Stochastic Congestion Game for Load Balancing in Mobile-Edge Computing

Fenghui Zhang and Michael Mao Wang

***Abstract*—Mobile-edge computing can reduce task execution delay and improve the Quality of Experience (QoE) for the network edge users. However, when there are multiple independent cloudlets in the network with the mobile users offloading tasks randomly, how to maintain the load balancing of the independent cloudlets, how to improve the quality of service and users' QoE are still issues need to be solved. To this end, we study these issues from the perspective of game theory and propose decentralized learning algorithms. First, we turn the cloudlets load-balancing issue into a competition that each user minimizes its task execution time, and then a stochastic congestion game with incomplete information is proposed. Second, based on the existence proof of the Nash equilibria by using potential game theory, we propose a multiuser decentralized learning algorithm to obtain the pure Nash equilibrium strategy of each user. Then, an ordinary differential equation is derived to prove the convergence of the algorithm. Finally, we propose two application scenarios, one is for static users and the other is for dynamic users, and then the performances of the algorithm in a static scenario is tested. In order to adapt to dynamic scenarios, and further improve the performance and reduce communication costs, we propose a decentralized learning algorithm with termination condition. The experiments show that this algorithm can improve the load balancing of the multicloudlet system, and enhance the quality of service.**

***Index Terms*—Incomplete information, load balancing, mobile-edge computing (MEC), potential game, stochastic game.**

I. Introduction

With the proliferation of smartphones and Internet-of-Things (IoT) devices, more and more new mobile applications, such as interactive gaming, face recognition, and augmented reality have emerged and drawn increasing interests [1]. These sophisticated applications usually require significant amounts of computation resources and energy which, however, cannot be directly afforded by most mobile devices due to their limited computation resources and battery capacities [2], [3]. Mobile cloud computing which allows mobile devices to offload tasks to cloud can significantly reduce execution time and energy consumption. However, the long distance between users and remote cloud server causes that many individual round trips take hundreds to thousands of milliseconds, which makes it difficult to run delay-sensitive applications in the remote clouds [4]. Therefore, in order to reduce the transmission delay, an alternative is to deploy cloudlets at the edge of the network to provide cloud computing services for nearby mobile devices, i.e., mobile-edge computing (MEC) [5].

With MEC, mobile devices can offload tasks to proximate cloudlets, which will significantly reduce transmission delay and improve user's Quality of Experience (QoE). Due to the enormous advantages of MEC, many cloud computing providers (CCPs) construct their cloudlets at the edge of networks. In some areas where there are massive IoT devices or mobile users, different CCPs may establish their cloudlets at the network edge, which are independent of each other [6], [7]. Consequentially, it brings the users more convenience, for they have more choices to select the favorite cloudlet for task offloading. However, another scenario will appear, where most of the users choose one or two cloudlets, while the other cloudlets are idle. This is because the independent users will not negotiate with each other before task offloading. Under this scenario, the users' QoE will reduce significantly for the average waiting time becomes longer, and it will also cause the waste of computing resources. At the same time, due to the mobility and state uncertainty of users, the number of offloading events is stochastic in different time slots. Until now, this is still an open issue need to be resolved, the main questions are how to maintain load balancing between cloudlets in a distributed manner? how to improve users' QoE with a lower cost?

Recently, different approaches are proposed to improve the load balancing between cloudlets, e.g., optimization, game, and auction [7]–[11]. However, these approaches require a controller in the system and need frequent communication between controller, cloudlet, and mobile device. In fact, the controller does not always exist in the system and the mobile devices always offload their randomly generated tasks without communicating with other devices. Hence, under this scenario, how to maintain load balancing is a huge challenge. To this end, we describe the process of cloudlets load balancing as a competition that the independent users decide their task offloading strategies to improve their utility. Then, a stochastic congestion game with incomplete information is employed to describe this competition. After that, a decentralized learning algorithm is proposed to improve the service quality of the

Manuscript received May 27, 2020; accepted June 30, 2020. Date of publication July 8, 2020; date of current version January 7, 2021. This work was supported in part by the National Natural Science Foundation of China under Grant 61771128; in part by the Natural Science Foundation of Anhui Province under Grant 1908085MF213; and in part by the Key Project of Anhui Education Department under Grant KJ2018A0411. *(Corresponding author: Michael Mao Wang.)*

Fenghui Zhang is with the School of Information Science and Engineering, Southeast University, Nanjing 211102, China (e-mail: fhzhang@seu.edu.cn).

Michael Mao Wang is with the School of Information Science and Engineering, Southeast University, Nanjing 211102, China (e-mail: wangmao@seu.edu.cn).

Digital Object Identifier 10.1109/JIOT.2020.3008009

system and the user's QoE. We also propose a decentralized learning algorithm with the termination condition to further improve the system performance. The main contributions of this article can be summarized as follows.

1) We consider the load-balancing issue under the scenario that multiuser offload tasks to multicloudlet randomly when all of them are independent. Based on the random analysis of the system, we model the load balancing of cloudlets and service quality improvement as a stochastic congestion game with incomplete information that users compete with each other to maximize their utility.
2) We rigorously prove that this stochastic congestion game is a potential game, and thus the existence of Nash equilibrium is guaranteed. Then a decentralized learning algorithm is proposed to solve the game. We prove the proposed algorithm converges to Nash equilibrium by describing it as an ordinary differential equation.
3) In order to adapt to dynamic scenarios and further improve system performance, we investigate the convergence characteristics of the game, and then a decentralized learning algorithm with termination condition is proposed. This algorithm can significantly reduce system communication costs and achieve load balancing of the cloudlets.
4) Different parameters are used to test the convergence of the algorithms. Compared with the existing approaches, the equilibrium strategy obtained from this game can improve system load balancing without a coordinator.

The remainder of this article is organized as follows. After reviewing the related works in Section II, we introduce the system model in Section III, the existence of Nash equilibrium points is proved in Section IV. In Section V, a decentralized learning algorithm is proposed and its convergence is analyzed. The application of the game is presented and its performance is demonstrated in Section VI. In Section VII, a decentralized learning algorithm with termination condition is proposed to further improve the system performance. Finally, Section VIII concludes this article with future works.

II. Related Works

In MEC, multiple cloudlets may be deployed at the edge of the network to provide cloud computing services. Ensuring that each cloudlet has high resource utilization is an important guarantee for MEC service quality. Therefore, load balancing as an important direction has attracted more and more attention. Optimization and scheduling are commonly used methods. For example, in [12], Dong *et al.* utilized a heuristic task clustering method and a glowworm swarm optimization algorithm to achieve long-term load balancing of edge data centers. Lin and Tsai [13] proposed a hierarchical edge-cloud SDN controller system to enhance network scalability and reduce computation delay. Similarly, Sotiriadis *et al.* [14] designed a dynamic reconfiguration system to ensure the load balancing of each cloud. Dai *et al.* [7] formulated the joint load balancing and offloading issue as a linear programming problem to maximize system utility. Moreover, in [15] and [16], multiple scheduling algorithms are tested to measure the performance of load balancing. However, in these approaches, a powerful central control node is needed to schedule and allocate tasks.

Having an appropriate task offloading scheme can also improve system performance and QoE of users. Delgado and Labeau [17] proposed a delay-aware load-balancing algorithm to reduce the multiaccess wireless network delay, but it is only applicable to a scenario where one user offloads tasks. In [10], Hong *et al.* proposed a multihop cooperative computation-offloading game for discrete multiple devices to minimize the latency and energy consumption. In [18] and [19], a Stackelberg game is used to formulate the interaction between a cloudlet and users to maximize the cloudlet revenue and minimize the cost of users. Jin *et al.* [11] proposed an auction approach to stimulate the cloudlet to provide more resources. These approaches can improve the QoE of users and encourage the cloudlet to provide resources, but only one cloudlet is included in the system. Zhang *et al.* [20] proposed a software-defined networking-based load-balancing task offloading scheme in FiWi enhanced vehicular edge computing networks. Sthapit *et al.* [8] used a network of queues and linear programming to solve load balancing in mobile *ad hoc* clouds. Lyu *et al.* [21] proposed a heuristic offloading decision algorithm to maximize system utility. In these schemes, users need to negotiate with each other to ensure a better task offload scheme which requires frequent information interaction between them.

In order to improve the overall system performance and the QoE of users. Dai *et al.* [7] proposed a resource allocation scheme in a multiuser multiserver vehicle edge computing system. Zhang *et al.* [22] presented a real-time distributed algorithm to achieve the maximum utility of the overall system. Du *et al.* [23] formulated a dual-side optimization to minimize the cost of vehicles and the MEC server at the same time. In [9] and [24], an optimization method was used for resource allocation in the edge computing system. In these approaches, a central controller is needed to coordinate the actions of multiple cloudlets and users. However, in some edge computing systems, the central controller does not exist. Meanwhile, due to the limitation of distance and energy, users will offload the randomly generated tasks without negotiation with others. Under this scenario, how to maintain load balancing is a huge challenge. To this end, we propose a stochastic congestion game to solve the load imbalance issue caused thereby.

III. System Model and Stochastic Congestion Game

In this section, we first introduce the composition of the system, and then give the task execution model of cloudlet. After that, we use cloudlet execution time to describe the cost of users who offload tasks to it. Finally, a stochastic congestion game with incomplete information is proposed to characterize the process of users' competition.

A. System Model

As shown in Fig. 1, we consider an MEC system consisting of multiple base stations, cloudlets, and a large number of

users. The set of cloudlets is $\mathcal{M} = \{1, \ldots, M\}$, while j is the index of cloudlet. The cloudlets are independent of each other owned by different CCPs. There are a set of $\mathcal{N} = \{1, \ldots, N\}$ users in the communication range of these base stations, while i is the index of users. The users can choose any of the cloudlets for offloading, once the selection is decided, they will offload tasks to the corresponding cloudlet. Although users are distributed in different cells and have different wireless resource environments, the workload of each cloudlet depends only on the number of users, the calculation of tasks, and the offloading action. Once these are determined, the workload of the cloudlet has been determined. Therefore, in this article, we only consider the impact of the user's offloading action on the cloudlet's load balancing, ignoring the user's wireless transmission.

In the system, not every user has to offload tasks in each time slot, if it has, then it will offload. We define a user who offloads tasks in a time slot as an active user. Therefore, the number of active users in each time slot is stochastic. Each offloaded task has two properties which are data size (in bits) and computing size (in CPU cycles). The data size of each task is independently and identically distributed in every time slot with the mean value z_i for user i [25]. We further consider the data processing density as g (in cycles/b), which is the number of CPU cycles required for processing a unit bit data. The value of g depends on the application type which is considered to be independent and identically distributed with mean value g_i for user i [26]. The maximum value of the data size z_i is $z_{\max}$ and the maximum value of the processing density g_i is $g_{\max}$.

We consider that the task execution of cloudlets is performed in a time-slot manner. At the beginning of each time slot, the active users offload tasks to their preference cloudlets, then the cloudlets execute these tasks and return. For in each time slot, the number of active users is stochastic and the offloading actions are uncertain, the workload of each cloudlet may be quite different. We use c_j to denote the CPU processing capability of cloudlet j. Thus, if there are n tasks arrive and each of them has the same workload G, the time required to complete all of the tasks is (nG/c_j) [27]. When the number of tasks increases, the total workloads become heavier, and then the total execution time of the cloudlet becomes longer. So when all the users offload tasks to cloudlet j, the total execution time for cloudlet j is

$$r_j = \frac{\sum_{i=1}^{N} z_i g_i}{c_j}. \tag{1}$$

B. Stochastic Congestion Game With Incomplete Information

After the cloudlets complete all tasks, they append the execution time behind the results and transmit to users. Since each user enjoys a low computational delay, they will offload tasks to the cloudlet with shorter execution time. However, the users do not know if other users have tasks to be offloaded, nor do they know about other users' offloading strategies. Thus, each time slot the number of active users is stochastic, we can use the stochastic congestion game to model this scenario. Furthermore, all users are independent, they do not know the payoff of the other users, and even more, they do not know if other users are existent, therefore, this game is a stochastic congestion game with incomplete information. In the game, each user has a strategy vector, which is a probability distribution of the actions. When the elements in the strategy vector are 0 or 1, the strategy is a pure strategy, we use Σ_i to denote the pure strategy set of user i. We use a_i to denote an action of player i, and $\varphi_j(a_i)$ to denote an event that user i offloads tasks to cloudlet j, so the $\varphi_j(a_i)$ can be represented as

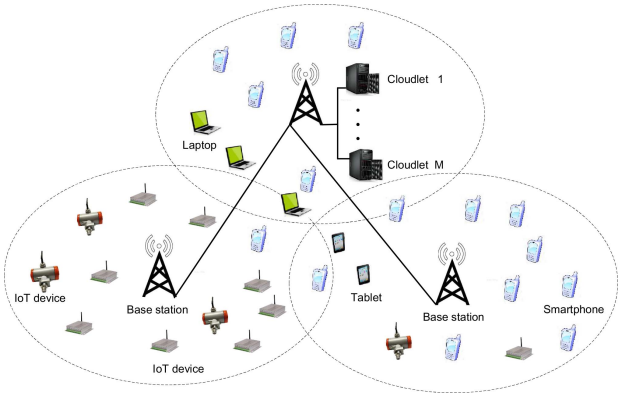


Fig. 1. Task offloading in the multicloudlet system.

$$\varphi_j(a_i) = \begin{cases} 1, & \text{if } a_i = j \\ 0, & \text{if } a_i \neq j. \end{cases} \tag{2}$$

For the convenience of analysis, we define a probability space as $(\Omega, \mathcal{D}, \mathcal{P})$, where Ω is a sample space, $\mathcal{D}$ is a minimal σ-algebra on subsets of Ω, and $\mathcal{P}$ is a probability measure on $(\Omega, \mathcal{D})$. In the game, Ω is the set of users $\mathcal{N}$, an element of $\mathcal{D}$ represent a user is active or inactive. We use $d_i = 1$ to denote the user i is an active user, while using $d_i = 0$ to denote the user i is an inactive user who have no task to offload. Let $w(t)$ denote an event in the probability space $(\Omega, \mathcal{D}, \mathcal{P})$ in time slot t. So in an event $w(t)$, the execution time of cloudlet j is

$$r_j(\mathbf{a}|w(t)) = \frac{\sum_{i=1}^{N} \varphi_j(a_i) d_i z_i g_i}{c_j} \tag{3}$$

where $\mathbf{a}$ represents action vector for all users, when $d_i = 0$, $a_i = 0$. We define l_j as the total workloads which are offloaded to cloudlet j in an event $w(t)$, so

$$l_j(\mathbf{a}|w(t)) = \left\{ \sum_{i=1}^{N} d_i \varphi(a_i) g_i z_i | a_i = j \right\}. \tag{4}$$

It is practical to consider that each user enjoys a low computational delay. So we can define the execution time of cloudlet as the cost of a user. For an event $w(t)$, when user i takes action a_i, its cost function can be described as

$$\begin{aligned} \tilde{v}_i(a_i, a_{-i}|w(t)) &= \sum_{j=1}^{M} \varphi_j(a_i) r_j(a_i, a_{-i}|w(t)) \\ &= \sum_{j=1}^{M} \varphi_j(a_i) \frac{l_j}{c_j} \end{aligned} \tag{5}$$

where $\tilde{v}_i$ is an instant cost for user i in an event $w(t)$, a_{-i} represents the action of all other users except user i. $w(t)$ represents an specific event, which is random at different time

slots. We define the event-based utility of user i as

$$\tilde{u}_i(a_i, a_{-i}|w(t)) = U - \tilde{v}_i(a_i, a_{-i}|w(t)) \tag{6}$$

where $\tilde{u}_i$ denotes the instant utility in an event $w(t)$, U represents the maximum tolerable cost of the users. It can be deemed that if the cost exceeds U, users will not offload tasks. In this article, we assume that $U > \max[\tilde{v}_i(a_i, a_{-i}|w(t))]$, for the users always benefit from task offloading.

Since the game is a stochastic game, we cannot obtain the optimal action in a time slot, however, through the average long-term utility, we can achieve the optimal strategy. The average utility of user i can be expressed as

$$\begin{aligned} u_i(a_i, a_{-i}) &= E\big[\tilde{u}_i(a_i, a_{-i}|w(t))\big] \\ &= \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T}\big[U - \tilde{v}_i(a_i, a_{-i}|w(t))\big] \\ &= U - \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T}\big[\tilde{v}_i(a_i, a_{-i}|w(t))\big]. \end{aligned} \tag{7}$$

Thus, we can formulate this noncooperative stochastic congestion game with incomplete information as $\mathcal{G} = \{\mathcal{N}, \mathcal{M}, \{\Sigma_i\}_{i\in\mathcal{N}}, \{\mathcal{U}_i\}_{i\in\mathcal{N}}\}$. Then, the proposed game can be expressed as

$$\begin{aligned} \max_{a_i\in\mathcal{A}_i}\ & u_i(a_i, a_{-i}) \\ \text{s.t.}\ & (2)-(4) \\ & \forall i \in \mathcal{N}, j \in \mathcal{M} \end{aligned} \tag{8}$$

where $\mathcal{N}$ and $\mathcal{M}$ represent the set of users and the set of cloudlets, respectively. Σ_i indicates the pure strategy set of user i, $\mathcal{U}_i$ denotes the set of the ith user's utility and $\mathcal{A}_i$ represents its action set.

IV. Nash Equilibrium Analysis

Before solving this stochastic congestion game, we should determine whether the game has Nash equilibria. Hence, in this section, we present the definition of Nash equilibrium point and the potential game. Then, we prove that the cloudlet selection game is a potential game that has Nash equilibria.

Definition 1: A cloudlet selection profile $\mathbf{a}^* = \{a_1^*, \ldots, a_N^*\}$ is a pure strategy Nash equilibrium point of the noncooperative game, if no player can improve its utility by deviating unilaterally

$$u_i\big(a_i^*, a_{-i}^*\big) \geq u_i\big(a_i, a_{-i}^*\big). \tag{9}$$

When a game is a potential game, it means that the deviation in the utility of every single user is consistent with the deviation in the potential function [28]. If a game is an exact potential game, it must satisfy Definition 2.

Definition 2: A game is an exact potential game if the deviation in the utility of an arbitrary player i can be reflected by the deviation in function $\Phi(a_1, \ldots, a_N)$

$$\begin{aligned} u_i(a_i, a_{-i}) - u_i(x_i, a_{-i}) &= \Phi(a_i, a_{-i}) - \Phi(x_i, a_{-i}) \\ &\forall i \in \mathcal{N}, a_i, x_i \in \mathcal{A}_i. \end{aligned} \tag{10}$$

In Theorem 1, we will find a potential function $\Phi(a_1, \ldots, a_N)$ to prove this game is an exact potential game.

Theorem 1: The cloudlet selection game is an exact potential game which has at least one pure strategy Nash equilibrium point.

Proof: We construct a potential function of this congestion game as

$$\Phi(a_i, a_{-i}) = NU - \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{M}\left(\frac{l_j^1 + \cdots + l_j^k}{c_j}\right) \tag{11}$$

where l_j^1 and l_j^k denote the workloads of cloudlet j when one and k users offload tasks to it, respectively, where $k \leq N$.

Generally, we use a_i to represent that user i offloads task to cloudlet m, $m \in \mathcal{M}$, while using another action x_i to represent the action of offloading task to the cloudlet $m+1$, $m+1 \leq M$. So when the other users do not change their action a_{-i}, the potential function is

$$\begin{aligned} &\Phi(a_i, a_{-i}) \\ &= NU - \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{M}\left(\frac{l_j^1 + \cdots + l_j^k}{c_j}\right) \\ &= NU - \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T}\left[\sum_{j=1}^{m-1}\left(\frac{l_j^1 + \cdots + l_j^k}{c_j}\right)\right. \\ &\qquad + \frac{l_m^1 + \cdots + l_m^k}{c_m} + \frac{l_{m+1}^1 + \cdots + l_{m+1}^k}{c_{m+1}} \\ &\qquad \left. + \sum_{j=m+2}^{M}\left(\frac{l_j^1 + \cdots + l_j^k}{c_j}\right)\right]. \end{aligned} \tag{12}$$

When the user i takes action x_i, the potential function is

$$\begin{aligned} &\Phi(x_i, a_{-i}) \\ &= NU - \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{M}\left(\frac{l_j^1 + \cdots + l_j^k}{c_j}\right) \\ &= NU - \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T}\left[\sum_{j=1}^{m-1}\left(\frac{l_j^1 + \cdots + l_j^k}{c_j}\right)\right. \\ &\qquad + \frac{l_m^1 + \cdots + l_m^{k-1}}{c_m} + \frac{l_{m+1}^1 + \cdots + l_{m+1}^{k+1}}{c_{m+1}} \\ &\qquad \left. + \sum_{j=m+2}^{M}\left(\frac{l_j^1 + \cdots + l_j^k}{c_j}\right)\right]. \end{aligned} \tag{13}$$

So when the user i change its action, the deviation of the potential function is

$$\Phi(a_i, a_{-i}) - \Phi(x_i, a_{-i}) = \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T}\left(\frac{l_{m+1}^{k+1}}{c_{m+1}} - \frac{l_m^k}{c_m}\right). \tag{14}$$

When the user i changes its action, the deviation of its utility is

$$\begin{aligned} &u_i(a_i, a_{-i}) - u_i(x_i, a_{-i}) \\ &\quad = U - \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T}\big[\tilde{v}_i(a_i, a_{-i}|w(t))\big] - U \end{aligned}$$

$$+ \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \left[\tilde{v}_i(x_i, a_{-i}|w(t))\right] = \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \left(\frac{l_{m+1}^{k+1}}{c_{m+1}} - \frac{l_m^k}{c_m} \right). \quad (15)$$

Then, we can obtain the following equation:

$$u_i(a_i, a_{-i}) - u_i(x_i, a_{-i}) = \Phi(a_i, a_{-i}) - \Phi(x_i, a_{-i}). \quad (16)$$

While a_i and x_i can be any actions of user i, so for each user and its action, (16) holds. Therefore, the deviation in the utility of any player i is equal to the deviation in potential function Φ. This congestion game is an exact potential game with Φ serving as the potential function. According to [28], any global or local maxima of the potential function constitute a pure strategy Nash equilibrium point of this game. Therefore, Theorem 1 is proven. ∎

In the system, we consider the cost of a cloudlet in an event is the execution time, thus the cost of cloudlet j can be expressed as

$$\tilde{C}_j(\mathbf{a}|w(t)) = \frac{\sum_{i=1}^{N} \varphi_j(a_i) d_i z_i g_i}{c_j} = \frac{l_j}{c_j}. \quad (17)$$

If there is a controller in the system to coordinate the action of users for minimizing the maximum execution time of these cloudlets. Then, the edge computing system can reach its optimal service quality. The optimization equation can be expressed as

$$\min_{a_1,\ldots,a_N} \max\left(\frac{l_1}{c_1}, \ldots, \frac{l_j}{c_j}, \ldots, \frac{l_M}{c_M} \right) \quad \text{s.t.} \quad (2)-(4) \quad \forall a_i \in \mathcal{A}_i \quad \forall j \in \mathcal{M}, \quad i \in \mathcal{N}. \quad (18)$$

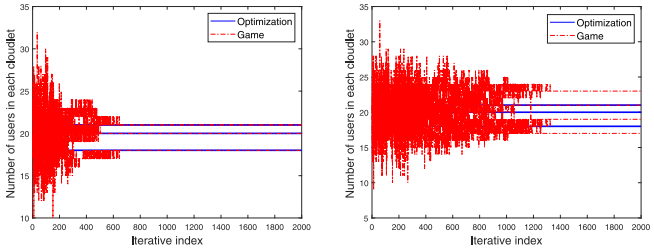
Theorem 2: The optimal pure Nash equilibrium strategy of the game can minimize the maximum execution time of the cloudlet in an event $w(t)$.

Proof: From (8), we know that the object of each user is to maximize the average long-term utility. Since the maximum tolerable cost U is fixed, so we can rewrite (8) in an event as

$$\min_{a_i\in\mathcal{A}_i} \tilde{v}_i(a_i, a_{-i}|w(t)) \quad \text{s.t.} \quad (2)-(4) \quad \forall i \in \mathcal{N}, j \in \mathcal{M}. \quad (19)$$

The above equation means user i chooses its offloading action to minimize the execution time of cloudlet j. As in (5), it is to minimize (l_j/c_j). While in the game all users will offload tasks to the cloudlet which has the minimum execution time, consequentially, the maximum execution time of the cloudlets will be reduced, which is also the object of (18). The difference between them is that the behavior of (19) is distributed, and the behavior of (18) is centralized. However, this potential game may have many pure strategy Nash equilibria, only the optimal actions can achieve the object of (18). ∎

In order to show Theorem 2 clearly, we demonstrate two instances in Fig. 2. In the simulation, the number of active users is 80, each task has the same g and z, and there are four



(a) (b)

Fig. 2. Comparisons of the game approach and optimization approach in an event. (a) Task allocation of the two methods is the same. (b) Task allocation of the two methods is inconsistent.

cloudlets in the network edge. Fig. 2(a) shows a case that by using the game approach, the number of tasks in each cloudlet is equal to that by using the optimization approach. Fig. 2(b) shows a case that the number of tasks in each cloudlet is inconsistent between the game approach and the optimization approach. But we can see that the average difference is only about 1 task between the two approaches. Although the effects of the two approaches are inconsistent in most cases, the game can also achieve favorable task allocation if the optimization approach cannot be used.

From Theorem 2, we know that the optimal pure Nash equilibrium strategy of the game can minimize the maximum execution time of the cloudlet in an event $w(t)$, which means that the optimal equilibrium can achieve globe optimal. Since the object of the game is load balancing of the system, it is also quite important to find the upper bound of the maximum deviation between different cloudlets' execution time when the game is at an equilibrium point.

We first assume that the game has already converged to Nash equilibrium. At the equilibrium point, there are n_j users offload tasks to cloudlet j and their workload is $s_1^j, \ldots, s_{n_j}^j$. We consider i is the index of user and $s_1^j \le \ldots \le s_{n_j}^j$. Therefore, the execution time of cloudlet j is $[(\sum_{i=1}^{n_j} s_i^j)/c_j]$ and we assume $[(\sum_{i=1}^{n_1} s_i^1)/c_1] \ge \ldots \ge [(\sum_{i=1}^{n_M} s_i^M)/c_M]$. According to the definition of Nash equilibrium, no player can improve its utility by deviating unilaterally from the Nash equilibrium point. Thus, if a user deviates from equilibrium, i.e., the user offloads its tasks s_1^1 to cloudlet j, according to the definition of Nash equilibrium, we can get

$$\frac{\sum_{i=1}^{n_1} s_i^1}{c_1} \le \frac{\sum_{i=1}^{n_j} s_i^j + s_1^1}{c_j} \quad (20)$$

which means if the user offloads workload s_1^1 to other cloudlets, its waiting time will become longer. Accordingly, we know that s_1^1 is the smallest workload of cloudlet 1, if the workload s_1^1 is offloaded to cloudlet M with the shortest execution time, (20) still holds. So, we can get

$$\frac{\sum_{i=1}^{n_M} s_i^M}{c_M} \le \frac{\sum_{i=1}^{n_1} s_i^1}{c_1} \le \frac{\sum_{i=1}^{n_M} s_i^M + s_1^1}{c_M}. \quad (21)$$

We know the maximum difference in the execution time of the cloudlets is $[(\sum_{i=1}^{n_1} s_i^1)/c_1] - [(\sum_{i=1}^{n_M} s_i^M)/c_M]$, which is less than or equal to $[(\sum_{i=1}^{n_M} s_i^M + s_1^1)/c_M] - [(\sum_{i=1}^{n_M} s_i^M)/c_M] = (s_1^1/c_M)$ according to (21). Therefore, when the game is in

Algorithm 1 Decentralized Learning Algorithm for Multiuser Task Offloading

Require: z_i, g_i, c_j, U, the initial mixed strategy for each user $\{\mathbf{p}_1(0), \ldots, \mathbf{p}_N(0)\}$
Ensure: Action of each user $\{a_1, \ldots, a_N\}$
While The mixed strategy $\mathbf{p}_i$ is not a pure strategy $\mathbf{e}_i$
1: At time slot t, each active user i selects the offloading action $a_i(t)$ according to its current strategy vector $\mathbf{p}_i(t)$. The inactive users keep silent.
2: After offloading, the active users evaluate their utility $\tilde{u}_i$ according to (6).
3: All active users update their strategy vector according to the following rule

$$\mathbf{p}_i(t+1) = \mathbf{p}_i(t) + b\tilde{\mathbf{u}}_i(t)(\mathbf{e}_{ai} - \mathbf{p}_i(t)),$$

where $0 < b < 1$ is a step size, $\mathbf{e}_{ai}$ is a unit vector of appropriate dimension with M component unity. When user i offload task to cloudlet j, The corresponding position of $\mathbf{e}_{ai}$ is 1, and other position are 0. Meanwhile, the inactive mobile users keep their strategy vector unchanged.
4: Update $t = t + 1$.
End while
The active users select actions with a corresponding position of 1 in their strategy vector.

Nash equilibrium, the upper bound of the maximum deviation of these cloudlets' execution time is (s_1^1/c_M), where s_1^1 is the minimum workload offloaded to the cloudlet with the longest execution time, and c_M is the CPU processing capability of the cloudlet with the shortest execution time.

V. Decentralized Learning Approach and the Convergence

In this stochastic congestion game with incomplete information, a user is not conscious of the states and actions of the others. For each of them will choose cloudlet according to its own preference, there may be many users choose one cloudlet for task offloading while the other cloudlets are idle. This will induce load imbalance, which will reduce the quality of service for the edge computing system. To solve this issue, we propose a decentralized learning approach. The basic idea of the algorithm is to gradually learn from the initial mixed strategy to obtain the pure Nash equilibrium strategy for each user. Let $\mathbf{p}_i(t) = \{p_{i1}(t), \ldots, p_{iM}(t)\}$ denote the mixed strategy vector of the ith user, $\sum_{j=1}^{M} p_{ij} = 1$. $p_{ij}(t)$ denotes the probability that the ith user chooses the j^{th} action at time slot t. $\mathbf{p}_i(0)$ is the initial mixed strategy of player i, which can be an arbitrarily distributed vector according to the ith user preference. Each play of the game consists that each active user automatically chooses an action independently and randomly according to its current strategy vector. The learning algorithm adopted by each user is given in Algorithm 1.

A. Convergence Analysis

We first rewrite the update rule of step 3 in Algorithm 1

$$\begin{aligned}\mathbf{p}_i(t+1) &= \mathbf{p}_i(t) + b\tilde{\mathbf{u}}_i(t)\big(\mathbf{e}_{ai} - \mathbf{p}_i(t)\big) \\ &= \begin{cases} p_i(t) + b\tilde{u}_i(t)(1 - p_i(t)), & \text{if } a_i = j \\ p_i(t) - b\tilde{u}_i(t)p_i(t), & \text{if } a_i \neq j. \end{cases}\end{aligned} \tag{22}$$

The first case is the update rule of the strategy element that the user selects the corresponding action, another case is the update rule of the other strategy elements.

From (22), we know that the algorithm is following the Markov process, which means the current strategy update is only related to the previous time slot. Therefore, the procedure for proving the convergence of the algorithm has three steps. First, an ordinary differential equation is derived, whose performance can approximate the asymptotic behavior of the algorithm. Second, we prove that the stationary points of the ordinary differential equation are Nash equilibrium points of the game. Third, we prove that the algorithm converges to the stationary points rather than to other points.

The algorithm is an iterative process, so we can rewrite (22) as

$$P(t+1) = P(t) + bG\big[P(t), \mathbf{a}(t), \tilde{\mathbf{u}}(t)\big] \tag{23}$$

where $P(t) = \{\mathbf{p}_1, \ldots, \mathbf{p}_N\}$, $P(t) \in \mathcal{P}$. $\mathbf{p}_i$ denotes the strategy vector of user i, $\mathbf{a}(t) = \{a_i(t), \ldots, a_N(t)\}$ represents all the players' action at time slot t, $\tilde{\mathbf{u}}(t) = \{\tilde{u}_i(t), \ldots, \tilde{u}_N(t)\}$ represents the utility of the player under $\mathbf{a}(t)$, b is step size of the algorithm. $G(\cdot)$ is a incremental function specified by (22).

Consider a piecewise-constant interpolation of $P(t)$, $P^b(\cdot)$ is defined as

$$P^b(k) = P(t) \quad k \in [bt, b(t+1)) \tag{24}$$

where $P^b(\cdot)$ is right continuous and have left hand limits. According to [29, Th. 3.1], we can derive the following theorem.

Theorem 3: With a sufficiently small step size $b \to 0$, the sequence of interpolated processes $\{P^b(\cdot)\}$ will converge weakly to the solution of the following ordinary differential equation:

$$\frac{dP}{dt} = f(P) \tag{25}$$

the initial state $P(0)$ is an arbitrary probability distribution, and $f(P)$ is the mean of interpolated equation $G(\cdot)$

$$f(P) = E\big[G\big(P(t), \mathbf{a}(t), \tilde{\mathbf{u}}(t)\big)|P(t) = P\big]. \tag{26}$$

Therefore, the algorithm can be represented as (25). We define P is a stationary point when the differential equation $(dP/dt) = 0$. Then we prove that the stationary points of this ordinary differential equation are Nash equilibrium points.

Theorem 4: All Nash equilibria of the game are stationary points of the ordinary differential equation when the step size b is sufficiently small.

Proof: According to Theorem 3, we can get

$$\frac{dp_{ij}}{dt} = f_{ij}(P) \tag{27}$$

where p_{ij} is the probability that user i offloads tasks to cloudlet j, $0 \le p_{ij} \le 1$.

By using (22) and (26), we can get

$$\begin{aligned} f_{ij}(P) = {} & p_{ij}\big(1 - p_{ij}\big)E\big[\tilde{u}_i(a_i(t), a_{-i}(t))|P(k) = P, a_i(t) = j\big] \\ & + \sum_{j'=1, j' \neq j}^{M} p_{ij'}\big(-p_{ij}\big)E\big[\tilde{u}_i(a_i(t), a_{-i}(t))|P(k) = P \\ & \qquad a_i(t) = j', a_i(t) \neq j\big] \end{aligned}$$

$$= p_{ij}\left(\sum_{j=1}^{M} p_{ij} - p_{ij}\right) h_{ij}(P) + \sum_{j'=1, j'\neq j}^{M} p_{ij'}\left(-p_{ij}\right) h_{ij'}(P)$$
$$= p_{ij}\sum_{j'=1}^{M} p_{ij'}\left[h_{ij}(P) - h_{ij'}(P)\right] \quad (28)$$

where $j, j' \in \mathcal{M}$. $h_{ij}(P)$ denotes the expected utility of user i when it offloads task to cloudlet j, while the other users i' employ the mixed strategy under each action $a_{i'j}$ with probability $p_{i'j}$

$$h_{ij}(P) = \sum_{a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_N} \tilde{u}_i\left(a_1,\ldots,a_{i-1},a_{ij},a_{i+1},\ldots,a_N\right) \times \prod_{i'\neq i, i'\in\mathcal{N}} p_{i'j}. \quad (29)$$

By using (27) and (28), the ordinary differential (25) in Theorem 3 can be rewritten as

$$\frac{dp_{ij}}{dt} = p_{ij}\sum_{j'=1}^{M} p_{ij'}\left[h_{ij}(P) - h_{ij'}(P)\right] \quad \forall i \in \mathcal{N}, j, j' \in \mathcal{M}. \quad (30)$$

From (30), we can get the following equation:

$$\frac{dp_{ij}}{dt} = p_{ij}\left[h_{ij}(P) - g_i(P)\right] \quad (31)$$

where $h_{ij}(P)$ is (29), and $g_i(P)$ is the average utility when user i adopts mixed strategy

$$g_i(P) = \sum_{a_1,\ldots,a_N} \tilde{u}(a_1,\ldots,a_N)\prod_{i\in\mathcal{N}} p_{ij} \quad (32)$$

when $(dp_{ij}/dt) = 0$, $h_{ij}(P) - g_i(P) = 0$. According to [30, Th. 3.1], when $h_{ij}(P) = g_i(P)$, P is the Nash equilibrium point of the game. Then Nash equilibrium point is stationary point of the ordinary differential equation. ■

From Theorem 3, we know that in the algorithm, the asymptotic behavior of $P(t)$ can be regarded as an ordinary differential equation. Theorem 4 shows that the stationary points of this ordinary differential equation are Nash equilibrium points. Then we will prove that this algorithm is convergent and eventually converges to the stationary points rather than to other points.

Theorem 5: If there exists a bounded differential function F, such that for some positive constant c

$$\frac{\partial F}{\partial p_{ij}}(P) = c \cdot h_{ij}(P). \quad (33)$$

Then the learning algorithm converges to the stationary points.

Proof: We first prove that there exists a bounded equation that makes the algorithm converge to stationary points. Then, we prove that the bounded equation is the potential equation of the game

$$\frac{dF}{dt} = \sum_{i\in\mathcal{N}, j\in\mathcal{M}} \frac{\partial F(P)}{\partial p_{ij}}\frac{dp_{ij}}{dt}. \quad (34)$$

Substituting (30) into (34)

$$\frac{dF}{dt} = \sum_{i\in\mathcal{N}, j\in\mathcal{M}} \frac{\partial F(P)}{\partial p_{ij}} p_{ij}\sum_{j'\in\mathcal{M}} p_{ij'}\left[h_{ij}(P) - h_{ij'}(P)\right]$$
$$= c\sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{M}} h_{ij}(P)p_{ij}\sum_{j'\in\mathcal{M}} p_{ij'}\left[h_{ij}(P) - h_{ij'}(P)\right]$$
$$= c\sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{M}}\sum_{j'\in\mathcal{M}} p_{ij}p_{ij'}\left[\left(h_{ij}(P)\right)^2 - h_{ij'}(P)h_{ij}(P)\right] \quad (35)$$

when $j = j'$, $(h_{ij}(P))^2 - h_{ij'}(P)h_{ij}(P) = 0$, so we can rewrite (35) as

$$\frac{dF}{dt} = c\sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{M}}\sum_{j'\in\mathcal{M}, j'>j} p_{ij}p_{ij'}\left[\left(h_{ij}(P)\right) - h_{ij'}(P)\right]^2 \geq 0. \quad (36)$$

Thus, F is nondecreasing along the trajectories of the ordinary differential equation, so the algorithm is convergent until $(dF/dt) = 0$

$$\text{When} \quad \frac{dF}{dt} = 0$$
$$\text{then} \quad p_{ij}p_{ij'}\left[\left(h_{ij}(P)\right) - h_{ij'}(P)\right] = 0$$
$$f_{ij}(P) = 0.$$

Therefore, P is a stationary point of the ordinary differential equation.

We define the potential equation of the game for the mixed strategy is

$$\bar{\Phi}(a_1,\ldots,a_N) = E[\Phi(a_1,\ldots,a_N)|P] = \sum_{a_{ij}, j\in\{\mathcal{M}+1\}}\sum_{a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_N} \tilde{u}_i \times \left(a_1,\ldots,a_{i-1},a_{ij},a_{i+1}\ldots,a_N\right)\prod_{i'\neq i, i'\in\mathcal{N}} p_{i'j}. \quad (37)$$

Then, we can get

$$\frac{\partial\bar{\Phi}(a_1,\ldots,a_N)}{\partial p_{ij}} = \sum_{a_1,\ldots,a_{i-1},a_{i+1},\ldots,a_N} \tilde{u}_i \times \left(a_1,\ldots,a_{i-1},a_{ij},a_{i+1},\ldots,a_N\right) \times \prod_{i'\neq i, i'\in\mathcal{N}} p_{i'j} = h_{ij}(P). \quad (38)$$

Therefore, the potential function with mixed strategies $\bar{\Phi}(a_1,\ldots,a_N)$ can be deemed as the function F, which has bounds. From Theorems 3 and 4, we conclude that the algorithm finally converges to the Nash equilibrium points. ■

According to the analysis in [31]–[33], for a single user, each computation is the update of the strategy vector according to (22), so the number of computations is M. If the number of iterations is S, the total computational complexity is MS. Therefore, the complexity of the algorithm is extremely low and suitable for IoT devices with lower computational capabilities.

VI. Application and Performance Analysis of the Algorithm

In this section, we first give the application of the decentralized learning algorithm in a static scenario. Then, we conduct simulations to validate the proposed algorithm and its performances.

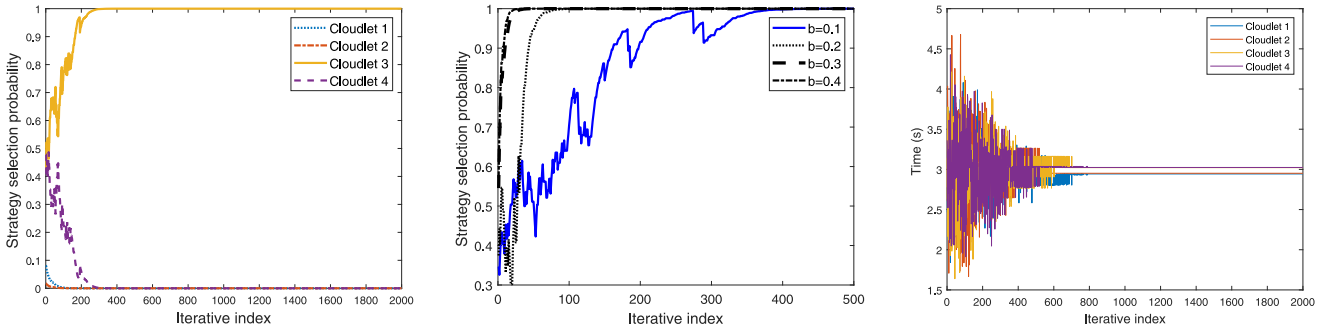


Fig. 3. Convergence of the decentralized learning algorithm. (a) Convergence of a user's strategy. (b) Influence of the step size b on the convergence of the algorithm. (c) Convergence of cloudlets' execution time.

A. Application of the Algorithm

In the coverage area of the base stations, if users need to offload tasks, they will choose one of the cloudlets for offloading according to their strategy vector. After the cloudlets finish all the tasks, they return the results to the users who offload tasks to them. In the result, a byte is added which represents the total execution time of the corresponding cloudlet. After the user receives the result, it updates the strategy vector according to (22). When another task is generated, it will offload according to the new strategy vector, until the vector gradually evolves into a unit vector. Then the user offloads tasks according to the unit vector.

B. Simulation Assumptions

We set up an MEC scenario by using MATLAB where a number of N users are randomly scattered in the coverage of these base stations, $N \in [50, 80]$. For easier observation, we set the probability of active users is equal to each other as 0.6 per time slot, i.e., in a time slot each user has a probability of 0.6 to offload task. Each user has a different average data size z_i at interval [0.15, 0.25] (in Mb), the computing size for each task is at interval [2, 5] (in K cycles/b) [25]. There are 4 cloudlets connected to the base stations, each of them has different computing ability at interval [3, 4] (in Ghz), which are belonged to different CCPs.

C. Convergence of the Algorithm

The convergence of the algorithm is shown in Fig. 3. In the simulations, the number of users is 80, the step size b is 0.1. Fig. 3(a) shows the convergence of a user's strategy vector. We can see that with the iteration of the algorithm the probability for the user selecting cloudlet 3 gradually becomes 1, and the probability for selecting other cloudlets gradually becomes 0. Since other users use the same algorithm, their strategy vectors also gradually converge to a unit vector. The element with a value of 1 in the vector is the cloudlet they will offload.

As we know in the algorithm, the users perform task offloading with a certain probability in each iteration, which indicates that when the more iterations of the algorithm, each user will conduct the more offloading attempts. So to reduce the number of iterations is also the object of the algorithm. In Fig. 3(b), we use step size b to reduce the number of iterations. As we can see from it, with b gradually increases, the number of iterations gradually decreases. When $b = 0.4$, it just uses about 20 iterations when the algorithm achieves the Nash equilibrium. Although the set of each simulation is the same, with different step size the algorithm may converge to different Nash equilibrium points. This is because the game has many Nash equilibrium points. When solving the game, the update speed of the strategy vector is determined by the step size and the different update speeds of strategies will lead to the different probability of strategy selection. Accordingly, different step size may cause the strategy vector to converge to different Nash equilibrium points. Fig. 3(b) shows a special case, when $b = 0.1$ the Nash equilibrium strategy of the user is to offload tasks to cloudlet 2, while b is 0.2, 0.3, and 0.4, the Nash equilibrium strategy is to offload tasks to cloudlet 4. However, despite converging to different Nash equilibrium points with different step size b, the conclusion is determined that the number of iterations decreases as the step size increases.

Fig. 3(c) demonstrates the convergence of cloudlets' execution time. We can see that with the iteration of the algorithm the execution time of each cloudlet gradually stabilizes. After the convergence of the algorithm, the execution times of the cloudlets are approximately equal to each other. The execution time of cloudlets 3 and 4 is about 3 s, while the execution time of cloudlets 1 and 2 is about 2.97 s. Therefore, by using the algorithm in each time slot, system load balancing can be achieved.

D. Performance Analysis

In order to verify the pure Nash equilibrium strategy obtained from the algorithm, we compare the average utility of users in Fig. 4. In each time slot, users offload tasks with a certain probability, so the amount of tasks performed by each cloudlet is random in each time slot. Therefore, in order to show the experiments more clearly, we display the average utility of the users for every ten time slots and 100 time slots in Fig. 4(a) and (b), respectively. Although the average utility of users is still a random value, from Fig. 4(a), we can see that the utility of each user by using the pure Nash equilibrium strategy is higher than by using the original mixed strategy in most of time slots. This is because when the original mixed

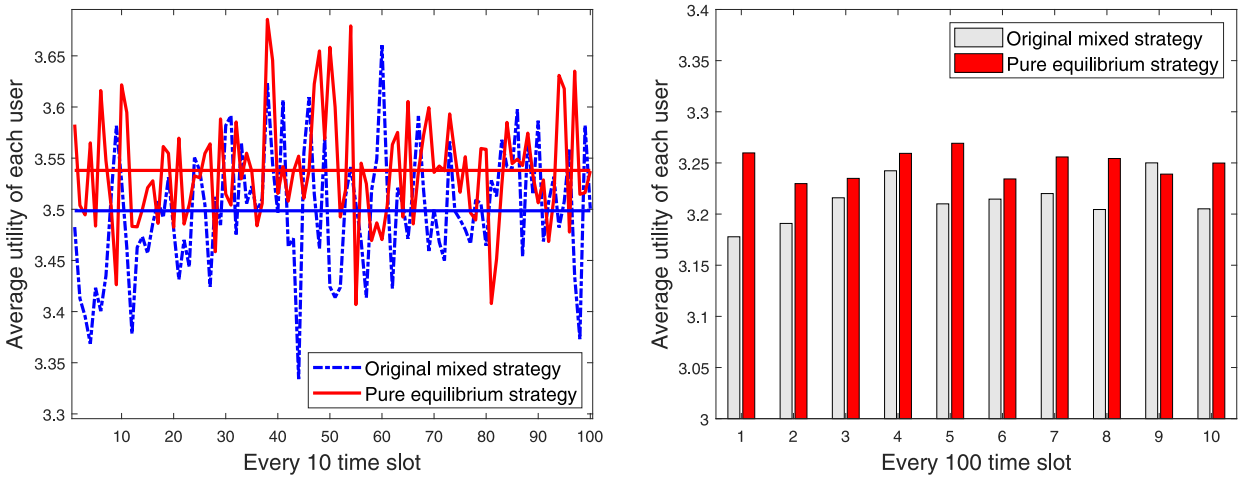


Fig. 4. Comparison of each user's average utility under different approaches. Comparison of each user's average utility every (a) ten time slots and (b) 100 time slots.

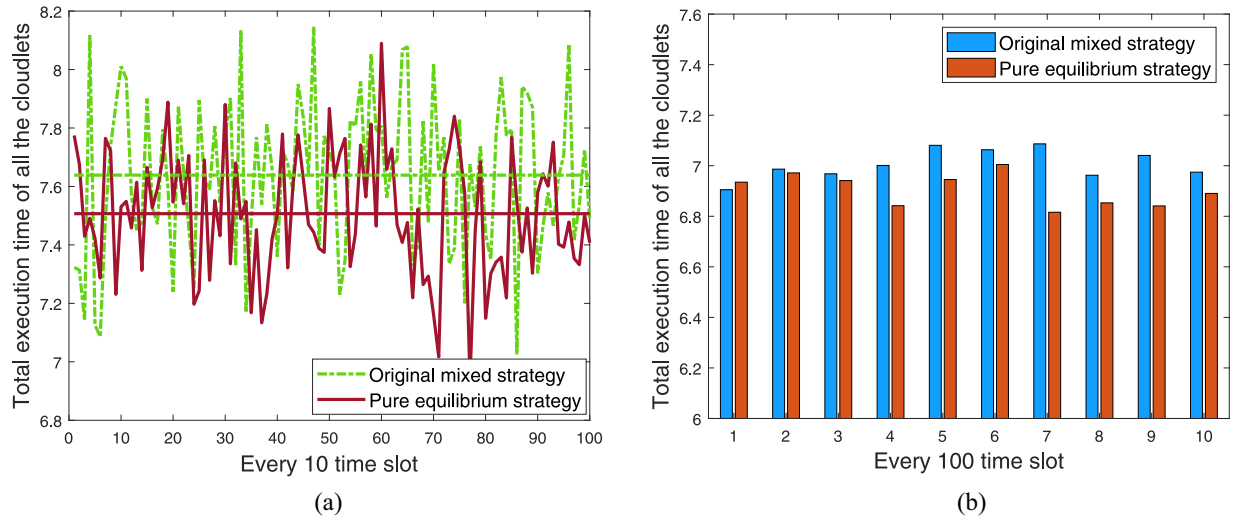


Fig. 5. Total execution time of the cloudlets under different schemes. Comparison of the total execution time of the cloudlets every (a) ten time slots and (b) 100 time slots.

strategy is adopted, each user will offload tasks only according to its preference, which may cause some cloudlets overload, resulting in longer running time. In Fig. 4(a), the two straight lines represent the average utility of the users over the entire execution cycle. We can see that by using the pure equilibrium strategy obtained from the game, each user acquires a higher utility than the original mixed strategy. Fig. 4(b) shows that the average utility of each user getting from the pure equilibrium strategy is higher than from the original mixed strategy every 100 time slots in most cases. In Fig. 4(b), there are only one of the ten tests is lower than the original mixed strategy. Therefore, from Fig. 4, we can conclude that by using the stochastic congestion game approach each user can improve the average utility.

Fig. 5 shows the total execution time of the cloudlets, while Fig. 5(a) and (b) demonstrates the total execution time for every ten time slots and 100 time slots, respectively. From Fig. 5(a), we also can see that by using pure equilibrium strategy, the total execution time of all the cloudlets is lower than by using the original mixed strategy. Fig. 5(a) and (b) shows that when using pure equilibrium strategy, the total execution time is lower in most cases not only in every ten time slots but also in 100 time slots. This is because the pure equilibrium strategy obtained from the game can improve the execution efficiency of the cloudlet system and thus reduce execution time.

We know that there is no coordinator in the cloudlet system, if it has, the coordinator can optimize task assignment, and then the cloudlet system can achieve the optimal service quality. In Fig. 6, the execution time of the cloudlet system is compared between the three approaches, which are using the original mixed strategy, stochastic game, and an optimization approach, respectively. As we can see from it, when the number of users increases, the execution time of

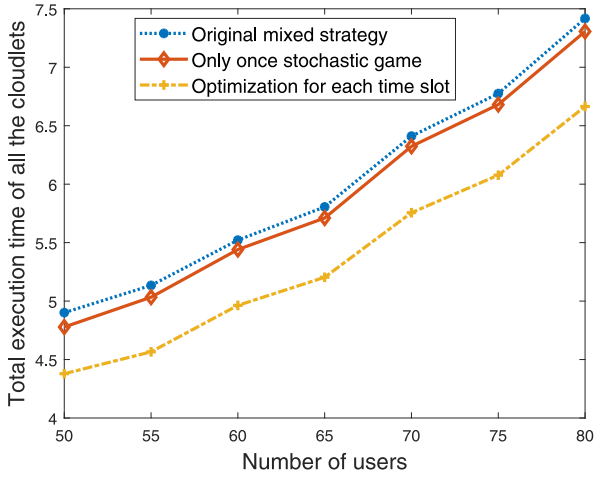


Fig. 6. Comparison of the system' total execution time under different approaches.

the cloudlet system becomes longer. When there is a coordinator in the system and optimization method is used, the system can achieve the optimal execution time among the three approaches. By using the Nash equilibrium strategy obtained from the stochastic game, the task execution time is lower than by using the original mixed strategy. Although by using stochastic game, the system will spend more time than there is a coordinator in the system, the huge advantage is that by using stochastic game, the system has little communication cost and does not require a coordinator, which is more suitable for the independent cloudlet system.

VII. Decentralized Learning Approach With Termination Condition and the Performance

From Fig. 6, we know that with Algorithm 1 the system spends more time than by using a coordinator. From Fig. 5, the workload of the cloudlets still has large fluctuations between different time slots. In addition, in a dynamic scenario, when the number of users in the system changes significantly, the performance of Algorithm 1 will deteriorate. So we need to further improve the system performance. An alternative is to apply the game in each slot. However, to achieve Nash equilibrium, the users need to negotiate with the cloudlets multiple times in each time slot, which will occupy a lot of communication resources. From Fig. 3(b), we know that when $b = 0.1$, it may need about 300 iterations to reach the Nash equilibrium. Even though a larger step size $b = 4$ can be used, it also needs about 20 iterations to reach the Nash equilibrium, which means the user needs about 20 negotiations with cloudlet. Therefore, the key to further reducing the system execution time is to reduce the number of iterations.

From Fig. 3(a) and (b), we can see that the probability of a user choosing a cloudlet is monotonously rising or decreasing when it exceeds a certain value. This feature means that when the probability that a user selects a cloudlet is greater than a certain value, it will largely offload tasks to the cloudlet. Thus, based on this feature, we design a decentralized learning algorithm with termination condition to further reduce the number of iterations, which is shown in Algorithm 2. In Algorithm 2,

Algorithm 2 Decentralized Algorithm for Multiuser Tasks Offloading With Termination Condition

Require: z_i, g_i, c_j, U, the initial mixed strategy of each user $\{\mathbf{p}_1, \ldots, \mathbf{p}_N\}$, termination condition f.

Ensure: Action of each user $\{a_1, \ldots, a_N\}$.

While The mixed strategy $\mathbf{p}_i$ of each active user is not a pure strategy $\mathbf{e}_i$

1: Each active user i selects its offloading action a_i according to its current strategy vector $\mathbf{p}_i$. Then they communicate to the corresponding cloudlets about their decision.

2: The cloudlets calculate the total execution time of the tasks according to equation (3), and then transfer the results to the active users.

3: The active users evaluate their utility $\tilde{u}_i$ according to equation (6).

4: All active users update their strategy vectors according to the following rules:

if $\max(p_{ij}) \geq f, p_{ij} \in \mathbf{p}_i, j \in \mathcal{M}$ **then**

$$\begin{cases} p_{ij} = 1 \\ p_{ik} = 0, k = 1, \ldots, j-1, j+1, \ldots, M \end{cases}$$

else

$$\mathbf{p}_i' = \mathbf{p}_i + b\tilde{\mathbf{u}}_i(\mathbf{e}_{ai} - \mathbf{p}_i)$$

end if

where $0 < b < 1$ is a step size, $\mathbf{p}'$ is the new strategy vector of user i and $\mathbf{e}_{ai}$ is a unit vector of appropriate dimension with M component unity. When user i offload task to cloudlet j, The corresponding position of $\mathbf{e}_{ai}$ is 1, and other positions are 0. Meanwhile, the inactive mobile users keep their strategy selection probabilities unchanged.

End while

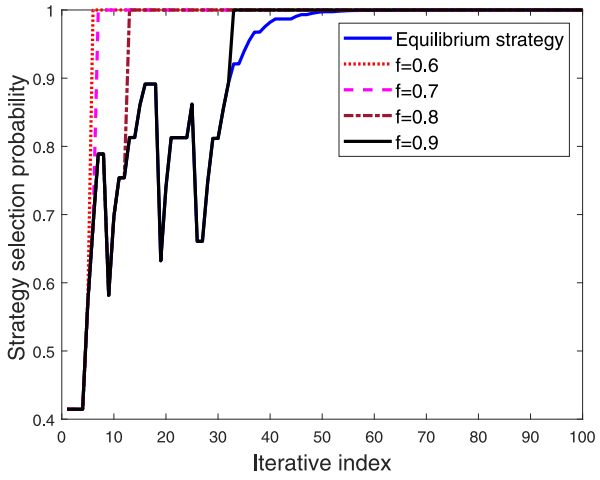
In this time slot, the active users select actions with a corresponding position of 1 in their strategy vectors.

TABLE I
Impact of Different Termination Conditions

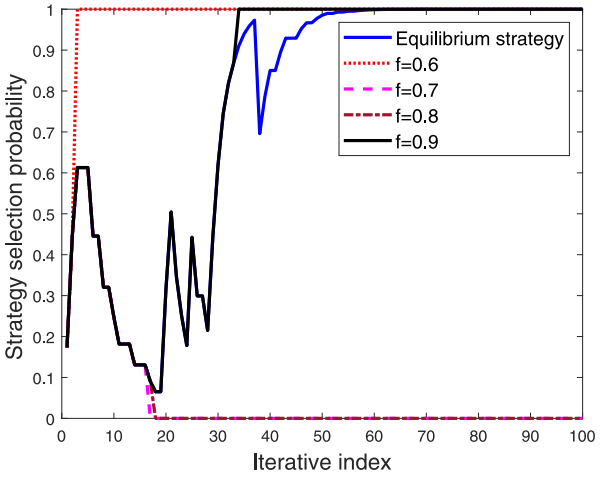
termination condition	f=0.6	f=0.7	f=0.8	f=0.9
average deviation ratio	0.3008	0.2215	0.1667	0.0915
average number of iterations	4.1565	7.1314	10.9593	17.3333

the termination condition f is introduced. When the maximum element of the mixed strategy vector is greater than f, we force this element to 1 and other elements to 0, i.e., when the probability of a user selecting a cloudlet is greater than f, we force the user to offload tasks to this cloudlet instead of others. After employing f, the convergence speed is greatly improved, but it also brings another problem, i.e., the user's action deviates from the equilibrium action.

Therefore, in order to verify this conclusion, we conduct a number of experiments. The results are shown in Table I. In the tests, the number of users is 80, the step size $b = 0.4$, and the number of experiments is 100. The average deviation ratio indicates that the probability of offloading to other cloudlets compare to the total numbers. We can see that when the termination condition becomes smaller, the average deviation ratio increases. If there are 100 users offloading tasks, when $f = 0.9$, there will be about nine users offloading tasks to other cloudlets. When $f = 0.6$, this number increases to about 30. But the number of iterations decreases as the termination condition f decreases. When $f = 0.6$, the algorithm terminates only about four iterations, and when $f = 0.9$, the algorithm takes about 17 times to terminate. The average number of iterations also represents the average number of negotiations between the user and the cloudlet. Therefore, when the termination condition f is smaller, the communication cost of the system will become smaller.



(a)



(b)

Fig. 7. Convergence of Algorithm 2. (a) User's action is consistent with the equilibrium action. (b) User's action is inconsistent with the equilibrium action.

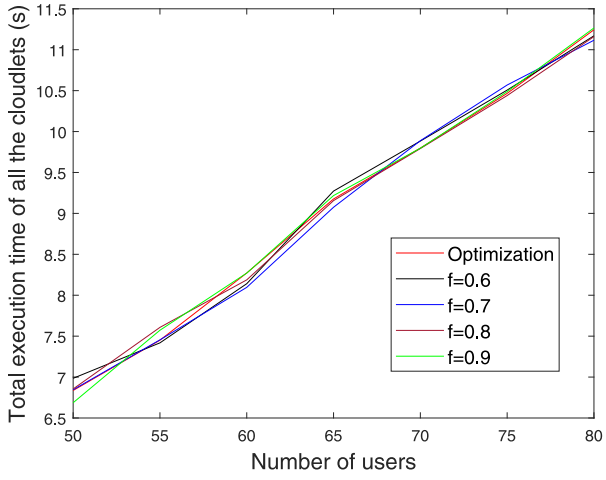


Fig. 8. Comparison of the system' total execution time between Algorithm 2 and optimization methods.

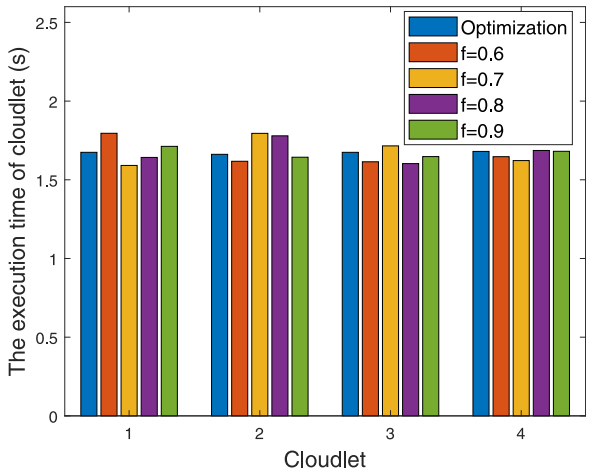


Fig. 9. Execution time of each cloudlet.

Fig. 7 demonstrates the convergence of Algorithm 2. In Fig. 7(a), a case is shown that a user offloads task to the same cloudlet as the equilibrium action, regardless of the termination conditions. Another case is shown in Fig. 7(b), which demonstrates that at different termination condition, a user may offload the task to other cloudlets which deviate from equilibrium action, e.g., when $f = 0.6$ and $f = 0.9$ the action of the user is the same as the equilibrium action, while when $f = 0.7$ and $f = 0.8$ it will offload the task to other cloudlets. Since each user's action is random, it is difficult to determine when the user's action will deviate from the equilibrium action. Consequently, we show the statistics of the deviation rate in Table I.

By using Algorithm 2, users have to negotiate with cloudlets multiple times in a task offloading. This approach will increase communication cost, however, it can also improve the system performance. In Fig. 8, we compare Algorithm 2 with the optimization method. In the simulation, the number of active users is in [50, 80]. When the number of active users increases, the total execution time of the cloudlet system is almost a straight line by using the optimization method, while applying Algorithm 2, the total execution time is a polygonal line. When the termination condition is small, the deviation between the polyline and the line is slightly larger. We also can see that when adopting different termination conditions f, the total execution times for the cloudlet system are approximate to the optimization method. Compared with Fig. 6, we can conclude that by using Algorithm 2 the system can achieve a better performance than Algorithm 1.

Fig. 9 demonstrates the execution time of each cloudlet under Algorithm 2 and the optimization method, where 50 active users are used. While in the proposed algorithm different termination conditions are tested and in the optimization method a coordinator is used. We can see that when the optimization approach is used, the execution time of each cloudlet is roughly equivalent, while the execution time of each cloudlet is a little fluctuating by using Algorithm 2. But it can also achieve an excellent load balance between cloudlets. When termination conditions $f = 0.6$ is used, fluctuations in execution time between cloudlets is larger than termination condition $f = 0.9$. However, it can significantly

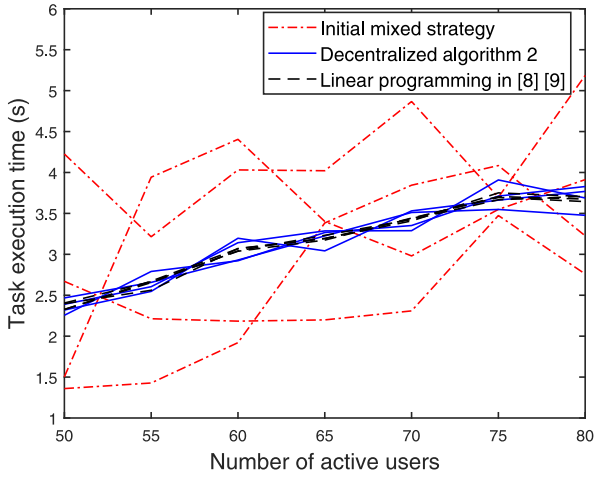


Fig. 10. Comparison of different load-balancing approaches.

reduce the number of communications. Therefore, when using Algorithm 2, considering that fluctuations in execution time are within an acceptable range, a smaller termination condition should be used to reduce communication costs.

Fig. 10 shows the load balancing of the cloudlets under different approaches, the number of active users is in [50, 80]. We can see that with the number of active users increases, the execution time of each cloudlet increases. The execution time of cloudlets is equal to each other approximately by using a linear programming algorithm in [8] and [7] which can guarantee the load balance of each cloudlet. With Algorithm 2, the execution time of each cloudlet will fluctuate a little compare with the linear programming algorithm. However, it can also achieve load balancing approximately. The significant advantage of the game approach is that it does not require a control node to coordinate the actions of all parties, while the optimization algorithm needs.

VIII. Conclusion

In this article, we adopt the stochastic congestion game approach to resolve the issue of load balancing and service quality improvement in a multicloudlet edge computing system without a coordinator. We first consider the load balancing of the cloudlets as an incomplete information multiplayer game in which users choose offloading strategies to achieve their maximum utility without knowing if other users exist. Then the pure strategy Nash equilibrium is proved to be existed by using potential game theory. Second, we propose a decentralized learning algorithm to solve this game. Then the convergence of the algorithm is proved by using an ordinary differential equation approach. After that, we provide a static application of the algorithm and test its performance, which can reduce the execution time of the cloudlet system with extremely low communication cost. Finally, to adapt to dynamic scenarios and further improve the performance, we propose a decentralized learning algorithm with termination condition, which can further reduce the execution time of the system and achieve the load balancing approximately. In future work, we will comprehensively consider wireless transmission, users' location, and cloudlet computing capability to further improve the efficiency of the edge computing system with multiple cloudlets.

References

[1] C.-K. Tham and B. Cao, "Stochastic programming methods for workload assignment in an ad hoc mobile cloud," *IEEE Trans. Mobile Comput.*, vol. 17, no. 7, pp. 1709–1722, Jul. 2018.

[2] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.

[3] P. Wang, C. Yao, Z. Zheng, G. Sun, and L. Song, "Joint task assignment, transmission, and computing resource allocation in multilayer mobile edge computing systems," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2872–2884, Apr. 2019.

[4] K. Xie *et al.*, "Distributed multi-dimensional pricing for efficient application offloading in mobile cloud computing," *IEEE Trans. Services Comput.*, vol. 12, no. 6, pp. 925–940, Nov./Dec. 2019.

[5] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2017.

[6] T. Q. Dinh, Q. D. La, T. Q. S. Quek, and H. Shin, "Learning for computation offloading in mobile edge computing," *IEEE Trans. Commun.*, vol. 66, no. 12, pp. 6353–6367, Dec. 2018.

[7] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint load balancing and offloading in vehicular edge computing and networks," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4377–4387, Jun. 2019.

[8] S. Sthapit, J. Thompson, N. M. Robertson, and J. R. Hopgood, "Computational load balancing on the edge in absence of cloud and fog," *IEEE Trans. Mobile Comput.*, vol. 18, no. 7, pp. 1499–1512, Jul. 2019.

[9] Y. Wu, Y. He, L. P. Qian, J. Huang, and X. Shen, "Optimal resource allocations for mobile data offloading via dual-connectivity," *IEEE Trans. Mobile Comput.*, vol. 17, no. 10, pp. 2349–2365, Oct. 2018.

[10] Z. Hong, H. Huang, S. Guo, W. Chen, and Z. Zheng, "QoS-aware cooperative computation offloading for robot swarms in cloud robotics," *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 4027–4041, Apr. 2019.

[11] A. Jin, W. Song, and W. Zhuang, "Auction-based resource allocation for sharing cloudlets in mobile cloud computing," *IEEE Trans. Emerg. Topics Comput.*, vol. 6, no. 1, pp. 45–57, Jan. 2018.

[12] Y. Dong, G. Xu, Y. Ding, X. Meng, and J. Zhao, "A 'joint-me' task deployment strategy for load balancing in edge computing," *IEEE Access*, vol. 7, pp. 99658–99669, 2019.

[13] F. P. Lin and Z. Tsai, "Hierarchical edge-cloud SDN controller system with optimal adaptive resource allocation for load-balancing," *IEEE Syst. J.*, vol. 14, no. 1, pp. 265–276, Mar. 2020.

[14] S. Sotiriadis, N. Bessis, C. Amza, and R. Buyya, "Elastic load balancing for dynamic virtual machine reconfiguration based on vertical and horizontal scaling," *IEEE Trans. Services Comput.*, vol. 12, no. 2, pp. 319–334, Mar./Apr. 2019.

[15] A. Hussain, M. Aleem, M. A. Islam, and M. A. Iqbal, "A rigorous evaluation of state-of-the-art scheduling algorithms for cloud computing," *IEEE Access*, vol. 6, pp. 75033–75047, 2018.

[16] L. Liu, S. Chan, G. Han, M. Guizani, and M. Bandai, "Performance modeling of representative load sharing schemes for clustered servers in multiaccess edge computing," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4880–4888, Jun. 2019.

[17] O. Delgado and F. Labeau, "Delay-aware load balancing over multipath wireless networks," *IEEE Trans. Veh. Technol.*, vol. 66, no. 8, pp. 7485–7494, Aug. 2017.

[18] M. Liu and Y. Liu, "Price-based distributed offloading for mobile-edge computing with computation capacity constraints," *IEEE Wireless Commun. Lett.*, vol. 7, no. 3, pp. 420–423, Jun. 2018.

[19] K. Wang, F. C. M. Lau, L. Chen, and R. Schober, "Pricing mobile data offloading: A distributed market framework," *IEEE Trans. Wireless Commun.*, vol. 15, no. 2, pp. 913–927, Feb. 2016.

[20] J. Zhang, H. Guo, J. Liu, and Y. Zhang, "Task offloading in vehicular edge computing networks: A load-balancing solution," *IEEE Trans. Veh. Technol.*, vol. 69, no. 2, pp. 2092–2104, Feb. 2020.

[21] X. Lyu, H. Tian, C. Sengul, and P. Zhang, "Multiuser joint task offloading and resource optimization in proximate clouds," *IEEE Trans. Veh. Technol.*, vol. 66, no. 4, pp. 3435–3447, Apr. 2017.

[22] F. Zhang, R. Deng, and H. Liang, "An optimal real-time distributed algorithm for utility maximization of mobile ad hoc cloud," *IEEE Commun. Lett.*, vol. 22, no. 4, pp. 824–827, Apr. 2018.

[23] J. Du, F. R. Yu, X. Chu, J. Feng, and G. Lu, "Computation offloading and resource allocation in vehicular networks based on dual-side cost minimization," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1079–1092, Feb. 2019.
[24] J. Ren, G. Yu, Y. Cai, and Y. He, "Latency optimization for resource allocation in mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 17, no. 8, pp. 5506–5519, Aug. 2018.
[25] J. Kwak, Y. Kim, J. Lee, and S. Chong, "Dream: Dynamic resource and task allocation for energy minimization in mobile cloud systems," *IEEE J. Sel. Areas Commun.*, vol. 33, no. 12, pp. 2510–2523, Dec. 2015.
[26] S. Chen, Y. Wang, and M. Pedram, "A semi-Markovian decision process based control method for offloading tasks from mobile devices to the cloud," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2013, pp. 2885–2890.
[27] L. Yang, H. Zhang, X. Li, H. Ji, and V. C. M. Leung, "A distributed computation offloading strategy in small-cell networks integrated with mobile edge computing," *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, pp. 2762–2773, Dec. 2018.
[28] D. Monderer and L. S. Shapley, "Potential games," *Games Econ. Behav.*, vol. 14, no. 1, pp. 124–143, 1996.
[29] P. Sastry, V. Phansalkar, and M. Thathachar, "Decentralized learning of Nash equilibria in multi-person stochastic games with incomplete information," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 24, no. 5, pp. 769–777, May 1994.
[30] J. Wang, *The Theory of Games*. Oxford, U.K.: Clarendon, 1988.
[31] J. Wang, C. Jiang, K. Zhang, X. Hou, Y. Ren, and Y. Qian, "Distributed Q-learning aided heterogeneous network association for energy-efficient IIoT," *IEEE Trans. Ind. Informat.*, vol. 16, no. 4, pp. 2756–2764, Apr. 2020.
[32] J. Wang, C. Jiang, H. Zhang, Y. Ren, K. Chen, and L. Hanzo, "Thirty years of machine learning: The road to Pareto-optimal wireless networks," *IEEE Commun. Surveys Tuts.*, early access, Jan. 13, 2020, doi: 10.1109/COMST.2020.2965856.
[33] H. Zhang, C. Jiang, J. Wang, L. Wang, Y. Ren, and L. Hanzo, "Multicast beamforming optimization in cloud-based heterogeneous terrestrial and satellite networks," *IEEE Trans. Veh. Technol.*, vol. 69, no. 2, pp. 1766–1776, Feb. 2020.

Fenghui Zhang is currently pursuing the Ph.D. degree with the School of Information Science and Engineering, Southeast University, Nanjing, China.

He also an Associate Professor with the School of Electronics and Information Engineering, West Anhui University, Lu'an, China. He was a Visiting Scholar with the Institute of Technology Tallaght, Dublin, Ireland, in 2013, and the University of Alberta, Edmonton, AB, Canada, in 2017. His research interests include mobile cloud computing and vehicular network.

Mr. Zhang serves as an Editor for the *International Journal of Wireless Communications and Mobile Computing*. He also serves as a reviewer for several IEEE journals and conferences.

Michael Mao Wang received the master's degree in biomedical engineering and the Ph.D. degree in electrical engineering and computer science from the University of Kentucky, Lexington, KY, USA, in 1992 and 1995, respectively.

He was a Distinguished Member of Technical Staff with the Motorola Advanced Radio Technology Group, Arlington Heights, IL, USA, from 1995 to 2003, and joined Qualcomm Research, San Diego, CA, USA, in 2003. He became a Professor with the School of Information Science and Engineering and the National Mobile Communications Research Laboratory, Southeast University, Nanjing, China, in 2015. He is also an Adjunct Professor with the School of Electronic and Optical Engineering, Nanjing University of Science and Technology, Nanjing. He holds more than 90 U.S., patents and has over 40 IEEE journal publications. His research interests include communication theory and wireless networking.