Joint Computation Offloading and Coin Loaning for Blockchain-empowered Mobile-Edge Computing

Zhen Zhang, Zicong Hong, Wuhui Chen, *Member, IEEE,*
Zibin Zheng, *Senior Member, IEEE,* and Xu Chen, *Senior Member, IEEE*

***Abstract*—The Blockchain-empowered mobile-edge computing (MEC) is a promising solution for enhancing the computation capabilities of mobile equipments (MEs) to process computation-intensive tasks such as the real-time data processing tasks and mining tasks. However, because of the "cold start" and "long return" problems, efficient computation offloading cannot be achieved in blockchain-empowered MEC because the MEs do not always have enough coins to afford the offloading service cost. In this paper, we study the joint computation-offloading and coin-loaning problem for blockchain-empowered MEC to minimize the total cost of all MEs. We introduce the banks that can provide loan services to the MEs to address the above two issues. We formulate the problem as a non-cooperative game to model the competitions between the myopic MEs. By using a potential game method, we prove the existence of a pure-strategy Nash equilibrium and design a distributed algorithm to achieve the Nash equilibrium point with low computational complexity. We also provide an upper bound on the price of anarchy of the game by theoretical proof. Besides, two smart contracts are designed to automatically perform the computing resource trading and coin loaning processes. Lastly, our simulation results show that our proposed algorithm can significantly reduce the total cost of all MEs, has better performance compared with other solutions, and scales well as the number of mobile equipments increases. Moreover, the financial cost for executing the two smart contracts on the Ethereum network is low.**

***Index Terms*—Blockchain, edge computing, computation offloading, coin loaning, potential game, Nash equilibrium.**

I. Introduction

MORE and more emerging smart applications such as augmented reality, natural language processing, and object recognition are gaining enormous attention with the development of various smart mobile equipments (MEs) [1], [2]. However, executing the computation-intensive and resource-hungry applications may result in large delays and significant energy consumption because of the limited onboard computation capabilities and battery life of MEs. To address the contradiction between the resource limitation of MEs and the requirements of the computation-intensive applications, mobile-edge computing (MEC) has been introduced as an enabler to provide limited computing resources with shorter delays close to the network edge to solve different types of tasks (e.g., computation-intensive applications, blockchain mining, and consensus process). In MEC, the requirements of short delays and energy-saving of the MEs are met via offloading the computation tasks to the edge servers deployed by telecom operators. As in previous works [3], [4], [5], solutions on efficiently designing the computation-offloading scheme and resource allocation manner in MEC to reduce the time cost and energy cost have been carefully studied. However, because data exchanging and computing-resource trading inevitably occur between MEs and edge servers (they have conflicting interests with each other), the trust problem between MEs and edge servers remains to be solved.

Fortunately, blockchain technology provides an opportunity to solve this problem. Blockchain technology, which allows MEs to maintain information transparency and build up trust with edge servers via blockchain's decentralized, tamper-proof, secure, and traceable characteristics, can provide feasible solutions for trust enhancement for MEC. Therefore, the blockchain-empowered MEC is introduced to address the trust problem still existing in traditional MEC [6], [7], [8]. In the blockchain-empowered MEC, the MEs can offload their computation tasks to the edge servers after affording the service cost for the computing services of edge servers, and all the transactional data are recorded in the blockchain to build up trust. For example, Jiao et al. [9] designed secure auction mechanisms in edge-computing resource allocation for blockchain-empowered MEC. However, efficient computing-resource trading cannot be achieved for blockchain-empowered MEC because of the "cold start" and "long return" issues, where "cold start" means that a newly added ME has no coins to afford the service cost for the computing services of edge servers and "long return" means that auditing and verifying the transaction records for new blocks in the consensus process would be costly in terms of time.

To overcome the "cold start" and "long return" challenges for blockchain-empowered MEC, we introduce banks to provide loan services to the MEs. Each bank sets its own *rate of return*, which increases with an increasing number of MEs that borrow from it, to gain revenue [10]. Hence, if a ME borrows from a bank, it needs to repay the principal and the additional fees caused by the rate of return to the bank. We then study

Affiliation of author(s): This work was supported in part by the National Key Research and Development Plan under Grant 2018YFB1003800, in part by the National Natural Science Foundation of China under Grant 11801595, in part by the Natural Science Foundation of Guangdong under Grant 2018A030310076, and in part by the Program for Guangdong Introducing Innovative and Entrepreneurial Teams under Grant 2017ZT07X355.(*Corresponding author: Wuhui Chen.*)

Z. Zhang, Z. Hong, W. Chen, Z. Zheng, and X. Chen are with School of Data and Computer Science, Sun Yat-sen University, China (e-mail: zhangzh297@mail2.sysu.edu.cn; hongzc@mail2.sysu.edu.cn; chenwuh@mail.sysu.edu.cn; zhzibin@mail.sysu.edu.cn; chenxu35@mail.sysu.edu.cn).

the joint computation-offloading and coin-loaning problem for blockchain-empowered MEC to minimize the total cost of all MEs, in which the MEs can choose either to offload their computation tasks to an edge server and borrow from a bank or to compute their computation tasks locally. In our joint computation-offloading and coin-loaning problem, there are three questions that need to be answered. 1) The first question is how should a ME choose between local computing and edge computing (via computation offloading)? 2) The second question is that if a ME chooses to offload its computation task to an edge server with limited computing capabilities, then which edge server should be chosen to efficiently process its computation task? 3) The third question is that if a ME chooses to borrow from a bank, then which bank should it choose to borrow from to minimize its loan cost?

In this paper, we aim to address the joint computation-offloading and coin-loaning problem to minimize the cost of all MEs based on a game-theory-based method. To answer the presented three questions, we formulate our joint computation-offloading and coin-loaning problem as a non-cooperative game in which the rational MEs make decisions based on their own interests and compete with each other to search for the optimal decisions to minimize their cost. In the game, if a ME chooses local computing, then its computation task is computed with its onboard computing capabilities. If a ME chooses to offload its computation task to an edge server with limited computing capabilities and borrows from a bank, then the ME jointly considers an edge server and a bank to minimize the computation cost. Besides, we also design a distributed algorithm by which the game can quickly converge to a stable state, i.e., a Nash equilibrium (NE) point. To our best knowledge, we are the first to study the joint computation-offloading and coin-loaning problem for blockchain-empowered MEC. In conclusion, the main contributions are summarized as follows:

- The role of a bank is introduced to provide loan services to the MEs to overcome the "cold start" and "long return" problems in blockchain-empowered MEC. In such a scenario, we formulate the joint computation-offloading and coin-loaning problem as a non-cooperative game to minimize the cost of each ME in terms of computing time, energy, and digital coin, in which each ME shows its economic property.
- Based on a potential game method, the existence of an NE point is proved, and an efficient distributed algorithm is designed to quickly achieve the NE point. The game theory can solve the optimization problem by allowing each ME minimizes their own cost in a distributed manner, which is more efficient than existing centralized optimization methods when the number of MEs is large. When considering the partial offloading model, we theoretically prove that the case of partial offloading can be reduced into the case of binary offloading.
- To support our model in blockchain environment, two smart contracts are designed to automatically perform the computing resource trading and coin loaning processes. By deploying the contracts on the Ethereum network, we attain and analyze the financial cost for executing each function in the contracts to show their performance.
- Lastly, the extensive simulation results demonstrate that the proposed algorithm can significantly reduce the cost of all MEs and possesses much better performance than other existing solutions under different parameter settings. Besides, the results of the real experiment in Ethereum environment show that the economic cost for using the smart contracts on the Ethereum network is low.

The remainder of this paper is organized as follows. We discuss related work in Section II. We describe the system model in Section III. We formulate the joint computation-offloading and coin-loaning problem as a non-cooperative game in Section IV. We extend our model to the case of partial offloading in Section V. We design a distributed algorithm to achieve the NE point of the game in Section VI. The results of simulations are presented in Section VIII. Lastly, the conclusion is summarized in Section IX.

II. Related Work

The computation-offloading problem for MEC has been studied extensively [11]–[18]. For example, Chen et al. [11] proposed a dynamic computation-offloading algorithm to solve the problem in a distributed manner for MEC. Sun et al. [12] proposed an efficient task-scheduling scheme in the vehicular cloud by jointly considering the instability of resources, the heterogeneity of vehicular computing capabilities, and the interdependency of computing tasks. Mansouri et al. [13] used a computation-offloading game to model the competition between mobile users who aim to maximize their own quality of experience in a hierarchical fog-cloud computing paradigm. Zheng et al. [14] investigated the problem of multiuser computation offloading for mobile-cloud computing under dynamic environments and proposed a stochastic game-theory approach for dynamic computation offloading. Guo et al. [15] proposed an iterative searching-based task offloading scheme that jointly optimizes task offloading, computational frequency scaling, and transmit-power allocation. Zhang et al. [16] developed a multi-queue model to explore the impact of offloading policies on the performance of MEs with the computing resources supported by an edge-computing server. Guo et al. [17] studied the computation-offloading problem in a mobile-edge-computing framework and proposed a two-tier game-theory greedy offloading scheme as the solution. Shatri et al. [18] proposed a distributed algorithm on computation offloading in multi-hop networks that aims to minimize the total network energy consumed. However, the trust problem remains to be solved.

Recently, a few works such as [19]–[21] have investigated the computation-offloading problem for blockchain-empowered MEC to solve the computation-intensive proof-of-work puzzle and enhance trust between MEs and edge servers. Chatzopoulos et al. [19] proposed a multilayer computation-offloading framework, FlopCoin, to integrate a distributed incentive scheme and a distributed reputation mechanism for mobile devices. Liu et al. [20] proposed a novel mobile-edge-computing-enabled wireless blockchain framework where computation-intensive mining tasks can be offloaded to nearby edge-computing nodes and where cryptographic hashes of blocks can be cached in the edge servers. Xiong et al. [21] studied the edge-computing resource management and pricing problem for a proof-of-work-based blockchain network using a game-theory-based approach. In addition, the coin-loaning problem for blockchain-empowered MEC has been studied. For example, Li et al. [22] proposed a double-auction-based computing-resource trading approach to maximize the economic benefits while protecting privacy for edge-cloud-assisted IoT. Jiao et al. [23] proposed an auction-based edge-computing resource allocation mechanism for the edge-computing service provider to support mobile blockchain-mining tasks. Li et al. [24] addressed the optimal loan allocation based on the double-auction algorithm for fast computing-resource trading in blockchain-enabled edge-assisted internet of thing. However, the computation-offloading and coin-loaning problems for blockchain-empowered MEC have been studied separately. Most studies have not considered the joint computation-offloading and coin-loaning problem, which is of great significance to blockchain-empowered MEC. In this paper, we study the joint computation-offloading and coin-loaning problem for blockchain-empowered MEC to minimize the total cost of MEs.

III. System Model and Problem Statement

As shown in Fig. 1, we consider a blockchain-empowered MEC framework consisting of MEs, edge servers, banks, and blockchain. We assume that there are N MEs in the network, denoted by $\mathcal{N} : (1, 2, \ldots, N)$. Each ME has a computation task to complete, and then attains some coins in return. The task is denoted by $A_n : (z_n, \gamma_n)$, where z_n is the input data size of this task and γ_n is the number of CPU cycles required to complete the task, and this number depends on the computing density of the task and is known to the user before the task is executed. We assume that constant ρ is the price per unit for the service demands of MEs and is charged by the edge servers. By affording the service cost for the computing services of edge servers, the MEs can offload their computation tasks to the edge servers. If the MEs do not have sufficient coins to afford the service cost, then they can borrow from the banks. All transactions occurring in the network are recorded into blockchain blocks so the transactional records cannot be tampered with, by which security and reliability are guaranteed. Similar to previous works [25], [26], [27], we assume that the MEs change slowly when offloading tasks.

To present this research more clearly, we list the important notations used in this manuscript in Table I.

A. Loan model

We suppose that there are M banks in the framework, denoted by $\mathcal{M} : (1, 2, \ldots, M)$. For the MEs that decide to offload but do not have sufficient coins, they need to borrow from the banks to afford the service cost for the computing services of edge servers. We define the binary variable $b_{n,m} \in \{0, 1\}$ for all ME $n \in \mathcal{N}$, where $b_{n,m} = 1$ denotes that ME n borrows from bank m, and $b_{n,m} = 0$ denotes that ME n does not borrow from bank m. We further define a *loaning vector* $\mathbf{b}_n = (b_{n,1}, b_{n,2}, \ldots, b_{n,M})$ for all ME $n \in \mathcal{N}$. We also assume that each ME can borrow from

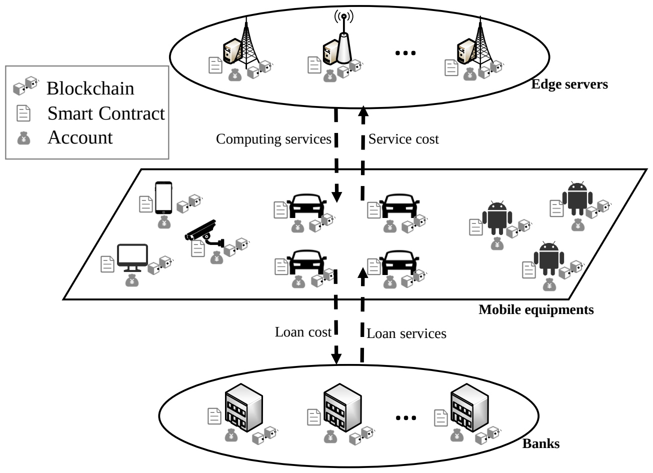

Fig. 1: An illustration of the system model.

at most one bank, so we have $\sum_{m \in \mathcal{M}} b_{n,m} \leq 1$ for all ME $n \in \mathcal{N}$. The banks gain revenue by setting their own *rates of return* that obey the linear models $R_m(x)$ for all bank $m \in \mathcal{M}$. Taking bank m as an example, its rate of return is set as:

$$R_m(x) = \mu_{1,m} + \mu_{2,m} x, \tag{1}$$

where x denotes the number of MEs that borrow from bank m and $\mu_{1,m}, \mu_{2,m} > 0$ are coefficients set by bank m.

B. Communication model

We assume that there are S edge servers in the framework, denoted by $\mathcal{S} : (1, 2, \ldots, S)$. Each ME connects with some edge servers through a cellular network, and we assume that each computation task should either be entirely computed by a ME or entirely offloaded to an edge server. We define the binary variable $a_{n,s} \in \{0, 1\}$ for all ME $n \in \mathcal{N}$ to denote their offloading decisions, where $a_{n,s} = 1$ denotes that the computation task of ME n is entirely offloaded to an edge server $s \in \mathcal{S}$ and $a_{n,s} = 0$ denotes that the computation task of ME n is entirely computed using onboard computing capabilities. We also define an *offloading vector* $\mathbf{a}_n = (a_{n,1}, a_{n,2}, \ldots a_{n,S})$ for all ME $n \in \mathcal{N}$. Because the computation task can be offloaded to at most one edge server, we must have $\sum_{s \in \mathcal{S}} a_{n,s} \leq 1$ for all ME $n \in \mathcal{N}$. We further define binary variable $l_{n,s} \in \{0, 1\}$ for all ME $n \in \mathcal{N}$, where $l_{n,s} = 1$ denotes that ME n can connect with edge server s and $l_{n,s} = 0$ denotes that ME n is too far away from edge server s to connect with it. Thus, based on the connectivity model, we have $a_{n,s} \leq l_{n,s}$ for all ME $n \in \mathcal{N}$ to ensure the offloading behavior can be completed.

We know that ME n needs to transmit z_n amount of data belonging to its task to edge server s if it decides to offload its task to edge server s. We use $\omega_{n,s}$ to denote the uplink rate of the link between ME n and edge server s. As in [26], [28], we consider that at most one wireless communication interface is used between each ME $n \in \mathcal{N}$ and edge server $s \in \mathcal{S}$, and the rate $\omega_{n,s}$ depends on the type of wireless interface and the distance between ME n and edge server s. The time that ME n takes to transmit data to edge server s is the following:

$$t_n^{off,s} = \frac{z_n}{\omega_{n,s}}. \tag{2}$$

TABLE I: The notations used in this paper.

Notations	Meanings
$\mathcal{N}$	The set of the MEs in the network.
N	The number of MEs in set $\mathcal{N}$.
A_n	The computation task of ME n.
z_n	The input data size of A_n.
γ_n	The number of CPU cycles required to complete A_n.
$\mathcal{S}$	The set of edge servers in the network.
S	The number of edge servers in set $\mathcal{S}$.
$\mathbf{a}_n$	The offloading vector of ME n.
$a_{n,s}$	The binary variable denoting whether ME n offloads its computation task to edge server s or not.
$l_{n,s}$	The binary variable denoting the connectivity between ME n and edge server s.
$\mathcal{M}$	The set of banks in the network.
M	The number of MEs in set $\mathcal{M}$.
$\mathbf{b}_n$	The loaning vector of ME n.
$b_{n,m}$	The binary variable denoting whether ME n borrows from bank m or not.
$R_m(x)$	The rate of return of bank m.
d_n	The decision of ME n.
$\mathbf{d}$	The decision profile including the decisions of all MEs.
$\mathbf{d}_{-n}$	The decisions of all MEs excluding ME n.
$n_s^S(\mathbf{d})$	The number of MEs that offload tasks to edge server s in $\mathbf{d}$.
$n_m^M(\mathbf{d})$	The number of MEs that borrow from bank m in $\mathbf{d}$.
$\omega_{n,s}$	The uplink rate of the link between ME n and edge server s.
$t_n^{off,s}$	The time that ME n takes to transmit data to edge server s.
p_n	The transmission power of ME n.
$e_n^{off,s}$	The energy costs that ME n incurs to transmit data to edge server s.
f_n^0	The computing capabilities of ME n.
t_n^0	Time cost for locally completing the task A_n of ME n.
υ_n	The energy consumption per CPU cycle of ME n.
e_n^0	Energy cost for locally completing the task A_n of ME n.
f_s^S	The computing capabilities of edge server s.
$t_n^{com,s}(\mathbf{d})$	The time that ME n takes to process task A_n of ME n with the computing services of edge server s.
θ_n	The service cost that is charged by an edge server.
$\pi_n^m(\mathbf{d})$	Loan cost when ME n borrows from bank m.
c_n^0	The total cost of ME n in the case of local computing.
$c_n^{s,m}(\mathbf{d})$	The total cost of ME n when ME offloads its task to edge server s and borrows from bank m in $\mathbf{d}$.
$c_n(\mathbf{d})$	The total cost of ME n in $\mathbf{d}$.
$c(\mathbf{d})$	The system cost.

We denote by p_n the transmission power of ME n to model its energy cost when transmitting data. The energy that ME n consumes to transmit data to edge server s is the product of transmission power and transmission time and can be calculated as:

$$e_n^{off,s} = \frac{p_n z_n}{\omega_{n,s}}. \tag{3}$$

C. Computation model

We denote by $d_n = (\mathbf{a}_n, \mathbf{b}_n)$ the *decision* of ME n, where $\mathbf{a}_n$ denotes the offloading vector of ME n and $\mathbf{b}_n$ denotes the loaning vector of ME n. If $\mathbf{a}_n = \mathbf{0}$, where $\mathbf{0}$ is a zero vector of S dimensions, ME n completes its computation task locally. Otherwise, ME n offloads its computation task to an edge server. We further refer to the set $\mathbf{d} = \{d_n\}_{n\in\mathcal{N}}$ as a decision profile including the decisions of all MEs. In the decision profile $\mathbf{d}$, we denote by $n_s^S(\mathbf{d}) = \sum_{n\in\mathcal{N}} a_{n,s}$ the number of MEs that offload tasks to edge server s. Similarly, we denote by $n_m^M(\mathbf{d}) = \sum_{n\in\mathcal{N}} b_{n,m}$ the number of MEs that borrow from bank m in the decision profile $\mathbf{d}$.

In the following, we introduce the time cost, energy cost, service cost, and loan cost in the case of local computing and in the case of computation offloading.

1) Local computing: In this case, the computation task is entirely computed with the onboard computing capabilities of MEs. We consider the computing capabilities of ME n as f_n^0. Hence, the computing capabilities f_n^0 together with the number of CPU cycles of task A_n determines the time consumed for locally completing the task A_n of ME n:

$$t_n^0 = \frac{\gamma_n}{f_n^0}. \tag{4}$$

The energy consumption is related to the number of CPU cycles required to complete the task and can be known based on the conclusion in [26], [29]. We assume the energy consumed per CPU cycle for ME n is υ_n. Thus, the number of CPU cycles of task A_n together with υ_n determines the energy consumed for locally computing the task A_n of ME n:

$$e_n^0 = \gamma_n \upsilon_n. \tag{5}$$

2) Computation offloading: In this case, the computation tasks are entirely offloaded to the edge servers and processed with the computing services of the edge servers. We denote by f_s^S the computing capabilities of edge server s. As in [26], [30], we consider $f_s^S / n_s^S(\mathbf{d})$ as the computing capabilities assigned to each ME that offloads a task to edge server s. Thus, the time required to process task A_n of ME n with the computing services of edge server s is calculated as:

$$t_n^{com,s}(\mathbf{d}) = \frac{\gamma_n}{f_s^S} \sum_{n'\in\mathcal{N}} a_{n',s}. \tag{6}$$

The edge servers with limited computing capabilities are introduced to support the computation-intensive tasks offloaded to them. The service demand γ_n together with the service price ρ determines ME n's service cost $\gamma_n\rho$ that is charged by an edge server. Thus, if ME n offloads its computation task to an edge server, then the service cost that is charged by the edge server is calculated as:

$$\theta_n = \gamma_n \rho. \tag{7}$$

In our model, we assume the following condition hold:

Assumption 1. *All the coins that the MEs receive are spent (e.g., paying the service cost and loan cost), meaning that the assets of each ME are 0.*

Based on Assumption 1, when ME n decides to offload, it needs to borrow an amount θ_n from a bank, which incurs an additional cost because of the rate of return of the bank. When ME n borrows from bank m in the case of computation offloading, its loan cost can be calculated as:

$$\pi_n^m(\mathbf{d}) = \theta_n \cdot R_m\Big(n_m^M(\mathbf{d})\Big). \tag{8}$$

D. Cost model

In previous works, [25], [26], [27], the authors modeled the total cost of each ME as a linear combination of computing time and energy because the edge servers do not charge the MEs for computing service. In this research, we aim to minimize the total cost of each ME in terms of the computing time, energy, and digital coin, so each ME shows its economic property. Therefore, the total cost of each ME is modeled as a linear combination of computing time, energy, and digital coin. We consider the weight of time as λ^T, weight of energy as λ^E, and weight of digital coin as 1. In what follows, we discuss the total cost of each ME in the case of local computing and in the case of computation offloading.

For the case of local computing, the time cost and energy cost for locally computing the task A_n of ME n determine the total cost of ME n:

$$c_n^0 = \lambda^T t_n^0 + \lambda^E e_n^0. \tag{9}$$

For the case of computation offloading, if ME n decides to offload its computation task to edge server s and to borrow from bank m, then the corresponding time cost, energy cost, service cost, and loan cost for offloading the tasks A_n of ME n to edge server s and processing the task in edge server s determine the total cost of ME n:

$$c_n^{s,m}(\mathbf{d}) = \theta_n + \pi_n^m(\mathbf{d}) + \lambda^T t_n^{com,s}(\mathbf{d}) + \lambda^T t_n^{off,s} + \lambda^E e_n^{off,s}. \tag{10}$$

Similar to previous works [25], [31], [32], we do not model the cost for returning the results from edge server s to the original ME n into $c_n^{s,m}(\mathbf{d})$ because the size of the results is much smaller than those of the input data z_n for many applications such as face recognition. Besides, when the MEC system is overloaded or broken, we adopt the recovery schemes in [33] to solve the MEC failure problems and improve the quality of experience.

As a consequence, the total cost of ME n in decision profile $\mathbf{d}$ can be calculated as:

$$c_n(\mathbf{d}) = c_n^0 I_{\{\mathbf{a}_n=\mathbf{0}\}} + c_n^{s,m}(\mathbf{d}) I_{\{\mathbf{a}_n\neq\mathbf{0}\}}, \tag{11}$$

where $\mathbf{0}$ is a zero vector of S dimensions and $I_{\{C\}}$ is an indicator function with $I_{\{C\}} = 1$ if the condition C is true and $I_{\{C\}} = 0$ if the condition C is false.

Then, the system cost can thus be calculated as $c(\mathbf{d}) = \sum_{n\in\mathcal{N}} c_n(\mathbf{d})$.

E. Problem statement

Based on the discussion in the previous section, the problem that we are addressing in this work is giving a decision profile $\mathbf{d}$ that makes the total cost of all MEs in the framework reach the minimum value. We then describe the optimization problem as follows:

$$\begin{aligned}
\mathbf{P1}: \quad & \min_{\mathbf{d}} \ c(\mathbf{d}), && (12)\\
s.t.: \quad & \sum_{s\in\mathcal{S}} a_{n,s} \le 1, \quad \forall n \in \mathcal{N} && (12a)\\
& \sum_{m\in\mathcal{M}} b_{n,m} \le 1, \quad \forall n \in \mathcal{N} && (12b)\\
& a_{n,s} \in \{0,1\}, \quad \forall n \in \mathcal{N}, s \in \mathcal{S} && (12c)\\
& a_{n,s} \le l_{n,s}, \quad \forall n \in \mathcal{N}, s \in \mathcal{S} && (12d)\\
& b_{n,m} \in \{0,1\}, \quad \forall n \in \mathcal{N}, m \in \mathcal{M}. && (12e)
\end{aligned}$$

Here, we observe that the optimization problem **P1** is actually a 0–1 integer programming problem, so it is a NP-hard problem. Besides, the decisions of the MEs affects each other in the problem, which can be found in formulas (6) and (8). The approaches based on centralized optimization had been introduced in [34], [35]. However, the MEs owned by different parties may not want to obey the centralized optimization solution because they pursue their own interests, making the approaches based on centralized optimization not always effective in practice. Therefore, we solve the optimization problem **P1** using game theory, by which the MEs can make decisions in a distributed manner and thus ease the heavy burden of computation and management at the center.

IV. Existence of Nash Equilibrium

We formally define the game of optimization problem **P1** as $\mathcal{G} \triangleq \left(\mathcal{N}, \{\mathcal{D}_n\}_{n\in\mathcal{N}}, \{c_n\}_{n\in\mathcal{N}}\right)$, where the set $\mathcal{N}$ of the MEs is the set of players; $\mathcal{D}_n = \mathbf{A}_n \bigotimes \mathbf{B}_n$ is the decision space of ME n; $\mathbf{A}_n$ and $\mathbf{B}_n$ are the decision spaces of offloading vector $\mathbf{a}_n$ and loaning vector $\mathbf{b}_n$ of ME n, respectively; and c_n is the cost function of ME n. Each ME aims to minimize its own cost in response to the decisions of other MEs and competes to search for an optimal decision d_n^* that satisfies the following conditions:

$$\begin{aligned}
& d_n^* \in \arg \min_{d_n\in\mathcal{D}_\mathbf{n}} \ c_n(d_n, \mathbf{d}_{-n}) && (13)\\
s.t.: \quad & \sum_{s\in\mathcal{S}} a_{n,s} \le 1, \quad \forall n \in \mathcal{N} && (13a)\\
& \sum_{m\in\mathcal{M}} b_{n,m} \le 1, \quad \forall n \in \mathcal{N} && (13b)\\
& a_{n,s} \in \{0,1\}, \quad \forall n \in \mathcal{N}, s \in \mathcal{S} && (13c)\\
& a_{n,s} \le l_{n,s}, \quad \forall n \in \mathcal{N}, s \in \mathcal{S} && (13d)\\
& b_{n,m} \in \{0,1\}, \quad \forall n \in \mathcal{N}, m \in \mathcal{M}. && (13e)
\end{aligned}$$

where $\mathbf{d}_{-n}$ denotes the decisions of all MEs except ME n. We provide the definition of the *best response decisions* as follows:

Definition 1. *Given $\mathbf{d}_{-n}$ that denotes the decisions of all MEs excluding ME n, the best response decisions of ME n are the decisions that satisfy the conditions in (13).*

In the best response decisions, each ME calculates current optimal decisions according to the information received from banks and edge servers, while other MEs do not change their decisions, i.e., for given $\mathbf{d}_{-n}$, ME n calculates its best response decisions based on the conditions in (13). When the game $\mathcal{G}$ reaches the NE point, the dynamic of the best response decisions converges to the NE point, in which each ME cannot find a better decision than the current one when decisions for the other MEs remain unchanged. This property is important for addressing the distributed problems using the game-theory-based method because each ME makes a decision based on its own interest.

Definition 2. *When ME n decides to offload its computation task to an edge server in the decision profile $\mathbf{d}$, a beneficial computation-offloading ME occurs if $c_n^{s,m}(\mathbf{d}) < c_n^0$ is satisfied.*

Based on the definition of the beneficial computation-offloading ME, we know that ME n improves its performance by offloading its computation task to an edge server if it is a beneficial computation-offloading ME. Thus, as the number of beneficial computation-offloading MEs in our model increases, the performance of our model improves. In our model, we need to ensure that a ME that offloads its task to an edge server must be a beneficial computation-offloading ME. Otherwise, this ME changes to local computing to reduce its computation cost.

Based on the definition of the game $\mathcal{G}$, the following is the definition of the NE point.

Definition 3. *If a decision profile $\mathbf{d}^*$ is an NE point of the game $\mathcal{G}$, then no ME can further reduce its computation cost by changing its decision unilaterally:*

$$c_n(d_n^*, \mathbf{d}_{-n}^*) \leq c_n(d_n, \mathbf{d}_{-n}^*), \quad \forall d_n \in \mathcal{D}_n, n \in \mathcal{N}. \tag{14}$$

According to the definition of the NE point, we consider decision d'_n a better decision compared with decision d_n if $c_n(d'_n, \mathbf{d}_{-n}) > c_n(d_n, \mathbf{d}_{-n})$ is satisfied. At the NE point, each ME cannot find a better decision than the current one when other MEs do not change their decisions. Therefore, the dynamic of the best response decisions converges to the NE point, and no ME can further reduce its computation cost by changing its decision unilaterally and has an incentive to deviate the system unilaterally.

Remark 1: In the game $\mathcal{G}$, if a ME n decides to offload its computation task to an edge server (i.e., $\mathbf{a}_n \neq \mathbf{0}$) at the NE point, then ME n must be a beneficial computation-offloading ME. This result is because if ME n is not a beneficial computation-offloading ME, then $c_n^{s,m}(\mathbf{d}) > c_n^0$. Hence, ME n would have an incentive to switch to local computing to reduce its computation cost, contradicting the fact that the game $\mathcal{G}$ has achieved the NE point.

Because the decision spaces of the players are discrete, we cannot prove the existence of the NE point in the game $\mathcal{G}$ by differential means. Here, we use the concept of a weighted potential game to prove the existence of the NE point. First, we introduce the concept of a weighted potential game as follows:

Definition 4. *A weight vector $\mathbf{W} = (w_n)_{n\in\mathcal{N}}$, where $w_n > 0$ for all ME $n \in \mathcal{N}$, is considered. The game is a weighted potential game if there exists a w-potential function $\Phi(\mathbf{d})$ such that the following equation holds for all $d_n, d'_n \in \mathcal{D}_n$:*

$$\begin{aligned} c_n(d'_n, \mathbf{d}_{-n}) - & c_n(d_n, \mathbf{d}_{-n}) \\ &= w_n\Big(\Phi(d'_n, \mathbf{d}_{-n}) - \Phi(d_n, \mathbf{d}_{-n})\Big). \end{aligned} \tag{15}$$

In the weighted potential game, when any ME changes its current decision from d_n to d'_n, the variation of its cost function $c_n(d'_n, \mathbf{d}_{-n}) - c_n(d_n, \mathbf{d}_{-n})$ can be mapped into the variation of the w-potential function $\Phi(d'_n, \mathbf{d}_{-n}) - \Phi(d_n, \mathbf{d}_{-n})$ with a weight w_n. This is important for the property of the finite-time convergence of an iterated game towards an NE point. The following theorem shows that the game $\mathcal{G}$ in our model is a weighted potential game.

Theorem 1. *The game $\mathcal{G}$ in our model is a weighted potential game with a weight vector $W = (\gamma_n)_{n\in\mathcal{N}}$.*

Proof. See Appendix A. ■

Theorem 2. *The game $\mathcal{G}$ admits a pure-strategy NE point and has the finite improvement property (FIP).*

Proof. See Appendix B. ■

V. Extension to Partial Offloading

As the discussion in Section III, we assume that the computation tasks of the MEs either are entirely computed locally or entirely offloaded to the edge servers. This offloading manner is called *binary offloading*. This section considers another offloading manner: *partial offloading* [2]. In partial offloading, the computation task of each ME can be divided into two arbitrary parts, one of which is computed locally and the other is offloaded to an edge server. Each ME can achieve higher efficiency in processing its computation task in this manner.

This work uses weight α_n to represent the proportion of ME n's task A_n computed locally and weight $1-\alpha_n$ to represent the proportion of ME n's task A_n offloaded to an edge server. Correspondingly, the offloading vector of each ME is modified to $\overline{\mathbf{a}}_n = (\overline{a}_{n,1}, \overline{a}_{n,2}, \ldots \overline{a}_{n,S})$, where $\|\overline{\mathbf{a}}_n\|_0 \leq 1$ means that at most one of the elements in $\overline{\mathbf{a}}_n$ is not equal to 0 and $\overline{a}_{n,s} \in [0,1]$ for all ME $n \in \mathcal{N}$. Thus, the decision of ME n can be redefined as $\overline{d}_n = (\overline{\mathbf{a}}_n, \mathbf{b}_n)$. Then, the cost of ME n in the decision profile $\overline{\mathbf{d}} = (\overline{d}_n)_{n\in\mathcal{N}}$ can be calculated as:

$$\overline{c}_n(\overline{\mathbf{d}}) = \alpha_n c_n^0 + (1-\alpha_n) c_n^{s,m}(\mathbf{d}). \tag{16}$$

In the case of partial offloading, the following theorem is obtained:

Theorem 3. *In the case of partial offloading, the best choice of α_n for each ME is $\alpha_n = 0$ or $\alpha_n = 1$.*

Proof. See Appendix C. ■

In the case of partial offloading, we construct a new game $\overline{\mathcal{G}} \triangleq \Big(\mathcal{N}, \{\overline{\mathcal{D}}_n\}_{n\in\mathcal{N}}, \{\overline{c}_n\}_{n\in\mathcal{N}}\Big)$, where $\overline{\mathcal{D}}_n = \overline{\mathbf{A}}_n \bigotimes \mathbf{B}_n$ is the decision space of ME n; $\overline{\mathbf{A}}_n$ and $\mathbf{B}_n$ are the decision spaces of offloading vector $\overline{\mathbf{a}}_n$ and loaning vector $\mathbf{b}_n$ of ME n, respectively, and $\overline{c}_n$ is the cost function of ME n. We further refer to the decisions of all MEs excluding ME n as $\overline{\mathbf{d}}_{-n}$. Then, we propose the following theorem:

Theorem 4. *The game $\overline{\mathcal{G}}$ can be reduced to game $\mathcal{G}$, i.e., the case of partial offloading can be reduced to the case of binary offloading.*

Proof. See Appendix D. ■

Based on Theorems 3 and 4, we can extend our model to the case of partial offloading. Moreover, the game $\overline{\mathcal{G}}$ in the case of partial offloading can be reduced to game $\mathcal{G}$ in the case of binary offloading.

VI. Distributed Algorithm

A. Algorithm design

Because we have proven that the game $\mathcal{G}$ always possesses a pure-strategy NE point and has the FIP, we can design our distributed algorithm based on this property. The main idea of the algorithm in Algorithm 1 is that we allow one and only one

Algorithm 1: Distributed algorithm

```
1  initialization:
2  each ME sets its decision to be local computing;
3  iteration:
4  iterate for each time slot $t$ and each ME $n$ in parallel:
5  receive the information on $n_s^S(\mathbf{d}(t-1))$, $n_m^M(\mathbf{d}(t-1))$ of
     each edge server and bank, respectively;
6  calculate the best response decision set $\Lambda_n(t)$;
7  if $\Lambda_n(t) \neq \emptyset$ then
8      send update request to the server to complete the update
         opportunity;
9      if receive permit response then
10         select a decision $d_n(t) \in \Lambda_n(t)$ for slot $t$;
11     else
12         keep $d_n(t) = d_n(t-1)$ for slot $t$;
13 else
14     keep $d_n(t) = d_n(t-1)$ for slot $t$;
15 until the NE point is achieved;
```

ME to update its decision at a time. Besides, the slotted time structure is used for smoothly implementing the algorithm into practice. We describe the process of the proposed algorithm as follows.

Each ME first needs to set its decision to be local computing (line 2). In each time slot t, before calculating the best response decision set, each ME n receives the information on $n_s^S(\mathbf{d}(t-1)$ of each edge server and $n_m^M(\mathbf{d}(t-1))$ of each bank, respectively (line 5). Then, each ME calculates the best response decision set $\Lambda_n(t)$ based on (13) (line 6). If $\Lambda_n(t) \neq \emptyset$, ME n sends an update request to the server to complete the update opportunity (lines 7–8). If $\Lambda_n(t) = \emptyset$, the current decision of ME n is optimal, so it maintains $d_n(t) = d_n(t-1)$ (line 14). After receiving enough update requests, the server randomly selects a ME n that has sent the update request and sends a permit response to it to update its decision at time slot t. Thus, ME n chooses a decision $d_n(t) \in \Lambda_n(t)$ as its decision at time slot t (lines 9–10). As for the MEs that have already sent update requests to the server but do not receive the permit response from the server, they keep their previous decisions, i.e., $d_n(t) = d_n(t-1)$ (line 12). When the server does not get any update requests within a specified period of time, the NE point has been achieved (line 15). Then, the server sends a response to all MEs to declare that the game is ended.

After executing our proposed algorithm, the decisions of all MEs are determined. Based on the decision of each ME, each edge server performs the computing resource trading with corresponding MEs by deploying the smart trading contract on the blockchain, and each bank performs coin loaning with corresponding MEs by deploying the smart loan contract on the blockchain.

B. Convergence analysis

Because our algorithm is designed according to the FIP, it enables MEs to achieve a mutually satisfactory state, i.e., the NE point of the game $\mathcal{G}$. Therefore, the proposed algorithm in this paper is convergent.

The computation load of our proposed algorithm described in Algorithm 1 is mainly concentrated on line 6, and the other parts only involve some basic arithmetic operations. At line 6, we need to calculate the best response decision set Λ_n. First, the proposed algorithm iterates the feasible decision space $\mathcal{D}_n$ of ME n to calculate all possible cases of computation cost, so the computational complexity of this operation is $O(S * M)$. Next, we use the quick-sort algorithm to sort the computation cost in all cases and then choose the best response decisions, so the computing complexity of this operation is $O(KlogK)$, where $K = S * M$. Therefore, our proposed algorithm can achieve the NE point of the game $\mathcal{G}$ in polynomial time.

C. Performance analysis

In this section, we analyze the efficiency of NE calculated from our proposed algorithm by learning the important concept of price of anarchy (PoA). PoA is the ratio of the system cost in a worst NE point calculated from our proposed algorithm and the system cost calculated from a centralized optimal solution, and PoA demonstrates the effects of MEs' selfish behaviors on the system cost. According to the descriptions in [25], [28], [30], PoA is calculated as:

$$PoA = \frac{\max_{\mathbf{d}\in\mathbf{D}^{\mathrm{NE}}} c(\mathbf{d})}{\min_{\mathbf{d}\in\mathbf{D}} c(\mathbf{d})}, \tag{17}$$

where $\mathbf{D} = \prod_{n\in\mathcal{N}} \mathcal{D}_n$ is the decision space of all MEs and $\mathbf{D}^{\mathrm{NE}}$ is the set of all possible NEs resulted from our proposed algorithm. For our model, as the value of PoA decreases, the performance of the proposed algorithm increases. Thus, the theorem on the bound of PoA is the following:

Theorem 5. *For game $\mathcal{G}$ in our model, the PoA of the system cost is bounded by the following:*

$$1 \leq PoA \leq \frac{\sum_{n\in\mathcal{N}} c_n^0}{\sum_{n\in\mathcal{N}} \min\{c_n^0, \hat{c}_n^{min}\}}. \tag{18}$$

Proof. See Appendix E. ∎

VII. Implementation Using Blockchain Smart Contracts

In this section, we leverage the smart contract to implement the computing resource trading mechanism and loan mechanism without a trusted third party. The smart contract uses cryptocurrencies as the currency exchange between the provider and buyers [36]. Because of the transparency and traceability of blockchain, the blockchain can trace the misbehaving participants in the contracts and punish them.

1) Smart trading contract: When computing resource trading happens, the buyers (MEs) need to pay coins from their wallets to the wallet addresses (WA) of providers (edge servers). We use the smart contract to implement the trading process. After the smart trading contract is deployed, the buyers can participate in the trade. The smart trading contract is automatically triggered to complete the trading process when the trade occurs, and the key events on the blockchain are listed as follows:

- A new smart trading contract is created on the blockchain.
- The provider advertises the item and price.

```
Init:      $WA_s, \rho, blist = [], D = [], WA\_list = []$
Create:    On ($WA_s$, $\rho$) from $s$
             Check $WA_s$
CommitD:   On ($WA_{bID}$, $\gamma_{bID}$) from some $bID$
             add($blist$, $bID$)
             add($D$, $\gamma_{bID}$)
             Check $WA_{bID}$
             add($WA\_list$, $WA_{bID}$)
Trade:     Upon receiving () from some $bID$
             Verify $bID \in blist$
             send($WA_{bID}$, $-\gamma_{bID} * \rho$)
             send($WA_s$, $\gamma_{bID} * \rho$)
```

Fig. 2: Smart trading contract.

- Potential buyers monitor the blockchain for new trades and submit their demands.
- The smart trading contract is triggered to complete the trading process.
- The new transaction records are stored into blockchain blocks and verified.

Fig. 2 shows a brief overview of the functions in the smart trading contract. Here, "on receiving message from some *bID*" means that the smart trading contract accepts a message from any buyer who wants to participate in the trade, "on receiving message from s" means that the smart trading contract accepts a message from a seller, and "upon receiving () from some *bID*" means that the smart trading contract accepts a transaction without any parameter from any buyer to execute function.

Init & **Create**. The $Init$ function defines and stores all variables associated with a new trade, including the WA of the seller WA_s, unit price of computing resource ρ, list of buyers $blist$, set of the demand of each buyer D, and set of the WA of each buyer WA_list. The $Create$ function creates a new smart contract on the blockchain and returns the address of the contract. After the smart trading contract is deployed on the blockchain, it runs independently and can be accessed by any participator in the trade.

The seller first executes the $Init$ function to initialize a new trade and then executes the $Create$ function to start a new trade. With $Create$, the seller provides its WA WA_s and the unit price of computing resource ρ. After the $Create$ function is executed, the contract is ready to accept buyers.

CommitD & **Trade**. The buyers that monitor the blockchain can commit their commands to the contract after the $Create$ function is executed. The $CommitD$ function accepts tuple $(bID, \gamma_{bID}, WA_{bID})$ from the buyer *bID*, and the demand γ_{bID} and WA_{bID} are recorded in the contract. The trading process can be automatically performed by triggering the $Trade$ function. During $Trade$, the buyers must pay coins from their wallets to the WA of the seller. Then, the seller can verify the payments by obtaining the new transaction records generated by the buyers. After verifying the payments, the seller allows the buyers to offload their computation tasks to it.

2) Smart loan contract: The MEs that do not have enough coins for offloading need to borrow from banks. In the process, the ME acts as the borrower and the bank acts as the loaner. We use the smart contract to implement the loan process. After the smart loan contract is deployed, the borrowers can participate in the loan. When the loan occurs, the smart loan contract is automatically triggered to complete the loan process, and the key events on the blockchain are listed as follows:

```
Init:      $WA_l, r, blist = [], D = [], WA\_list = []$
Create:    On ($WA_l$, $r$) from $l$
             Check $WA_l$
CommitD:   On ($WA_{bID}$, $l_{bID}$) from some $bID$
             add($blist$, $bID$)
             add($D$, $l_{bID}$)
             Check $WA_{bID}$
             add($WA\_list$, $WA_{bID}$)
Loan:      Upon receiving () from some $bID$
             Verify $bID \in blist$
             send($WA_{dID}$, $l_{bID}$)
             send($WA_l$, $-l_{bID}$)
Repay:     Upon receiving () from some $bID$
             Verify $bID \in blist$
             send($WA_{bID}$, $-l_{bID} * (1 + r)$)
             send($WA_l$, $l_{bID} * (1 + r)$)
```

Fig. 3: Smart loan contract.

- A new smart loan contract is created on the blockchain.
- The loaner advertises its rate of return.
- Potential loaners monitor the blockchain for new loans and submit their demands.
- The smart trading contract is triggered to complete the loan process.
- The new transaction records are stored into blockchain blocks and verified.

Fig. 3 provides a detailed overview of the functions in the smart loan contract. Similar to the smart trading contract, "on receiving message from *bID*" means that the contract receives a message from any borrower who wants to participate in the loan, "on receiving message from l" means that the contract receives a message from a loaner, and "upon receiving () from some *bID*" means that the contract receives a transaction without any parameter from any borrower to execute function.

Init & **Create**. The $Init$ function defines and stores all variables associated with the smart loan contract, including the WA of the loaner WA_l, rate of return r, list of borrowers $blist$, set of the demand of each borrower D, and set of the WA of each borrower WA_list. The $Create$ function deploys a smart loan contract on the blockchain and returns the address of the contract. The loaner needs to send its WA and rate of return to the $Create$ function. After the $Create$ function is executed, the potential borrowers can participate in the loan.

CommitD & **Loan**. The borrowers that monitor the blockchain can commit their loan commands after $Create$. The $CommitD$ function receives and records the tuple (bID, WA_{bID}, l_{bID}) from borrower *bID*, where l_{bID} is the loan demand. The loan process is automatically performed by triggering the $Loan$ function. During $Loan$, the loaner transforms the coins to the borrowers. After $Loan$, the contract generates an IOU (I owe you) from each borrower to the loaner.

Repay. After the borrowers earn coins by completing the data processing tasks or mining tasks, they can repay both principal and interest to the loaners by calling the $Repay$ function. After verifying the payments, the contract destroys any corresponding IOUs.

VIII. Number Results

A. Simulation setup

In the simulations, we placed the MEs and the edge servers into the region of 600 m $\times$ 600 m. We assume that the coverage of each edge servers is 80 m. In the region, there are several banks that provide loan services to the MEs, and we assume that each ME can borrow from any bank. We suppose that each ME is equipped with long-term evolution (LTE) interfaces and can connect with the edge servers via LTE interfaces. Without loss of generality, we suppose that the uplink rate $\omega_{n,s}$ only depends on the type of communication interface and the distance between ME n and edge server s. We consider that the channel gain for ME n to edge server s is proportional to $d_{n,s}^{-\theta}$, where $d_{n,s}$ denotes the distance between ME n and edge server s and θ is the pass loss factor that is set based on the path loss model for the urban and suburban cellular radio environment [26] [37]. Given the background noise $\overline{p}_0$, the uplink rate between ME n and edge server s can be calculated as $\omega_{n,s} = B_s log(1 + p_n d_{n,s}^{-\theta}/\overline{p}_0)$. Besides, the energy consumed per CPU cycle is set as $10^{-11}(f_n^0)^2$ [26], [30]. The main simulation parameters are listed in Table II.

TABLE II: Simulation parameters.

Parameter	Value
The transmit power of the MEs n	0.4 W
The pass loss factor θ	4
The bandwidth B_n	5 MHz
The power of noise $\overline{p}_0$	-120 dBm/Hz
Computing capabilities of the MEs f_n^0	0.5-1 GHz
Computing capabilities of the servers f_s^S	8-16 GHz
Input data size z_n	0.6-1.2 MB
The number of CPU cycles γ_n	0.8-1.5 Gcycles
Weights λ^T, λ^E	0-1, 0-1
Coefficients $\mu_{1,m}$, $\mu_{2,m}$	0.02-0.05, 0.001-0.005
The unit price of computing resources ρ	0.15 token/Gcycle

B. Baseline solutions

In this section, we compare our proposed algorithm with the following solutions:

1) *Genetic algorithm [38]:* Genetic algorithm is an advanced random-searching-based technique, and the algorithm has been shown to be effective in seeking near-optimal solutions of complicated combinational optimization problems. After a sufficient number of iterations, the genetic algorithm can search for a near-optimal result. However, in our problem, the searching space for the genetic algorithm is very large. It takes a large amount of time for the genetic algorithm to search for a near-optimal result in our problem, which is intolerable for the computation tasks. Therefore, in our simulations, we set the number of the generations of the genetic algorithm to be 200,000 and the population size for each generation to be 150. Then, we compared the final searching results in the genetic algorithm with the results of our proposed algorithm.

2) *Local computing by all MEs [25]:* Each ME chooses to compute its computation task using the onboard computing capacities. In this scenario, each ME can avoid the potential performance loss caused by the impacts of the decisions of other MEs.

3) *Computation offloading by all MEs [25]:* Each ME offloads its computation task to an edge server and borrows from a bank in a random manner. In this scenario, because each irrational ME does not consider the impacts of the decisions of other MEs, they may suffer performance loss.

C. Convergence analysis

To demonstrate the convergence of our proposed distributed algorithm, we plot the dynamics of the system cost and the value of w-potential function over time slot in Fig. 4.

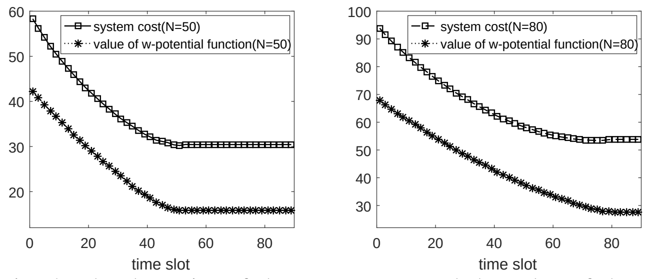

Fig. 4: The dynamics of the system cost and the value of the w-potential function over time slot.

The figure shows that the proposed algorithm can ensure the system cost decreases and converges to a stable state, i.e., the NE point. At the same time, the value of the w-potential function also decreases as the time slot increases. Moreover, the system cost and the value of the w-potential function reach a stable state simultaneously, indicating that the variation of the cost function of each ME can be mapped into the variation of the w-potential function. Thus, the proposed algorithm enables the system cost to reach a stable point where no MEs can reduce their cost by changing their decisions unilaterally, i.e., the NE point of the game $\mathcal{G}$, demonstrating that the proposed algorithm has good convergence.

To evaluate the computational complexity of the proposed algorithm, in Fig. 5, the number of iterations of the proposed algorithm under different numbers of MEs, edge servers, and banks are presented. As shown in Fig. 5, as the number of MEs in the network increases, the number of iterations needed to reach the NE point increases. This relationship exists because as the number of MEs in the network increases, more MEs have better decisions to update. Moreover, as the number of MEs increases, the number of iterations for convergence increases slowly. When there are more edge servers and banks, the number of iterations required to reach the NE point increases. This is because as the numbers of edge servers and banks in the network increase, there are more decisions for each ME to make. However, the increase is very small, demonstrating that the proposed algorithm can converge to the NE point in a fast manner and scales well as the number of MEs increases.

D. Performance analysis

This section analyzes the performance of the proposed distributed algorithm. Fig. 4 shows that as the time slot increases, the proposed algorithm can continue to decrease the system cost before reaching the NE point. When the

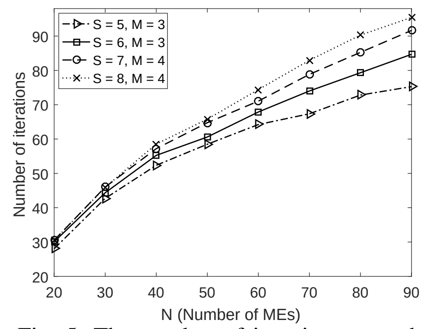

Fig. 5: The number of iterations over the number of MEs.

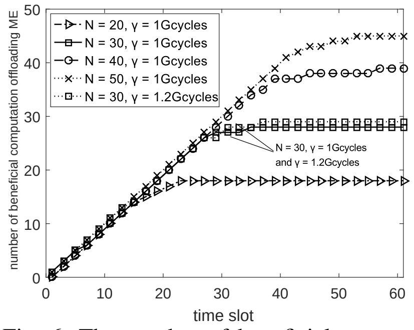

Fig. 6: The number of beneficial computation offloading MEs over time slot.

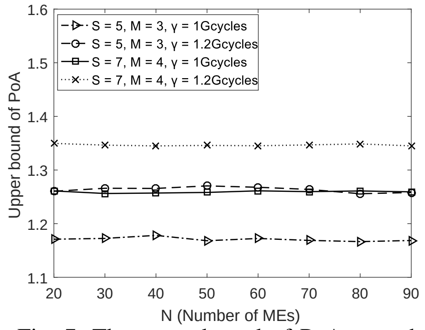

Fig. 7: The upper bound of PoA over the number of MEs.

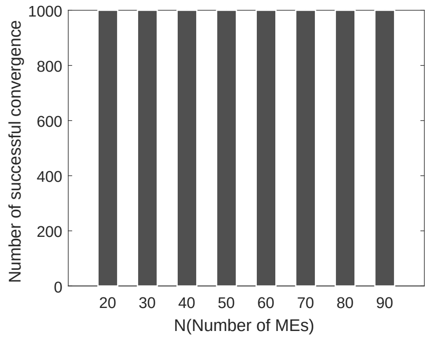

Fig. 8: The number of successful convergences of the proposed algorithm in the extended model per 1000 simulations.

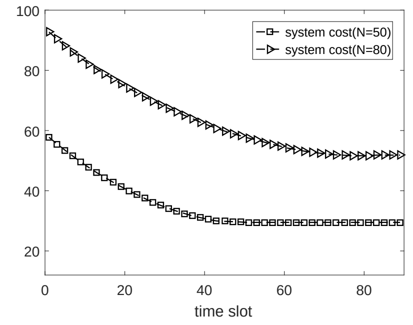

Fig. 9: The dynamics of the system cost over time slot in the extended model.

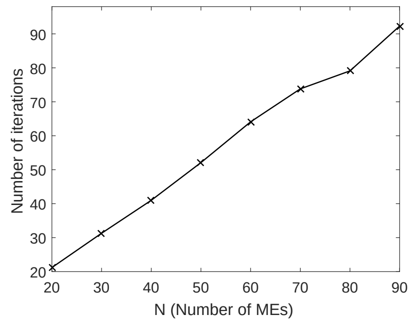

Fig. 10: The number of iterations over the number of MEs in the extended model.

NE point is reached by the proposed algorithm, the system cost of the network remains constant, demonstrating that the proposed algorithm can effectively reduce the system cost of the network. For different network densities ($N = 50$ and $N = 80$ in Fig. 4), as the number of MEs increases, the system cost increases. However, because of the game between different MEs, the system cost at the NE point increases slowly as the network densities increase. This relationship demonstrates that the proposed algorithm can effectively improve the computation performance of the network.

Fig. 6 illustrates the number of beneficial computation-offloading MEs over time slot under different network densities and computation loads. Fig. 6 shows that the number of beneficial computation-offloading MEs becomes constant finally because the proposed algorithm converges to the NE point. Before reaching the NE point, there are more MEs that can benefit by offloading their computation tasks to the edge servers because the edge servers have more computing resources than the MEs. Moreover, if the size of input data increases under the same number of MEs, the number of beneficial computation-offloading MEs also increases. This relationship is because when the size of input data increases, the computation cost of local computing increases, which is shown in formula (9). Thus, more MEs offload their computation tasks to the edge servers to reduce computation cost.

Fig. 7 shows the variation of the upper bound on PoA in the game $\mathcal{G}$ as the number of MEs increases under different numbers of edge servers, banks, and computation loads. The results in Fig. 7 show that as the number of MEs increases, the upper bound on the PoA remains almost unchanged. However, when the numbers of edge servers and banks increase, the upper bound on PoA under the same network density increases. This relationship is because when the numbers of edge servers and banks increase, there are more MEs that offload their computation tasks to the edge servers and borrow from banks. Besides, if the computation loads (i.e., the number of CPU cycles) increase, the upper bound on PoA increases as well. This relationship is because more MEs offload their computation tasks to the edge servers because of the higher computation cost in local computing.

E. Performance of partial loan

Based on Assumption 1, the initial model considers the case of fully depending on a loan to pay the offloading service cost. We now extend the partial loan case into the initial model by considering the inherent asset for each ME. We use Δ_n to denote the inherent asset of ME n. Hence, if ME n decides to offload, the amount l_n that ME n needs to borrow from a bank is calculated as:

$$l_n = \begin{cases} 0, & \theta_n \leq \Delta_n, \\ \theta_n - \Delta_n, & \theta_n > \Delta_n. \end{cases} \tag{19}$$

The above equation means that if the inherent coins of ME n can cover the cost of offloading, it does not borrow from any bank. Otherwise, ME n must borrow from a bank to attain sufficient coins for offloading. Thus, when ME n decides to borrow from bank m, its loan cost can be calculated as:

$$\widetilde{\pi}_n^m = l_n \cdot R_m\Big(n_n^M(\mathbf{d})\Big). \tag{20}$$

Unfortunately, it is difficult to theoretically prove the existence of the NE point for this extended model, because the inherent asset Δ_n of ME n has nothing to do with γ_n which is a weight of the weighted potential game. Therefore,

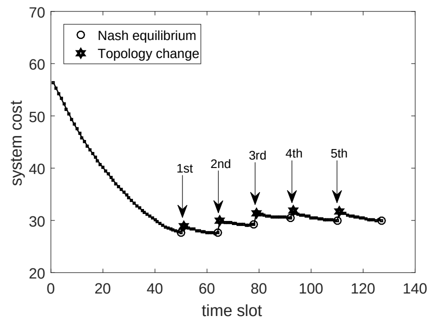

Fig. 11: The proposed algorithm adapts to the topology changes.

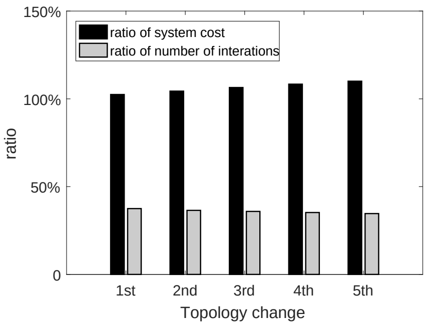

Fig. 12: An illustration of the universality of the phenomenon shown in Fig. 11.

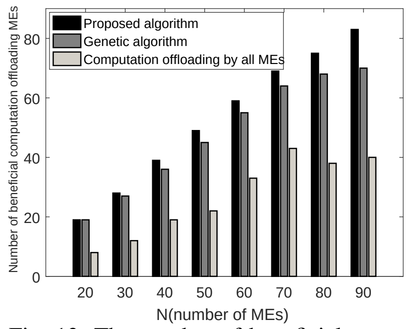

Fig. 13: The number of beneficial computation offloading MEs over the number of MEs.

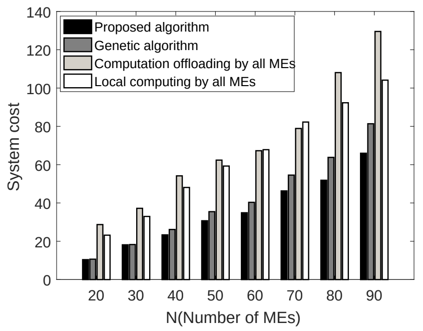

Fig. 14: The system cost over the number of MEs.

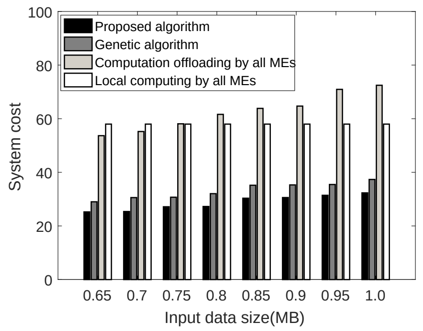

Fig. 15: The system cost over the size of the input data of the computation task.

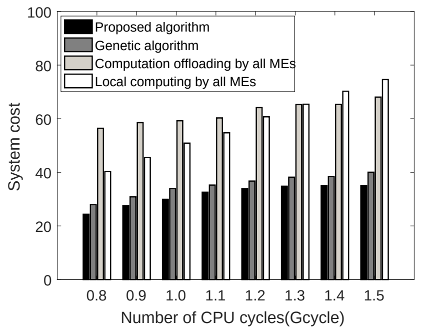

Fig. 16: The system cost over the number of CPU cycles of the computation task.

we show the convergence of the proposed algorithm in the extended model by extensive simulations. In the simulations, we draw Δ_n from a continuous uniform distribution with parameters $[0, 1]$. Fig. 8 provides the number of successful convergences of the proposed algorithm in the extended model per 1000 simulations. To our surprise, the results in Fig. 8 show that the proposed algorithm under different numbers of MEs always converges in the 1000 simulations. Fig. 9 shows the dynamics of the system cost over time slots in the extended model. The results in Fig. 9 demonstrate that our proposed algorithm can significantly reduce the system cost in the extended model. Fig. 10 shows the number of iterations required for the proposed algorithm to converge in the extended model. The results in Fig. 10 demonstrate that the proposed algorithm can converge quickly in the extended model. The reason why the proposed algorithm shows pretty good performance in the extended model may be that the change of the cost function of each ME can still be mapped into a global function.

F. Topology-varying adaptation

In a practical mobile scenario, the topology of the network changes over time with the move of the MEs, so we need to run the proposed algorithm continuously. To evaluate the performance of our proposed algorithm when the topology changes, we conduct a simulation to imitate how our algorithm adapts to the varying topology.

In Fig. 11, we first execute the proposed algorithm to achieve the NE point of the game, as performed in the simulation shown in Fig. 4. After 50 time slots, the MEs reach the NE point (denoted by the left-most cycle). Then, we allow the MEs to move freely at a speed of 20 m/s for 2 seconds [27]. The movement of the MEs causes the decisions of some MEs to no longer be optimal, so the system cost increases (denoted by the left-most hexagram), as shown in Fig. 11. To our surprise, the increase in the system cost is fairly small. Besides, the MEs can reach a new NE point again only using 15 time slots (denoted by the second circle), which is much faster than the first time. Similarly, we see that for the other four cases, topology changes yielded similar results.

To show the universality of the phenomenon shown in Fig. 11, we conduct the simulation shown in Fig. 11 repeatedly, and the average results are recorded in Fig. 12. In Fig. 12, the black bars denote the ratios of the number of iterations needed to reach the NE point after the 1st, 2nd, 3rd, 4th, and 5th topology changes to that needed to reach the NE point before the topology changes. The gray bars denote the ratios of the system cost at the NE point after the 1st, 2nd, 3rd, 4th, and 5th topology changes to that at the NE point before the topology changes. Fig. 12 shows that the heights of the black bars are about 100% and the heights of the gray bars are about 30%. These results demonstrate that after changing the topology, our proposed algorithm can quickly converge to the NE point again, and the change in the system cost at the NE point is very small.

Overall, the proposed algorithm can adapt to the varying topology quickly and maintain high performance in a practical mobile scenario.

G. Impacts of parameter settings

This section investigates the performance of the proposed distributed algorithm under different parameter settings. We plot the number of beneficial computation-offloading MEs versus the number of MEs in Fig. 13. The figure shows that

TABLE III: The cost for the seller and buyers in executing the smart trading contract when there are 20 buyers.

Functions	Transaction Cost in Gas	Execution Cost in Gas	Total Cost in Gas	Total Cost in $
Init	607525	424464	1031989	5.7791
Create	61993	40529	102522	0.5741
CommitD	23383	703	24086	0.1349
Trade	39450	16770	56220	0.3148
Seller Total	669518	464993	1134511	6.3533
Buyer Total	62833	17473	80306	0.4497
Trading Total	732351	482466	1214817	6.8030

TABLE IV: The cost for the loaner and borrowers in executing the smart loan contract when there are 20 borrowers.

Functions	Transaction Cost in Gas	Execution Cost in Gas	Total Cost in Gas	Total Cost in $
Init	796848	568600	1365448	7.6465
Create	61993	40529	102522	0.5741
CommitD	23427	747	24174	0.1354
Loan	38301	15621	53922	0.3020
Repay	38685	16005	54690	0.3063
Loaner Total	858841	609129	1467970	8.2206
Borrower Total	100413	32373	132786	0.7436
Loan Total	959254	641502	1600756	8.9642

as the number of MEs increases, the number of beneficial computation-offloading MEs in the proposed algorithm and the genetic algorithm increases, and the results of the proposed algorithm are better than those of the genetic algorithm. While the number of MEs increases to 70, the number of beneficial computation-offloading MEs in the solution of computation offloading by all MEs decreases. This relationship is because when the network density increases to a certain level, the impacts between the decisions of each ME become serious in this solution.

Fig. 14 shows the system cost calculated from each solution under different network densities (i.e., different numbers of MEs). As shown in Fig. 13, when the number of MEs increases, the system cost in each solution increases as well. In contrast, in the proposed algorithm, because of the game between different MEs, the system cost increases slowly as the network density increases. Moreover, the system cost in the proposed algorithm under different network densities remains the smallest, demonstrating that the proposed algorithm can significantly reduce the system cost.

To analyze the impacts of the input data sizes of the computation tasks, we conducted the simulation shown in Fig. 15. The figure shows that the system cost in the proposed algorithm, the genetic algorithm and the solution of computation offloading by all MEs increases as the input data sizes of the computation tasks increase. This relationship is because when the input data sizes of the computation tasks increase, transmitting the computation tasks consumes more time and energy. However, the system cost in the solution of local computing by all MEs remains constant, because the computation cost in the case of local computing has nothing to do with the input data sizes of the computation tasks, and this relationship is shown in formula (9).

In Fig. 16, we investigate the impacts of the computation loads of the computation tasks (i.e., the number of CPU cycles). The figure shows that when the computation load increases, the system cost in each solution also increases. This relationship is because computing the computation tasks consumes more time and energy if the computation loads of the computation tasks increase. In contrast, the MEs that offload computation tasks to the edge servers borrow more from banks, as demonstrated in formula (8).

Lastly, Figs. 13, 14, 15, and 16 show that the proposed distributed algorithm can significantly improve the computation performance of the network. Comparing with the solutions of computation offloading by all MEs and local computing by all MEs, there is no doubt that the proposed algorithm possesses much better performance. The genetic algorithm can search for a near-optimal result after sufficient iterations, but it would take a long period of time. While in the simulation we have set the number of the generations of the genetic algorithm to be 200,000, and the population size for each generation to be 150, the genetic algorithm still takes much more time than the proposed algorithm. Besides, the MEs owned by different parties may not want to obey the centralized optimization solution because they pursue their own interests. Therefore, the proposed algorithm shows better performance by comparison, demonstrating the efficiency of the proposed algorithm.

H. Performance of smart contracts

To demonstrate the performance of the smart contracts on the blockchain, we deploy these smart contracts in the Ethereum environment. In Table III, we show the cost of executing each function of the smart trading contract on the Ethereum network when there are 20 buyers. In Table IV, we show the cost of executing each function of the smart loan contract on the Ethereum network when there are 20 borrowers. The cost is in the amount of gas on the Ethereum network, and then we convert the cost in gas into the cost in dollars by using an exchange rate of 1 Gas $\approx$ 0.00000002 Ether and 1 Ether $\approx$ \$ 280 in July 2019. As we can see, for the smart trading contract, the total cost is 1.13 million gas (\$ 6.35) for each seller and about 80 thousand gas (\$ 0.45) for each buyer; and for the smart loan contract, the total cost is about 1.47 million gas (\$ 8.22) for each loaner and about 132 thousand gas (\$ 0.74) for each borrower. Therefore, the financial cost for executing these smart contracts on the Ethereum network is low, demonstrating the practicality of the smart contracts.

IX. Conclusion

In this paper, we aim to optimize the total cost of all MEs in the joint computation-offloading and coin-loaning problem in which the role of a bank is introduced to address the "cold start" and "long return" issues for blockchain-empowered MEC. We solve the optimization problem in a distributed manner by designing an efficient distributed algorithm based on a potential game method. After extending the partial offloading case into the initial model, we theoretically prove that the case of partial offloading can be reduced to the case of binary offloading. Besides, a smart trading contract is designed for automatic computing resource trading between MEs and edge servers, and a smart loan contract is designed for automatic loan and repayment between MEs and banks. Lastly, our simulation results demonstrate that the proposed algorithm can quickly converge to the NE point, achieve superior computation offloading performance, and adapt to the topology changes. In addition, by deploying the smart contracts on the Ethereum network, the results demonstrate that the financial cost for executing the contracts in Ethereum environment is low.

For future work, we will utilize the Stackelberg game to optimize the interests of the MEs, edge servers, and banks at the same time.

Appendix A
Proof of Theorem 1

We first define the w-potential function in the game $\mathcal{G}$ as follows:

$$\begin{aligned}\Phi(\mathbf{d}) = \lambda^T \sum_{s\in\mathcal{S}} \sum_{n=1}^{n_s^S(\mathbf{d})} \frac{n}{f_s^S} + \rho \sum_{m\in\mathcal{M}} \sum_{n=1}^{n_m^M(\mathbf{d})} n\mu_{2,m} \\ + \sum_{n\in\mathcal{N}} o_n(\mathbf{d}) I_{\{\mathbf{a}_n \neq \mathbf{0}\}} + \sum_{n\in\mathcal{N}} q_n I_{\{\mathbf{a}_n = \mathbf{0}\}},\end{aligned} \tag{21}$$

where $o_n(\mathbf{d}) = \frac{\lambda^E e_n^{off,k_n(\mathbf{d})}(\mathbf{d}) + \lambda^T t_n^{off,k_n(\mathbf{d})}(\mathbf{d}) + \mu_{1,k_n(\mathbf{d})}\theta_n}{\gamma_n}$, $k_n(\mathbf{d})$ denotes the bank that ME n borrows from in the decision profile $\mathbf{d}$, and $q_n = \frac{c_n^0 - \theta_n}{\gamma_n}$.

We show in the following five cases that the function $\Phi(\mathbf{d})$ is a w-potential function that satisfies the equation in Definition 4. Based on our model, there are five different changing models of decision profile $\mathbf{d}$. We calculate the variation of cost function and w-potential function based on the changing models of decision profile $\mathbf{d}$. If ME n decides to update its decision in the decision profile $\mathbf{d}$, we consider its decision before updating as d_n and its decision after updating as d'_n.

For case 1), ME n that locally computes its computation task updates its current decision to offload its computation task to edge server s' and borrow from bank m'. Then, the variation of its cost function when updating its decision from d_n to d'_n can be calculated as:

$$\begin{aligned}&c_n(d'_n, \mathbf{d}_{-n}) - c_n(d_n, \mathbf{d}_{-n}) \\ &= (\rho \sum_{n'\in\mathcal{N}} b_{n',m'}\mu_{2,m'} + \lambda^T \sum_{n'\in\mathcal{N}} \frac{a_{n',s'}}{f_{s'}^S})\gamma_n \\ &\quad + (1+\mu_{1,m'})\gamma_n\rho + \frac{(\lambda^E p_n + \lambda^T) z_n}{\omega_{n,s'}} - c_n^0.\end{aligned}$$

According to (21), for this case, we can calculate the variation of the w-potential function:

$$\begin{aligned}&\Phi(d'_n, \mathbf{d}_{-n}) - \Phi(d_n, \mathbf{d}_{-n}) \\ &= \rho \sum_{n'\in\mathcal{N}} b_{n',m'}\mu_{2,m'} + \lambda^T \sum_{n'\in\mathcal{N}} \frac{a_{n',s'}}{f_{s'}^S} \\ &\quad + \frac{(\lambda^E p_n + \lambda^T) z_n}{\gamma_n \omega_{n,s'}} - \frac{c_n^0 - (1+\mu_{1,m'})\gamma_n\rho}{\gamma_n} \\ &= \frac{1}{\gamma_n}\Big(c_n(\mathbf{d}') - c_n(\mathbf{d})\Big).\end{aligned}$$

For case 2), ME n that offloads its computation task to edge server s and borrows from bank m updates its current decision to offload its computation task to edge server s' and borrow from bank m. Then, the variation of its cost function when updating its decision from d_n to d'_n can be calculated as:

$$\begin{aligned}&c_n(d'_n, \mathbf{d}_{-n}) - c_n(d_n, \mathbf{d}_{-n}) \\ &= \lambda^T\gamma_n \sum_{n'\in\mathcal{N}} (\frac{a_{n',s'}}{f_{s'}^S} - \frac{a_{n',s}}{f_s^S}) + (\lambda^E p_n + \lambda^T)(\frac{z_n}{\omega_{n,s'}} - \frac{z_n}{\omega_{n,s}}).\end{aligned}$$

In this case, the variation of the w-potential function can be calculated as:

$$\begin{aligned}&\Phi(d'_n, \mathbf{d}_{-n}) - \Phi(d_n, \mathbf{d}_{-n}) \\ &= \lambda^T \sum_{n'\in\mathcal{N}} (\frac{a_{n',s'}}{f_{s'}^S} - \frac{a_{n',s}}{f_s^S}) + \frac{\lambda^E p_n + \lambda^T}{\gamma_n}(\frac{z_n}{\omega_{n,s'}} - \frac{z_n}{\omega_{n,s}}) \\ &= \frac{1}{\gamma_n}\Big(c_n(\mathbf{d}') - c_n(\mathbf{d})\Big).\end{aligned}$$

For case 3), ME n that offloads its computation task to edge server s and borrows from bank m updates its current decision to offload its computation task to edge server s and borrow from bank m'. Then, the variation of its cost function when updating its decision from d_n to d'_n can be calculated as:

$$\begin{aligned}&c_n(d'_n, \mathbf{d}_{-n}) - c_n(d_n, \mathbf{d}_{-n}) \\ &= \gamma_n\rho \sum_{n'\in\mathcal{N}} (b_{n',m'}\mu_{2,m'} - b_{n',m}\mu_{2,m}) + \frac{\theta_n}{\gamma_n}(\mu_{1,m'} - \mu_{1,m}).\end{aligned}$$

In this case, the variation of the w-potential function is the following:

$$\begin{aligned}&\Phi(d'_n, \mathbf{d}_{-n}) - \Phi(d_n, \mathbf{d}_{-n}) \\ &= \rho \sum_{n'\in\mathcal{N}} (b_{n',m'}\mu_{2,m'} - b_{n',m}\mu_{2,m}) + \theta_n(\mu_{1,m'} - \mu_{1,m}) \\ &= \frac{1}{\gamma_n}\Big(c_n(\mathbf{d}') - c_n(\mathbf{d})\Big).\end{aligned}$$

For case 4), ME n that offloads its computation task to edge server s and borrows from bank m updates its current decision to offload its computation task to edge server s' and loan from bank m'. Then, the variation of its cost function when updating its decision from d_n to d'_n can be calculated as:

$$\begin{aligned}&c_n(d'_n, \mathbf{d}_{-n}) - c_n(d_n, \mathbf{d}_{-n}) \\ &= \gamma_n\Big(\lambda^T \sum_{n'\in\mathcal{N}} (\frac{a_{n',s'}}{f_{s'}^S} - \frac{a_{n',s}}{f_s^S}) \\ &\quad + \rho \sum_{n'\in\mathcal{N}} (b_{n',m'}\mu_{2,m'} - b_{n',m}\mu_{2,m})\Big) \\ &\quad + (\lambda^E p_n + \lambda^T)(\frac{z_n}{\omega_{n,s'}} - \frac{z_n}{\omega_{n,s}}) + \frac{\theta_n}{\gamma_n}(\mu_{1,m'} - \mu_{1,m}).\end{aligned}$$

In this case, the variation of the w-potential function can be calculated as:

$$\begin{aligned}&\Phi(d'_n,\mathbf{d}_{-n})-\Phi(d_n,\mathbf{d}_{-n})\\&=\lambda^T\sum_{n'\in\mathcal{N}}\Big(\frac{a_{n',s'}}{f^S_{s'}}-\frac{a_{n',s}}{f^S_s}\Big)\\&\quad+\rho\sum_{n'\in\mathcal{N}}(b_{n',m'}\mu_{2,m'}-b_{n',m}\mu_{2,m})\\&\quad+\frac{\lambda^E p_n+\lambda^T}{\gamma_n}\Big(\frac{z_n}{\omega_{n,s'}}-\frac{z_n}{\omega_{n,s}}\Big)+\theta_n(\mu_{1,m'}-\mu_{1,m})\\&=\frac{1}{\gamma_n}\Big(c_n(\mathbf{d}')-c_n(\mathbf{d})\Big).\end{aligned}$$

For case 5), ME n that offloads its computation task to edge server s and borrows from bank m updates its current decision to local computing. Similar to the arguments in case 1), $\Phi(\mathbf{d}')-\Phi(\mathbf{d})=\frac{1}{\gamma_n}\Big(c_n(\mathbf{d}')-c_n(\mathbf{d})\Big)$ holds in this case.

In terms of the results in the presented five cases of the changing models of decision profile $\mathbf{d}$, the game $\mathcal{G}$ in our model is a weighted potential game with the weight vector $\mathbf{W}=(\gamma_n)_{n\in\mathcal{N}}$. Therefore, Theorem 1 is proven.

Appendix B
Proof of Theorem 2

This theorem can be proven according to the corollaries on the potential game shown in [13], [39]: *every finite potential game possesses a pure-strategy NE point and has the FIP.* Because the decision spaces of the players in the game $\mathcal{G}$ are finite and we have proven that the game $\mathcal{G}$ is a weighted potential game, the pure-strategy NE point exists and the FIP holds, completing the proof of Theorem 2.

Appendix C
Proof of Theorem 3

Because the formula (16) is continuously differentiable with respect to variable α_n, we obtain the following:

$$\frac{\partial\overline{c}_n(\overline{\mathbf{d}})}{\partial\alpha_n}=c^0_n-c^{s,m}_n(\mathbf{d}).\tag{22}$$

According to above partial derivative, we know that if $c^0_n>c^{s,m}_n(\mathbf{d})$, then $\frac{\partial\overline{c}_n(\overline{\mathbf{d}})}{\partial\alpha_n}>0$ and $\overline{c}_n(\overline{\mathbf{d}})$ reach a minimum value at point $\alpha_n=0$, meaning that task A_n of ME n is entirely offloaded to edge server m. If $c^0_n<c^{s,m}_n(\mathbf{d})$, then $\frac{\partial\overline{c}_n(\overline{\mathbf{d}})}{\partial\alpha_n}<0$ and $\overline{c}_n(\overline{\mathbf{d}})$ reaches a minimum value at point $\alpha_n=1$, meaning that task A_n of ME n is entirely computed locally. If $c^0_n=c^{s,m}_n(\mathbf{d})$, then $\frac{\partial\overline{c}_n(\overline{\mathbf{d}})}{\partial\alpha_n}=0$, meaning that $\overline{c}_n(\overline{\mathbf{d}})$ is a constant function with α_n, so we can choose $\alpha_n=0$ or $\alpha_n=1$ for ME n in this case. Therefore, Theorem 3 holds.

Appendix D
Proof of Theorem 4

We first define the w-potential function in the game $\overline{\mathcal{G}}$ as follows:

$$\begin{aligned}\overline{\Phi}(\overline{\mathbf{d}})=\lambda^T\sum_{s\in\mathcal{S}}\sum_{n=1}^{n^S_s(\mathbf{d})}\frac{n}{f^S_s}+\rho\sum_{m\in\mathcal{M}}\sum_{n=1}^{n^M_m(\mathbf{d})}n\mu_{2,m}\\+\sum_{n\in\mathcal{N}}o_n(\mathbf{d})(1-\alpha_n)+\sum_{n\in\mathcal{N}}q_n\alpha_n,\end{aligned}\tag{23}$$

where $o_n(\mathbf{d})=\frac{\lambda^E e^{off,k_n(\mathbf{d})}_n(\mathbf{d})+\lambda^T t^{off,k_n(\mathbf{d})}_n(\mathbf{d})+\mu_{1,k_n(\mathbf{d})}\theta_n}{\gamma_{n'}}$, $k_n(\mathbf{d})$ denotes the bank that ME n borrows from in the decision profile $\mathbf{d}$, and $q_n=\frac{c^0_n-\theta_n}{\gamma_n}$.

According to Theorem 3, the best choice of α_n for each ME is $\alpha_n=0$ or $\alpha_n=1$ in the case of partial offloading. Therefore, the changing models of decision profile $\overline{\mathbf{d}}$ in the game $\overline{\mathcal{G}}$ are the same as the changing models of decision profile $\mathbf{d}$ in the game $\mathcal{G}$. To prove that the function defined above satisfies the equation in Definition 4, we consider a changing model of decision profile $\overline{\mathbf{d}}$, in which ME n that computes its computation task locally wants to update its current decision to offload its computation task to edge server s' and borrow from bank m'. Based on Theorem 3, we have $\alpha_n=0$ in this case. We consider the decision of ME n before updating as $\overline{d}_n$ and the decision of ME n after updating as $\overline{d}'_n$. Then, the variation of its cost function when updating its decision from $\overline{d}_n$ to $\overline{d}'_n$ can be calculated as:

$$\begin{aligned}&\overline{c}_n(\overline{d}'_n,\overline{\mathbf{d}}_{-n})-\overline{c}_n(\overline{d}_n,\overline{\mathbf{d}}_{-n})\\&=\Big((\rho\sum_{n'\in\mathcal{N}}b_{n',m'}\mu_{2,m'}+\lambda^T\sum_{n'\in\mathcal{N}}\frac{a_{n',s'}}{f^S_{s'}})\gamma_n\\&\quad+(1+\mu_{1,m'})\gamma_n\rho+\frac{(\lambda^E p_n+\lambda^T)z_n}{\omega_{n,s'}}-c^0_n\Big)(1-\alpha_n).\end{aligned}$$

Based on (23), for this case, we can calculate that the variation of the w-potential function in this case is the following:

$$\begin{aligned}&\overline{\Phi}(\overline{d}'_n,\overline{\mathbf{d}}_{-n})-\overline{\Phi}(\overline{d}_n,\overline{\mathbf{d}}_{-n})\\&=\rho\sum_{n'\in\mathcal{N}}b_{n',m'}\mu_{2,m'}+\lambda^T\sum_{n'\in\mathcal{N}}\frac{a_{n',s'}}{f^S_{s'}}\\&\quad+\Big(\frac{(\lambda^E p_n+\lambda^T)z_n}{\gamma_n\omega_{n,s'}}-\frac{c^0_n-(1+\mu_{1,m'})\gamma_n\rho}{\gamma_n}\Big)(1-\alpha_n).\end{aligned}$$

Because we have $\alpha_n=0$, $\overline{\Phi}(\overline{d}'_n,\overline{\mathbf{d}}_{-n})-\overline{\Phi}(\overline{d}_n,\overline{\mathbf{d}}_{-n})=\frac{1}{\gamma_n}\Big(\overline{c}_n(\overline{d}'_n,\overline{\mathbf{d}}_{-n})-\overline{c}_n(\overline{d}_n,\overline{\mathbf{d}}_{-n})\Big)$ holds in this case. Similarly, we can easily obtain the same results in the other four cases of the changing models of decision profile $\overline{\mathbf{d}}$. Therefore, Theorem 4 is proven.

Appendix E
Proof of Theorem 5

We consider that $\mathbf{d}^{\text{NE}}$ is an arbitrary NE point attained using our algorithm and $\mathbf{d}^*$ is the optimal decision calculated using the centralized optimization solution. Because $\mathbf{d}^*$ makes the system cost reach a minimum value, we must have $PoA\geq 1$.

We give an upper bound of $\max_{\mathbf{d}\in\mathbf{D}^{\text{NE}}}c(\mathbf{d})$ by considering $\mathbf{d}^{\text{NE}}$ as the worst-case NE point. We prove that all MEs compute their computation tasks locally is the worst-case NE point resulting in the highest system cost. Because $\mathbf{d}^{\text{NE}}$ is an NE point, we know that $c_n(\mathbf{d}^{\text{NE}})\leq c^0_n$ holds for all ME $n\in\mathcal{N}$. Otherwise, if a ME $n\in\mathcal{N}$ exists such that $c_n(\mathbf{d}^{\text{NE}})\geq c^0_n$, then ME n switches to local computing for smaller computation cost, contradicting the assumption that $\mathbf{d}^{\text{NE}}$ is an NE point. Therefore, $c_n(\mathbf{d}^{\text{NE}})\leq c^0_n$ holds for all ME $n\in\mathcal{N}$, and the worst-case NE point is that all MEs choose local computing. Thus, $\max_{\mathbf{d}\in\mathbf{D}^{\text{NE}}}c(\mathbf{d})$ is upper bounded by $\sum_{n\in\mathcal{N}}c^0_n$.

Next, we derive the lower bound of system cost in the centralized optimal solution. If ME n computes its computation task locally, then its computation cost is c_n^0. Otherwise, if ME n offloads its computation task to edge server s and borrows from bank m, in the best case, there is no competition with other MEs, i.e., $n_s^S(\mathbf{d}) = 1$ and $n_m^M(\mathbf{d}) = 1$. Therefore, based on (10), we can get the following:

$$c_n^{s,m}(d_n, \mathbf{d}_{-n}) \geq \frac{\lambda^T \gamma_n}{f_s^S} + (1 + \mu_{1,m} + \mu_{2,m})\gamma_n \rho + \frac{(\lambda^E p_n + \lambda^T) z_n}{\omega_{n,s}} = \hat{c}_n^{s,m}. \tag{24}$$

For all edge server $s \in \mathcal{S}$ and bank $m \in \mathcal{M}$, we calculate the minimum cost $\hat{c}_n^{min} = \min_{s\in\mathcal{S}, m\in\mathcal{M}} \hat{c}_n^{s,m}$. Then, we have $c_n(d_n, \mathbf{d}_{-n}) \geq \min\{c_n^0, \hat{c}_n^{min}\}$ and $\sum_{n\in\mathcal{N}} c_n(d_n, \mathbf{d}_{-n}) \geq \sum_{n\in\mathcal{N}} \min\{c_n^0, \hat{c}_n^{min}\}$. Thus, we can conclude the following:

$$PoA \leq \frac{\sum_{n\in\mathcal{N}} c_n^0}{\sum_{n\in\mathcal{N}} \min\{c_n^0, \hat{c}_n^{min}\}}, \tag{25}$$

which completes the proof of Theorem 4.

References

[1] N. Dlodlo, O. Gcaba, and A. Smith, “Internet of things technologies in smart cities,” in *2016 IST-Africa Week Conference*, May, pp. 1–7, 2016.

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

[3] X. Ma, S. Wang, S. Zhang, P. Yang, C. Lin, and X. S. Shen, “Cost-efficient resource provisioning for dynamic requests in cloud assisted mobile edge computing,” *IEEE Transactions on Cloud Computing*, pp. 1–1, 2019.

[4] Z. Zhou, J. Feng, Z. Chang, and X. S. Shen, “Energy-efficient edge computing service provisioning for vehicular networks: A consensus admm approach,” *IEEE Transactions on Vehicular Technology*, pp. 1–1, 2019.

[5] W. Chen, B. Liu, H. Huang, S. Guo, and Z. Zheng, “When uav swarm meets edge-cloud computing: The qos perspective,” *IEEE Network*, vol. 33, no. 2, pp. 36–43, March 2019.

[6] K. Christidis and M. Devetsikiotis, “Blockchains and smart contracts for the internet of things,” *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[7] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, “Smart contract-based access control for the internet of things,” *IEEE Internet of Things Journal*, pp. 1–1, 2018.

[8] Z. Li, J. Kang, R. Yu, D. Ye, Q. Deng, and Y. Zhang, “Consortium blockchain for secure energy trading in industrial internet of things,” *IEEE Transactions on Industrial Informatics*, vol. 14, no. 8, pp. 3690–3700, Aug 2018.

[9] Y. Jiao, P. Wang, D. Niyato, and K. Suankaewmanee, “Auction mechanisms in cloud/fog computing resource allocation for public blockchain networks,” *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–1, 2019.

[10] J. Wood, “Lending and borrowing on the blockchainshould banks be scared?” https://medium.com/trivial-co/.

[11] Y. Chen, N. Zhang, Y. Zhang, and X. Chen, “Dynamic computation offloading in edge computing for internet of things,” *IEEE Internet of Things Journal*, pp. 1–1, 2018.

[12] F. Sun, F. Hou, N. Cheng, M. Wang, H. Zhou, L. Gui, and X. Shen, “Cooperative task scheduling for computation offloading in vehicular cloud,” *IEEE Transactions on Vehicular Technology*, vol. 67, no. 11, pp. 11 049–11 061, Nov 2018.

[13] H. Shah-Mansouri and V. W. Wong, “Hierarchical fog-cloud computing for iot systems: A computation offloading game,” *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 3246–3257, 2018.

[14] J. Zheng, Y. Cai, Y. Wu, and X. S. Shen, “Dynamic computation offloading for mobile cloud computing: A stochastic game-theoretic approach,” *IEEE Transactions on Mobile Computing*, pp. 1–1, 2018.

[15] H. Guo, J. Zhang, J. Liu, and H. Zhang, “Energy-aware computation offloading and transmit power allocation in ultra-dense iot networks,” *IEEE Internet of Things Journal*, pp. 1–1, 2018.

[16] Y. Zhang, B. Feng, W. Quan, G. Li, H. Zhou, and H. Zhang, “Theoretical analysis on edge computation offloading policies for iot devices,” *IEEE Internet of Things Journal*, pp. 1–1, 2018.

[17] H. Guo, J. Liu, J. Zhang, W. Sun, and N. Kato, “Mobile-edge computation offloading for ultra-dense iot networks,” *IEEE Internet of Things Journal*, pp. 1–1, 2018.

[18] H. Al-Shatri, S. Müller, and A. Klein, “Distributed algorithm for energy efficient multi-hop computation offloading,” in *2016 IEEE International Conference on Communications (ICC)*, pp. 1–6, 2016.

[19] D. Chatzopoulos, M. Ahmadi, S. Kosta, and P. Hui, “Flopcoin: A cryptocurrency for computation offloading,” *IEEE Transactions on Mobile Computing*, vol. 17, no. 5, pp. 1062–1075, 2018.

[20] M. Liu, R. Yu, Y. Teng, V. C. Leung, and M. Song, “Computation offloading and content caching in wireless blockchain networks with mobile edge computing,” *IEEE Transactions on Vehicular Technology*, 2018.

[21] Z. Xiong, S. Feng, D. Niyato, P. Wang, and Z. Han, “Edge computing resource management and pricing for mobile blockchain,” *arXiv preprint arXiv:1710.01567*, 2017.

[22] Z. Li, Z. Yang, and S. Xie, “Computing resource trading for edge-cloud-assisted internet of things,” *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2019.

[23] Y. Jiao, P. Wang, D. Niyato, and Z. Xiong, “Social welfare maximization auction in edge computing resource allocation for mobile blockchain,” in *IEEE International Conference on Communications*, pp. 1–6, 2018.

[24] Z. Li, Z. Yang, S. Xie, W. Chen, and K. Liu, “Credit-based payments for fast computing resource trading in edge-assisted internet of things,” *IEEE Internet of Things Journal*, pp. 1–1, 2019.

[25] 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.

[26] S. Jošilo and G. Dán, “A game theoretic analysis of selfish mobile computation offloading,” in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pp. 1–9, 2017.

[27] Z. Hong, H. Huang, S. Guo, W. Chen, and Z. Zheng, “Qos-aware cooperative computation offloading for robot swarms in cloud robotics,” *IEEE Transactions on Vehicular Technology*, pp. 1–1, 2019.

[28] H. Shah-Mansouri and V. W. Wong, “Hierarchical fog-cloud computing for iot systems: A computation offloading game,” *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 3246–3257, 2018.

[29] Y. Wen, W. Zhang, and H. Luo, “Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones,” in *2012 Proceedings Ieee Infocom*, pp. 2716–2720, 2012.

[30] S. Jošilo and G. Dán, “Selfish decentralized computation offloading for mobile cloud computing in dense wireless networks,” *IEEE Transactions on Mobile Computing*, vol. 18, no. 1, pp. 207–220, 2019.

[31] N. Li, J.-F. Martinez-Ortega, and V. H. Diaz, “Distributed power control for interference-aware multi-user mobile edge computing: A game theory approach,” *IEEE Access*, vol. 6, pp. 36 105–36 114, 2018.

[32] N. Li, J.-F. Martinez-Ortega, and G. Rubio, “Distributed joint offloading decision and resource allocation for multi-user mobile edge computing: A game theory approach,” *arXiv preprint arXiv:1805.02182*, 2018.

[33] D. Satria, D. Park, and M. Jo, “Recovery for overloaded mobile edge computing,” *Future Generation Comp. Syst.*, vol. 70, pp. 138–147, 2017.

[34] Y. Mao, J. Zhang, S. Song, and K. B. Letaief, “Power-delay tradeoff in multi-user mobile-edge computing systems,” in *2016 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, 2016.

[35] Y. Mao, J. Zhang, S. H. Song, and K. B. Letaief, “Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems,” *IEEE Transactions on Wireless Communications*, vol. 16, no. 9, pp. 5994–6009, Sep. 2017.

[36] S. Wu, Y. Chen, Q. Wang, M. Li, C. Wang, and X. Luo, “Cream: A smart contract enabled collusion-resistant e-auction,” *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 7, pp. 1687–1701, July 2019.

[37] T. S. Rappaport *et al.*, *Wireless communications: principles and practice*. prentice hall PTR New Jersey, 1996.

[38] L. Davis, “Handbook of genetic algorithms,” 1991.

[39] D. Monderer and L. S. Shapley, “Potential games,” *Games and economic behavior*, vol. 14, no. 1, pp. 124–143, 1996.