A New Task Offloading Algorithm in Edge Computing

Zhenjiang Zhang[1](corresponding author), Chen Li[1], ShengLung Peng[2], Xintong Pei[1]
[1]School of Electronic and Information Engineering, Key Laboratory of Communication and Information Systems, Beijing Municipal Commission of Education, Beijing Jiaotong University, China
[2]National Taipei University of Business, China
zhangzhenjiang@bjtu.edu.cn, 17120075@bjtu.edu.cn, slpeng@ntub.edu.tw, 19120103@bjtu.edu.cn,

Abstract—In the last few years, the Internet of Things (IOT), as a new disruptive technology, has gradually changed the world. With the prosperous development of the mobile Internet and the rapid growth of the Internet of Things, various new applications continue to emerge, such as mobile payment, face recognition, wearable devices, driverless, VR/AR, etc. Although the computing power of mobile terminals is getting higher and the traditional cloud computing model has higher computing power, it is often accompanied by higher latency and cannot meet the needs of users. In order to reduce user delay to improve user experience, and at the same time reduce network load to a certain extent, edge computing, as an application of IOT, came into being. In view of the new architecture after dating edge computing, this paper focuses on the task offloading in edge computing, from task migration in multi-user scenarios and edge server resource management expansion, and proposes a multi-agent load balancing distribution based on deep reinforcement learning DTOMALB, a distributed task allocation algorithm, can perform a reasonable offload method for this scenario to improve user experience and balance resource utilization. Simulations show that the algorithm has a certain adaptability compared to the traditional algorithm in the scenario of multi-user single cell, and reduces the complexity of the algorithm compared to the centralized algorithm, and reduces the average response delay of the overall user. And balance the load of each edge computing server, improve the robustness and scalability of the system.

Keywords： Edge Computing, Internet of Things, Task Offload, Load Balancing, Deep Reinforcement Learning

1. Introdation

Over the past decade, mobile communication technology has evolved from the third/fourth generation to the fifth generation of mobile communication today. The rapid development of communication technology brings re-transmission speed and shorter delay. At the same time, integrated circuits continue to develop to smaller sizes and integration levels, while the computing power and storage capacity of chips are still rising. All kinds of intelligent hardware such as smart phones, tablet PCs, VR devices, wearable smart devices, etc. were born in the moment. And human society is also moving from the era of mobile Internet to the era of Internet of Everything. For the purpose of perceiving and driving the world, the IoT has gradually developed into a multi-disciplinary ecosystem, which is widely used in various scenarios requiring real-time data processing and feedback. Driven by the IoT, various smart edge devices have been rapidly popularized, and various new mobile applications such as navigation, mobile payment, face recognition, VR/AR, etc. have followed one after another [1] . However, it is accompanied by exponential growth of data traffic and the generation of a large number of computationally intensive tasks, which poses huge challenges to network bandwidth and servers. The traditional computing paradigms adopted by traditional cloud computing are mostly centralized computing models. Linearly expanded cloud computing services cannot efficiently handle the massive data and computing tasks generated by exponentially growing edge devices [2]. It faces problems such as real-time, accumulation, and bandwidth occupation. Therefore, in order to meet the needs of real-time operation, low latency requirements and high quality of service (QoS) scenarios, edge computing emerged as an application paradigm of the IoT.

As a new distributed computing paradigm, edge computing enables computing and data to be stored closer to edge devices, thereby changing the response time of computing tasks and greatly reducing the pressure on network bandwidth and cloud centers, reducing the productivity of edge devices and improving service quality for users.

Task offloading refers to the user equipment processing some computationally intensive applications and uploading the data processing these applications to the edge server through wireless transmission under the condition of weighing continuous or other indicators. Resource allocation refers to the edge server for these uploads the processing application allocates certain computing resources, in this way to obtain continuous or gradual replacement, providing a better user experience.

The problem to be solved by task offloading is for each user, and determine whether the tasks generated by it need to be offloaded, and considering the dense deployment of edge servers, the restricted computational load of distributed servers, so it must be solved for each offload processing the question of how much computing resources are allocated by the application. These two issues need to be considered together to produce better results.

Usually, an initial part of computing offloading and resource allocation is also a key part of deciding whether to offload, that is, offloading decision. After determining whether to uninstall, the next question to consider is how much and what should be uninstalled. In general, the possible decisions to calculate the offload may have the following situations, as shown in Figure 1:

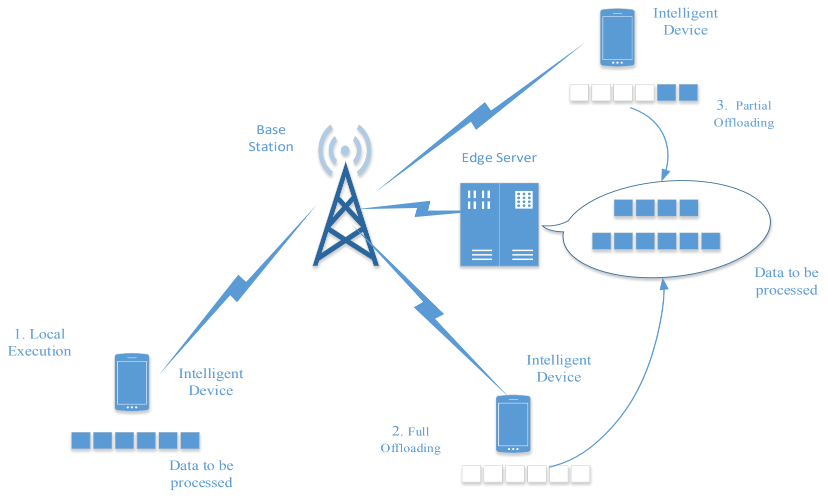


Figure 1 Task offloading method comparison

(1) Local Execution: That is, the entire calculation process is completed locally. This situation is generally aimed at tasks with low computing power requirements.

(2) Full Offloading: The entire calculation is connected to the base station via wireless channels and then offloaded and migrated to the edge server for calculation and processing. This method is also called the complete uninstallation problem and the binary uninstallation problem. This problem assumes that the application of the edge service cannot be split, and can only choose to perform local computing or offload to the edge server to perform the calculation.

(3) Partial Offloading: Under the premise that the calculation can be split, part of the calculation is processed locally, and the other part is offloaded to the edge server for processing.

In this paper, the task offloading mechanism under the multi-user single-cell scenario is mainly studied. In the scenario of a single cell, multiple users are connected to an edge server through a single LTE macro base station, and the edge server can schedule tasks to other edge computing servers connected thereto. Aiming at the competition and selfishness that may occur when multiple users uninstall their computing tasks, a global load balancing penalty factor is introduced to minimize the response time of global user tasks and make the load on each edge server relatively balanced. In addition, in view of the characteristics of dimensional explosion, scalability and poor dynamics faced by the centralized task scheduling as the number of users increases, an algorithm model for centralized training and distributed operation is proposed. By establishing each user as a Markov game model, a distributed task offloading algorithm DTOMALB based on multi-agent and load balancing is proposed.

The following is the structure of this paper. The second section is related work. The third section is the system model. This part mainly describes the single-cell multi-user scenario, communication model, and calculation model, and transforms the task offloading problem into a target optimization problem. The fourth section mainly proposes new algorithms. A task offloading algorithm based on multi-agent is proposed to optimize the user's response delay and balance the load, and improve the server's resource utilization. And through the centralized training and distributed operation mode, it solves the various problems that the centralized scheduling will produce and improves the robustness of the system. The fifth section is the experiment and results and discussions, mainly include the experimental method to verify the feasibility and effectiveness of DTOMALB, and the analysis of the simulation results. It also includes some discussions after the analysis of the results, discussing the meaning of the research results in the context of existing research, and emphasizing the limitations of the research. The last section is the conclusion.

2. Related work

In an IoT network based on edge computing, task offloading is a major way to solve the limitation of edge device computing, storage, and power resources in edge computing. The edge device can offload part or all of the computing tasks to the edge computing server, thereby speeding up the processing speed of the tasks, saving the energy of the device, and reducing the response time. The main issues that need attention are whether to uninstall, when to uninstall, how many computing tasks should be uninstalled, and where to uninstall. The offload calculation will bring additional communication overhead, such as transmission delay and energy consumption caused by communication. A lot of research is devoted to the optimal offloading strategy for different scenarios and different optimization goals.

Wang et al. proposed a joint base station cache and D2D offloading algorithm based on Q learning for edge computing architecture. They applied Q learning to distributed cache replacement strategies according to content popularity [3].

Flores et al. designed and implemented a task delegation or code offloading framework for a mobile cloud computing model. When task delegation, resource-intensive mobile tasks are delegated asynchronously by directly invoking services. When the code is uninstalled, the mobile application is partitioned and analyzed, complex computing operations are identified, and they are offloaded to cloud-based computing to improve the overall system performance [4].

You et al. studied multi-user edge computing scenarios based on time division multiple access (TDMA) and orthogonal frequency division multiple access (OFDMA). Under the constraint of computing delay, the optimal resource allocation problem is expressed as a convex optimization problem that minimizes weighting and mobile energy consumption. For a cloud with limited capacity, a suboptimal resource allocation algorithm is proposed to reduce the complexity of the calculation threshold [5].

Bi et al. studied the offloading problem in multi-user scenarios, decoupled the combination of multi-user computing mode selection and the strong coupling of transmission time allocation, and proposed a simple two-segment search algorithm to obtain conditional optimal time allocation And on this basis, a coordinate descent method was designed to optimize the mode selection [6].

Guo et al. proposed an energy-saving dynamic offloading and resource scheduling strategy to reduce energy consumption and shorten the application completion time. Under the premise of meeting the task dependency requirements and completion time limit constraints, the problem is transformed into an energy efficiency cost minimization problem, and the problem is decoupled into three subtasks of calculation offload selection, clock frequency control, and transmission power allocation. [7] .

In response to the current MEC offloading problem, Wang et al. proposed a new offloading framework based on deep reinforcement learning, which can automatically infer the optimal offloading strategy in different scenarios according to the characteristics of the offloading task to minimize The overall service delay [8].

Xu et al. incorporated renewable energy into mobile edge computing and proposed an effective resource management algorithm based on reinforcement learning to learn dynamic optimal strategies for dynamic load offloading and edge server configuration to minimize long-term system costs [9].

Dong et al. proposed an intelligent offloading system for vehicle edge computing based on deep reinforcement learning. The offloading system includes a task scheduling module and a resource allocation module. With the goal of maximizing the quality of experience (QoE), a joint optimization problem of two modules was established, and this problem was solved through DQN [10].

The scene studied by Liu et al. is the Vehicle Edge Computing (VEC) network. In this paper, an efficient computational offloading scheme is proposed for the user, which is transformed into an optimization problem to maximize the use of the proposed VEC network, taking into account the randomness of vehicle traffic. In this paper, dynamic communication requirements and time-varying communication conditions are expressed as semi-Markov processes, and an algorithm based on Q-learning is proposed. Later, in order to avoid the problem of dimensionality guarantee, an algorithm based on Deep Reinforcement Learning (DRL) was proposed to obtain the optimal computing offload and resource allocation strategy [11].

3. System Model

In a single-cell multi-user scenario, multiple users N={1,2,3,...,N} perform wireless communication through a single Long Term Evolution (LTE) macro base station. The base station can be divided into multiple sub-channels for different users. At the same time, the base station is connected to multiple edge servers M={1,2,3,...,M} to provide users with computing resources to help them process computing tasks, as shown in Figure 2. Each user can be a mobile phone, computer, wearable smart device, etc., all with different computing capabilities. At the same time, the computing power and load of each edge server are also different. In addition, computing tasks generated from multiple user equipments may also be competitive for limited computing resources. Since each user is selfish, he wants to offload his user tasks to the best performing server for calculation. However, when a large number of concurrent user tasks reach a certain high-performance edge server at the same time, the server will not be able to allocate the required computing resources for all user tasks, but it will affect the user experience. Although some servers have low performance, they are in a relatively idle state and can provide relatively superior computing resources, but they are not utilized, resulting in a waste of resources. Therefore, when the task distribution is relatively balanced, not only can the resource utilization rate of each server be improved, but also the overall user service quality can be improved. Under the constraints of limited resources, for heterogeneous user equipment and different user tasks, how to reasonably allocate the computing resources of the edge server to each task, and to ensure that the user experience of the user to balance the load of each edge server is urgent to solve The problem.

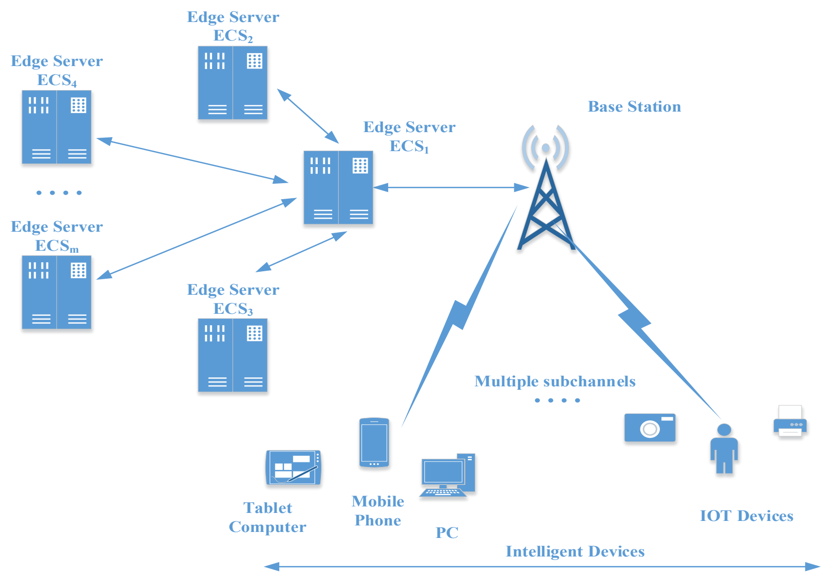


Figure 2 Single-cell multi-user scenario in edge computing

We assume that there are n={1,2,3,...,N} user devices UD (User Device) and an evolved base station (eNodeB, eNB) in a total cell. It is directly connected to an edge computing server ECS through optical fiber, with the number m=1. At the same time, the edge server is also connected to other edge computing servers of the cell number m={2,3,4...,M} through optical fibers, which can provide users with computing resources. It is assumed that at each moment, each user equipment will generate a computationally intensive task, and can choose to calculate locally or offload to the edge server directly connected to the base station through the base station or the remaining edge servers connected to the edge server. Define the uninstall decision vector of the nth user as:

$$\mathcal{A}_n = [a_{n,0}, a_{n,1}, a_{n,2}, \dots, a_{n,M}] \tag{1}$$

$a_{n,m} \in \{0,1\}$。$a_{n,m} = 0$ means do not offload to the m-th edge computing server，$a_{n,m} = 1$ means offload to the m-th edge computing server. $a_{n,1} = 1$ means the task will be offloaded to the edge computing server directly connected to the eNB. Especially, $a_{n,0} = 1$ indicates that the task is calculated locally. Assuming that the task cannot be split, for the task generated by user n, the offload vector has the following constraints:

$$\sum_{m=0}^{M} a_{n,m}(t) = 1, \ \forall n \in N \tag{2}$$

The decision space of all users is:

$$\mathcal{A} = [\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \dots, \mathcal{A}_N] = \begin{bmatrix} a_{0,0}, a_{0,1}, a_{0,2}, \dots, a_{0,M} \\ a_{1,0}, a_{1,1}, a_{1,2}, \dots, a_{1,M} \\ a_{2,0}, a_{2,1}, a_{2,2}, \dots, a_{2,M} \\ \cdot \\ \cdot \\ \cdot \\ a_{N,0}, a_{N,1}, a_{N,2}, \dots, a_{N,M} \end{bmatrix} \tag{3}$$

We use the length of the task and the number of calculation cycles required by the task to describe it.

$$S_n \triangleq (L_n, C_n) \tag{4}$$

L_n is the length of the calculation task, including the calculation task request, data, code, etc. C_n is the number of CPU cycles required to complete the calculation task.

A. Communication model

In the current scenario, multiple users share an eNB, so interval interference can be ignored. Assuming that N users choose to offload the calculation tasks generated at the same time, the bandwidth of the wireless channel will be evenly distributed to the UD. Then the upload rate available for the nth UD is:

$$r_n = \frac{W}{N} log_2(1 + \frac{p_n g_n}{\frac{W}{N} N_0}) \tag{5}$$

Where W is the bandwidth of the entire wireless channel, p_n is the upload power of user equipment n, g_n is the gain of the wireless channel allocated to user equipment n, and N_0 is the variance of the complex Gaussian white noise channel

B. Computation model

(1) Local computation model

For user n, when the offloading variable $a_{n,0} = 1$, local operation will be performed. According to its task tag S_n, the total number of cycles C_n required to execute the task can be known. Then the time delay for the user task S_n to execute locally can be expressed as:

$$t_n^{local} = \frac{C_n}{f_{local}} \tag{6}$$

(2) Offloading computation model

For a task generated by a user n, when the offloading variable $a_{n,m}(t) = 1$, it means that the task offloads the task to the m-th edge server. If m=1, the m-th base station and the corresponding edge computing server perform calculations, otherwise it will be forwarded again to reach the m-th server. User tasks are uninstalled through the following three steps:

1. User n uploads the task to the edge server ECS$_1$ through the uplink allocated by the base station after the time t_n^{up}:

$$\mathrm{t}_{\mathrm{n}}^{\mathrm{up}} = \frac{L_n}{r_n} \tag{7}$$

Where r_n is upload rate.

2. Determine whether m is 1. If it is 1, then calculate in ECS$_1$, and return the calculation result after the computation time $t_{n,1}^{comp}$.

3. If not, through ECS$_1$, the task is forwarded to the edge computing server ECS$_m$ through time $t_{n,m}^{comp}$, and after the calculation time, the result is obtained and returned.

For each edge server, assuming that it uses the first-come-first-served strategy to server the arriving computing tasks, the response delay in the server is the delay in queue plus the calculation delay:

$$t_{n,m}^{resp} = t_{n,m}^{wait} + t_{n,m}^{comp} = \frac{\sum_{i\in k} C_i}{f_m} + \frac{C_n}{f_m} \tag{8}$$

If m！=1, it will take $t_{n,m}^{tran}$ to forward the task：

$$t_{n,m}^{tran} = \frac{L_n}{\mu} \tag{9}$$

Where μ is the forwarding rate between edge servers.

The delay in the entire task offloading process can be expressed as:

$$t_n^{off} = \begin{cases} \mathrm{t}_{\mathrm{n}}^{\mathrm{up}} + t_{n,m}^{resp}, m = 1 \\ \mathrm{t}_{\mathrm{n}}^{\mathrm{up}} + t_{n,m}^{resp} + t_{n,m}^{tran}, m! = 1 \end{cases} \tag{10}$$

The delay that users spend in computing their tasks is:

$$t_n = \begin{cases} \frac{C_n}{f_{local}}, m = 0 \\ \mathrm{t}_{\mathrm{n}}^{\mathrm{up}} + t_{n,m}^{resp}, m = 1 \\ \mathrm{t}_{\mathrm{n}}^{\mathrm{up}} + t_{n,m}^{resp} + t_{n,m}^{tran}, m! = 1 \end{cases} \tag{11}$$

For the overall offloading process, for each user, it is desirable to minimize the delay of their respective users. For the entire system, it is desirable to allocate user tasks reasonably so that the overall average user delay is minimized. In summary, the optimization problem can be modeled as:

$$P1: min\ \frac{1}{N}\sum_{n\in N} t_n$$

$$\mathrm{s.t.} \sum_{m=0}^{M} a_{n,m}(t) = 1$$

$$\forall n \in N, \forall m \in M \tag{12}$$

The above problem is an objective optimization problem. In essence, it is a combinatorial optimization problem. It is NP-hard to solve the complexity. Traditional methods are more difficult to solve, mostly brute force search or heuristic algorithms. In order to improve the adaptability of the model, this chapter also uses reinforcement learning to solve this problem.

4. Deep Reinforcement Learning-based Solution

Reinforcement learning is a field in machine learning. The biggest difference from supervised learning is that it does not require artificial labeling. The core lies in the exploration of the unknown environment and the use of known knowledge. Reinforcement learning through continuous trial and error, continuous interaction with the environment, and get rewards or punishment from the environment, and then obtain learning information to update its own model. It learns a behavior strategy. After a certain amount of training, it can make decisions to maximize long-term returns based on environmental conditions. After the introduction of deep learning, reinforcement learning can handle high-dimensional actions and states, learning efficiency is greatly improved, and to a certain extent, the limitations of reinforcement learning are broken. In this paper, in order to make the model adaptively learn to offload decisions, deep reinforcement learning algorithms will also be used to solve the task offloading problem in edge computing.

Deep reinforcement learning In order to apply deep reinforcement learning algorithms, a Markov decision process model needs to be established. The three elements of the model are system state, action and reward function.

A. System State

In the current state, a total of $n \in \{1,2,...N\}$ users will offload tasks, and the number of available edge computing servers is $m \in \{1,2,...M\}$. In each state $t \in \{0,1,2,...T\}$, each user has a computation-intensive task $u_n(t)$ that needs to be calculated.

The edge server ECS$_m$ may provide computing resources for multiple user tasks at the same time, and the user tasks are queued in sequence in the computing queue. Assuming that there are currently k user tasks queued in the task queue of ECS$_m$, the computing power of ECS$_m$ is and the computing power of ECS$_m$ is f_m. Define the current load factor of the edge server ECS$_m$ as the ratio of the required computing cycle of all tasks in the current edge server to the computing capacity of the server:

$$LD_m(t) = \frac{\sum_{i \in k} c_i}{f_m} \quad (13)$$

Define the current system state as the system load factor of each edge server, that is, how many tasks are queued and calculated in the system:

$$\chi(t) = [LD_1(t), LD_2(t), LD_3(t), \ldots, LD_m(t)] \quad (14)$$

Assuming that each edge computing server has k user tasks in the current state, its load factor is:

$$LD_m(t) = \frac{\sum_{i \in k} c_i}{f_m} \quad (15)$$

Since each user has m servers that can be uninstalled, the entire state space is M^{N}. It can be seen that as the number of users increases, the state space of the entire system will show an exponential growth.

B. System Action

Define the current system action as an offload vector for all user tasks:

$$\mathcal{A}_{\mathrm{n}}(\mathrm{t}) = [a_{n,0}, a_{n,1}, a_{n,2}, \ldots, a_{n,M}] \quad (16)$$

And have the following constraints:

$$\sum_{m=0}^{M} a_{n,m}(t) = 1, \ \forall n \in N \quad (17)$$

The action space of the entire system is:

$$\mathcal{A}(\mathrm{t}) = [\mathcal{A}_1(\mathrm{t}), \mathcal{A}_2(\mathrm{t}), \mathcal{A}_3(\mathrm{t}), \ldots, \mathcal{A}_{\mathrm{N}}(\mathrm{t})] =$$

$$\begin{bmatrix} a_{0,0}, a_{0,1}, a_{0,2}, \dots, a_{0,M} \\ a_{1,0}, a_{1,1}, a_{1,2}, \dots, a_{1,M} \\ a_{2,0}, a_{2,1}, a_{2,2}, \dots, a_{2,M} \\ \cdot \\ \cdot \\ \cdot \\ a_{N,0}, a_{N,1}, a_{N,2}, \dots, a_{N,M} \end{bmatrix} \tag{18}$$

As can be seen from the above formula, the action space of the entire system is also quite large, and its dimension is $\mathrm{M} \times \mathrm{N}$. The action space is M^{N}. It can be seen that as the number of users increases, the dimension of system actions will continue to increase, and the system action space will increase exponentially.

C. System reward function

Since each user is selfish, it is easy for multiple users to select the same edge server for uninstallation at the same time, resulting in a decline in the user experience of each user. In order to improve the sociality of user decision-making and the adaptability of the model, we propose a task offloading algorithm based on load balancing and reinforcement learning (TOLB-RL) . That is, when setting the reward function in reinforcement learning, the system load balancing coefficient is introduced, so that tasks can be more evenly distributed, the training speed of the model is accelerated, and the average delay of the overall system user is reduced.

For the administrators of edge computing servers, they hope to improve the resource utilization of the servers and make the load of each server relatively balanced, so as to avoid all tasks being concentrated on individual servers for calculation, resulting in waste of resources. Define the load balancing factor of the system as the variance of the load of each server:

$$var(LD) = \frac{\sum_{m\in M}(LD_m - |E|)^2}{M} \tag{19}$$

Where $|\mathrm{E}|$ is the average of all edge server load factors.

Whenever the system takes an action in the current state, it will get an instant feedback of the action. In reinforcement learning, it is generally expected to maximize rewards. According to question P1, every user wants to minimize their task offload delay. Therefore, the inverse number of the delay can be used as a reward after the decision. At the same time, considering the load balancing coefficient of the system, when defining the reward function, the load balancing coefficient is used as the penalty term of the reward function. If the action causes the overall load balancing coefficient to increase, the penalty is increased, otherwise the penalty is reduced. We define the instant reward function as:

$$R(t)_n = -\omega_1 t_n - \omega_2 var(LD) \tag{20}$$

This reward function comprehensively considers user delay and system load balancing coefficient. The value depends on the current system status and the actions taken by the user. Where ω_1 and ω_2 represent the average task response time and the weight of the edge server load balancing, respectively. Here, since the optimization goal is to minimize the time delay and minimize the load factor, reinforcement learning is generally to maximize the reward during optimization. Therefore, the entire reward function is a negative number. When the maximum reward of the entire system is obtained, it is equivalent to a minimum delay and a load balancing factor. Define the system utility as the overall system reward.

D. Multi-agent Reinforcement Learning

The centralized DQN[12] needs to observe the global state, so each user's offload decision requires agent observation, which has poor scalability and causes the problem of dimensional explosion. The speed of training and execution is relatively slow. In addition, for the centralized algorithm, once the central agent node making the decision fails, the entire system will be paralyzed and unable to operate, and its reliability is relatively low. Therefore, we will introduce a multi-agent reinforcement learning algorithm, which treats each user as a separate agent and can make task offload decisions for its own tasks, rather than relying solely on a centralized single agent.

For multi-agent reinforcement learning, first, a Markov Game model needs to be established. Markov game can be described by $(n, S, A_1, \dots, A_n, R_1, \dots, R_n,)$:

- n is the number of agents, in this section is the number of users N.
- S is the system state, generally refers to the joint state of multiple agents, that is, the joint state of each agent. In this section, users share the load status of the current edge computing server, which can be expressed as:

$$\chi(t) = [LD_1(t), LD_2(t), LD_3(t), \dots, LD_m(t)] \tag{21}$$

- R_i is the instant reward function of each agent, that is, the reward obtained in the next system state s' after the joint action $(A_1, \dots, A_n)$ taken by multiple agents in the current state s.

The reward function completely describes the relationship between multiple agents. It should be noted that the reward function here is the reward function of each agent. When the reward function of each agent is consistent, that is $R_1 = R_2 = \dots = R_n$, it means that there is a complete cooperative relationship between the agents; when there are only two agents, and the reward function is opposite, that is $R_1 = -R_2$, it means that the agent's The relationship is completely competitive; when the reward function is between the two, it is a mixed relationship of competition and cooperation.

Figure 3 is a multi-agent reinforcement learning system. Multi-agents act at the same time, and under joint action, the entire system will be transferred, and each agent will be rewarded immediately.

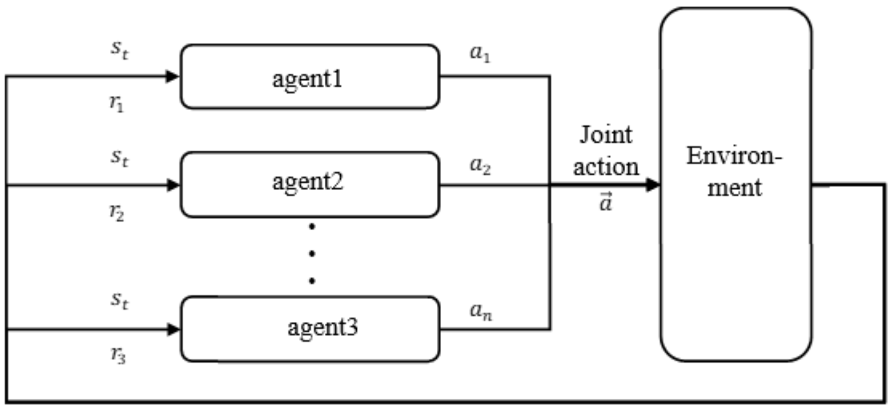


Figure 3 Multi-agent reinforcement learning system

Directly adopting the centralized reinforcement learning algorithm DQN will cause the action dimension to be too large and difficult to converge. In order to solve the above problems, the DTOMALB algorithm is proposed to solve the above problems, which is the DDPG[15] algorithm in the case of multi-agents combined with the load balancing factor.

DDPG is a deterministic strategy gradient algorithm based on the actor-critic model. It also combines the advantages of DQN, using Target network and experience playback.

The reinforcement learning method based on value function introduces the action value function $\hat{q}$ by approximating the value function. This function is often approximated by a neural network with parameter w, and accepts state s and action a as inputs. After calculation, the approximate action value is obtained, and then the action is selected by the maximum value:

$$\hat{q}(s,a,w) \approx q_{\pi}(s,a) \tag{22}$$

In the policy-based reinforcement learning method, a policy function is learned, and the action probability is directly obtained through the input state. At this time, the strategy π can be described as a function containing the parameter θ, namely

$$\pi_{\theta} = P(a|s,\theta) \approx \pi(a|s) \tag{23}$$

The optimization goal is:

$$J(\theta) = V_{\pi\theta}(s_1) = E_{\pi\theta}(G_1) = E_{\pi\theta}[(r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots] \tag{24}$$

Finally, the gradient to derive θ can be expressed as:

$$\nabla_{\theta} J(\theta) = E_{\pi\theta}(\nabla_{\theta} log\pi_{\theta}(s,a) Q_{\pi}(s,a)) \tag{25}$$

When using the Monte Carlo method to update, for each time step in each Monte Carlo sequence, the formula is used to update the parameters of the network:

$$\theta = \theta + \alpha \nabla_{\theta} log\pi_{\theta}(s_t,a_t) v_t \tag{26}$$

When using the time difference method to learn the approximate function $Q_{\pi}(s,a)$ of the real action value function, it evolves into the Actor-Critic algorithm. The Actor network is responsible for selecting actions, and the Critic network is responsible for learning the real action value function, so as to guide the Actor to select actions.

The deterministic strategy adopted in the DDPG algorithm is to directly output actions according to the current state:

$$a = \pi(s|\theta^{\mu}) \tag{27}$$

Random strategy, the strategy output is the probability of action. For example, the A2C[13] algorithm uses a normal distribution to sample the action, that is, each action has a probability of being selected. Random strategy integrates exploration and improvement into one strategy, but requires a lot of training data.

In deterministic strategies, the output of the strategy is an action, which requires less data to be sampled and high algorithm efficiency, but it is impossible to explore the environment. Since the deterministic strategy cannot explore the environment, the DDPG algorithm utilizes the off-policy learning method. Off-policy means that the sampling strategy and the improved strategy are not the same strategy. Similar to DQN, samples are generated using a random strategy and stored in the experience playback mechanism. Samples are randomly selected during training. The improvement is the current deterministic strategy. The entire deterministic strategy learning framework adopts the AC method.

In DDPG, a deep neural network with parameters θ^{μ} and $\theta^{Q}$is used to represent the deterministic strategy $a = \pi(s|\theta^{\mu})$ and the action value function $Q(s,a|\theta^{Q})$. Among them, the strategy network is used to update the strategy, corresponding to the actors in the AC framework; the value network is used to approximate the value function of the state action pair, and provides gradient information, corresponding to the critics in the AC framework. The objective function is defined as the total return with discounts:

$$J(\theta^{\mu}) = E_{\theta^{\mu}}[r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots] \tag{28}$$

The objective function is optimized end-to-end by stochastic gradient method. Silver et al. [16] proved that the gradient of the objective function with respect to θ^{μ} is equivalent to the expected gradient of the Q-value function with respect to θ^{μ}:

$$\frac{\partial J(\theta^{\mu})}{\partial \theta^{\mu}} = E_s[\frac{\partial Q(s,a|\theta^{\mu})}{\partial \theta^{\mu}}] \quad (29)$$

According to the deterministic strategy $a = \pi(s|\theta^{\mu})$, the following formula can be obtained:

$$\frac{\partial J(\theta^{\mu})}{\partial \theta^{\mu}} = E_s[\frac{\partial Q(s,a|\theta^{\mu})}{\partial \theta^{\mu}} \frac{\partial \pi(s|\theta^{\mu})}{\partial \theta^{\mu}}] \quad (30)$$

The Actor network will update the parameters of the strategy network in the direction of increasing the Q value.

For the Critic network, it will be updated by updating the value network in DQN. The gradient information is::

$$\frac{\partial L(\theta^{Q})}{\partial \theta^{Q}} = E_{s,a,r,s\prime\sim D}[(TargetQ - Q(s,a|\theta^{Q}))\frac{\partial Q(s,a|\theta^{Q})}{\partial \theta^{Q}}] \quad (31)$$

$$TargetQ = r + \gamma Q'(s', \pi(s'|\theta^{\mu'})|\theta^{Q'}) \quad (32)$$

For the MADDPG [17] algorithm that introduces multi-agents, the system will adopt a centralized training and distributed execution framework. In the training process, learn by using some global information, and in the final test run, you only need to use local information to make decisions when applying. This is a disadvantage of Q-learning. Q-learning must use the same information when learning and applying. This algorithm improves the DDPG algorithm, Critic network adds the strategy information of other agents, and Actors can only access local information. After the training is completed, only Actors are used in the execution phase, and each Actor is executed in a distributed manner.

Let $\theta = \{\theta_1, \theta_2, ..., \theta_N\}$ denote the parameters of n agent strategies, and $\pi = \{\pi_1, \pi_2, ..., \pi_N\}$ denote the strategy of n agents. For the cumulative expected reward of the i-th agent, for a random strategy, the strategy gradient is:

$$\nabla_{\theta_i} J(\theta_i) = E_{s\sim p^{\mu}, a_i \sim \pi_i}[\nabla_{\theta_i} \log \pi_i(a_i|o_i)\, Q_i^{\pi}(x, a_1, ..., a_N)] \quad (33)$$

Here $Q_i^{\pi}(x, a_1, ..., a_N)$ is a centralized action value function, which takes action $a_1, ..., a_N$ of all agents, plus state information x as input, and then outputs the Q value of agent i.

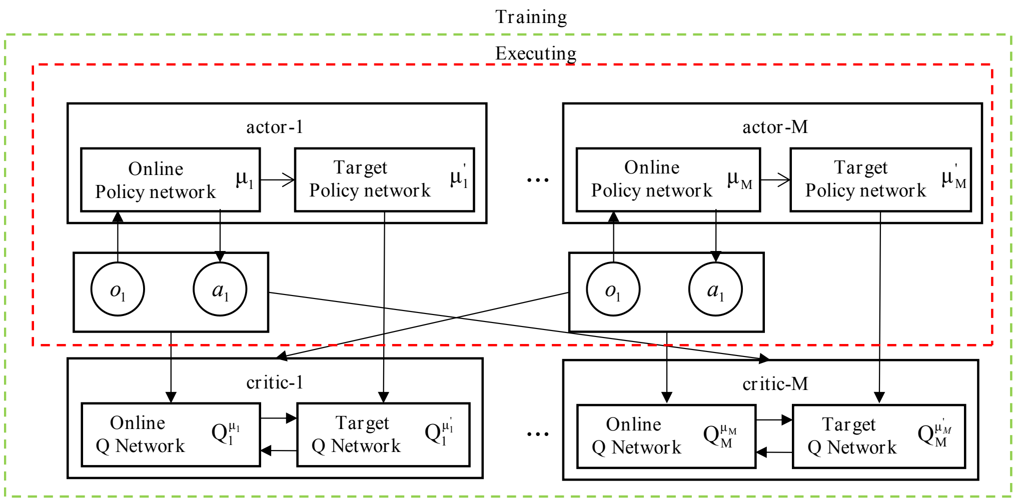
The gradient formula obtained according to the deterministic strategy μ_{θ_i} is as follows:

$$\nabla_{\theta_i} J(\mu_i) = E_{x,a\sim D}[\nabla_{\theta_i}\mu_i(a_i|o_i)\nabla_{a_i} Q_i^{\pi}(x, a_1, ..., a_N)]|_{a_i=\mu_i(o_i)} \quad (34)$$

The experience playback buffer D contains a tuple $(x, x', a_1, ..., a_N, r_1, ..., r_N)$, which records the experience of all agents. The centralized action value function Q_i^{μ} is updated as follows:

$$L(\theta_i) = E_{x,a,r,x'}[(Q_i^{\pi}(x, a_1, ..., a_N) - y)^2], y = r_i + \gamma Q_i^{\mu'}(x', a_1', ..., a_N')|_{a_j'=\mu_j'(o_j)}$$

(35)



Training
Executing
actor-1
Online Policy network μ_1
Target Policy network μ_1'
actor-M
Online Policy network μ_M
Target Policy network μ_M'
o_1
a_1
o_1
a_1
critic-1
Online Q Network $Q_1^{\mu_1}$
Target Q Network $Q_1^{\mu_1'}$
critic-M
Online Q Network $Q_M^{\mu_M}$
Target Q Network $Q_M^{\mu_M'}$
…
…

Figure 4 Schematic diagram of the network structure of the DTOMALB algorithm

Because DDPG is generally used to solve the problem of continuous action input, and the action space in us is discrete. Therefore, Gumbel-Softmax [17] network is used to convert discrete actions into continuous action estimates.

Figure 4 is a schematic diagram of the network structure of the DTOMALB algorithm.

The overall algorithm flow is shown in Table 1.

Table 1 DTOMALB Algorithm Description

```
Algorithm DTOMALB
1    Initialize memory playback library D ;
2    For user=1,2,...,N:
3        Initialize actor network, including target and eval;
4        Initialize critic network, including target and eval ;
5    End For;
6    For episode =1, 2, …, M:
7        Initialize random variables $\mathcal{N}_t$ used for exploration and initial states$s_0$
8        For step =1,2,…, T:
9            For user i=1,2,...,N:
10               for agent i, $a_i = \mu_{\theta_i} + \mathcal{N}_t$. Choose an action based on current strategy and
                 exploration mechanism, and get rewards $r_i$ through environmental feedback.
11           End For;
12           Get joint action $a = [a_1, a_2, \dots, a_n]$, reward $r = [r_1, r_2, \dots, r_n]$ and new state x';
13           Store $(s, a, r, x')$ in memory playback library D and update the system status
14           For agent i=1,2,...,N:
15               Randomly choose a minibatch sample set $(s, a, r, x')$ from the memory;
16               Calculate $y^i = r_i^j + \gamma Q_i^{\mu'}(x'^j, a'_1, \dots a'_N)$
                 Update crtic network by minimizing the mean square error:
17                   $L(\theta_i = \frac{1}{S}\sum_J (y^i - Q_i^{\mu}(x^j, a_1, \dots a_N))^2$
18               Update actor network by using the strategy gradient method:
```

```
                $\nabla_{\theta_i} J = -\frac{1}{S}\sum_j \nabla_{\theta_i}\mu_i(o_i^j \nabla_{a_i} Q_i^{\mu'}(x'^j, a'_1, \ldots a'_N)|a_i = \mu_i(o_i^j)$
19              End For;
            Update the target network parameters of each agent:
20              $\theta_i' = \tau\theta_i + (1-\tau)\theta_i'$
                $\theta_j' = \tau\theta_j + (1-\tau)\theta_j'$
21          End For;
22      End For
```

5. Experiment, Results and Discussions

In this section, the examples are given to evaluate the algorithm from four aspects. The experiment mainly includes the simulation of the DTOMALB algorithm and the comparison with other algorithms in four index levels to test the availability and superiority of the algorithm proposed in this paper, and the effect of adding some variables on the whole training process of the algorithm.

In a single-cell multi-user scenario, there are 10 users and 5 edge servers on this experiment. More detailed simulation parameters settings are shown in Table 2.

Table 2 Simulation parameter setting

Parameter	Definition	Value
M	Number of edge servers	5
N	Number of users	10
f	Edge server computing power	～U[12,20]Ghz
f_{local}	local computing ability.	～U[300,500]Mhz
L	Task transfer data size	～U[300,500]300KB
C	CPU resources required by the task	～U[800,1200]M cycles
B	Link bandwidth	20Mhz
p^{up}	User upload power	500mw

After the algorithm is simulated, the effectiveness of the algorithm is analyzed from four aspects: training convergence, load balancing coefficient, comparison of system utility and number of users, and task load balancing results. The simulation results are analyzed as follows, and the meaning of the research results and the limitations of the research are reflected in the result analysis.

(1) Training convergence analysis

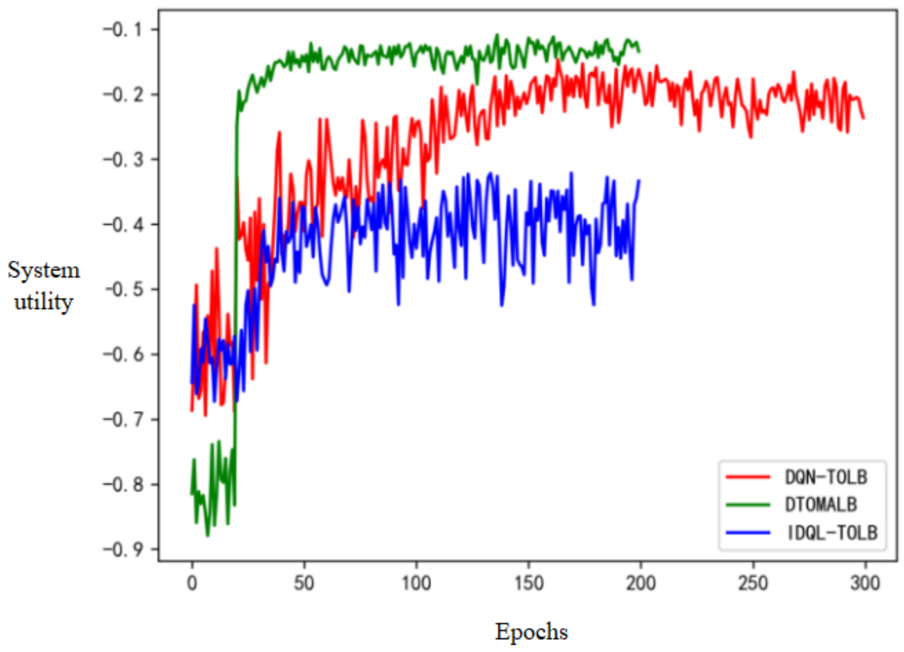


Figure 5 The convergence of the algorithm

In Figure 5 , DQN-TOLB and IDQL-TOLB[14] are the centralized DQN algorithm introducing load factor and the independent DQN algorithm in multi-agent. To facilitate comparison with the centralized DQN-TOLB algorithm, we define the system utility of the two multi-agent algorithms as the average of the cumulative rewards of multiple agents. It can be seen from Figure 5 that the system utility of the three algorithms changes with the training process. It can be seen that the proposed DTOMALB algorithm has the fastest convergence and the highest utility. Observing the centralized training DQN-TOLB algorithm, it can be found that it can also converge in the end, but it falls into the local optimum, and its convergence speed is slow. This is because its action space is huge, and it requires a lot of exploration to converge and find the optimal value. For the IDQL-TOLB algorithm, the training process is very unstable, and it can be seen that its fluctuation range is the largest. This is because each agent makes a decision independently, which makes the entire system environment dynamically change, so the training process is unstable, and Reward fluctuates at a lower level.

(2) With or without load balancing coefficient training

Figure 6 is a comparison of the training situation of the DTOMALB algorithm and the DTOMA algorithm without the introduction of load balancing coefficients. It can be seen that when the load balancing factor is introduced, the DTOMALB algorithm can converge faster and can achieve higher system utility. However, the DTOMA algorithm without introducing load balancing coefficients has a long training process, and the system utility is relatively low. This is because after the introduction of the load balancing factor, the selfishness of each user is reduced, and each user will tend to choose an edge server with a lower load, thereby reducing unnecessary competition, thereby accelerating convergence, and improving the effectiveness of the system . Without the introduction of load balancing coefficients, competition will occur more easily, so that training requires more rounds to explore the optimal strategy, and it is easier to fall into the local optimal. Therefore, it is necessary to introduce load balancing coefficients, which can accelerate the training speed and promote the overall average delay reduction of all users.

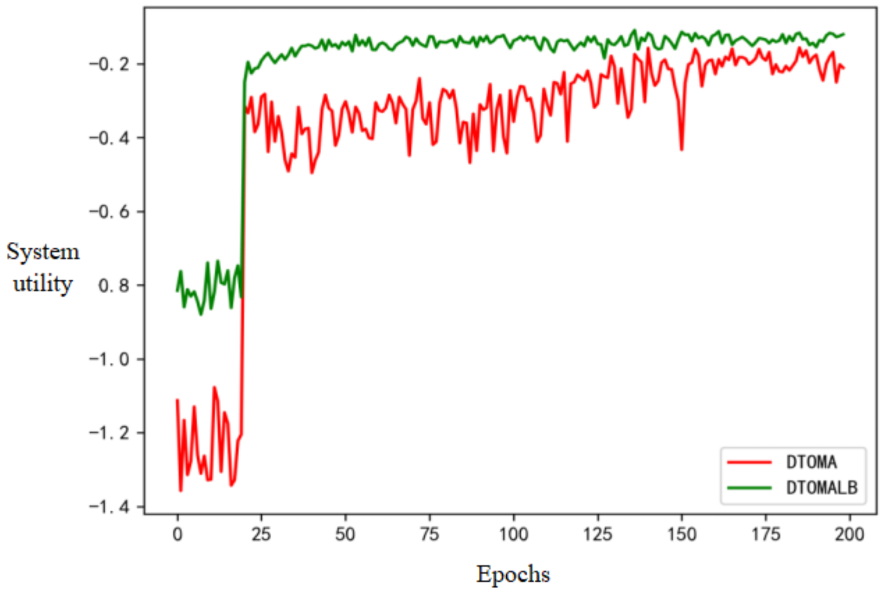


Figure 6 Comparison of training with and without load balance coefficient

(3) Comparison of system utility and number of users

As the total number of user equipment increases, the overall utility of the system decreases, because with the increase of user equipment, more and more tasks need to be uninstalled. On the one hand, as the number of users increases, each user occupies fewer and fewer communication resources, so the offload rate will decrease, and the offload delay will increase, resulting in reduced system utility. On the other hand, as the number of users increases, the computing resources that each user can allocate also decrease, resulting in an increase in latency and therefore a decrease in the overall system utility. It can be seen from Figure 7 that the overall utility of the resource scheduling scheme based on DTOMALB is higher, reflecting the superiority of the algorithm.

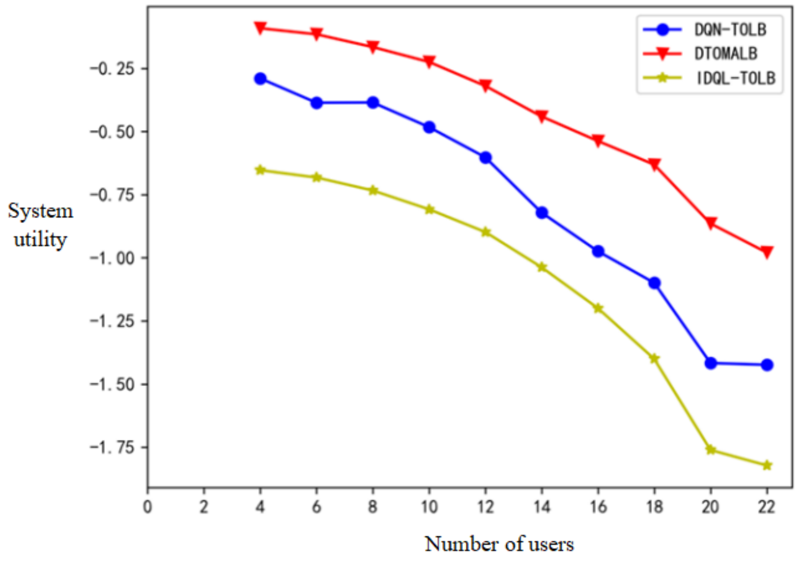


Figure 7 The relationship between system utility and number of users

(4) Task load balance result analysis

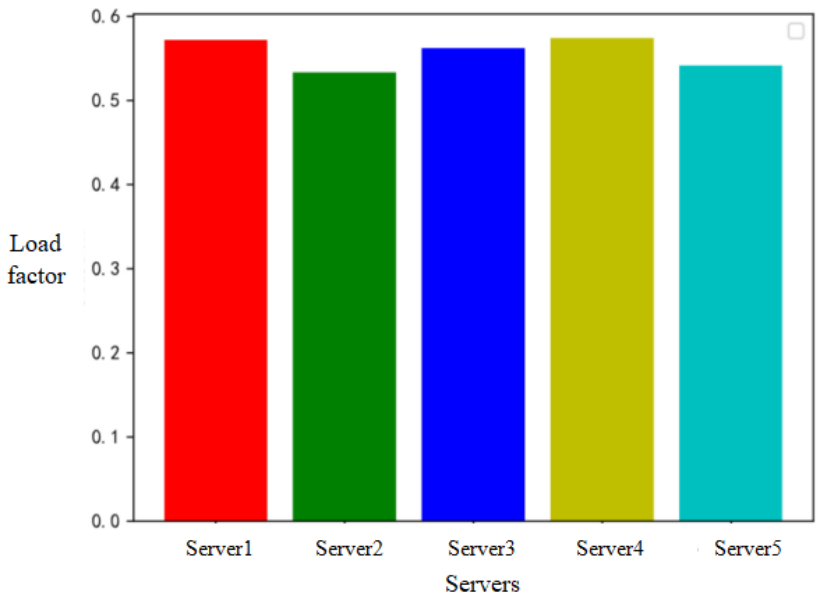


Figure 8 Edge server load balancing

Figure 8 shows the load balancing results achieved by the DTOMALB algorithm. The computing power of servers 1 to 5 is [12, 14, 16, 18, 20] Ghz. It can be seen that the load of each server is more balanced, the server with strong computing power is assigned to relatively more computing tasks, and the server with weak computing power is assigned less computing tasks. The load factor of each server is almost the same, and the load balancing is realized.

The limitations of this research mainly focus on model training, which can be improved from the offline training level.

6. Conclusion

This chapter studies task offloading and resource allocation in a multi-user single-cell scenario. In a single cell scenario, multiple users are connected to an edge server through a single LTE macro base station, and the edge server can schedule tasks to other servers connected to it. Aiming at the resource waste caused by the uneven load of multiple users in the process of uninstalling their computing tasks, and considering that the response time of the user tasks should be minimized, the problem of user offloading and resource allocation in this scenario is turned into more Goal optimization problem. At the same time, in view of the characteristics of dimensional explosion, scalability and poor dynamics faced by the centralized task scheduling with the increase of the number of users, an algorithm model for centralized training and task offloading of distributed operations is proposed. By establishing each user as a Markov game model and introducing load balancing coefficients, a DTOMALB algorithm based on multi-agent is proposed. Through simulation experiments, comparing the centralized algorithm and the independent multi-agent algorithm, the DTOMALB algorithm proposed in this chapter can effectively reduce the response delay of all users and make the load of each edge computing server relatively balanced, improving the robustness and scalability of the system.

7. Declarations

Abbreviations

IoT: Internet of Things; DTOMALB: distributed task offloading algorithm based on multi-agent and load balancing; VEC: Vehicle Edge Computing; DRL: Deep Reinforcement Learning; LTE: Long Term Evolution.

Availability of data and materials

The datasets used and/or analysed during the current study are available from the corresponding author on reasonable request.

Competing interests

The authors declare that they have no competing interests.

Funding

This work was supported by the National Natural Science Foundation of China (NO. 61772064), the National Key Research and Development Program of China (2018YFC0831900).

Authors' contributions

SP, ZZ conceived and designed the study. CL and XP performed the simulation experiments. ZZ and CL wrote the paper. ZZ and CL reviewed and edited the manuscript. All authors read and approved the final manuscript.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (NO. 61772064), the National Key Research and Development Program of China (2018YFC0831900).

8. References

[1] Ashton K.That 'Internet of things' thing[J]. RFiD Journal,2009,22(7):97-1

[2] Buyya R, Yeo C S, Venugopal S, et al. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility[J]. Future Generation Computer Systems, 2009, 25(6):599-616.

[3] Wang W, Lan R, Gu J, et al. Edge Caching at Base Stations With Device-to-Device Offloading[J]. IEEE Access, 2017, 5(99):6399-6410.

[4] Flores H, Srirama S N, Buyya R. Computational offloading or data binding? Bridging the cloud infrastructure to the proximity of the mobile user[C]. 2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud). IEEE, 2014: 10-18.

[5] You C, Huang K, Chae H, et al. Energy-efficient resource allocation for mobile-edge computation offloading[J]. IEEE Transactions on Wireless Communications, 2017, 16(3): 1397-1411.

[6] Bi S, Zhang Y J A. Computation Rate Maximization for Wireless Powered Mobile-Edge

Computing with Binary Computation Offloading[J]. IEEE Transactions on Wireless Communications, 2018:1-1.

[7] Guo S, Xiao B, Yang Y, et al. Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing[C]. IEEE INFOCOM 2016 - IEEE Conference on Computer Communications. IEEE, 2016.

[8] Wang J, Hu J, Min G, et al. Computation Offloading in Multi-Access Edge Computing Using a Deep Sequential Model Based on Reinforcement Learning[J]. IEEE Communications Magazine, 2019, 57(5):64-69.

[9] Xu J, Chen L, Ren S. Online learning for offloading and autoscaling in energy harvesting mobile edge computing[J]. IEEE Transactions on Cognitive Communications and Networking, 2017, 3(3): 361-373.

[10] Dong P , Wang X X , Rodrigues J J P C , et al. Deep Reinforcement Learning for Vehicular Edge Computing: An Intelligent Offloading System[J]. ACM Transactions on Intelligent Systems and Technology, 2019, 10(6).

[11] Liu Y, Yu H, Xie S, et al. Deep Reinforcement Learning for Offloading and Resource Allocation in Vehicle Edge Computing and Networks[J]. IEEE Transactions on Vehicular Technology, 2019, 99:1-1.

[12] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. nature, 2015, 518(7540): 529-533.

[13] Mnih V, Badia A P, Mirza M, et al. Asynchronous methods for deep reinforcement learning[C]//International conference on machine learning. 2016: 1928-1937.

[14] Tampuu, Ardi, et al. Multiagent cooperation and competition with deep reinforcement learning. PloS one 12.4 (2017): e0172395.

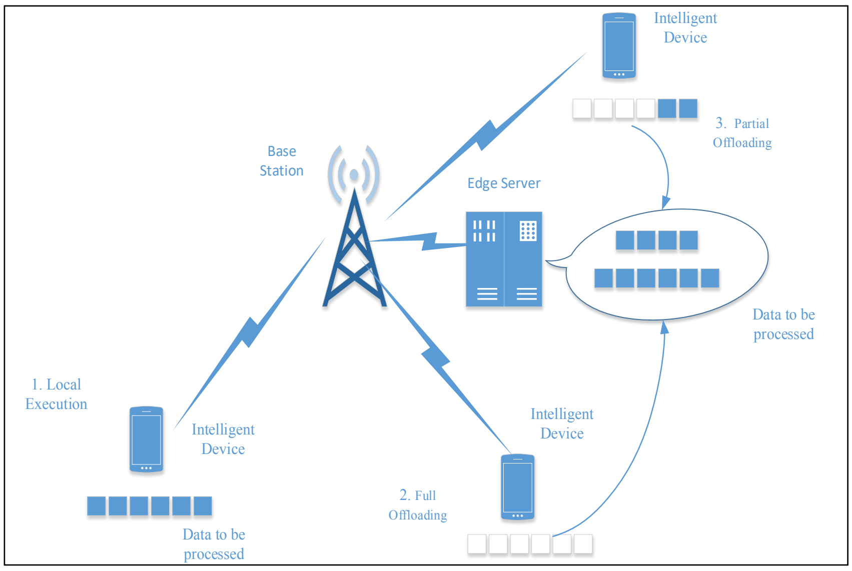
[15] Lillicrap T P, Hunt J J, Pritzel A, et al. Continuous control with deep reinforcement learning[J]. arXiv preprint arXiv:1509.02971, 2015.

[16] Silver D, Lever G, Heess N, et al. Deterministic policy gradient algorithms[C]. 2014.

[17] Ryan Lowe. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments[J]. 2017.

Figure Title and Legend

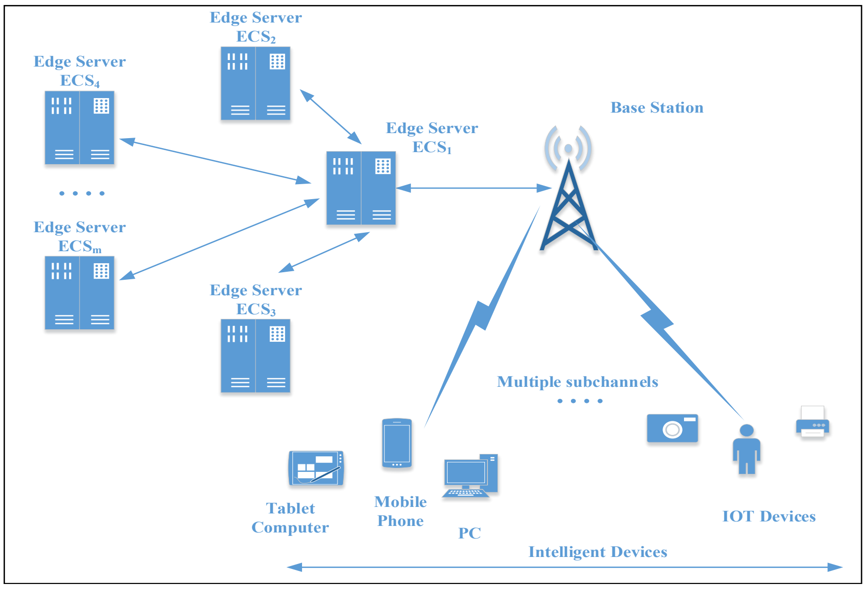
Figure 1:



Short title: Task offloading method comparison

Detailed legend: The possible decisions to calculate the offload may have three situations, local execution, full offloading and partial offloading.

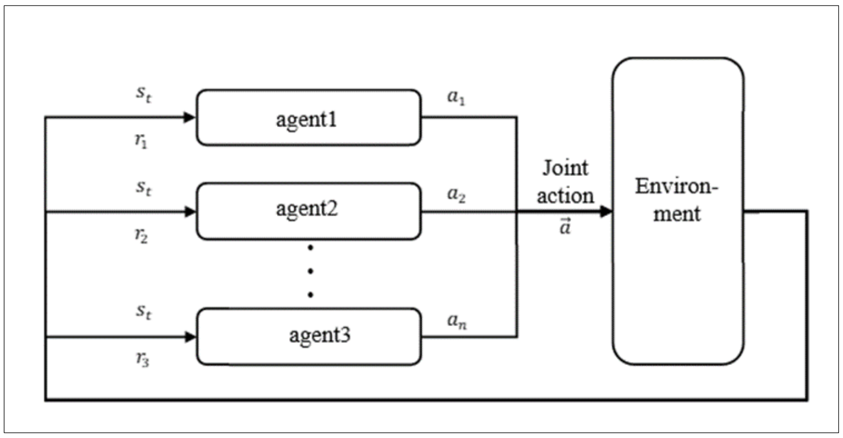
Figure 2:



Short title: Single-cell multi-user scenario in edge computing

Detailed legend: In a single-cell multi-user scenario, the base station can be divided into multiple sub-channels for different users. At the same time, the base station is connected to multiple edge servers to provide users with computing resources to help them process computing tasks.

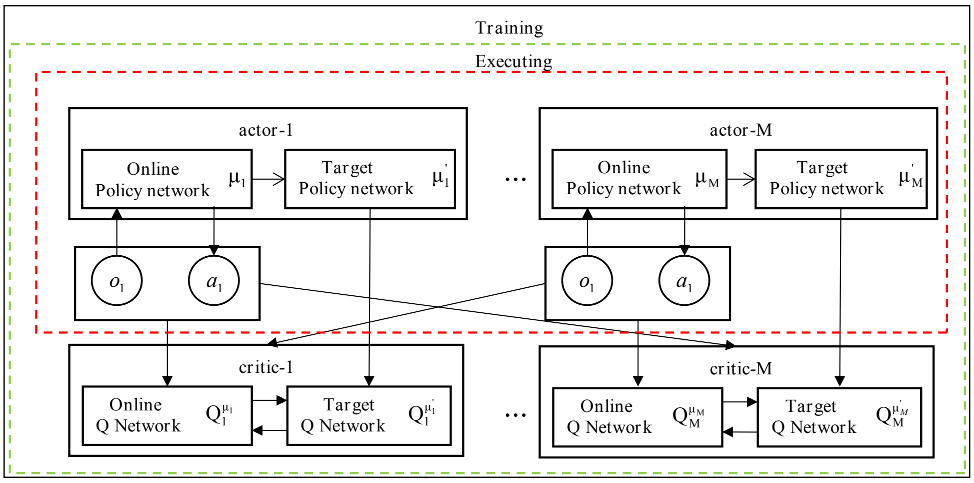
Figure 3:



Short title: Multi-agent reinforcement learning system

Detailed legend: Multi-agents act at the same time, and under joint action, the entire system will be transferred, and each agent will be rewarded immediately.

Figure 4:

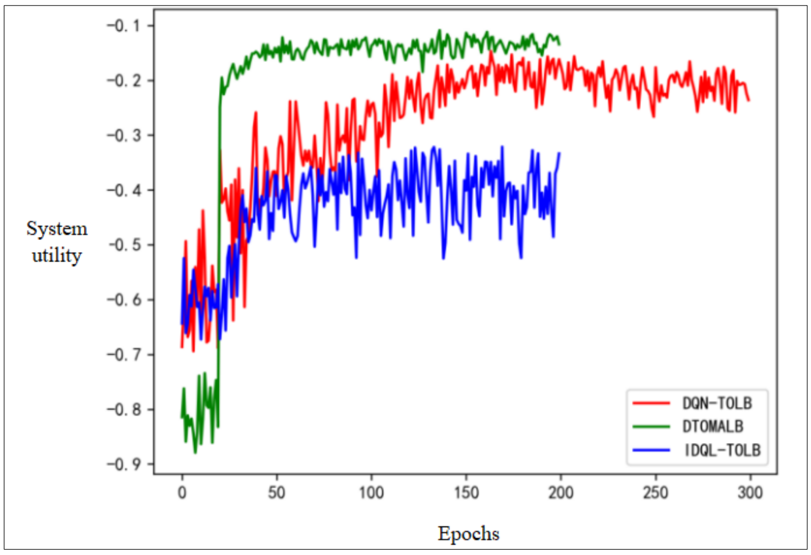


Training
Executing
actor-1
Online Policy network μ_1
Target Policy network μ_1'
actor-M
Online Policy network μ_M
Target Policy network μ_M'
o_1
a_1
o_1
a_1
critic-1
Online Q Network $Q_1^{\mu_1}$
Target Q Network $Q_1^{\mu_1'}$
critic-M
Online Q Network $Q_M^{\mu_M}$
Target Q Network $Q_M^{\mu_M'}$

Short title: Schematic diagram of the network structure of the DTOMALB algorithm

Detailed legend: How the distributed task offloading algorithm based on multi-agent and load balancing works.

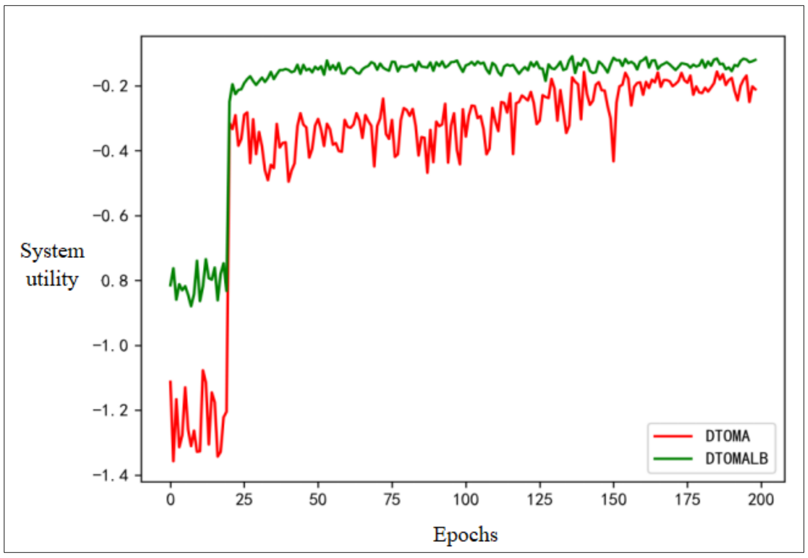
Figure 5:



Short title: The convergence of the algorithm

Detailed legend: Compare the convergence of the DQN-TOLB and IDQL-TOLB algorithms with the centralized DQN-TOLB algorithm.

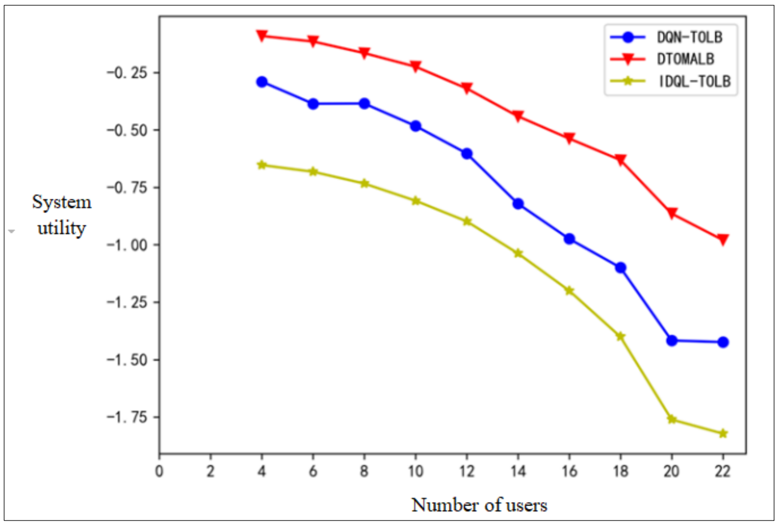
Figure 6:



Short title: Comparison of training with and without load balance coefficient

Detailed legend: A comparison of the training situation of the DTOMALB algorithm and the DTOMA algorithm without the introduction of load balancing coefficients..

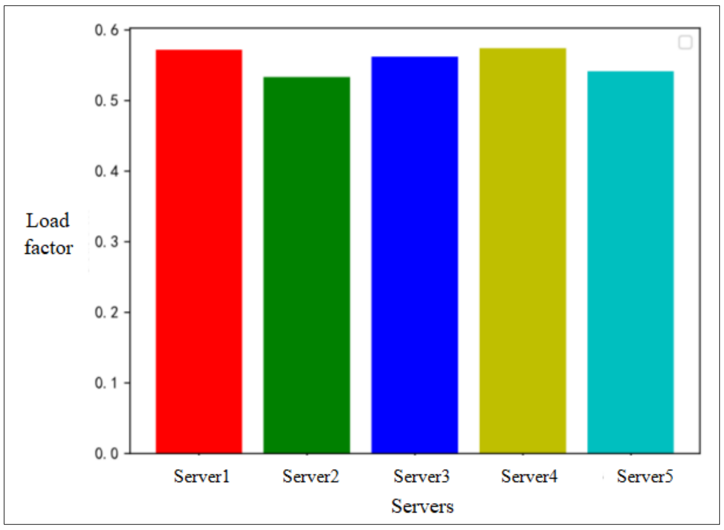
Figure 7:



Short title: The relationship between system utility and number of users

Detailed legend: The superiority of the algorithm is reflected bcause the overall utility of the resource scheduling scheme based on DTOMALB is higher.

Figure 8:



Short title: Edge server load balancing

Detailed legend: The load factor of each server is almost the same, and the load balancing is realized by the DTOMALB algorithm.