Hindawi
Complexity
Volume 2020, Article ID 7016307, 14 pages
https://doi.org/10.1155/2020/7016307

WILEY | Hindawi

Research Article

Distributed Task Offloading Game in Multiserver Mobile Edge Computing Networks

Shuang Chen,[1] Ying Chen,[1] Xin Chen,[1] and Yuemei Hu[2]

[1]*School of Computer Science, Beijing Information Science and Technology University, Beijing, China*
[2]*Network Engineering Department, Information Science and Technology School, QuFu Normal University, Rizhao Campus, Shandong, China*

Correspondence should be addressed to Ying Chen; chenying@bistu.edu.cn

Received 15 February 2020; Accepted 28 March 2020; Published 4 May 2020

Guest Editor: Xuyun Zhang

With the explosion of data traffic, mobile edge computing (MEC) has emerged to solve the problem of high time delay and energy consumption. In order to cope with a large number of computing tasks, the deployment of edge servers is increasingly intensive. Thus, server service areas overlap. We focus on mobile users in overlapping service areas and study the problem of computation offloading for these users. In this paper, we consider a multiuser offloading scenario with intensive deployment of edge servers. In addition, we divide the offloading process into two stages, namely, data transmission and computation execution, in which channel interference and resource preemption are considered, respectively. We apply the noncooperative game method to model and prove the existence of Nash equilibrium (NE). The real-time update computation offloading algorithm (RUCO) is proposed to obtain equilibrium offloading strategies. Due to the high complexity of the RUCO algorithm, the multiuser probabilistic offloading decision (MPOD) algorithm is proposed to improve this problem. We evaluate the performance of the MPOD algorithm through experiments. The experimental results show that the MPOD algorithm can converge after a limited number of iterations and can obtain the offloading strategy with lower cost.

1. Introduction

With the advent of the 5G era, a large number of mobile applications have emerged and become increasingly popular [1, 2]. With the explosion of data traffic, computation-intensive applications require stronger computing power and energy storage [3]. However, resource constraints on mobile devices cannot meet the computing requirements of computation-intensive applications, thereby affecting the quality of experience (QoE) [4–7]. Cloud computing as a solution comes into being. However, the exponential growth in data traffic means that cloud computing, the centralized way of processing tasks, can no longer guarantee the quality of services (QoS) [8, 9]. Mobile edge computing (MEC) is considered as a promising paradigm for the application of distributed ideas. Edge servers are deployed on edge base stations near users to provide computing and storage capacity [10, 11]. In the field of mobile networks, services are an important indicator of performance [12]. Offloading computing tasks to the edge of the network for processing not only reduces the task processing burden of mobile devices but also improves computing QoS and QoE through low latency and low energy consumption [13].

This idea of distributed offloading of MEC can be applied to many different scenarios [14]. In recent years, vehicle network is popular because of its wide application. Most of these vehicle services have strict time delay requirements [15]. However, the current limited processing and storage capacity of on-board equipment may not fully meet the requirements [16]. Therefore, the introduction of MEC into a vehicle network is a promising method to solve these problems. MEC servers with powerful computing and storage capabilities are installed on the edge of the vehicle network, greatly reducing task processing time while sharing computing tasks for local devices [15]. In addition, with the rapid development of science and technology, users

increasingly value the quality of life. Some online games are getting a lot of attention. This kind of computationally intensive application requires a lot of computing power and energy reserves to quickly handle a large number of computing tasks. To meet this requirement for real-time response, offloading compute-intensive tasks to edge servers is a solution [17]. Offloading computing tasks to resource-rich MEC servers can not only greatly improve the QoE, but also enhance the capabilities of mobile devices to run a variety of resource-intensive applications [18, 19]. In addition, from the perspective of the application field, MEC is also widely applied in the data mining field to reduce the delay of the prediction process and equipment's energy consumption [20–25].

During computation offloading, the user offloads the computing task to the nearest edge server for execution [26]. In 5G environment, the intensive deployment of base stations makes the service scope of edge servers overlap [27]. Due to the limited computing power of edge servers, resource preemption occurs when a large number of users offload tasks to the same server [28]. Therefore, users make a server decision according to the offloading situation. In addition, the offloading of computing tasks to the edge requires channel transmission, and the transmission of multiple users in the same channel would cause interference and affect the interaction between users [29, 30]. The overinterference will affect the data transmission rate and the offloading delay.

Recently, there are a lot of research studies on computation offloading. Computation offloading problems can be expressed as different types of problems to be solved. Liu et al. described the offloading problem as a mixed-integer nonlinear program problem and optimized it from two aspects of communication and calculation with the goal of minimizing the system overhead [31]. In [32], the authors defined the offloading problem as a game problem and proposed a distributed computing offloading algorithm to obtain the offloading strategy of each user. Chen et al. expressed the task offloading problem as a stochastic optimization problem and optimized the problem with the goal of reducing the energy consumption of task offloading [33]. In [34], the authors formulated the joint optimization problem of bandwidth division and computational resource allocation and decomposed the original problem into two subproblems. With the goal of minimizing the system overhead, an iterative algorithm was proposed to obtain the global optimal solution. Due to the limitation of the capacity of the channel and the computing resources of the edge server during the transmission process, there is a competitive relationship between the offloaded users, which can be well described by game theory.

Game theory is a powerful tool to solve the decision-making problem of computation offloading. Guo et al. proposed a highly computationally efficient two-layer game theory greedy approximation offloading scheme and proved its superior performance [35]. In [36], the authors describe the offloading problem as a dynamic random game between small cells, taking into account the randomness of channel state. In [37], the authors applied the game theory method to realize the effective computing offloading in a distributed way and designed a distributed computing shunt algorithm to solve.

In this paper, we apply game theory to model the computation offloading problem of multiple users. Our main contributions are as follows:

(i) We construct a MEC network scenario with dense distribution of edge servers and introduce the MEC operator role in addition to the mobile users. We mainly study the computation offloading decision of mobile users in the service overlap area.

(ii) We divide the whole offloading process into two parts: computation and transmission. Considering channel interference and resource preemption, we describe the problem as two noncooperative game models, which make decisions on offloading server and channel, respectively. In addition, we prove the existence of NE in two models.

(iii) The real-time update computation algorithm (RUCO) is proposed to solve the problem. Then, in order to solve the high complexity problem of the RUCO, we improved the algorithm and proposed the MPOD algorithm. Experimental results show that the MPOD algorithm has good convergence and can get lower cost offloading strategy.

The rest of the paper is organized as follows. System model and problem formulation are carried out in Section 2. Section 3 proves the existence of NE in two game models and proposes the RUCO algorithm and the MPOD algorithm, and the evaluation is described in Section 4. Finally, Section 5 is the conclusion.

2. Computation Offloading Model and Noncooperative Game Formulation

2.1. Computation Offloading Model. We consider a multi-server, multiuser MEC computing offload scenario, as shown in Figure 1. In this scenario, edge servers are deployed in a highly intensive manner to perform a large number of computing tasks. Due to the dense distribution of edge servers, service areas of servers overlap. Here, we study the computation offloading of MUs in the overlapping service areas. In a MEC system, MUs can choose to offload compute-intensive tasks to any edge servers serving the location to perform the computation. Specifically, in Figure 1, MU_1 can offload tasks to M_1 or M_2 to perform computational tasks. We weighed each user's offloading decision by considering all MUs in the overlap areas and single-server service areas of the system. During the computation in the edge servers, the edge server operators can allocate computing resources to the MUs and charge the MUs a certain resource rental fee. This is a component of the offloading cost of MUs and also an important basis for MU to make offloading decisions.

We consider $N = \{1, 2, 3, \ldots, n\}$ to represent a group of mobile users and $M = \{1, 2, 3, \ldots, m\}$ to represent a set of mobile edge servers where MUs offload the computation

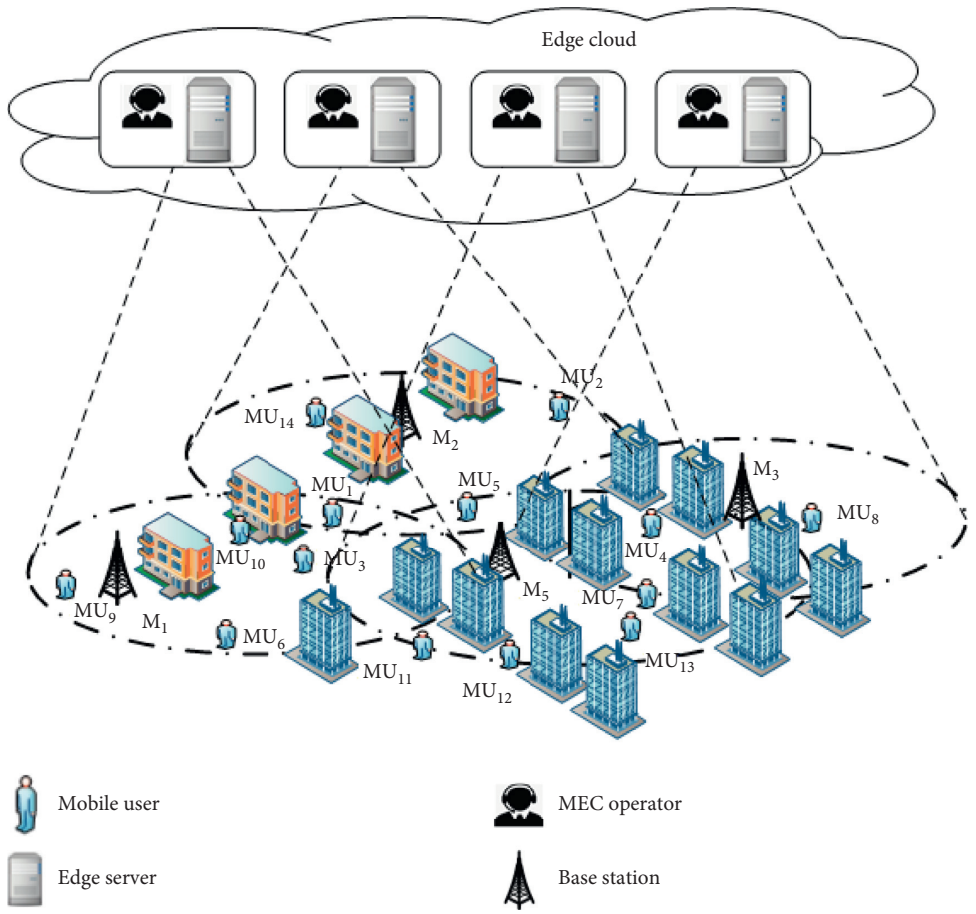

Figure 1: Mobile edge computing system with multiuser multiserver.

tasks to complete the computation. For ease of study, we assume that the user is stationary at this moment and the position does not change. Edge servers have more computing power than local computing. We define the number of instructions completed by edge servers per unit time as the computing power of the edge servers F_M. MUs transmits the computation task to edge servers through the channel. For MU i, we assume that there are s noninterfering subchannels to choose, represented as $C_i = \{c_1, c_2, \ldots, c_s\}$. Then, the sum of selectable channel sets for all MUs can be expressed as $C = \{C_1, C_2, \ldots, C_n\}$. When MUs choose the same channel to offload tasks to the same edge server, interference will be generated to affect the transfer rate of the task. Therefore, channel interference is also an important factor affecting offloading decisions.

In our model, each MU has a computationally intensive task. We define the computation task as $T_i = \{s_i, b_i\}$, where s_i represents the number of instructions to be executed to complete the task i and b_i represents the amount of data to be transferred to offload the task i to the edge server.

Each MU makes offloading decisions based on the actual situation of its task. MUs need to play two games in the whole offloading process. In the first game, the MU decides which server to offload the task to. In the second game, the user decides which channel to choose for data transmission. According to the location of each MU, we can obtain the feasible strategy profile of each MU and the feasible set of all MU is represented as A. For MU i, we represent its feasible strategy set as $A_i = \{A_i^0, A_i^{1,1}, \ldots, A_i^{m,s}\}$, $A_i \subset A$, where A_i^0 means MU i chooses to perform computation locally and $A_i^{m,s}$ means MU i chooses to offload the task to edge server m and chooses to transmit data through channel s. Each strategy $A_i^{m,s}$ consists of two elements $A_i^{m,s} = \{a_i^m, a_i^s\}$. a_i^m is the set of servers that MU i can select in the first game. a_i^s is the channel set that MU i can choose to carry out data transmission in the second game. Each MU chooses offloading strategy according to its own situation and the state of the game environment. $D_i = \{d_i^m, d_i^s\}$, $D_i \in A_{i \in N}$, is defined as the offloading strategy selected for the MU i, where d_i^m represents the server to be offloaded and d_i^s is the channel to be selected for data transmission. Due to the limited number of edge servers deployed, the number of MUs is much higher than the number of edge servers, so multiple MUs choose to offload to the same edge server and several MUs choose the same channel for data transmission. For MU i, we define the MUs set to be α_i and the number of MU to be $n(\alpha) = \sum_{j \in \alpha_i} \varrho\{d_i^m = d_j^m\}$ to perform the computation task offloaded to the same server as the MU i. Similarly, we define the number of MUs that selects the same channel for MU i to carry out data transmission as β_i. The number of MUs is expressed as $n(\beta) = \sum_{j \in \beta_i} \varrho\{d_i^s = d_j^s\}$, $n(\beta_i) < n(\alpha_i)$, where $I\{x\}$ is an index function. When x condition is true,

the function value is 1; otherwise, the value is 0. The main notations involved in the model are given in Table 1.

2.1.1. Communication Model. MUs choose to offload the task to the edge cloud for computation. From the perspective of communication, the whole offloading process is divided into four parts, namely, uplink communication delay, backhaul link delay, downlink delay, and cloud task processing delay. The backhaul link is very fast and much faster than the uplink, so the backhaul link delay is negligible. In addition, compared with the data transmission delay of the uplink, the downlink returns the computation results to the MUs, and the delay in this process is far less than the uplink communication delay. Therefore, in our study, only uplink and cloud task computing are considered in terms of time delay.

We assume that the MU i decides to offload computing tasks to the edge server for execution, and then, the rate of data transmission during the offload process needs to be considered. The selection of channel d_i^s has a direct impact on the transmission rate. In addition, the factors affecting the rate include the state of game environment, bandwidth, and background noise. The following is the transmission rate of data transmission through channel m selected by MU i:

$$\omega_i\left(d_i^m, d_i^s\right) = B\log_2\left(1 + \frac{p_i g_i}{\sum_{j\in\beta_i} p_j g_j I\left\{d_i^s = d_j^s\right\} + \sigma}\right). \quad (1)$$

We represent the bandwidth of the channel as B and the transmission power of MU i as p_i. We define the channel gain as g_i and the background noise as σ. $\sum_{j\in\beta_i} p_j g_j \varrho\left\{d_i^m = d_j^m\right\}$ represents the interference generated by MUs who choose the same channel as MU i for data transmission. Obviously, interference has a significant impact on the data transmission rate of MU i. Specifically, the greater the interference from other MUs to the MU i is, the lower the data transmission rate of the MU i is.

In the process of data transmission, we consider the cost of MUs from two aspects. On the one hand, MUs can generate time delay in the process of data transmission, and high time delay would lead to the decline of the QoE, and then, it would increase the cost. On the other hand, due to the limited energy stored by mobile devices, the energy consumption during task transmission deserves attention. The cost increases with the increase in energy consumption. Specifically, MU i offloads the computing task T_i to the edge server m, and the time consumption during transmission is expressed as $T_i^{c_2}$, as shown in the following equation:

$$T_i^{c_2}\left(d_i^m, d_i^s\right) = \frac{b_i}{\omega_i}. \quad (2)$$

The energy consumption E_i^c of MU i during offloading is computed as follows:

$$E_i^c = \frac{b_i p_i}{\omega_i}. \quad (3)$$

Through the above communication model, we can get several factors that affect the cost in the communication process, such as the amount of data transferred by the task, the transmission power, and the data transmission rate. Transmission rate is one of the things we will focus on in the following study.

Table 1: Notations.

Symbol	Description
MU	Mobile user
M	Edge server
m	Wireless channel
A_i	Available policy sets for MU_i
D_i	MU_i selected strategy
D_{-i}	The strategy set for users other than MU_i
d_i^m	MU_i selects server M for offloading
d_i^s	MU_i selects channel m for offloading
α_i	Set of users who select the same server as the MU_i
β_i	Set of users who select the same channel as the MU_i
p_i	The transmission power of MU_i
ω_i	MU_i uplink data rate
π	The fee charged to complete an instruction
z_i^l	The local calculation costing
$z_i^{c_1}$	The cloud computing costing
$z_i^{c_2}$	The cloud transmission costing
z_i	MU_i total costing

2.1.2. Computation Model. MUs can take all factors into consideration to decide how to complete the computation tasks. MUs have the option of offloading tasks to the edge server for processing or performing computations directly locally. If MUs choose to execute the computation task locally, we consider the computation delay and the energy consumption during the computation as the cost for the MUs to complete the tasks. When MU i performs the computing task T_i locally, it is represented by T_i^l as follows:

$$T_i^l = \frac{s_i}{F_i^0}, \quad (4)$$

where the computing power of the local server is represented by F_i^0. The computing power is determined by the actual hardware configuration of the server and is constant.

The energy consumption generated by MU i performing the task T_i locally is expressed as follows:

$$E_i^l = v_i s_i, \quad (5)$$

where v_i denotes the average amount of energy consumption to complete an instruction.

Compared to the low computing power of the local server, the edge server is more powerful. Therefore, most MUs would prefer to offload computing tasks to edge servers. During the computation on the edge server, we consider not only the delay in completing the task, but also the computation resource leasing fee charged by MEC server operators to MUs.

The delay caused by the MU i completing the task T_i on the edge server m is expressed as follows:

$$T_i^{c_1} = \frac{s_i + \sum_{j\in\alpha_i} s_j I\left\{d_j^m = d_i^m\right\}}{f_M}. \quad (6)$$

$\sum_{j\in\alpha_i} s_j \varrho\{d_j^M = d_i^M\}$ represents the preemption of the edge server computing resources by other MUs who offload to the same server as MU i. This causes the MU i to take more time to complete the task computation.

MUs rent computing resources of edge servers to complete the task, and MEC server operators would charge a certain computing resources leasing fee to MUs who need computing resources. MU i computes task T_i in the edge server, and the rental fee charged by the MEC server operators is expressed as V_i, as shown in the following equation:

$$V_i = \pi s_i, \tag{7}$$

where π represents the fee charged by the MEC operator to complete an instruction.

2.1.3. Cost Model. We define the components of cost as three types, namely, delay, energy consumption, and payment. The delay includes transmission delay and computation delay. Energy consumption is the energy consumption of mobile devices during computation offloading or local calculation. Payment is a fee that MUs pay to perform computing task using the computing resources of the MEC server. In addition, the cost of time delay and energy consumption is allocated linearly by the weight factor. We define the weighting factor of time delay as γ_i^T and the weighting factor of energy consumption as γ_i^E. The advantage of this method is that it can be adjusted according to the actual requirements of the computing task. Specifically, when the calculation task is time delay sensitive, the setting of weight factor should meet $\gamma_i^T > \gamma_i^E$, while in the face of tasks with high energy consumption, the weight is allocated as $\gamma_i^T < \gamma_i^E$.

If MU i chooses local computation, then the cost is expressed as follows:

$$z_i^l = \gamma_i^E E_i^l + \gamma_i^T T_i^l = \left(\frac{\gamma_i^T}{F_i^0} + \gamma_i^E \nu_i\right) s_i. \tag{8}$$

Since there are two games in the whole offloading process, we describe the cost of the two games separately as the cost basis of offloading decision. First, we consider the definition of the cost of the server decision. In this game, we consider the time delay caused by the computation of the task in the edge server and the cost of computing resources. In addition, there is a correlation between server selection and channel selection. When making channel decisions, it has been determined which server channel is selected for data transmission. Therefore, probabilistic selection of channels is also included in the cost definition of server decisions. The following is the cost function of the server decision:

$$z_i^{c_1}(d_i^m, \alpha_i) = \gamma_i^T T_i^{c_1}(d_i^m) + \pi s_i + p_m(d_i^m) z_i^{c_2}. \tag{9}$$

We define the cost of channel decision making including transmission delay and transmission energy consumption. The channel decision cost of MU i is given by

$$z_i^{c_2}(d_i^s, \beta_i) = \gamma_i^E E_i^c(d_i^s) + \gamma_i^T T_i^{c_2}(d_i^s). \tag{10}$$

Thus, the total cost function of MU i can be expressed as follows:

$$z_i(d_i^m, d_i^s) = \begin{cases} z_i^l(d_i^0), & D_i = D_0 \\ \gamma_i^E E_i^c + \gamma_i^T\left(T_i^{c_1} + T_i^{c_2}\right) + \pi s_i, & D_i \neq D_0 \end{cases} \tag{11}$$

2.2. Problem Formulation. We consider that each MU is selfish and wants to reduce its cost by choosing the most appropriate offloading strategy. In other words, the ultimate goal of the MU is to minimize the cost of the whole computation offloading process:

$$\min_{d_i^m \in a_i^m, d_i^s \in a_i^s} z_i(d_i^m, d_i^s, \alpha_i, \beta_i), \quad \forall i \in N. \tag{12}$$

In the whole offloading process, MUs need to make two decisions, server decision and channel decision. We define the two decisions' problem as the following two subproblems.

In the process of server decision making, we consider the relevance between servers and channels because the feasible channel set of MUs in the next stage decision is determined by the edge servers of this decision. Therefore, all possible channel selection costs are considered probabilistic in this stage. To sum up, the goal of this stage is to minimize cost $z_i^{c_1}$:

$$\min_{d_i^m \in a_i^m} z_i^{c_1}(d_i^m, a_i^s, \alpha_i), \quad \forall i \in N. \tag{13}$$

From the above equation, we can find that the value of cost $z_i^{c_1}$ is determined by the server decision, the feasible strategy profile of channels and the MU set that chooses the same server.

In the decision-making process, MUs are independent of each other. MUs cannot know other users' decisions and can only judge the current environment according to its actual situation and prior experience. Since the computing resources of edge servers are limited, the more the MUs that choose the same server for computation are, the greater the computation delay consumption is. Therefore, each MU wants to find the server with the lowest computation delay, and everyone wants to find the server with the least computation task to offload. MUs are regarded as participants, and there exists a noncooperative game as follows:

$$\Gamma_1 = \langle N, \{a_i^m\}_{i\in N}, \{Z_i^{c_1}\}_{i\in N} \rangle. \tag{14}$$

In the process of channel decision, the MU decision is mainly decided by the task specification and the number of users transmitted in the same channel. The larger the number of MUs transmitted on the same channel is, the lower the data transmission rate of the MU is. Therefore, MUs want to conduct data transmission through channels with a small number of MUs, which can reduce interference and achieve the purpose of reducing cost. The goal of each MU in this phase is given by

$$\min_{d_i^s \in a_i^s} z_i^{c_2}(d_i^s, \beta_i), \quad \forall i \in N. \tag{15}$$

Each MU wants to perform computational offloading through the channel with the least interference, and thus, MUs are willing to gain decisions to reduce their costs through autonomous learning. There is a competitive relationship between MUs, and we build a noncooperative game model as follows:

$$\Gamma_2 = \langle N, \{a_i^s\}_{i\in N}, \{Z_i^{c_2}\}_{i\in N}\rangle. \tag{16}$$

In game models Γ_1 and Γ_2, there is a competitive game relationship between MUs. Each MU is pursuing a smaller cost by changing strategy, while other MUs are also trying to reduce their cost. In the end, all players in the game will reach an equilibrium state. In this state, no one can further reduce cost by changing strategies. In other words, the game reaches NE.

3. Nash Equilibrium and Algorithm

3.1. Equilibrium Analysis. We consider whether MUs can further reduce the cost by changing the offloading strategy. Therefore, the next step is to study the existence of NE in Γ_1 and Γ_2.

Definition 1. For the noncooperative game, given a strategy profile $D^* = (D_1^*, D_2^*, \ldots, D_N^*)$, if no MU can further decrease its offloading cost by changing its offloading strategy, we call D^* as a NE of the game, i.e.,

$$z_i(D_i^*, D_{-i}^*) \leqslant z_i(D_i, D_{-i}^*), \quad \forall D_i \in A_i, i \in N. \tag{17}$$

MUs in the equilibrium state can achieve a offloading strategy profile, where all the MUs have no incentive to change their strategies unilaterally. Therefore, NE equilibrium has good stability.

In what follows, we show that Γ_1 and Γ_2 can achieve a NE. To prove this, we introduce the concept of the exact potential game. The game is an exact potential game. In the exact potential game, there exists a potential function, and when the strategy changes, the potential function has the same change trend as the MUs' cost function. In addition, if Γ_1 is an exact potential game, it has at least one pure NE and satisfies the finite improvement property, which means that an NE can be achieved by a finite number of iterations of changing strategies.

Definition 2. A game is the exact potential game, if and only if there is a potential function $\Phi(a)$ which obeys the following condition:

$$z_i(d_i, d_{-i}) - z_i(d_i', d_{-i}) = \Phi(d_i, d_{-i}) - \Phi(d_i', d_{-i}), \tag{18}$$

where z_i is the cost of participant i and d_i is the strategy for participant i. Then, d_{-i} denotes the strategies of the other participants.

Theorem 1. *The game Γ_1 is an exact potential game which has at least one pure NE strategy.*

Proof. We first prove that Γ_1 is an exact potential game and then prove the existence of NE in Γ_1 through the properties of exact potential game.

According to (9), the cost function of Γ_1 is first given as follows:

$$\begin{aligned} z_i^{c_1}(d_i^m, \alpha_i) &= \gamma_i^T T_i^{c_1}(d_i^m) + \pi s_i + p_m(d_i^m) z_i^{c_2} \\ &= \gamma_i^T \frac{s_i + \sum_{j\in\alpha_i} s_j I\{d_j^m = d_i^m\}}{f_M} \\ &\quad + \pi s_i + p_m(d_i^m) z_i^{c_2}. \end{aligned} \tag{19}$$

As an exact potential game, we give the potential function of Γ_1 as follows:

$$\begin{aligned} \Phi_k^{c_1}(d_k^m, \alpha_i\alpha) &= \frac{1}{2}\sum_{k\in N} \gamma_k^T \frac{s_k + \sum_{j\in\alpha_k} s_j I\{d_j^m = d_k^m\}}{f_M} \\ &\quad + \sum_{k\in N} \pi s_k + \sum_{k\in N} p_m(d_k^m) z_k^{c_2}. \end{aligned} \tag{20}$$

MU i has resource preemption for other MUs offloaded to the same server. On the contrary, other MUs also preempt MU i's computing resources. We would carry out equivalent transformation for (20):

$$\begin{aligned} \Phi_i^{c_1}(d_i^m, \alpha_i) &= \frac{1}{2}\gamma_i^T \frac{s_i + \sum_{j\in\alpha_i} s_j I\{d_j^m = d_i^m\}}{f_M} \\ &\quad + \frac{1}{2}\sum_{j\in N} \gamma_j^T \frac{s_j + \sum_{i\in\alpha_i} s_i I\{d_j^m = d_i^m\}}{f_M} \\ &\quad + \frac{1}{2}\sum_{k\in N/\{k\neq i\}} \gamma_i^T \frac{s_k + \sum_{j\in\alpha_i/\{j\neq i\}} s_j I\{d_j^m = d_i^m\}}{f_M} \\ &\quad + \pi s_i + \sum_{k\in N/\{k\neq i\}} \pi s_k + p_m(d_i^m) z_i^{c_2} \\ &\quad + \sum_{k\in N/\{k\neq i\}} p_m(d_k^m) z_k^{c_2}. \end{aligned} \tag{21}$$

Resource preemption is mutual, so there is an equality relationship in (21), $\gamma_i^T((s_i + \sum_{j\in\alpha_i} s_j I\{d_j^m = d_i^m\})/f_M) = \sum_{j\in N}\gamma_j^T((s_j + \sum_{j\in\alpha_j} s_i I\{d_j^m = d_i^m\})/f_M)$.

Therefore, we can simplify the potential function as follows:

$$\Phi_i^{c_1}(d_i^m, \alpha_i) = \gamma_i^T \frac{s_i + \sum_{j\in\alpha_i} s_j I\{d_j^m = d_i^m\}}{f_M} + \frac{1}{2}\sum_{k\in N/\{k\neq i\}} \gamma_i^T \frac{s_k + \sum_{j\in\alpha_i/\{j\neq i\}} s_j I\{d_j^m = d_k^m\}}{f_M} + \pi s_i + \sum_{k\in N/\{k\neq i\}} \pi s_k + p_m(d_i^m) z_i^{c_2} + \sum_{k\in N/\{k\neq i\}} p_m(d_k^m) z_k^{c_2}. \tag{22}$$

When the offloading strategy d_i^m of MU i changes to $d_i^{m'}$, the change trend of potential function is as follows:

$$\begin{aligned}
&\Phi_i^{c_1}(d_i^m, \alpha_i) - \Phi_i^{c_1}\left(d_i^{m'}, \alpha_i\right) \\
&= \gamma_i^T \frac{s_i + \sum_{j\in\alpha_i} s_j I\{d_j^m = d_i^m\}}{f_M} + \frac{1}{2}\sum_{k\in N/\{k\neq i\}} \gamma_i^T \frac{s_k + \sum_{j\in\alpha_i/\{j\neq i\}} s_j I\{d_j^m = d_k^m\}}{f_M} + \pi s_i + \sum_{k\in N/\{k\neq i\}} \pi s_k + p_m(d_i^m) z_i^{c_2} + \sum_{k\in N/\{k\neq i\}} p_m(d_k^m) z_k^{c_2} \\
&\quad - \gamma_i^T \frac{s_i + \sum_{j\in\alpha_i} s_j I\{d_j^m = d_i^{m'}\}}{f_M} - \frac{1}{2}\sum_{k\in N/\{k\neq i\}} \gamma_i^T \frac{s_k + \sum_{j\in\alpha_i/\{j\neq i\}} s_j I\{d_j^m = d_k^m\}}{f_M} - \pi s_i - \sum_{k\in N/\{k\neq i\}} \pi s_k - p_m\left(d_i^{m'}\right) z_i^{c_2} - \sum_{k\in N/\{k\neq i\}} p_m(d_k^m) z_k^{c_2}.
\end{aligned} \tag{23}$$

We find that when the MU i strategy changes, some factors unrelated to MU i in (22) have no change. Therefore, simplified form of (23) is as follows:

$$\begin{aligned}
&\Phi_i^{c_1}(d_i^m, \alpha_i) - \Phi_i^{c_1}\left(d_i^{m'}, \alpha_i\right) \\
&= \gamma_i^T \frac{s_i + \sum_{j\in\alpha_i} s_j I\{d_j^m = d_i^m\}}{f_M} + p_m(d_i^m) z_i^{c_2} - \gamma_i^T \frac{s_i + \sum_{j\in\alpha_i} s_j I\{d_j^m = d_i^{m'}\}}{f_M} - p_m\left(d_i^{m'}\right) z_i^{c_2} \\
&= z_i(d_i^m, \alpha_i) - z_i\left(d_i^{m'}, \alpha_i\right).
\end{aligned} \tag{24}$$

By Definition 2, we know that Γ_1 is an exact potential game. Therefore, Theorem 1 is proved.

Then, we consider Γ_2. □

Theorem 2. *Γ_2 is an exact potential game, and the potential function is as follows:*

$$\Phi^{c_2}(d_i^s, \beta_i) = \frac{1}{2}\sum_{i\in N} \gamma_i^T \frac{b_i}{B\log_2\left(1 + \left(p_i g_i / \left(\sum_{j\in\beta_i} p_j g_j I\{d_i^s = d_j^s\} + \sigma\right)\right)\right)} + \sum_{i\in N} \gamma_i^{Ev_i s_i}. \tag{25}$$

The proof is similar to the proof of Theorem 1, so we simplify the proof.

Proof. The cost function of Γ_2 is expressed as follows:

$$z_i^{c_2}(d_i^s, \beta_i) = \gamma_i^E v_i s_i + \gamma_i^T \frac{b_i}{B\log_2\left(1 + \left(p_i g_i / \sum_{j\in\beta_i} p_j g_j I\{d_i^s = d_j^s\} + \sigma\right)\right)}. \tag{26}$$

By equivalent substitution, we get the following equivalence relation:

$$\Phi^{c_2}(d_i^s, \beta_i) - \Phi^{c_2}\left(d_i^{'s}, \beta_i\right) = z_i^{c_2}(d_i^s, \beta_i) - z_i^{c_2}\left(d_i^{'s}, \beta_i\right). \tag{27}$$

So Theorem 2 is proved. □

3.2. The Real-Time Update Computation Offloading Algorithm. The above proves that NE exists in the two game models we established. Next, we study the solution of NE offloading strategy. In the game model, all MUs reduce the offloading cost by changing their offloading strategy. Since the MUs in the game are selfish, each MU pursues its own cost minimization. Therefore, we need a decentralized algorithm to implement our distributed computing offload solution and ultimately make MUs reach a goal that all parties are satisfied with.

Γ_1 and $\Gamma_2$are exact potential games and have the finite improvement property (FIP). Specifically, a set of NE strategies can be obtained after a finite number of iterations to reach an equilibrium state. We propose the real-time update computation offloading algorithm (RUCO) to obtain the NE strategy profile. The main idea of the algorithm is to use the convergence of the model to reach the NE state of the whole game system through finite iterations. All MUs execute the decision process simultaneously before starting to perform the compute offload task. In order to realize the selfish behavior of different MUs at the same time, we propose an idea of updating the game status in real time. MUs initiate a request message to each MU to update its status when a better strategy is available during the game. However, at each iteration, a single update request is approved with a confirmation message so that only one decision is made at a time.

The channel decision steps are similar to the server decision process, and thus, the whole decision process is not given concretely. The specific content of the server game

stage is shown in Algorithm 1. The main decision-making steps are given as follows:

(i) Initializing: at the initial state $t = 0$, MU i randomly selects the initial strategy d_i^m, and the initial strategy is obtained from the feasible profile a_i^m, i.e., $d_i^m \in a_i^m$. According to the cost function, the initial offloading cost $z_i^{c_1}(t = 0)$ is given. The calculation basis of the cost value is as follows:

$$z_i^{c_1}(t=0) = \begin{cases} z_i^l(d_i^0), & d_i^m = 0, \\ \gamma_i^T T_i^{c_1}(d_i^m) + \pi s_i + p_m(d_i^m) z_i^{c_2}, & d_i^m \neq 0. \end{cases} \tag{28}$$

(ii) Selecting a new best strategy: MU i seeks its new best strategy in the current game environment. The goal of MU i is to minimize the offloading cost in the current state, i.e.,

$$d_i^m = \arg\min_{d_i^m \in a_i^m} z_i^{c_1}(d_i^m, \alpha_i). \tag{29}$$

(iii) Sending request to update: MU i selects the latest best strategy and sends a request to other MUs to update the offloading strategy. During each iteration, the MU i can only request the strategy update once, so the strategy updating cannot begin until the request is acknowledged. Strategy updating obeys the following rules:

$$d_i^m(t=0) = \begin{cases} d_i^m(t=0), & \text{request accepted}, \\ d_i^m(t), & \text{request rejected}. \end{cases} \tag{30}$$

(iv) Termination: if we get an equilibrium strategy through iteration, iteration stops.

3.3. The Multiuser Probabilistic Offloading Decision Algorithm. The RUCO algorithm updates the offloading strategy profile iteratively through the idea of real-time update, and finally, the Nash equilibrium strategy is obtained. However, when the MU i generates a new strategy, all MUs receive an update request from the MU i for the strategy update operation. In one iteration, it is possible for each MU to send an update request, so the game environment has a high number of updates. The complexity of the whole algorithm is very high, and the delay of game reaching equilibrium will increase.

In order to reduce the delay, we improve the RUCO algorithm. The update of user strategy is no longer required in the global form. We apply selection probability to represent the feasible strategy set of each user. When the cost is low, the probability of choosing a strategy increases. In the next iteration, the user will choose the high probability strategy to offload first. Based on this idea, we propose the multiuser probabilistic offloading decision (MPOD) algorithm. Users only need to modify the probability vector of strategy selection of their feasible strategy set for each strategy update, but they do not make a global update request.

The detailed process of the whole algorithm is shown in Algorithm 2. The main steps of the MPOD algorithm are given as follows:

(i) Initializing: we introduce the concept of strategy probability selection, which can be defined as the basis for each MU to choose offloading strategy. For each MU, we initially set a uniform distribution of the selection probability of each strategy in the feasible profile. MU i's strategy selection probability vector is as follows:

$$P_i^m(t=0) = \left(\frac{1}{m+1}, \frac{1}{m+1}, \ldots, \frac{1}{m+1}\right). \tag{31}$$

(ii) Updating the strategy: when the user chooses a strategy to offload, the cost of offloading will change the probability of choosing the strategy. In time period t, MU i makes the current decision based on its current strategy selection probability vector $P_i^m(t)$.

(iii) Calculating the current cost value: in the current state, each MU calculates the cost value $z_i^{c_1}$ based on (8) and directly calculates the value of z_i^l if the MU chooses to calculate locally.

(iv) Updating the strategy selection vector: each MU adheres to the following rules to update their strategy selection vectors:

$$P_i^m(t+1) = P_i^m(t) + \xi r_i^t \left(e_{d_i^m} - P_i^m(t)\right), \tag{32}$$

where ξ is the coefficient of learning ability, and $0 < \xi < 1$. $e_{d_i^m}$ is a unit vector of $(M+1)$ dimension, where d_i^m corresponds to an element value of 1. r_i^t represents the benefit obtained by the decision, expressed as $r_i^t = 1 - \varepsilon z_i^{c_1}(t)$, where ε is a minimum value to ensure the nonnegative benefit of the decision.

(v) Termination: until all MUs do not adjust their offloading strategy, the iteration stops.

4. Evaluation

In this section, we evaluate the performance of the MPOD algorithm through experiments. We considered a network of densely distributed edge servers, including 3 servers and 100 MUs, 30 of whom are in the service overlap area. We set channel bandwidth B to 5 MHz [38]. In the process of data transmission, the calculation method of channel gain is $g_i = d_{i,M}^{-\alpha}$, where α is the path loss index. According to the path loss model of city and suburb, we set the value of α to 4. To simplify the calculation, we assume that the distance between the edge servers is 10 m and that the data transmission power of each MU mobile device is 0.4 W. In local computation, we need to know the computing power of the local server, so we randomly generated MU's local server computing capacity in [0.5, 3] Gcycles. In addition, the cloud server has a powerful computing capacity of 50 Gcycles. For each compute-intensive task T, the size b of the transmitted

Input: $N, A, s_i, b_i, \gamma_i^E, \gamma_i^T, B, p_i, g_i, f_M, F_i^0, \pi, \sigma, v_i, t$;
Output:
The equilibrium strategy vector D_i^*;
(1) Initialize: each MU i selects its first strategy $d_i^m(t=0)$ from A_i to compute the initial value of cost function $z_i^{c_1}(t=0)$, according to (8) or (9);
(2) **for** each MU i **do**
(3) get the current game environment;
(4) select the new best strategy $d_i^m(t+1)$;
(5) compute the new value for $z_i^{c_1}(t+1)$;
(6) **if** $z_i^{c_1}(t+1) > z_i^{c_1}(t)$ **then**
(7) $z_i^{c_1}(t+1) = z_i^{c_1}(t)$;
(8) **else**
(9) send a request to update;
(10) **if** request accepted **then**
(11) update strategy d_i^m;
(12) **end if**
(13) **end if**
(14) until an Equilibrium strategy profile is gained;
(15) **end for**

Algorithm 1: The real-time update computation offloading algorithm (RUCO).

Input: $N, A, s_i, b_i, \gamma_i^E, \gamma_i^T, B, p_i, g_i, f_M, F_i^0, \pi, \sigma, v_i, t, \xi$;
Output:
The equilibrium strategy vector D_i^*;
(1) Initialize: each MU i sets the initial value of its offloading strategy selection probability to be uniformly distributed, i.e., $P_i^m(t=0) = ((1/m+1), (1/m+1), \ldots, (1/m+1))$;
(2) **for** each MU i **do**
(3) select an offloading strategy $d_i^m(t)$ according to $P_i^m(t)$;
(4) compute the new cost value for $z_i^{c_1}(t+1)$;
(5) update strategy selection probability as follows: $P_i^m(t+1) = P_i^m(t) + \xi r_i^t(e_{d_i^m} - P_i^m(t))$;
(6) until all MUs do not adjust their offloading strategies;
(7) **end for**

Algorithm 2: The multiuser probabilistic offloading decision (MPOD) algorithm.

data was generated randomly from [0.5, 5] MB and the number of instructions to execute to complete the calculation of 1 Mb data is generated randomly from [100, 500] cycles/MB. In addition, the energy consumption per CPU cycle is obtained through $10^{-11}(\mathrm{FM})^2$. The weighting factors of time and energy consumption can be set according to the actual situation and the balance between them can be made. In this experiment, we set both weight factors to 0.5. Furthermore, the background noise σ is set to −100 dBm. The main parameter settings are given in Table 2.

4.1. Parameter Analysis. Edge server operators perform server updates, maintenance, and other operations, so they would charge resource rental fees to MUs that offload tasks to edge servers for execution. As shown in Figure 2, we analyze the impact of resource lease price customization on the total offloading cost. The unit price of resource lease is calculated to be 0.5 and 1. We observe the change of total offloading cost as the number of MU participating in the game increases. As shown in the figure, with the increase of MUs quantity, the total cost tends to rise. The higher the cost of computing the resource lease is, the higher the cost of the MUs offloading tasks is. Therefore, the rental unit price has a direct impact on the offloading cost.

For resource leasing unit price customization, we also compare the equilibrium costs corresponding to different prices. Through a finite number of iterations, the offloading cost of each MU reaches an equilibrium state and no longer changes. In the whole game process, the total offloading cost no longer changes. As shown in Figure 3, after about 12 iterations, the total offloading cost reaches the convergence state. When the unit price of resource leasing is 1, the total offloading cost is much higher than that of 0.5. Therefore, we further prove the influence of different resource lease prices on the total offloading cost.

Now, we analyze the influence of learning ability ξ of the model on the whole game process. The game process is the selection process of offloading decision made by each MU. The selection of offloading strategy is represented by

Table 2: Parameters used in the evaluation.

Parameters	Value
The number of MUs	100
The number of servers	3
The channel bandwidth B	5 MHz
The path loss index α	4
The distance between the edge servers	10 m
The data transmission power	0.4 W
The MU's local server computing capacity	[0.5, 3] Gcycles
The cloud server's computing capacity	50 Gcycles
The size b of task T	[0.5, 5] MB
The number of instructions of 1 MB data	[100, 500] cycles/MB
The background noise σ	−100 dBm

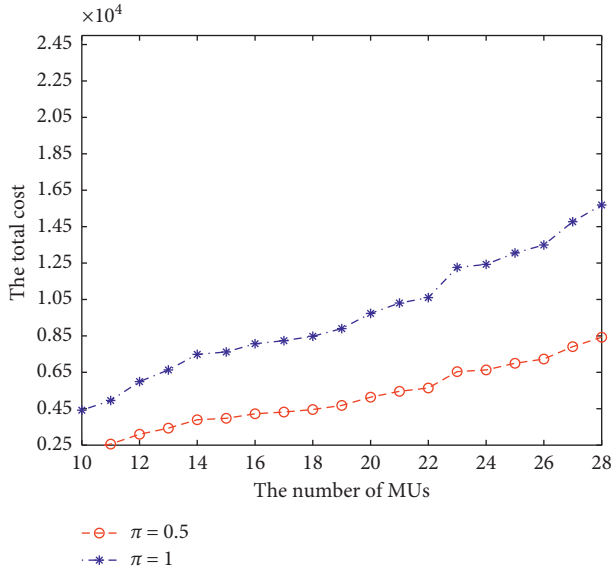

Figure 2: The impact of different unit price changes with quantity of MU on the total cost.

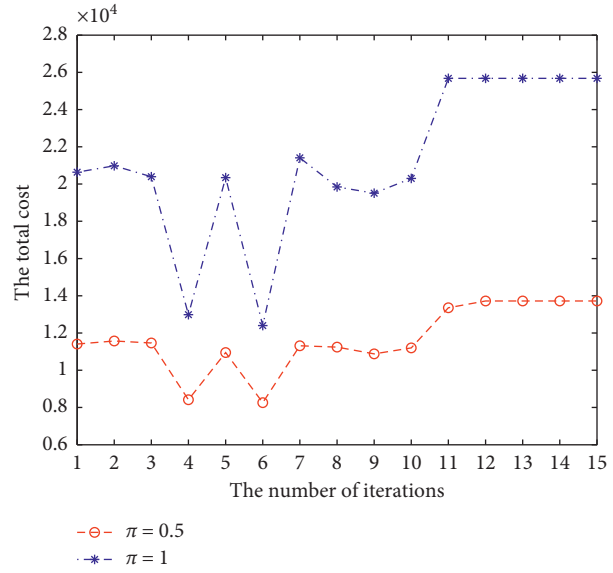

Figure 3: The impact of different unit prices on the total cost with the change of iteration times.

probability vector, and the strategy selection vector is taken as the basis of strategy selection in each iteration. Learning ability determines the degree to which MU gains in each decision process. Figure 4 describes the change of the probability of strategy selection of MU_1 choosing M_3 to offload with the change of the number of iterations. We can see that with the increase of the number of iterations, the probability of MU_1 choosing M_3 for offloading is on the rise, and the probability value is approaching to 1. Specifically, when the number of iterations is about 20, the probability value is basically 1. We can say M_3 is the equilibrium strategy for MU_1. We set the values of learning ability as 0.4, 0.5, and 0.6. As can be observed in Figure 4, when learning ability is 0.6, the probability of strategy selection reaches 1 at the fastest. Therefore, learning ability affects the rate of change of strategy selection probability.

The nonnegative coefficient ε is applied to determine the benefit of each decision according to the cost value, and the decision benefit is applied to determine the change of the strategy selection probability. We set ε values to 0.0001, 0.002, and 0.0003 for comparison. The value of ε should be small enough to ensure that the decision benefit is positive. In Figure 5, we analyze the change trend of decision benefit with cost under different values of ε. Overall, costs and benefits tend to move in opposite directions. Specifically, as the offloading cost increases, the decision cost decreases. From the change rate, we can see that the model with a large value of ε has a faster decline in decision benefits. Therefore, the value of 8 has a certain impact on the value of decision benefits and the rate of change.

In addition, we analyze the influence of nonnegative coefficient ε on the probability of strategy selection. As shown in Figure 6, after a certain number of iterations, the probability of strategy selection becomes 1 or 0 for any user in the game model. It can be clearly seen that the greater the value of ε is, the slower the strategy selection probability reaches equilibrium. As the strategy selection probability is affected by the decision benefit, the larger the decision benefit, the faster the strategy selection probability reaches the equilibrium state. Therefore, the value of ε indirectly affects the probability of strategy selection.

4.2. Convergence Analysis. We have proved that the model exists NE. Specifically, the model can reach an equilibrium state through a finite number of iterations. We further prove the convergence of the model by experiments. In Figure 7, we randomly select 3 MU to show the changing trend of offloading strategy selected by MU through iteration. We can observe that three MU offloading strategies reach the equilibrium state at the 11th iteration. Specifically, the offloading strategy does not change when MU_1 iterates 8 times. MU_2 reaches convergence state through 11 iterations. MU_3 also iterates 11 times to reach equilibrium. Therefore, the offloading strategy does not change after a limited number of iterations. From this strategy change trend, we can prove that the model is convergent.

Figure 8 proves the convergence of the model from another perspective. In the MPOD algorithm, we apply the

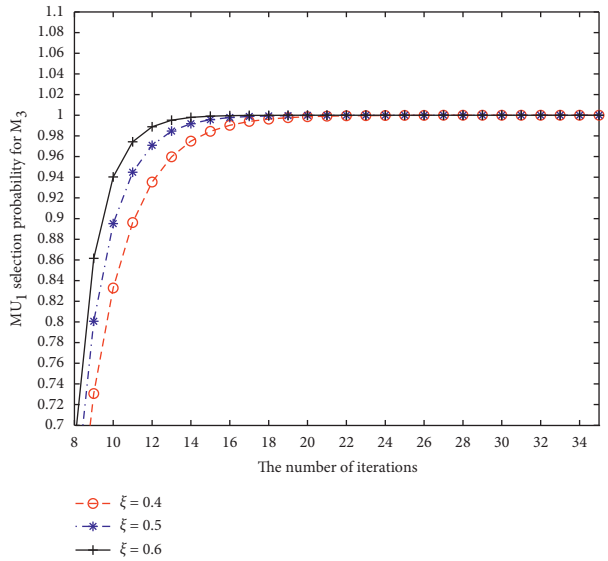

Figure 4: The impact of different values of learning ability ξ.

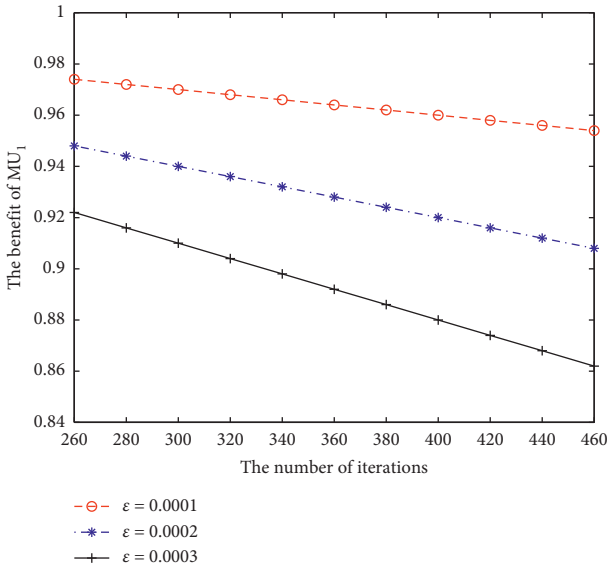

Figure 5: The impact of different ε on the benefits of MU_1.

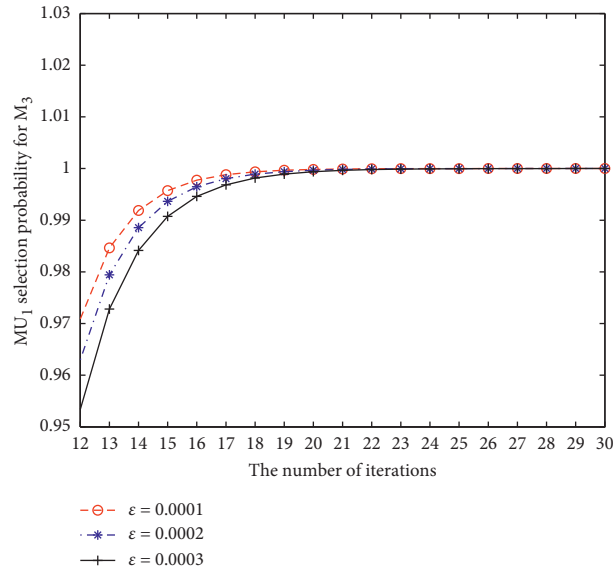

Figure 6: The impact of different ε on MU_1 selection probability for M_3.

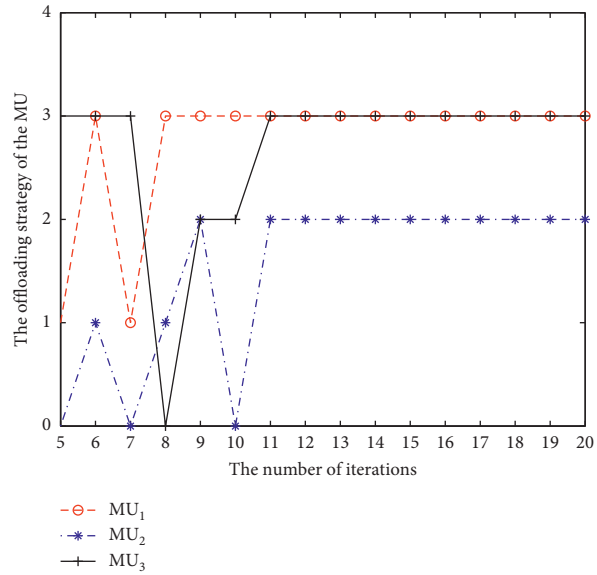

Figure 7: Convergence of the offloading strategies.

strategy selection vectors to represent the probability of selecting each strategy. In other words, as the probability of choosing one strategy increases, the probability of choosing other strategies decreases. Due to the presence of NE in the model, MUs would automatically select the equilibrium strategy after iterations and the selected strategies are no longer change. Therefore, the selection probability of equilibrium strategy would gradually approach and reach 1, while the selection probability of other strategies can be correspondingly decreased to 0. Figure 8 shows the strategy selection probability for each of MU_1's feasible policies. Specifically, around 17 iterations, the selection probability of strategy M_3 reaches 1, and in particular, M_3 is the equilibrium strategy of MU_1. In terms of the probability of strategy selection, we prove the convergence of the model once again.

4.3. Comparative Experiments. Compared with other algorithms, we analyze the performance of the MPOD algorithm. In the local execution completely algorithm (LECA), all MUs perform their tasks locally and do not offload the tasks. In the random selection execution algorithm (RSEA), each user offloads randomly selected strategies from its feasible strategy profile. Figure 9 shows the changing trend of average cost of MUs participating in the game with the increase of the number of iterations. Since the strategy of MUs to execute the task has no change in the LECA algorithm, the average cost is not affected by the number of iterations. The

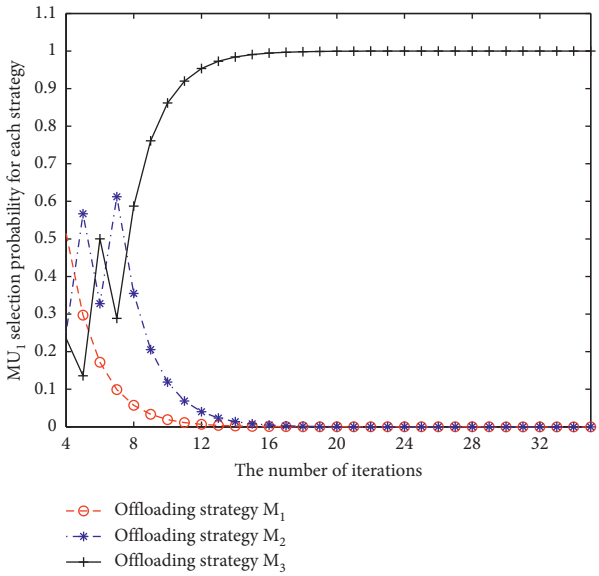

Figure 8: Convergence of the strategy selection probabilities.

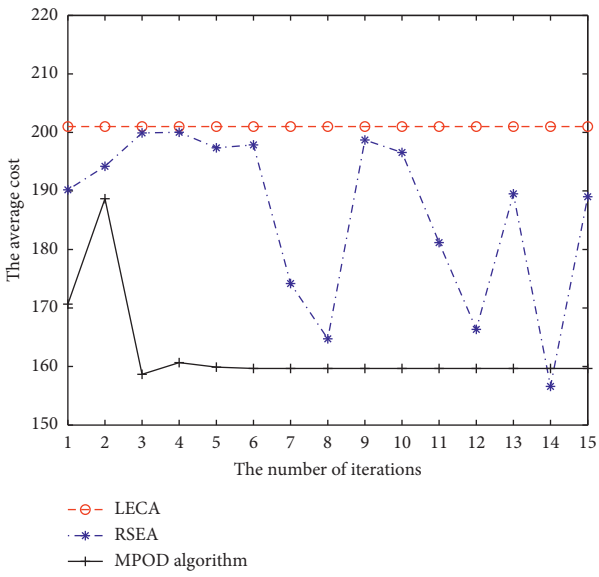

Figure 9: Performance comparison in terms of average cost for different iterations.

decision process of offloading applying the RSEA algorithm is uncertain, so the curve shows a broken line form of great change. In the MPOD algorithm, after about 6 iterations, the average cost no longer changes. In the seventh iteration, the average cost obtained by the MPOD algorithm is 8.7% lower than the RSEA algorithm and 20.6% lower than the LECA algorithm. Therefore, in terms of offloading cost, our proposed MPOD algorithm has obvious advantages.

Figure 10 shows the change in the number of MUs participating in the game and the total cost of different algorithms. According to the overall change trend of the three algorithms, with the increase of MUs, the total cost changes show an upward trend. The rate of total cost rise of

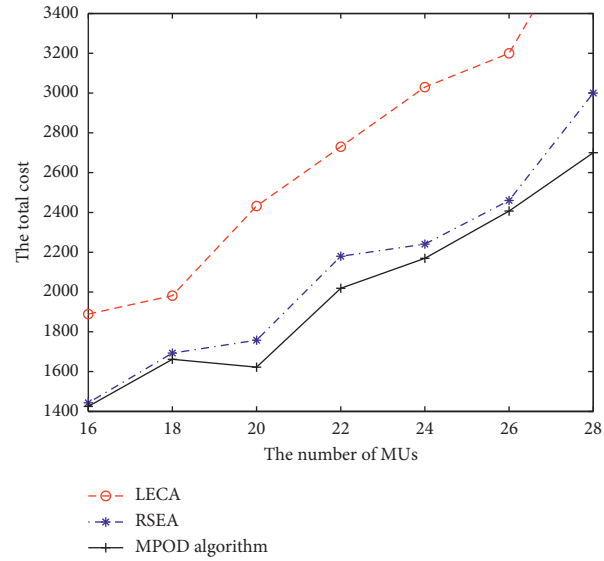

Figure 10: Performance comparison in terms of the total cost for different numbers of MUs.

the MPOD algorithm is the slowest, and the total cost is always lower than the LECA algorithm and the RSEA algorithm. When the number of MU increases to 28, the total cost of the MPOD algorithm is 10.8% lower than the RSEA algorithm and 31.8% lower than the LECA. This further proves that lower cost can be obtained through our algorithm.

5. Conclusion

In this paper, we study the task offloading problem in the scenario of dense distribution of edge servers. We focus on the offload strategy for mobile users in the edge server overlapping service areas. We divide the whole offloading process into data transmission and computation processing. We establish two noncooperative game models to solve the server and channel offloading decision, respectively, and prove the existence of NE in the model. The RUCO algorithm and the MPOD algorithm are proposed to obtain NE offloading strategy, and the latter is an improved algorithm to the former. In addition, we conduct experiments to prove the performance of the MPOD algorithm. The experimental results show that the MPOD algorithm has good convergence and can obtain lower offloading strategy.

Data Availability

Most of the simulation experimental data used for supporting the study of this article are included within the article. Further additional information about the data is available from the first author via chenshuang20192019@126.com.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was partly supported by the National Natural Science Foundation of China (nos. 61902029and 61872044), the Excellent Talents Projects of Beijing (no. 9111923401), and the Key Research and Cultivation Projects at Beijing Information Science and Technology University (no. 5211910958).

References

[1] Q. Ye, W. Zhuang, X. Li, and J. Rao, "End-to-end delay modeling for embedded VNF chains in 5G core networks," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 692–704, 2019.

[2] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: a survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2224–2287, 2019.

[3] J. Ren, Y. He, G. Huang, G. Yu, Y. Cai, and Z. Zhang, "An edge-computing based architecture for mobile augmented reality," *IEEE Network*, vol. 33, no. 4, pp. 162–169, 2019.

[4] A. Alnoman, A. S. Anpalagan, W. Ejaz, and A. Anpalagan, "Computing-Aware base station sleeping mechanism in H-CRAN-Cloud-Edge networks," *IEEE Transactions on Cloud Computing*, p. 1, 2019.

[5] Y. Zhang, K. Wang, Q. He et al., "Covering-based web service quality prediction via neighborhood-aware matrix factorization," *IEEE Transactions on Services Computing*, 2019.

[6] S. Yang, F. Li, M. Shen, X. Chen, X. Fu, and Y. Wang, "Cloudlet placement and task allocation in mobile edge computing," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5853–5863, 2019.

[7] Q. Ye, J. Li, K. Qu, W. Zhuang, X. S. Shen, and X. Li, "End-to-End quality of service in 5G networks: examining the effectiveness of a network slicing framework," *IEEE Vehicular Technology Magazine*, vol. 13, no. 2, pp. 65–74, 2018.

[8] Y. Chen, J. Huang, C. Lin, and J. Hu, "A partial selection methodology for efficient QoS-aware service composition," *IEEE Transactions on Services Computing*, vol. 8, no. 3, pp. 384–397, 2015.

[9] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge AI: intelligentizing mobile edge computing, caching and communication by federated learning," *IEEE Network*, vol. 33, no. 5, pp. 156–165, 2019.

[10] J. Huang, J. Liang, and S. Ali, "A simulation-based optimization approach for reliability-aware service composition in edge computing," *IEEE Access*, vol. 8, pp. 50355–50366, 2020.

[11] Y. Zhou, L. Tian, L. Liu, and Y. Qi, "Fog computing enabled future mobile communication networks: a convergence of communication and computing," *IEEE Communications Magazine*, vol. 57, no. 5, pp. 20–27, 2019.

[12] Y. Zhang, G. Cui, S. Deng, F. Chen, Y. Wang, and Q. He, "Efficient query of quality correlation for service composition," *IEEE Transactions on Services Computing*, 2018.

[13] Y. He, J. Ren, G. Yu, and Y. Cai, "D2D communications meet mobile edge computing for enhanced computation capacity in cellular networks," *IEEE Transactions on Wireless Communications*, vol. 18, no. 3, pp. 1750–1763, 2019.

[14] J. Huang, S. Li, and Y. Chen, "Revenue-optimal task scheduling and resource management for IoT batch jobs in mobile edge computing," *Peer-to-Peer Networking and Applications*, 2020.

[15] Z. Ning, J. Huang, X. Wang, J. J. P. C. Rodrigues, and L. Guo, "Mobile edge computing-enabled internet of vehicles: toward energy-efficient scheduling," *IEEE Network*, vol. 33, no. 5, pp. 1–8, 2019.

[16] T. Qiu, X. Wang, C. Chen, M. Atiquzzaman, and L. Liu, "TMED: a spider-web-like transmission mechanism for emergency data in vehicular ad hoc networks," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 9, pp. 8682–8694, 2018.

[17] J. Huang, C. Zhang, and J. Zhang, "A multi-queue approach of energy efficient task scheduling for sensor hubs," *Chinese Journal of Electronics*, vol. 29, no. 2, pp. 242–247, 2020.

[18] Y. Chen, N. Zhang, Y. Zhang, and X. Chen, "Dynamic computation offloading in edge computing for internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4242–4251, 2019.

[19] L. Ale, N. Zhang, H. Wu, D. Chen, and T. Han, "Online proactive caching in mobile edge computing using bidirectional deep recurrent neural network," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5520–5530, 2019.

[20] A. Ramlatchan, M. Yang, Q. Liu, M. Li, J. Wang, and Y. Li, "A survey of matrix completion methods for recommendation systems," *Big Data Mining and Analytics*, vol. 1, pp. 308–323, 2018.

[21] L. Qi, W. Dou, W. Wang, G. Li, H. Yu, and S. Wan, "Dynamic mobile crowdsourcing selection for electricity load forecasting," *IEEE Access*, vol. 6, pp. 46926–46937, 2018.

[22] D. Chen, N. Zhang, N. Cheng, K. Zhang, Z. Qin, and X. S. Shen, "Physical layer based message authentication with secure channel codes," *IEEE Transactions on Dependable and Secure Computing*, p. 1, 2019.

[23] Y. Zhang, C. Yin, Q. Wu, Q. He, and H. Zhu, "Location-aware deep collaborative filtering for service recommendation," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp. 1–12, 2020.

[24] W. Gong, L. Qi, and Y. Xu, "Privacy-aware multidimensional mobile service quality prediction and recommendation in distributed fog environment," *Wireless Communications and Mobile Computing*, vol. 2018, Article ID 3075849, 8 pages, 2018.

[25] Y. Xu, L. Qi, W. Dou, and J. Yu, "Privacy-preserving and scalable service recommendation based on SimHash in a distributed cloud environment," *Complexity*, vol. 2017, Article ID 3437854, 9 pages, 2017.

[26] Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu, and X. S. Shen, "Energy efficient dynamic offloading in mobile edge computing for internet of things," *IEEE Transactions on Cloud Computing*, p. 1, 2019.

[27] S. Chen, X. Chen, Y. Chen, and Z. Li, "Distributed computation offloading based on stochastic game in multi-server mobile edge computing networks," in *Proceedings of the 2019 IEEE International Conference on Smart Internet of Things (SmartIoT)*, pp. 77–84, Tianjin, China, August 2019.

[28] H. Sun, F. Zhou, and R. Q. Hu, "Joint offloading and computation energy efficiency maximization in a mobile edge computing system," *IEEE Transactions on Vehicular Technology*, vol. 68, pp. 3052–3056, 2019.

[29] Z. Ding, J. Xu, O. A. Dobre, and H. Vincent Poor, "Joint power and time allocation for NOMA-MEC offloading," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 6, pp. 6207–6211, 2019.

[30] X. Poor, H. Tian, L. Jiang et al., "Selective offloading in mobile edge computing for the green Internet of Things," *IEEE Network*, vol. 32, no. 1, pp. 54–60, 2018.

[31] J. Liu, P. Li, J. Liu, and J. Lai, "Joint offloading and transmission power control for mobile edge computing," *IEEE Access*, vol. 7, pp. 81640–81651, 2019.

[32] J. Zhao, Q. Li, Y. Gong, and K. Zhang, "Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 7944–7956, 2019.

[33] Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu, and X. S. Shen, "TOFFEE: task offloading and frequency scaling for energy efficiency of mobile devices in mobile edge computing," *IEEE Transactions on Cloud Computing*, p. 1, 2019.

[34] Q.-V. Pham, L. B. Le, S.-H. Chung, and W.-J. Hwang, "Mobile edge computing with wireless backhaul: joint task offloading and resource allocation," *IEEE Access*, vol. 7, pp. 16444–16459, 2019.

[35] H. Guo, J. Liu, J. Zhang, W. Sun, and N. Kato, "Mobile-edge computation offloading for ultradense IoT networks," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4977–4988, 2018.

[36] Y. Zhang, C. Yang, J. Li, and Z. Han, "Distributed interference-aware traffic offloading and power control in ultra-dense networks: mean field game with dominating player," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 9, pp. 8814–8826, 2019.

[37] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2016.

[38] H. Jin, X. Zhu, and C. Zhao, "Computation offloading optimization based on probabilistic SFC for mobile online gaming in heterogeneous network," *IEEE Access*, vol. 7, pp. 52168–52180, 2019.