computing and MEC and introducing DQN to design an offloading strategy. When the number of user terminals is 80, the maximum user delay is 1150 ms, which is better than other comparison strategies. On the other hand, although there are abundant resources in the cloud nodes, they are remote from the users and the transmission delay of the tasks is larger. Hence, the maximum user delay in [15] is more than 1240 ms. Therefore, when the number of computational tasks or users is within a certain range, the edge collaboration has better performance than the edge-cloud collaboration.

5.2.4. Relationship between User Terminal Computing Capacity and Maximum User Delay. The effect of user terminal computing capacity on the maximum user delay under different algorithms is depicted in Figure 7.

As shown in Figure 7, the maximum user delay in [15] decreases much more slowly, while the delay of the other three algorithms decreases with the increase of computing capacity. This is because the algorithm proposed in [15] does not consider local execution for users and mainly relies on cloud computing platforms; hence, changes in terminal computing capacity do not have a great effect on user delay. Additionally, it is also illustrated in Figure 7 that the difference between the collaborative scheme of [9] and the proposed algorithm and the non-collaborative scheme of [12] gradually becomes smaller, which is due to the fact that the advantage of the resource allocation algorithm will be less significant as the tasks tend to be executed locally at the terminal and fewer tasks are offloaded to the MEC servers when the terminal computing capacity improves.

Meanwhile, the proposed algorithm uses the multi-objective and HER-based DQN algorithm to find the optimal resource allocation strategy. And the maximum user delay is about 1300 ms when the user terminal computing capacity is

| Simulation parameters | Value |
|-----------------------|-------|
| Bandwidth (MHz)       | 15    |
| Noise power spectral density (dBm-Hz^{-1}) | -174 |
| Maximum transmission power (mW)       | 600   |
| Local maximum computing power (cycle-s^{-1}) | 1 \times 10^8 |
| Maximum computing power of MEC single core (cycle-s^{-1}) | 3 \times 10^8 |
| Slot length (ms)      | 1     |
| CPU cycles required for task calculation (cycle-bit^{-1}) | 1760 |
| Task arrival rate (kbit-slot^{-1}) | [1, 2, 3, 4] |

Figure 3: Convergence of Markov decision algorithm under low load.

Figure 4: Comparison of average end-to-end delay of different algorithms.

Figure 5: Effect of different numbers of MEC servers on average communication delay.
0.6 Gigacycles/s, which can show better performance even when the terminal computing capacity is low.

6. Conclusion

To address the high application delay caused by the limitation of hardware resources such as storage and computing capacity of servers, a resource allocation strategy using deep reinforcement learning in cloud-edge collaborative computing environment is proposed. Based on the collaborative MEC system model, the optimization objective of minimizing system delay is designed, which is solved using HER-improved DQN to obtain the optimal resource allocation scheme. The results based on the simulation platform show that improving DQN by HER can accelerate the learning efficiency of the agents. It converges rapidly to the optimal value after 100 iterations, so the total system utility is about 0.2.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

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

References

[1] C. Tang, C. Zhu, X. Wei, H. Wu, Q. Li, and J. Rodrigues, "Intelligent resource allocation for utility optimization in RSU-empowered vehicular network," *IEEE Access*, vol. 8, no. 1, pp. 94453–94462, 2020.
[2] H. Sun and H. Xi, “Resource optimization technology using genetic algorithm in UAV-assisted edge computing environment,” *Journal of Robotics*, vol. 2022, Article ID 3664663, 8 pages, 2022.
[3] C. Li, Y. Zhang, and Y. Luo, "Adaptive handover based on traffic balancing and multi-dimensional collaborative resource management in MEC environment," *The Journal of Supercomputing*, vol. 78, no. 5, pp. 6752–6787, 2022.
[4] S. Zhou, W. Jadoon, and J. Shuja, “Machine learning-based offloading strategy for lightweight user mobile edge computing tasks,” *Complexity*, vol. 2021, no. 3, pp. 1–11, 2021.
[5] S. K. U. Zaman, A. I. Jehangiri, T. Maqsood et al., "LiMPO: Lightweight Mobility Prediction and Offloading Framework Using Machine Learning for mobile Edge computing," *Cluster Computing*, 2022.
[6] W. Xia and L. Shen, "Joint resource allocation at edge cloud based on ant colony optimization and genetic algorithm," *Wireless Personal Communications*, vol. 117, no. 2, pp. 355–386, 2021.
[7] S. Janakiraman and M. D. Priya, “Improved artificial bee colony using monarch butterfly optimization algorithm for load balancing (IABC-MBOA-LB) in cloud environments,” *Journal of Network and Systems Management*, vol. 29, no. 4, pp. 1–38, 2021.
[8] T. Reis, M. Teixeira, J. Almeida, and A. C. D. Paiva, “A recommender for resource allocation in compute clouds using genetic algorithms and SVR,” *IEEE Lat Am T*, vol. 18, no. 6, pp. 1049–1056, 2020.
[9] K. Wang, “Migration strategy of cloud collaborative computing for delay-sensitive industrial IoT applications in the context of intelligent manufacturing,” *Computer Communications*, vol. 150, pp. 413–420, 2020.
[10] X. Gao, R. Liu, and A. Kaushik, “Hierarchical multi-agent optimization for resource allocation in cloud computing,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 692–707, 2021.
[11] Q. Han, S. Yang, X. Ren, C. Zhao, X. Zhang, and X. Yang, “OL4EL: online learning for edge-cloud collaborative learning on heterogeneous edges with resource constraints,” *IEEE Communications Magazine*, vol. 58, no. 5, pp. 49–55, 2020.
[12] J. Li, K. Zhao, X. Ding, and W. G. Shi, “Resource allocation strategy of SWIPT relay under general interference,” *Wireless Personal Communications*, vol. 112, no. 3, pp. 1–15, 2020.
[13] Q. Zhang, L. Gui, F. Hou, J. Chen, F. Zhu, and F. Tian, “Dynamic task offloading and resource allocation for mobile-edge computing in dense cloud RAN,” IEEE Internet of Things Journal, vol. 7, no. 4, pp. 3282–3299, 2020.

[14] C. Shi, L. Ding, F. Wang, S. Salous, and J. Zhou, “Low probability of intercept-based collaborative power and bandwidth allocation strategy for multi-target tracking in distributed radar network system,” IEEE Sensors Journal, vol. 20, no. 12, pp. 6367–6377, 2020.

[15] X. Liu, J. Yu, Z. Feng, and Y. Gao, “Multi-agent reinforcement learning for resource allocation in IoT networks with edge computing,” China Communications, vol. 17, no. 9, pp. 220–236, 2020.

[16] T. Pham and T. Nguyen, “Optimization of resource management for NFV-enabled IoT systems in edge cloud computing,” IEEE Access, vol. 8, no. 4, pp. 178217–178229, 2020.

[17] J. Lin, D. Cui, Z. Peng, Q. Li, and J. He, “A two-stage framework for the multi-user multi-data center job scheduling and resource allocation,” IEEE Access, vol. 8, no. 9, pp. 197863–197874, 2020.

[18] A. Montazerolghaem, M. H. Yaghmaee, and A. Leon-Garcia, “Green cloud multimedia networking: NFV/SDN based energy-efficient resource allocation,” IEEE Transactions on Green Communications and Networking, vol. 4, no. 3, pp. 873–889, 2020.

[19] N. C. Luong, Y. Jiao, P. Wang, D. Niyato, D. I. Kim, and Z. Han, “A machine-learning-based auction for resource trading in fog computing,” IEEE Communications Magazine, vol. 58, no. 3, pp. 82–88, 2020.

[20] M. Kolhar, F. Al-Turjman, J. Alameen, and A. A. Lameen, “Unified dynamic bandwidth allocation scheme for ethernet passive optical network,” IEEE Access, vol. 8, no. 3, pp. 216176–216184, 2020.

[21] Y. Liao, L. Shou, Q. Yu, Q. Ai, and Q. Liu, “Joint offloading decision and resource allocation for mobile edge computing enabled networks,” Computer Communications, vol. 154, no. 8, pp. 361–369, 2020.

[22] X. C. Chen, Y. C. Zhou, Y. Yang, and L. Lv, “Hybrid fog/cloud computing resource allocation: joint consideration of limited communication resources and user credibility,” Computer Communications, vol. 169, no. 6, pp. 48–58, 2021.

[23] A. Mahdi, Y. Mina, R. Milad, J. Alireza, and R. K. Mohammad, “Efficient resource management and workload allocation in fog–cloud computing paradigm in IoT using learning classifier systems,” Computer and Communications, vol. 153, no. 4, pp. 217–228, 2020.

[24] W. Zhang, Z. Zhang, S. Zeadally, H. C. Chao, and V. C. M. Leung, “Energy-efficient workload allocation and computation resource configuration in distributed cloud/edge computing systems with stochastic workloads,” IEEE Journal on Selected Areas in Communications, vol. 38, no. 6, pp. 1118–1132, 2020.

[25] F. Shokri Habashi, S. Yousefi, and B. G. Ghalebsaz Jeddí, “Resource allocation mechanisms for maximizing provider’s revenue in infrastructure as a service (IaaS) cloud,” Cluster Computing, vol. 24, no. 3, pp. 2407–2423, 2021.

[26] M. Mukherjee, S. Kumar, C. X. Mavromoustakis et al., “Latency-driven parallel task data offloading in fog computing networks for industrial applications,” IEEE Transactions on Industrial Informatics, vol. 16, no. 9, pp. 6050–6058, 2020.

[27] W. Hussain, O. Sohaib, M. Naderpour, and H. H. Gao, “Cloud marginal resource allocation: a decision support model,” Mobile Networks and Applications, vol. 25, no. 4, pp. 1418–1433, 2020.