Computation Offloading with Instantaneous Load Billing for Mobile Edge Computing

Mingjin Gao, Rujing Shen, Jun Li, Shihao Yan, Yonghui Li, Jinglin Shi, Zhu Han, and Li Zhuo

Abstract—Mobile edge computing (MEC) is a promising approach that can reduce the latency of task processing by offloading tasks from user equipments (UEs) to MEC servers. Existing works always assume that the MEC server is capable of executing the offloaded tasks, without considering the impact of improper load on task processing efficiency. In this paper, we present a two-stage computing offloading scheme to minimize the task processing delay while managing the server load properly. To minimize the task processing delay, each UE optimizes how much workload to be offloaded to the MEC server. To improve the task processing efficiency of the server, we arrange the processing order of offloading tasks by introducing an aggregative game with an instantaneous load billing mechanism. The proposed game can obtain the optimal task offloading and processing strategy with limited information and a small number of iterations. Simulation results show that our scheme approaches the optimal offloading strategy in terms of minimizing task processing delay for each UE and improving processing efficiency for the server.

Index Terms—Mobile edge computing, computation offloading, aggregative game, instantaneous load billing

1 Introduction

Ubiquitous user equipments (UEs) provide an increasing amount of mobile applications, such as face recognition or natural language processing. These services contribute to building a highly convenient society, and make soaring demand for strong computing capabilities of UEs. Although current UEs have larger storage and more powerful CPUs, the development of hardware still cannot catch up with the ever-growing demand of complicated applications (e.g., real-time games and online videos) [1–3].

To reduce the computing burden of UEs and to enable computing intensive applications, mobile edge computing (MEC) was proposed [2, 4, 5]. In MEC, tasks can be offloaded from UEs to nearby infrastructures (e.g. MEC servers), which can significantly extend UEs' computing capabilities [6–8]. MEC is a entity within edge computing, while edge computing is a constitution of the Cloud-Fog-Edge-Dew hierarchy. The Cloud-Fog-Edge-Dew hierarchy includes cloud computing, fog computing, edge computing and dew computing [9–11]. In cloud computing, tasks are offloaded to remote clouds, which may result in long transmission delay. Thus, clouding computing is inappropriate to delay sensitive applications. To solve this problem, fog computing and edge computing are emerging by extending the cloud and its services to the edge of the network. Fog computing extends data processing from the cloud to the fog (e.g. fog servers) of a network, while edge computing further extends data processing from the fog to the edge (e.g. edge servers) close to the data source (e.g. UEs) [12]. In dew computing, a dew device is positioned between the fog/edge server and the UE, aiming to provide micro services [13, 14]. In this context, the computing capability of fog/edge server is stronger than that of the dew device.

In this paper, we focus on the computation offloading in MEC, which is an important research topic and has drawn a certain amount of research effort [1, 15]. In [1], the authors provided a dynamic offloading and resource scheduling policy to achieve energy-efficient computation offloading under a strict constraint for application completion time. They also proposed a distributed algorithm to obtain the optimal policy. In [15], the authors formulated a multiobjective optimization problem to balance the energy consumption and execution delay during computation offloading, and they utilized queuing theory to identify the optimal offloading probability and transmit power for each mobile device. However, the aforementioned works only focus on the energy consumption or processing efficiency for UEs, but seldom discuss that for the MEC server, which is impractical and motivates our work. Generally, UEs do not care the processing order of offloading tasks at the server, because the server has powerful computing capability to finish offloading tasks with a short delay. As a result, they may offload their tasks to the MEC server simultaneously, leading to serious congestions. These congestions can

M. Gao, R. Shen and J. Shi are with the Institute of Computing Technology, Chinese Academy of Science as well as the Beijing Key Laboratory of Mobile Computing and Pervasive Device, Beijing, China. R. Shen is also with the University of Chinese Academy of Sciences, Beijing, China. J. Li is with the School of Electronic and Optical Engineering, Nanjing University of Science and Technology, Nanjing, China. He is also with the National Mobile Communications Research Laboratory, Southeast University, Nanjing, China, and with the School of Computer Science and Robotics, National Research Tomsk Polytechnic University, Tomsk, 634050, Russia. S. Yan is with the School of Engineering, Macquarie University, Sydney, NSW 2109, Australia. Y. Li is with the university of Sydney, Australia. He is also affiliated with Pengcheng Laboratory, Shenzhen, China. Z. Han is with the University of Houston, Houston, TX 77004 USA, and also with the Department of Computer Science and Engineering, Kyung Hee University, Seoul, South Korea, 446-701. L. Zhuo is with Beijing University of Technology, Beijing, China.

further increase the energy consumption and cause the processing efficiency reduction of the server. Thus, it is necessary to design an incentive mechanism to convince UEs to shift their peak-time processing computations to non-peak time for energy saving and efficiency improvement.

To address this challenge, we design the computation offloading process as an aggregative game with the instantaneous load billing scheme. Aggregative games have been adopted to design optimal strategies in wireless communication recently [16, 17]. In an aggregative game, each player aims to minimize its cost through action scheduling. In an aggregative game, its optimization problem is coupled with the aggregated actions of all players. In [16], the authors adopted the aggregative game to model spectrum sharing in large-scale, heterogeneous, and dynamic networks. Meanwhile, by utilizing past channel access experience, they proposed an online learning algorithm to improve the utility of each user. In [18], the authors proposed a day-ahead electric vehicle charging scheduling based on an aggregative game model, and proved the existence and uniqueness of the Nash Equilibrium (NE). However, above works usually achieve the NE through time consuming iterations or with the help of global information. Thus, how to accelerate the iteration process with limited information presents a major challenge.

In this paper, we use aggregative game to study the computation offloading problem between multiple UEs and one server in MEC. Since the tasks offloaded from multiple UEs will be executed simultaneously by one MEC server, it is necessary to arrange the task processing order properly to improve the efficiency of the server. Thus, we design an aggregative game model with an instantaneous load billing scheme to motivate UEs to shift their peak-time executing computation to non-peak time. Moreover, the optimal computation offloading strategy can be obtained with limited neighbor information and only a few iterations. The main contributions of this work are summarized as follows.

- We design a two-stage computing offloading scheme. To minimize the task processing delay, each UE optimizes how much workload to be offloaded to the MEC server. To improve the task processing efficiency of the server, we arrange the processing order of offloading tasks by introducing an aggregative game with an instantaneous load billing mechanism.
- We propose a novel aggregative game theoretic framework to make UEs offload their tasks in order voluntarily. We adopt an instantaneous load billing scheme to motivate UEs to shift their peak-time executing computation to non-peak time.
- We achieve the optimal computation offloading strategy with limited neighbor decision information and only a few iterations.

Furthermore, we compare the proposed scheme with a global optimal technique and a cloud computing technique in terms of transmission and processing delay. Simulation results demonstrate the proposed scheme can reduce transmission and processing delay with limited neighbor information compared with these existing techniques.

The rest of this paper is organized as follows. Section 2 reviews recent research related to computation offloading and billing schemes. Section 3 introduces the system model for the two-stage computing offloading scheme in MEC. In Section 4, we present the aggregative game theoretic framework to schedule processing strategy of the offloaded computation, and accelerate the method to obtain the NE. Next, we present numerical results and discussions in Section 5. Finally, we draw conclusions and introduce our future works in Section 6.

2 Related Work

In this section, we review recent research works on computation offloading in subsection 2.1 and billing schemes in subsection 2.2.

2.1 Computation Offloading

Computation offloading has been a popular research topic since 2012, when it was introduced by Cisco [19]. In computation offloading, a node with limited computing resources can offload tasks to other nodes or servers nearby.

2.1.1 Offloading Decision Targets

Offloading decision-making plays a critical role in computation offloading, which has recently attracted an increasing amount of research efforts. In general, offloading decisions are made for various optimization targets [20–22], such as latency minimization, energy consumption minimization, and efficient resource utilization. In [20], the authors investigated the latency-minimization problem in a multi-user time-division multiple access mobile-edge computation offloading (MECO) system by considering a joint communication and computation resource allocation. In [21], the authors proposed a dynamic task offloading scheduling to investigate the tradeoff between energy consumption and execution delay for an MEC system. They formulated the scheduling problem into an average weighted sum of energy consumption and execution delay minimization problem of mobile devices. Based on the Lyapunov optimization method, they obtained the optimal scheduling. In [22], the authors studied optimal resource allocation for a multiuser MECO system to minimize the weighted sum mobile energy consumption under a constraint on computation latency. However, the aforementioned works usually assume that servers are capable of executing the offloaded tasks without either energy or efficiency cost, which is unrealistic and impractical.

2.1.2 Offloading Techniques

There are lots of techniques to obtain computation offloading strategies, including game theory [23–26], heuristic algorithms [27], etc. In [23], the authors investigated the problem of multi-user computation offloading, formulated the offloading decision process in a dynamic environment as a stochastic game, and proposed a multi-agent stochastic learning algorithm to reach Nash Equilibrium. In [27], the authors studied the cooperative partial computation offloading with Internet of Things. They designed an iterative heuristic MEC resource allocation algorithm to make the offloading decision dynamically. However, the above traditional techniques are usually complicated and need many iterations. Moreover, they may require a significant amount of environment information to obtain their desired strategies. Different from the above traditional techniques, in this paper, we propose a novel aggregative game based algorithm to obtain the optimal task offloading and processing strategy with limited information and only a few iterations. Finally, with the help of instantaneous load billing scheme, the algorithm can motivate UEs to shift their peak-time processing computations to non-peak time, which improves the task processing efficiency and reduces the energy consumption on the server.

2.2 Billing Mechanisms

Billing mechanism is of great importance in our strategy design to motivate UEs to consider the energy consumption and processing efficiency of the server. Billing mechanism has been widely used in the area of economic management. In [28], the authors utilized the load billing scheme to investigate a practical demand side management scenario where the selfish consumers compete to minimize their individual energy cost through scheduling their future energy consumption profiles. However, only few works adopted billing mechanism in computation offloading, especially instantaneous load billing mechanism. We introduce instantaneous load billing mechanism [28–30] to computation offloading. Instantaneous load billing was proposed to charge customers based on its instantaneous load in each time slot during the operation period. As a result, the consumers will be charged more if they consume more computing resources during peak time. To avoid congestion of the offloaded tasks on the server, we develop an instantaneous load billing mechanism to motivate UEs to shift their peak-time processing computations to off-peak time.

3 System Model

In this section, we detail our two-stage computing offloading scheme system model. Specifically, we first show that how each UE determines its offloading computation and local computation to minimize its task processing delay in Section 3.1. Then, we discuss the aggregative game to arrange the processing order of the offloaded computations in the MEC server in Section 3.2. At last, we introduce the adopted instantaneous executing computing billing scheme in Section 3.3.

3.1 The First Stage of Designed Computing Offloading Scheme

In this work, we propose a two-stage computing offloading scheme. In this scheme, the computing requirements from N UEs can be offloaded to one MEC server. Specifically, the set of UEs is denoted by $\mathcal{N} = \{1, 2, \cdots, N\}$. In addition, we assume that there is a communication scheme where immediate connected UEs can communicate with each other at a low cost. Since the MEC server possesses much stronger computing and storage capability (e.g., faster processing speed, larger storage capacity) than UEs, it can significantly reduce computation burdens on UEs.

To simplify the model, we assume that every task can be split into multiple subtasks [31]. Thus, in the first stage, by offloading some subtasks to the MEC server, UEs can save processing time and release their local computation burdens. We define the size of each task's input data as $q_n (n \in \mathcal{N})$, whose unit is bit. We assume that the input data size of each task is known in advance and the input data includes program codes and input parameters [32]. The task can be split into two subtasks whose input data sizes are q_n^l and q_n^e respectively. The subtask with size q_n^l is executed locally, while the subtask with size q_n^e is offloaded from the UE n to the MEC server. To split tasks arbitrarily at runtime in real setting, we can refer to code offloading methods in [33–35].

For each UE, it will determine its task offloading strategy $\mathbf{q}_n = (q_n^l, q_n^e)^T$ in order to minimize its total task processing time which is denoted by T_n. To obtain the expression of T_n, we next analyze related latency during the task processing.

To obtain the local execution time, we first define the computing ability of mobile device n in one CPU cycle as f_n^l, and the required CPU cycles per bit of UE n's task as c_n. In general, CPU cycles per bit can be computed by using the method in [32, 36]. That is, given a task's input data size q_n, we first compute the required CPU instructions of the task by analyzing its program code and input parameters. Then, we can get the number of CPU cycles required by each CPU instructions through inquiring relevant CPU architecture information. Based on the above information, we can compute the total CPU cycles of the task. Thus, the CPU cycles per bit of the task can be derived by dividing the total CPU cycles by the input data size. We can derive the local execution time as [37]

$$T_n^l = \frac{c_n q_n^l}{f_n^l}. \tag{1}$$

Then, we turn to analyze the transmission delay of offloaded subtask q_n^e, which is denoted by T_n^t. Before

the analysis, we define the channel bandwidth, transmit power, channel gain of the channel between UE n and the MEC server as w_n, m_n, g_n, respectively [38–40]. Moreover, we define the spectral density of noise power in each channel as N_0. Then, we consider a setup where all UEs transmit their subtasks q_n^e in the uplink (UL), which has enough UL channels. Thus, UEs do not need to choose UL channels. Now, the achievable UL rate in the channel between UE n and the MEC server is given by [41]

$$r_n = w_n \log_2 \left(1 + \frac{m_n g_n}{w_n N_0}\right). \tag{2}$$

Based on (2), we obtain the time duration of transmitting computation q_n^e from UE n to the MEC server as follow [2]

$$T_n^t = \frac{q_n^e}{r_n}. \tag{3}$$

Next, we analyze the time duration of task processing at the MEC server. We assume that the MEC server has multiple cores and parallel processing capabilities. Once the subtasks offloaded by all UEs (i.e. q_n^e) are received, the MEC server has sufficient computing ability to process these workload in a short time T_p which is determined in advance. For simplicity, in this paper, we do not distinguish multiple cores explicitly, but treat them as a unified core.

As a result, we illustrate the relationship among T_n^l, T_n^t and T_p in Fig. 3, and we can define the total task processing delay as the maximum of local processing time and offloading processing time

$$T_n = \max\ \{T_n^l,\ T_n^t + T_p\}. \tag{4}$$

Based on (4), UE n ($\forall\ n \in \mathcal{N}$) can minimize its total task processing time T_n by selecting its optimal offloading strategy $\mathbf{q}_n = (q_n^l, q_n^e)^T$. Mathematically, UE n can obtain its optimal strategy by solving the following problem with input data size constraint

$$\min_{\mathbf{q}_n}\ T_n \quad \text{s.t. } q_n^l + q_n^e = q_n. \tag{5}$$

To obtain the solution of the problem (5), we present the following lemma.

***Lemma** 1:* The optimal task offloading strategy $\mathbf{q}_n^* = (q_n^{l*}, q_n^{e*})^T$ is given by

$$q_n^{l*} = \begin{cases} \frac{f_n^l q_n + T_p f_n^l r_n}{r_n c_n + f_n^l}, & T_p < \frac{q_n c_n}{f_n^l}, \\ q_n, & T_p \geq \frac{q_n c_n}{f_n^l}. \end{cases} \tag{6}$$

$$q_n^{e*} = \begin{cases} q_n - \frac{f_n^l q_n + T_p f_n^l r_n}{r_n c_n + f_n^l}, & T_p < \frac{q_n c_n}{f_n^l}, \\ 0, & T_p \geq \frac{q_n c_n}{f_n^l}. \end{cases} \tag{7}$$

Proof: To solve the problem (5), we first transfer the problem to the following equivalent problem with the help of equations (1), (3), and (4).

$$\min_{q_n^l} \max\ \left\{\frac{c_n q_n^l}{f_n^l}, \frac{q_n - q_n^l}{r_n} + T_p\right\}. \tag{8}$$

Then, we can solve the problem (8) by determining a q_n^{l*} (i.e., the optimal q_n^l) to minimize the value of function $\max\{f_1(q_n^l), f_2(q_n^l)\}$ under the constraint $0 \leq q_n^l \leq q_n$, where $f_1(q_n^l) = \frac{c_n q_n^l}{f_n^l}$, $f_2(q_n^l) = \frac{q_n - q_n^l}{r_n} + T_p$. For ease of understanding, we plot the function $\max\{f_1, f_2\}$ in Fig. 1 and Fig. 2, and obtain the minimum point $(\frac{f_n^l q_n + T_p f_n^l r_n}{r_n c_n + f_n^l}, \frac{c_n q_n + c_n T_p r_n}{r_n c_n + f_n^l})$ which is also the intersection of f_1 and f_2. For easy expression, we define the abscissa value of the minimum point as a. Considering the constraint $0 \leq q_n^l \leq q_n$, the next demonstration is divided into two circumstances, i.e., $a < q_n$ (Fig. 1) and $a \geq q_n$ (Fig. 2). In Fig. 1, where $a < q_n$ (i.e., $T_p < \frac{q_n c_n}{f_n^l}$), we can achieve the minimum of $\max\{f_1, f_2\}$ when $q_n^l = a$. Thus we have $q_n^{l*} = a$. In Fig. 2, where $a \geq q_n$ (i.e., $T_p \geq \frac{q_n c_n}{f_n^l}$), the function $\max\{f_1, f_2\}$ is declining in the interval $(0, q_n)$. Thus, we can achieve the minimum of $\max\{f_1, f_2\}$ when $q_n^l = q_n$, which means $q_n^{l*} = q_n$.

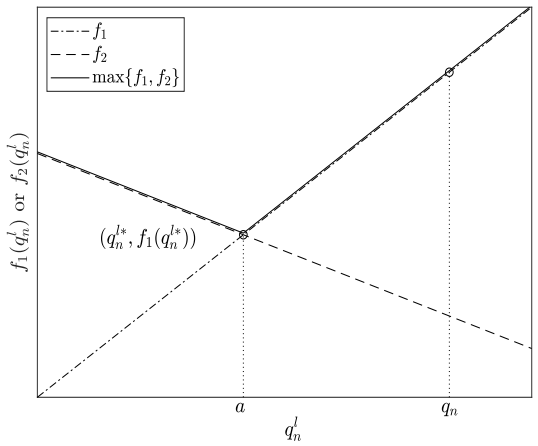

Fig. 1. The optimal task offloading strategy q_n^{l*} when $T_p < \frac{q_n c_n}{f_n^l}$.

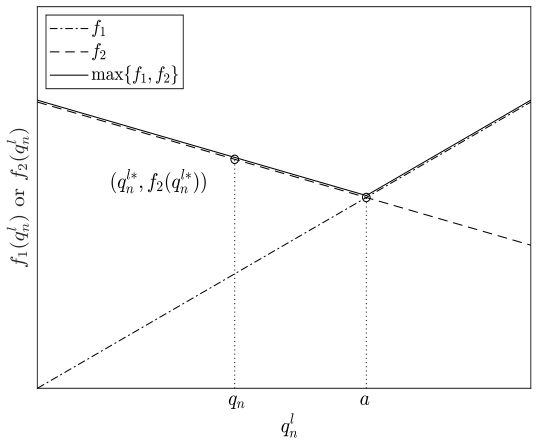

Fig. 2. The optimal task offloading strategy q_n^{l*} when $T_p \geq \frac{q_n c_n}{f_n^l}$.

Therefore, the optimal task offloading strategy $\mathbf{q}_n^* = (q_n^{l*}, q_n^{e*})^T$ can be given by equation (6). According to

the constraint $q_n^l + q_n^e = q_n$, we can further compute the optimal q_n^e as per equation (7). □

3.2 The Second Stage of Designed Computing Offloading Scheme

From the last subsection, we know that at a certain time point, an MEC server will receive many subtasks q_n^{e*} ($\forall n \in \mathcal{N}$) transmitted from UEs. Subsequently, it will process these subtasks in parallel within a future block time T_p, where T_p is determined in advance. However, within the block time T_p, if the computations processing arrangement is not proper, the efficiency of the MEC server will decrease. The reason is that, the disordering of offloading task execution means that a great many tasks may be processed in parallel in some periods. According to the analysis in [42], the larger the number of tasks that are concurrently executed, the longer the time it takes to execute each of the tasks on a server, which reduces the server efficiency. A numerical example is provided for demonstration. If three tasks are concurrently executed by a server, the effective computation rate decreases by 6% compared with one-task case. In addition, the power consumption of a server depends on not only computational devices such as a CPU, but also cooling devices. As the number of parallel tasks increases, the rotation speed of cooling devices (i.e. fans) increases in order to decrease the temperature of the server, which increases the power consumption of the server. Thus, in the second stage, to process these offloading subtasks efficiently within block time T_p, we propose a task split model to provide a possibility for arranging offloading computations to process in a proper order.

We first assume that time block T_p contains H time slots, and denote these time slots as $\mathcal{H} = \{1, 2, \cdots, H\}$. Each time slot can represent different timing horizons (e.g. one second) with different executing computation price. It means that each UE will be charged differently according to the time slot it chooses to process its offloading computation, and we will discuss the billing scheme in the next subsection. In this mechanism, each UE may split its offloading subtask into many computations, and allocate these computations into H time slots, so as to minimize its total offloading computation cost. Specifically, the UE n's offloading subtask q_n^{e*} can be split into H computations, and these computations will be processed in each time slot sequentially. We can formulate the executing computation profile as

$$\mathbf{x}_n = (x_n^1, x_n^2, \cdots, x_n^H)^T, \ \forall \ n \in \mathcal{N}, \tag{9}$$

where x_n^h ($h \in \mathcal{H}$) is UE n's computation processed by the MEC server in the h-th time slot. In addition, the computation profile in (9) is within a feasible set, which is given by

$$K_n = \{\mathbf{x}_n : \sum_{h=1}^{H} x_n^h = q_n^{e^*} \tag{10}$$

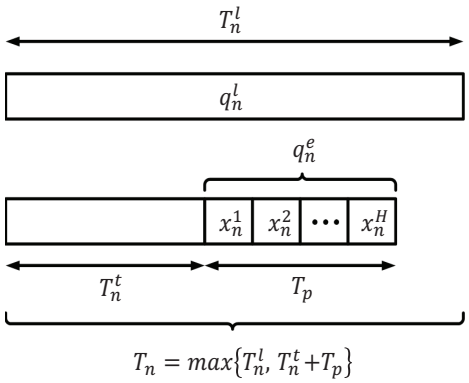

Fig. 3. The total delay of UE n, where $n \in \mathcal{N}$.

$$\text{and } x_n^h \leq S_h - \sum_{i=1}^{N} x_i^h \mathbb{I}_{i \neq n}, \forall h \in \mathcal{H}\},$$

where S_h is the upper limit of the MEC server's processable workload in time slot h, and is determined by the parallel computing ability of the MEC server. $\mathbb{I}_{i \neq n}$ is an indicator function, where $\mathbb{I}_{i \neq n} = 1$ if $i \neq n$ and $\mathbb{I}_{i \neq n} = 0$ if $i = n$. Then, the feasible computation set of all UEs can be expressed as

$$K = K_1 \times \cdots \times K_N. \tag{11}$$

To explain the proposed offloading task split model more clearly, we use Fig. 3 to show the executing computation in each time slot. In this figure, UE n determines to offload the subtask with size of q_n^e to the server, while the remaining subtask with size of q_n^l is executed locally. For the subtask executed locally, it takes T_n^l to finish executing. It takes $T_n^t + T_p$ to finish the offloading subtask, where T_n^t is transmission time to the server and T_p is processing time in the server. Then, the total delay for UE n to finish its task q_n is defined as $T_n = \max\{T_n^l, T_n^t + T_p\}$. In addition, the offloaded subtasks can be further divided into H computations with size of $x_n^1, x_n^2, \cdots, x_n^H$ respectively. Each of them is executed in different time slot.

3.3 Instantaneous Executing Computation Billing

Based on the computation split model proposed in the last subsection, we refer to the instantaneous load billing scheme [43–45] and design a fair charging scheme to incentive UEs to shift their peak-time executing computation to non-peak time. In this scheme, we assume that the executing computation price (the cost of one unit executing computation) of a certain time slot is set as an increasing and smooth function of the total demand in that time slot. Specifically, the executing computation price of the h-th ($h \in \mathcal{H}$) time slot is given by [46]

$$p_h = a_h(e^{b_h x_\Sigma^h} - 1), \tag{12}$$

where $x_\Sigma^h = \sum_{n=1}^{N} x_n^h$ is the total computation executed by the MEC server in time slot h, a_h denotes the linear

growth rate of the unit price, which is used for minor adjustment of unit price, and b_h denotes the exponential growth rate of the unit price, which is used for significant adjustment of unit price, note that a_h and b_h are for the control of the price increase range. We can find that, the price function in (12) can effectively convince UEs to shift their peak-time processing computations to non-peak time slots, since the increasing and convex price function ensures that the executing computation price will grow more rapidly as the aggregated processing demand increases. Therefore, the considered executing computation price model will improve the efficiency of the MEC server by flattening the overall processing computation demand curve.

Then, the UEs will be charged based on the price in (12) according to the amount of computation executed by the MEC server in each time slot. And the total offloading computation cost for each UE is the sum of the executing computation cost in each time slot, which is given by

$$B_n(\mathbf{x}_n, \mathbf{x}_\Sigma) = \sum_{h=1}^{H} p_h x_n^h, \tag{13}$$

where $\mathbf{x}_\Sigma = \sum_{i=1}^{N} \mathbf{x}_i$ denotes the aggregated computation profile of all UEs over future H time slots.

4 Executing Computation Scheduling Mechanism

In this section, we present an aggregative-game-based scheduling mechanism for the second stage of computation offloading, in order to improve the processing efficiency of the MEC server. We first introduce the aggregative game formulation in Section 4.1, based on which we propose our designed method with limited neighbor information to obtain the optimal scheduling strategy in Section 4.2.

4.1 Aggregative Game Formulation for Executing Computation Scheduling

After identifying the offloading strategy $\mathbf{q}_n^* = (q_n^{l*}, q_n^{e*})^T$ to achieve the minimal T_n for the UEs, the MEC server may receive many subtasks (i.e., $q_n^{e*}\ \forall n \in \mathcal{N}$) from these UEs. Then, the MEC server will charge UEs for executing their subtasks according to the instantaneous executing computing billing scheme. As a result, UEs spontaneously split their subtasks into computations, and shift these computations from the peak time slots to non-peak time slots. Thus, the processing efficiency of the MEC server can be significantly improved.

We assume that all UEs are selfish. It means that UE n ($\forall n \in \mathcal{N}$) aims to minimize its total offloading computation cost $B_n(\mathbf{x}_n, \mathbf{x}_\Sigma)$ by determining its own executing computation profile $\mathbf{x}_n$. Mathematically, this will involve UE n solving the following optimization problem

$$\min_{\mathbf{x}_n}\ B_n(\mathbf{x}_n, \mathbf{x}_\Sigma), \forall n \in \mathcal{N} \tag{14a}$$

$$\text{s.t.}\ \ \mathbf{x}_n \in K_n. \tag{14b}$$

According to the aggregative game theory, the optimization problem (14), which is coupled with the aggregation of executing computation of all UEs (i.e. $\mathbf{x}_n$), can be modeled by the following Nash equilibrium problem.

$$G = (\mathcal{N}, \{K_n\}_{n\in\mathcal{N}}, \{B_n(\mathbf{x}_n, \mathbf{x}_\Sigma)\}_{n\in\mathcal{N}}), \tag{15}$$

where $\mathcal{N}$ is the set of players (i.e. UEs), K_n is the set of strategies for player n, and the total offloading computation cost $B_n(\mathbf{x}_n, \mathbf{x}_\Sigma)$ is the cost function to be minimized by the player n.

To solve the problem G in (15), we turn to game theory and try to find the NE $\mathbf{x}^* = (\mathbf{x}_1^*, \cdots, \mathbf{x}_N^*)$ of G. Thus, we first introduce the important concept of NE.

***Definition** 1:* A strategy profile $\mathbf{x}^*$ is a NE of G if at the equilibrium $\mathbf{x}^*$, no UE can further reduce its cost function $B_n(\mathbf{x}_n, \mathbf{x}_\Sigma)$ by unilaterally changing its strategy, which means that

$$\begin{aligned} &B_n(\mathbf{x}_n^*, \mathbf{x}_\Sigma^*) \le B_n(\mathbf{x}_n, \mathbf{x}_\Sigma^*), \\ &\forall\ \mathbf{x}_n \in K_n, n \in \mathcal{N}. \end{aligned} \tag{16}$$

Before obtaining the NE, we need to prove the existence and uniqueness of NE. Thus, according to the game theory, we regard the following lemma as the game characteristics required for the existence of the NE.

***Lemma** 2:* For each $n \in \mathcal{N}$, set $K_n \in \mathbb{R}^H$ is compact and convex. Given a subset $K = K_1 \times \cdots \times K_N$ in $\mathbb{R}^{HN}$, each function $B_n(\mathbf{x}_n, \mathbf{x}_\Sigma)$ is continuously differentiable in $(\mathbf{x}_n, \mathbf{x}_\Sigma)$ over an open set containing set $K_n \times K$, while each function $\mathbf{x}_n \mapsto B_n(\mathbf{x}_n, \mathbf{x}_\Sigma)$ is convex over set K_n.

Proof: Based on equation (10), it is evident that the set $K_i \subset \mathbb{R}^H (\forall i \in \mathcal{N})$ is compact and convex for each fixed $\mathbf{x}_{-i}$. Then, according to (13), we can see that each function $B_i(\mathbf{x}_i, N\bar{\mathbf{x}})$ is continuously differentiable in $(\mathbf{x}_i, \bar{\mathbf{x}})$ over some open set containing the set $K_i \times \overline{K}$. Hence, we only need to prove the convexity of $B_i(\mathbf{x}_i, N\bar{\mathbf{x}})$ in $\mathbf{x}_i$ for every fixed $\mathbf{x}_{-i}$. This can be achieved by proving that the Hessian of $B_i(\mathbf{x}_i, N\bar{\mathbf{x}})$ is positive semidefinite [23]. After some algebraic manipulation, we have

$$\begin{aligned} &\triangle_{\mathbf{x}_i}^2 B_i(\mathbf{x}_i, N\bar{\mathbf{x}}) \\ =&diag[x_i^h p_h''(x_\Sigma^h) + 2p_h'(x_\Sigma^h)]_{h=1}^H. \end{aligned} \tag{17}$$

Note that (17) is a diagonal matrix with all diagonal elements being positive, the Hessian matrix of $B_i(\mathbf{x}_i, N\bar{\mathbf{x}})$ in $\mathbf{x}_i$ is positive semidefinite. □

Under Lemma 2, to proof the uniqueness of NE, we specify the solution to NE of the game G as a variational inequality (VI) problem [47] as follows.

***Definition** 2:* Given a subset $K = K_1 \times \cdots \times K_N$ in $\mathbb{R}^{HN}$ and a mapping $F : K \to \mathbb{R}^{HN}$, the VI problem, denoted by $VI(K, F)$, is to find a vector $\mathbf{x}^* \in K$ such that $(\mathbf{y} - \mathbf{x}^*)^T F(\mathbf{x}^*, \mathbf{x}_\Sigma^*) \ge 0, \forall \mathbf{y} \in K$, where $F(\mathbf{x}, \mathbf{x}_\Sigma)$ is

defined by

$$F(\mathbf{x}, \mathbf{x}_\Sigma) \triangleq \begin{bmatrix} F_1(\mathbf{x}_1, \mathbf{x}_\Sigma) \\ \vdots \\ F_N(\mathbf{x}_N, \mathbf{x}_\Sigma) \end{bmatrix}, \tag{18}$$

$$F_n(\mathbf{x}_i, \mathbf{x}_\Sigma) = \nabla_{\mathbf{x}_n} B_n(\mathbf{x}_n, \mathbf{x}_\Sigma) \text{ for all } n \in \mathcal{N}. \tag{19}$$

Following [48], $VI(K, F)$ admits a unique solution if the mapping $F(\mathbf{x}, \mathbf{x}_\Sigma)$ is strictly monotone over K since the feasible set K is compact and convex. Considering the monotonicity requirement of mapping $F(\mathbf{x}, \mathbf{x}_\Sigma)$, we have the following lemma.

***Lemma** 3:* If parameter b_h satisfies $0 < b_h < \frac{1}{S_h}$, $\forall h \in \mathcal{H}$, then the mapping $F(\mathbf{x}, \mathbf{x}_\Sigma)$ is strictly monotone over K, i.e.,

$$\sum_{h=1}^{H}\sum_{n=1}^{N}[(x_n^h - s_n^h)\big(\nabla_{x_n^h} B_n(\mathbf{x}_n, \mathbf{x}_\Sigma) - \nabla_{s_n^h} B_n(\mathbf{s}_n, \mathbf{s}_\Sigma)\big)] > 0, \text{for all } \mathbf{x}, \mathbf{s} \in K, \tag{20}$$

Proof: In order to keep the mapping $F(\mathbf{x}_n, \mathbf{x}_\Sigma)$ strictly monotone, we try to make the monotone constraint in (20) hold by adjusting parameters a_h and b_h. According to [28], we know that it is equivalent to make the Jacobian matrix of $g_h(\mathbf{x}^h)$ positive definite, where $\mathbf{x}^h = (x_1^h, \cdots, x_N^h)^T$ and $g_h(\mathbf{x}^h) = \nabla_{\mathbf{x}^h} p_h(x_\Sigma^h) x_n^h = (\nabla_{x_1^h} p_h(x_\Sigma^h) x_n^h, \cdots, \nabla_{x_N^h} p_h(x_\Sigma^h) x_n^h)^T$. Thus, we compute the (n, m)-th entry of the Jacobian matrix $G_h(\mathbf{x}^h) = \nabla_{\mathbf{x}^h} g_h(\mathbf{x}^h)$ as follows

$$[G_h(\mathbf{x}^h)]_{n,m} = \begin{cases} e_h[2 + b_h x_n^h], & \text{if } n = m, \\ e_h[1 + b_h x_n^h], & \text{if } n \neq m, \end{cases} \tag{21}$$

where, $e_h = a_h b_h \exp[b_h x_\Sigma^h]$.

Since the matrix $G_h(\mathbf{x}^h)$ may not be symmetric, we can prove its positive definiteness by showing that the symmetric matrix

$$G_h(\mathbf{x}^h) + G_h(\mathbf{x}^h)^T = e_h(\mathbf{z}^h \mathbf{1}^T + \mathbf{1}(\mathbf{z}^h)^T + 2\mathcal{I}) \tag{22}$$

is positive, where $\mathbf{z}^h = \mathbf{1} + b_h \mathbf{x}^h$. This is equivalent to showing that the smallest eigenvalue of this matrix is positive.

Note that being the sum of the outer product of two vectors and its transpose, the symmetric matrix $\mathcal{T}_b = \mathbf{z}^h \mathbf{1}^T + \mathbf{1}(\mathbf{z}^h)^T$ can have rank at most 2, and its rank is precisely 2 if $\mathbf{z}^h$ is not a multiple of $\mathbf{1}$. For the case when the rank is 1, the only nonzero eigenvalue of $\mathcal{T}_b$ is its trace, which is clearly positive. We henceforth assume that $\mathbf{z}^h$ and $\mathbf{1}$ are linearly independent.

Since $\mathcal{T}_b$ is symmetric, the eigenvectors corresponding to its two nonzero eigenvalues have to be linear combinations of its columns, and therefore have the form $\mu \mathbf{z}^h + \theta \mathbf{1}$, where μ and θ are constants. Hence, if y is an eigenvalue, we have the equation

$$(\mathbf{z}^h \mathbf{1}^T + \mathbf{1}(\mathbf{z}^h)^T)(\mu \mathbf{z}^h + \theta \mathbf{1}) = y(\mu \mathbf{z}^h + \theta \mathbf{1}), \quad \exists\ \mu \text{ and } \theta. \tag{23}$$

In order to obtain the solution of (23) better, we rewrite (23) as follows

$$\mathbf{z}^h[\mu(\mathbf{1}^T \mathbf{z}^h - y) + N\theta] + \mathbf{1}[\mu(\mathbf{z}^h)^T \mathbf{z}^h + \theta((\mathbf{z}^h)^T \mathbf{1} - y)] = \mathbf{0}. \tag{24}$$

Since $\mathbf{z}^h$ and $\mathbf{1}$ are linearly independent, their coefficients in the equation (24) should separately vanish, yielding the pair of equations as below

$$\mu(\mathbf{1}^T \mathbf{z}^h - y) + N\theta = 0, \tag{25}$$

$$\mu(\mathbf{z}^h)^T \mathbf{z}^h + \theta((\mathbf{z}^h)^T \mathbf{1} - y) = 0. \tag{26}$$

Considering the existence of a nontrivial solution for μ and θ, the determinant of their coefficient matrix in the above set of equations has to be zero, i.e., $(\mathbf{1}^T \mathbf{z}^h - y)^2 - N(\mathbf{z}^h)^T \mathbf{z}^h = 0$. This is a quadratic equation in terms of the unknown eigenvalues for the matrix $\mathcal{T}_b$ in (22), and its solutions are

$$\eta_{\mathcal{T}_b}^1 = N + b_h x_\Sigma^h + \sqrt{N(\mathbf{z}^h)^T \mathbf{z}^h}, \tag{27}$$

$$\eta_{\mathcal{T}_b}^2 = N + b_h x_\Sigma^h - \sqrt{N(\mathbf{z}^h)^T \mathbf{z}^h}. \tag{28}$$

Since $\eta_{\mathcal{T}_b}^1 \geq \eta_{\mathcal{T}_b}^2$, the smallest eigenvalue of the matrix $G_h(\mathbf{x}^h) + G_h(\mathbf{x}^h)^T$ can be expressed as

$$\eta_{min} = e_h \min(N + b_h x_\Sigma^h - \sqrt{N(\mathbf{z}^h)^T \mathbf{z}^h} + 2, 2), \tag{29}$$

where the second term 2 in the function (29) arises because the matrix $\mathcal{T}_b$ contains $N - 2$ zero eigenvalues in.

To further simplify (29), we have

$$N(\mathbf{z}^h)^T \mathbf{z}^h = N[\sum_{n=1}^{N}(1 + b_h x_n^h)^2] \tag{30}$$

$$= N[N + b_h^2 \sum_{n=1}^{N}(x_n^h)^2 + 2 b_h x_\Sigma^h] \tag{31}$$

$$\leq N[N + b_h^2 (x_\Sigma^h)^2 + 2 b_h x_\Sigma^h] \tag{32}$$

$$= N[(1 + b_h x_\Sigma^h)^2 - 1 + N] \tag{33}$$

$$\leq [(1 + b_h x_\Sigma^h)^2 - 1 + N]^2. \tag{34}$$

Substituting (30) into (29), we obtain

$$\eta_{min} \geq e_h \min(-b_h^2 (x_\Sigma^h)^2 - b_h x_\Sigma^h + 2, 2). \tag{35}$$

Since $e_h > 0$, we observe from the right hand side of (35) that $\eta_{min} > 0$ if $-b_h^2 (x_\Sigma^h)^2 - b_h x_\Sigma^h + 2 > 0$ for any $x_\Sigma^h \in [0, S_h]$. Considering that the quadratic function $\varphi(x_\Sigma^h) = -b_h^2 (x_\Sigma^h)^2 - b_h x_\Sigma^h + 2 > 0$ has two different zero points (i.e., $(-b_h)^2 - 4 \times (-b_h^2) \times 2 > 0$) and is a downward parabola with middle axis less than zero (i.e., $-\frac{-b_h}{2 \times (-b_h^2)} < 0$), we only need to keep $\varphi(S_h) > 0$ and $\varphi(0) > 0$. Thus, we a sufficient condition for the mapping $F(\mathbf{x}, \mathbf{x}_\Sigma)$ to be strictly monotone is given by

$$0 < b_h < \frac{1}{S_h}. \tag{36}$$

This completes the proof. □

According to [47], together with the compactness of K, Lemma 3 reveals the existence and uniqueness of a NE.

4.2 Algorithm to Obtain the Optimal Scheduling Strategy

Since the optimal problem in (14) is coupled with the aggregated executing computation of all UEs, we do not need to obtain the latest strategy of all the UEs after the UEs update their individual ones. We only need to obtain the latest information of the aggregated executing computation profile (i.e., $\mathbf{x}_\Sigma$). However, since there is no central unit to provide the UEs with correct $\mathbf{x}_\Sigma$, UEs may estimate $\mathbf{x}_\Sigma$ by exchanging their information with their immediate neighbors. Specifically, we denote the immediate neighbors of UE i as $\mathcal{N}_i$. For these setting, we develop a distributed algorithm with limited information, through which UEs are able to achieve the NE of game G.

Before introducing the algorithm, we first have the following assumption.

Assumption: The connection topology of the UEs is an undirected static graph.

Such an assumption is practical. For example, we can establish a virtual private network with the resources of cellular networks. Then the undirected static connection of UEs is feasible. We define UEs connected in the graph as immediate neighbors.

Based on the Assumption, we propose an neighbor communication model to model UEs' communication and exchange of the estimates for aggregate $\mathbf{x}_\Sigma$, which is inspired by the agreement protocol in [47]. In this model, we assume that a global clock holds. At each tick of the global clock, UEs may wake up according to some distribution (i.e. Poisson distribution). When an UEs wakes up, it contacts with its immediate neighbors. We use Z^k to denote the k-th tick time of the global clock whose total time slot is $[Z^0, Z^N)$. We discretize the global clock time so that instant k corresponds to the time slot $[Z^{k-1}, Z^k)$.

At time Z^k, assume that UE i $(\forall i \in \mathcal{N})$ wakes up and receives the estimated $\mathbf{v}_j^k$ from all the neighbors $j \in \mathcal{N}_i$. Then, it generates its intermediate estimate according to the following rule

$$\hat{\mathbf{v}}_i^k = w_{ii}(k)\mathbf{v}_i^k + \sum_{j\in\mathcal{N}_i} w_{ij}(k)\mathbf{v}_j^k,\ \forall i \in \mathcal{N}, \tag{37}$$

where $w_{ii}(k)$ is the nonnegative weight that UE i assigns to its own estimate, $w_{ij}(k)$ is the nonnegative weight that UE i assigns to UE j's estimate, and we set $w_{ii}(k) + \sum_{j\in\mathcal{N}_i} w_{ij}(k) = 1$.

By specifying $w_{ij}(k) = w_{ji}(k) = 0$ for $j \notin \mathcal{N}_i$ and $j \neq i$, we rewrite (37) as

$$\hat{\mathbf{v}}_i^k = \sum_{j=1}^{N} w_{ij}(k)\mathbf{v}_j^k, \tag{38}$$

$$\text{with } \mathbf{v}_j^0 = \mathbf{x}_j^0 \text{ for all } j \in \mathcal{N}, \text{for all } i \in \mathcal{N},$$

where $\mathbf{x}_j^0 \in K_j (j \in \mathcal{N})$ are initial random UE decisions. Based on (38), we derive the weight matrix $W(k)$ as follows

$$\begin{aligned} W(k) \in \mathcal{S} = &\{W \in \mathbb{R}^{NN} | W(k)\mathbf{1} = \mathbf{1}, \\ &\mathbf{1}^T W(k) = \mathbf{1}^T, w_{ij}(k) = w_{ji}(k) \\ &= 0 \text{ for } j \notin \mathcal{N}_i \text{ and } j \neq i\}, \end{aligned} \tag{39}$$

where $\mathbf{1}$ is an $N \times 1$ vector whose elements are all equal to one.

With its own iterate $\mathbf{x}_i^k$ and the average estimate $\hat{\mathbf{v}}_i^k$ in (38), UE i $(\forall i \in \mathcal{N})$ updates its iterate and average estimate according to the following rules

$$\mathbf{x}_i^{k+1} = \prod_{K_i}[\mathbf{x}_i^k - \alpha_{k,i} F_i(\mathbf{x}_i^k, N\hat{\mathbf{v}}_i^k)], \tag{40}$$

$$\mathbf{v}_i^{k+1} = \hat{\mathbf{v}}_i^k + \mathbf{x}_i^{k+1} - \mathbf{x}_i^k, \tag{41}$$

where $\alpha_{k,i}$ is the stepsize of UE i, $\prod_{K_i}$ denotes the Euclidean projection onto the set K_i and $F_i(\mathbf{x}_i^k, N\hat{\mathbf{v}}_i^k)$ is defined as

$$F_i(\mathbf{x}_i^k, N\hat{\mathbf{v}}_i^k) = \nabla_{\mathbf{x}_i^k} B_i(\mathbf{x}_i^k, N\hat{\mathbf{v}}_i^k). \tag{42}$$

Specifically, we have $\alpha_{k,i} = \frac{1}{k}$, where k denotes the number of updates that UE i has executed up to time k inclusively. The quantity $N\hat{\mathbf{v}}_i^k$ in (40) is the aggregate estimate that UE i uses instead of the true estimate $\sum_{i=1}^N \mathbf{x}_i^k$ of the UE decisions at time k.

According to [47], under stable conditions on UE's weight $W(k)$ (i.e., $W(k) \in \mathcal{S}$) and stepsize $\alpha_{k,i}$ (i.e., $\alpha_{k,i} = \frac{1}{k}$), the iterate matrix $\mathbf{x}^k$ can converge to the NE point $\mathbf{x}^*$.

We detail the algorithm for UE n's task split and offloading strategy in Algorithm 1. At the beginning, in order to minimize the total delay T_n, each UE determines its offloading strategy $\mathbf{q}_n^* = (q_n^{l*}, q_n^{e*})^T$ according to Lemma 1. Then, to improve processing efficiency, the server makes each UE to further divide its offloaded subtasks into H computations by instantaneous load billing scheme. As a result, UEs minimize their offloading computation cost in an aggregative game theoretic approach. Specifically, UEs wakes up according to Poisson distribution. If UE i wakes up at the k-th tick time, it contacts with immediate neighbors, and then obtains estimate of aggregated computation profile $\mathbf{x}_\Sigma$ (i.e. $N\hat{\mathbf{v}}_i^k$) by equation (38). At last, awake UEs update their iterates and estimates according to equations (40) and (41) until the variance of $\mathbf{x}^k$ is stable.

5 Numerical Results

In this section, we present the numerical results of the optimal scheduling strategy, based on which the impact of system parameters on the optimal scheduling strategy is examined and many useful insights are provided accordingly. According to the assumption that the connection topology of UEs is an undirected static graph,

Algorithm 1 Algorithm for UE n's Offloading and Task Split Strategy

Input: $q_n, c_n, f_n^l, r_n, T_p, \mathbf{x}^0$, and $W(k)$.
Output: $\mathbf{q}_n^*, \mathbf{x}_n^*$.
Steps:

1: **If** $T_p < \frac{q_n c_n}{f_n^l}$, **then** compute $q_n^{l*} = \frac{f_n^l q_n + T_p f_n^l r_n}{r_n c_n + f_n^l}$, $q_n^{e*} = q_n - \frac{f_n^l q_n + T_p f_n^l r_n}{r_n c_n + f_n^l}$ as the optimal offloading strategy $\mathbf{q}_n^* = ((q_n^0)^*, (q_n^{other})^*)$
2: **else** set $q_n^{l*} = q_n$, $q_n^{e*} = 0$
3: **end if**
4: Initiate $k = 0$, and set $\mathbf{v}^0 = \mathbf{x}^0$
5: **while** the variance of $\mathbf{x}^k$ is not stable **do**
6: At time k ($k \in \{1, 2, \cdots, N\}$), UEs wakes up according to Poisson distribution. If UE n wakes up at the k-th tick time, it communicates with its immediate neighbors and generates its intermediate estimate according to $\hat{\mathbf{v}}_n^k = \sum_{j=1}^N w_{nj}(k)\mathbf{v}_j^k \ \forall n \in \mathcal{N}$.
7: UE n updates its $(k+1)$-th iterate and average estimate according to
$\mathbf{x}_n^{k+1} = \prod_{K_n}[\mathbf{x}_n^k - \alpha_{k,n} F_n(\mathbf{x}_n^k, N\hat{\mathbf{v}}_n^k)]$,
$\mathbf{v}_n^{k+1} = \hat{\mathbf{v}}_n^k + \mathbf{x}_n^{k+1} - \mathbf{x}_n^k$.
8: $k = k + 1$.
9: **end while**

we randomly generate an undirected static graph with multiple UEs. Then, we number all UEs and define UEs connected directly as immediate neighbors. In addition, we believe that the termination criterion is satisfied when the total offloading computation cost for each UE $B_n(\mathbf{x}_n, \mathbf{x}_\Sigma)$ is almost unchanged. Without other statements, we detail values or ranges of used parameters in Table 1 [32, 37].

TABLE 1
Parameter Setting.

Parameters	Values
T_p	0.02sec
c_n	$100 \sim 1000$cycles/bit
f_s	100GHz
f_n^l	$50 \sim 100$GHz
$w_n N_0$	-25dbm
w_n	500MHz
m_n	$100 \sim 300$mW
q_n	$10 \sim 30$Mbit
g_n	-30db

To examine the performance of our aggregative-game-based scheduling mechanism for offloading computations, referred to as AGG, we first discuss the baselines for comparison. We choose the following representative task allocation schemes from recent literature as baselines.

- Congestion-game-based Edge Computing Task Allocation (COG): A congestion game based edge computing task allocation scheme minimizes the total cost of all UEs by information sharing among UEs [49]. That is, the UE has all information about others at any time.
- Centralized Server-based Allocation (CSA): It is a centralized task allocation scheme where each UE sends all their computation tasks to the server [50].

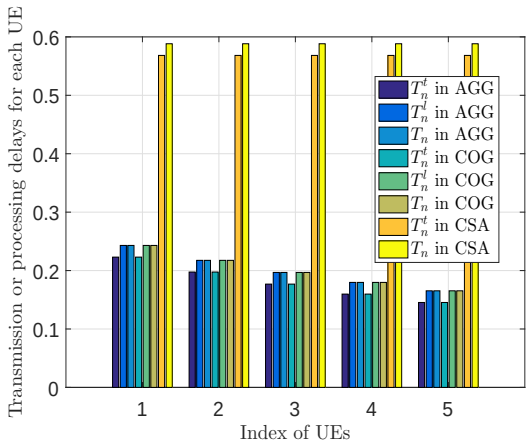

Fig. 4. Transmission or processing delays for each UE, where $N = 5$, $q_n = 20$ for all UEs, $f_1^l = 50$, $f_2^l = 60$, $f_3^l = 70$, $f_4^l = 80$, $f_5^l = 90$.

5.1 Numerical Results When $N = 5$

In this subsection, we present numerical results for N = 5 to show the detailed results of each UE with heterogeneous computing abilities.

5.1.1 Delay Comparison

In Fig. 4, we focus on how much delay is reduced by AGG compared to COG and CSA. Specifically, we evaluate the transmission delays, local processing delays and total task processing delays (i.e., T_n^t, T_n^l and T_n) for each UE under AGG, COG and CSA. Note that there is no local processing delay in CSA since each UE sends all their computation tasks to the server. On the one hand, we observe that AGG can significantly reduce all delays by more than 60% compared with CSA, which demonstrates that computing offloading can reduce task processing delay and accelerates UEs' service responses. We can also observe that, $T_n^l = T_n^t + T_p = T_n$ in AGG, which is reasonable considering expressions in (4) and (5). In order to minimize T_n, the UE n has to make two parallel delays (i.e., local processing delay T_n^l, transmission and server processing delay $T_p + T_n^t$) equal. On the other hand, we notice that T_n^t, T_n^l and T_n of AGG are almost the same as those of COG. This is due to the fact that we use the same task offloading strategy (i.e., $\mathbf{q}_n = (q_n^l, q_n^e)^T$) in AGG and COG to achieve the minimal total task processing delay. We also observe that, in AGG and COG, delays decrease with the increasing of local computing ability in one CPU cycle f_n^l. It is due to the fact that UEs with higher computing ability can process more tasks within a shorter time duration, which helps release the burden of server.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2020.2996764, IEEE Transactions on Services Computing

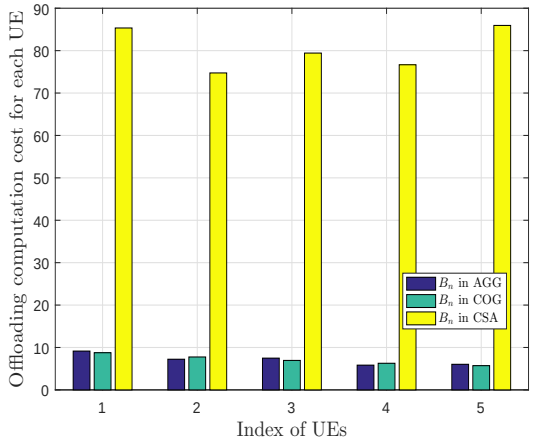

Fig. 5. Offloading computation cost for each UE, where $N = 5$, $H = 3$, $q_n = 20$ for all UEs, $f_1^l = 50$, $f_2^l = 60$, $f_3^l = 70$, $f_4^l = 80$, $f_5^l = 90$, $a_h = 10$, b_h=0.01 for all slots.

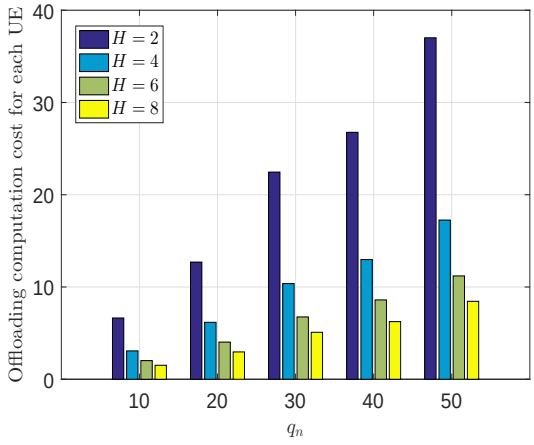

Fig. 6. Offloading computation cost for each UE, where $N = 5$, $f_n^l = 80$ for all UEs, $q_1 = 10$, $q_2 = 20$, $q_3 = 30$, $q_4 = 40$, $q_5 = 50$, $a_h = 10$, b_h=0.01 for all slots, iteration = 50.

5.1.2 Cost Analysis under Parameter Variation

Next, we try to examine the impact of system parameters on the cost of optimal scheduling strategy.

In Fig. 5, we examine the total offloading computation cost for each UE (i.e., $B_n(\mathbf{x}_n, \mathbf{x}_\Sigma)$) under AGG, COG and CSA. We observe that, compared with CSA which allocates offloaded tasks randomly among H time slots, AGG can significantly decrease $B_n(\mathbf{x}_n, \mathbf{x}_\Sigma)$ for each UE by nearly 90%. It can be explained by two reasons. First, in CSA, UEs offload all their tasks to the server, which causes heavy workload in the server and high execution cost. Second, in CSA, the task allocation among time slots is random, which may result in extremely high execution price in some time slots. Meanwhile, the figure shows that $B_n(\mathbf{x}_n, \mathbf{x}_\Sigma)$ under AGG and COG are almost the same. Actually, COG can be regarded as the optimal strategy since it can achieve minimal total cost of all UEs by information sharing among UEs. However, AGG only has aggregative information which is obtained by estimation. The result means that, AGG can achieve the minimal total cost of all UEs with incomplete information and limited number of iterations. Therefore, AGG is functionally equivalent but requires less information, and thus AGG is more practical. In addition, we notice that UEs with higher local computing ability in one CPU cycle f_n^l have less $B_n(\mathbf{x}_n, \mathbf{x}_\Sigma)$ in AGG and COG. It is because that UEs with higher f_n^l have less tasks to offload, thus they can pay less for execution in the server. However, in CSA, f_n^l has nothing to do with $B_n(\mathbf{x}_n, \mathbf{x}_\Sigma)$ since there is no task executed locally.

In Fig. 6, we set that there are 5 UEs with different offloading computations q_n, and we examine the effect of offloading computation differences and amount of slots H to the offloading computation cost for each UE. We can find that UEs with less q_n have less cost. Meanwhile, when q_n is same, UEs with smaller H have to pay more compared with those with larger H. It is because that when H is large, there would not be too many tasks allocated to the same slots, which reduces the price of each slots.

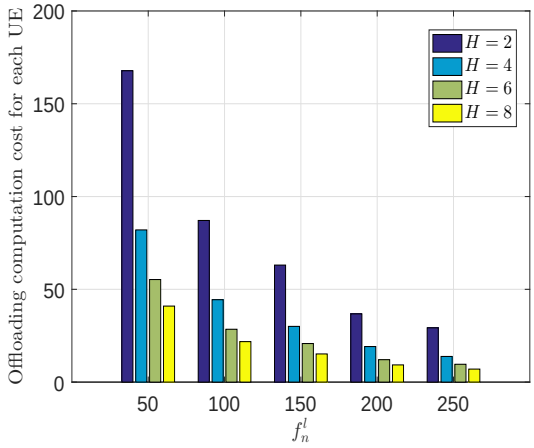

Fig. 7. Offloading computation cost for each UE, where $N = 5$, $q_n = 10$ for all UEs, $f_1^l = 50$, $f_2^l = 100$, $f_3^l = 150$, $f_4^l = 200$, $f_5^l = 250$, $a_h = 10$, b_h=0.01 for all slots, iterations = 50.

In Fig. 7, we set that there are 5 UEs with different local computing abilities f_n^l, and we examine the effect of local computing ability differences and amount of slots H to the offloading computation cost for each UE. We can find that UEs with stronger local computing ability have less cost. Meanwhile, when f_n^l is same, UEs with smaller H have to pay more compared with those with larger H, which is consistent with the analysis for Fig. 6.

In order to explain why AGG and COG can achieve minimal offloading computation cost, we use Fig. 8 to detail the offloading tasks allocation among each time slot. We can find that, although task allocation strategies under AGG, COG and CSA seem random, sum of computations in each time slot is same under AGG or COG. This guarantees even price in each time slot, which

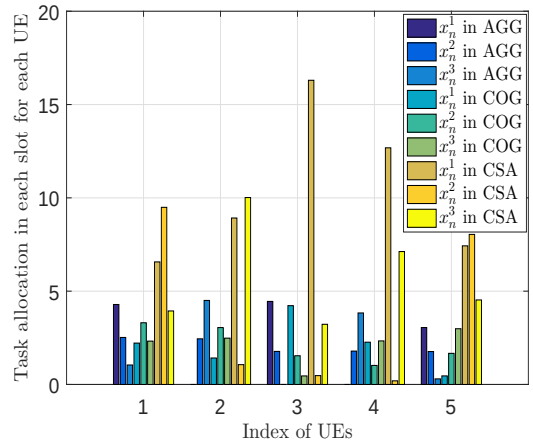

Fig. 8. Task allocation in each slot for each UE, where $N = 5$, $H = 3$, $q_n = 20$ for all UEs, $f_1^l = 50$, $f_2^l = 60$, $f_3^l = 70$, $f_4^l = 80$, $f_5^l = 90$, $a_h = 10$, b_h=0.01 for all slots.

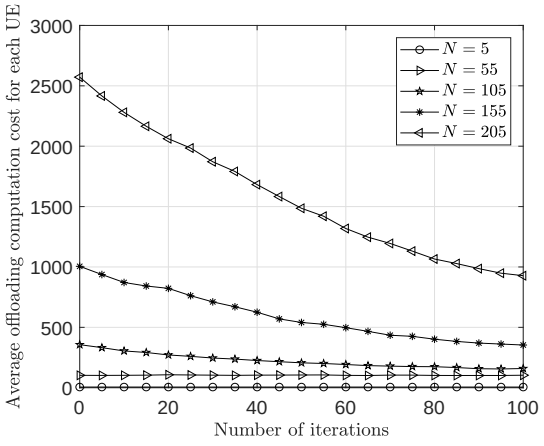

Fig. 10. Average offloading computation cost for each UE, where $H = 3$, $q_n = 20$, $f_n^l = 80$ for all UEs, $a_h = 10$, b_h=0.01 for all slots.

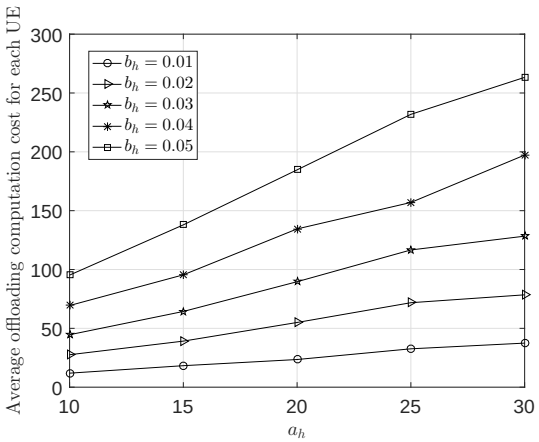

Fig. 9. Average offloading computation cost for each UE, where $N = 10$, $H = 3$, $q_n = 20$, $f_n^l = 80$ for all UEs, iterations=100.

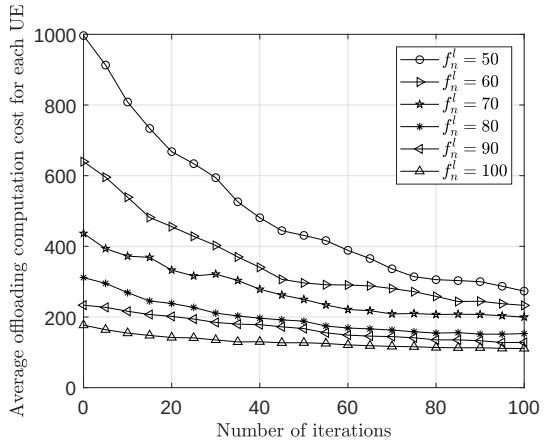

Fig. 11. Average offloading computation cost for each UE, where $N = 100$, $H = 3$, $q_n = 20$ for all UEs, $a_h = 10$, b_h=0.01 for all slots.

contributes to the minimal offloading computation cost.

In Fig. 9, we present the relationship between price parameters a_h, b_h and average offloading computation cost $\bar{B}_N$. We can observe that, $\bar{B}_N$ increases linearly with a_h or b_h, which is consistent with (12).

5.2 Numerical Analysis with Hundreds of UEs

5.2.1 Convergence Results

Next, we discuss the iterative convergence process of AGG in Fig. 10 and Fig. 11. In Fig. 10, we set N UEs with the same f_n^l. The figure presents how the average offloading computation cost $\bar{B}_N = \frac{1}{N}\sum_{n=1}^{N} B_n(\mathbf{x}_n, \mathbf{x}_\Sigma)$ changes with the growth of N. We can observe that, AGG has a fast convergence performance when N is small. When N gets larger, the algorithm can still converge within 100 iterations. In addition, we find that the average offloading computation cost increases with the growth of N. It is because the workload of server increases with N. The price of workload within each time slot also increases. In Fig. 11, we divide UEs into 6 groups. In each group, there are 100 UEs with the same f_n^l. It is shown that with the increasing number of iteration, the average offloading computation cost $\bar{B}_N$ decreases fast at the beginning, and then holds steady after 80 iterations. It further demonstrates that AGG has a fast convergence performance. In addition, when $\bar{B}_N$ maintains converging, we can find that $\bar{B}_N$ decreases with the growth of f_n^l. It suggests that, when f_n^l gets larger, UEs are willing to process more tasks locally, which consequently reduces the workload offloaded to the server.

5.2.2 Cost Analysis

In Fig. 12, we present the relationship between amount of UEs N and average offloading computation cost $\bar{B}_N$, as well as the relationship between the amount of time

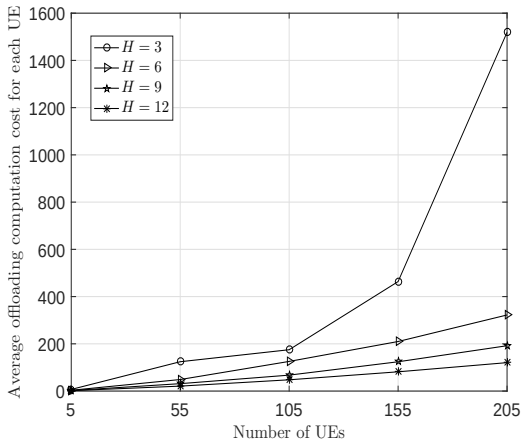

Fig. 12. Average offloading computation cost for each UE, where $q_n = 20$, $f_n^l = 70$ for all UEs, $a_h = 10$, b_h=0.01 for all slots, iterations=100.

slots H and $\bar{B}_N$. We can observe that, $\bar{B}_N$ increases exponentially with N. But when H is large, $\bar{B}_N$ increases almost linearly. The reason is that, according to the billing scheme in (12), the price is increasing exponentially with the growth of aggregated computation profile $\mathbf{x}_\Sigma$. When H is small, $\mathbf{x}_\Sigma$ is big, leading to a significant and exponential increase in $\bar{B}_N$. When H is large, $\mathbf{x}_\Sigma$ is small, leading to almost linear increase.

6 Conclusions

In this paper, we designed a two-stage computing offloading scheme to release computing burden for UEs. In the first stage, each UE approached its minimal task processing delay by determining how much workload to offload to the server. In the second stage, to improve the processing efficiency for the server, we used the aggregative game to motivate or force the UEs, who offloaded their subtasks to the server to shift their peak-time executing computation to non-peak time. In addition, to obtain the NE of aggregative game, we proposed a distributed algorithm, where the UEs communicated with their immediate neighbors to estimate the aggregative executing computation profile at each iteration. The algorithm was practical and fast-convergent. Simulation results showed that our scheme achieved the optimal offloading strategy that minimized the task processing delay for each UE while improving the processing efficiency of the server. Moreover, we analyzed the impact of model parameters on the offloading computation cost for each UE.

There are some interesting future works. Firstly, how to offload tasks from multiple UEs to multiple servers remains challenging. In this context, a matching algorithm between UEs and servers is required to associate each UE to a best server. Moreover, this work adopts the deterministic arrival model of tasks on the server. Driven by stochastic nature of task arrival, a stochastic arrival model is worthy of study. Secondly, how to design a computation offloading strategy among heterogeneous UEs deserves further investigation. In this context, we need to consider the heterogeneity of UEs, the dynamically changing topology of UEs, etc. Thirdly, there is a possibility that mobile devices have batch and online processing natures. In this context, costs for switching between tasks (e.g., memory storage) need to be considered.

References

[1] S. Guo, J. Liu, Y. Yang, B. Xiao, and Z. Li, "Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing," *IEEE Trans. Mobile Comput.*, vol. 18, pp. 319–333, Feb. 2019.

[2] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, pp. 2795–2808, Oct. 2016.

[3] M. Gao, J. Li, D. N. K. Jayakody, H. Chen, Y. Li, and J. Shi, "A super base station architecture for future ultra-dense cellular networks: Toward low latency and high energy efficiency," *IEEE Communications Magazine*, vol. 56, no. 6, pp. 35–41, Jun. 2018.

[4] H. Zhang, J. Gao, L. Yang, X. Li, and H. Ji, "Computation offloading considering fronthaul and backhaul in small-cell networks integrated with mec," in *IEEE Conference on Computer Communications Workshops*, Atlanta, GA, May 2017.

[5] S. Yu, R. Langar, X. Fu, L. Wang, and Z. Han, "Computation offloading with data caching enhancement for mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 67, pp. 11 098–11 112, Nov. 2018.

[6] D. Liu, L. Khoukhi, and A. Hafid, "Decentralized data offloading for mobile cloud computing based on game theory," in *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, 2017.

[7] H. Shah-Mansouri and V. W. S. Wong, "Hierarchical fog-cloud computing for iot systems: A computation offloading game," *IEEE Internet of Things Journal*, vol. 5, pp. 3246–3257, Aug. 2018.

[8] Y. Du, J. Li, L. Shi, T. Liu, F. Shu, and Z. Han, "Two-tier matching game in small cell networks for mobile edge computing," *Accepted by IEEE Transactions on Sevices Computing*, 2019.

[9] M. Hirsch, J. M. Rodriguez, A. Zunino, and C. Mateos, "Battery-aware centralized schedulers for cpu-bound jobs in mobile grids," *Pervasive and Mobile Computing*, vol. 29, pp. 73 – 94, Jul. 2016.

[10] C. Li and L. Li, "Exploiting composition of mobile devices for maximizing user QOS under energy constraints in mobile grid," *Information Sciences*, vol. 279, pp. 654 – 670, Sept. 2014.

[11] M. Hirsch, C. Mateos, and A. Zunino, "Augmenting computing capabilities at the edge by jointly exploiting mobile devices: A survey," *Future Generation Computer Systems*, Jun. 2018.

[12] S. N. Shirazi, A. Gouglidis, A. Farshad, and D. Hutchison, "The extended cloud: Review and analysis of mobile edge computing and fog from a security and resilience perspective," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2586 – 2595, Nov. 2017.

[13] P. P. Ray, "An introduction to dew computing: Definition, concept and implications," *IEEE Access*, vol. 6, pp. 723 – 737, Nov. 2017.

[14] M. Gusev, "A dew computing solution for iot streaming devices," in *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, May 2017.

[15] L. Liu, C. Zheng, X. Guo, S. Mao, and T. Ristaniemi, "Multiobjective optimization for computation offloading in fog computing," *IEEE Internet of Things Journal*, vol. 5, pp. 283 – 294, Dec. 2017.

[16] P. Zhou, W. Wei, K. Bian, D. O. Wu, Y. Hu, and Q. Wang, "Private and truthful aggregative game for large-scale spectrum sharing," *IEEE J. Sel. Areas Commun.*, vol. 35, pp. 463 – 477, Jan. 2017.

[17] Z. Han, D. Niyato, W. Saad, and T. Basar, *Game Theory for Next-Generation Wireless and Communication Networks: Modeling, Analysis, and Design*. Cambridge University Press, 2011.

[18] Z. Liu, Q. Wu, S. Huang, L. Wang, M. Shahidehpour, and Y. Xue, "Optimal day-ahead charging scheduling of electric vehicles through an aggregative game model," *IEEE Transactions on Smart Grid*, vol. 9, no. 5, pp. 5173 – 5184, Sept. 2018.

[19] D. Xu, Y. Li, X. Chen, J. Li, P. Hui, S. Chen, and J. Crowcroft, "A survey of opportunistic offloading," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 2198 – 2236, Thirdquarter 2018.

[20] J. Ren, G. Yu, Y. Cai, and Y. He, "Latency optimization for resource allocation in mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 17, no. 8, pp. 5506 – 5519, Aug. 2018.

[21] G. Zhang, W. Zhang, Y. Cao, D. Li, and L. Wang, "Energy-delay tradeoff for dynamic offloading in mobile-edge computing system with energy harvesting devices," *IEEE Trans Ind. Informat.*, vol. 14, no. 10, pp. 4642 – 4655, Oct. 2018.

[22] C. You, K. Huang, H. Chae, and B. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1397 – 1411, Mar. 2017.

[23] J. Zheng, Y. Cai, Y. Wu, and X. Shen, "Dynamic computation offloading for mobile cloud computing: A stochastic game-theoretic approach," *IEEE Trans. Mobile Comput.*, vol. 18, no. 4, pp. 771 – 786, Apr. 2019.

[24] M. Gao, R. Shen, S. Yan, J. Li, H. Guan, Y. Li, J. Shi, and Z. Han, "Heterogeneous computational resource allocation for c-ran: A contract-theoretic approach," *Accepted by IEEE Transactions on Sevices Computing*, 2019.

[25] T. Liu, J. Li, F. Shu, H. Guan, S. Yan, and D. N. K. Jayakody, "On the incentive mechanisms for commercial edge caching in 5g wireless networks," *IEEE Wireless Communications*, vol. 25, no. 3, pp. 72–78, Jun. 2018.

[26] T. Liu, J. Li, F. Shu, M. Tao, W. Chen, and Z. Han, "Design of contract-based trading mechanism for a small-cell caching system," *IEEE Trans. Wireless Commun.*, vol. 16, no. 10, pp. 6602–6617, Oct. 2017.

[27] Z. Ning, P. Dong, X. Kong, and F. Xia, "A cooperative partial computation offloading scheme for mobile edge computing enabled internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4804 – 4814, Jun. 2019.

[28] H. Chen, Y. Li, R. H. Y. Louie, and B. Vucetic, "Autonomous demand side management based on energy consumption scheduling and instantaneous load billing: An aggregative game approach," *IEEE Trans. Smart Grid*, vol. 5, no. 4, pp. 1744–1754, Jul. 2014.

[29] Z. Baharlouei, M. Hashemi, H. Narimani, and H. Mohsenian-Rad, "Achieving optimality and fairness in autonomous demand response: Benchmarks and billing mechanisms," *IEEE Trans. Smart Grid.*, vol. 4, pp. 968–975, Jun. 2013.

[30] I. Atzeni, L. G. Ordonez, G. Scutari, D. P. Palomar, and J. R. Fonollosa, "Demand-side management via distributed energy generation and storage optimization," *IEEE Trans. Smart Grid.*, vol. 4, pp. 866–876, Jun. 2013.

[31] B. D. Burge, R. N. Isaac, and J. Ueda, "Personal wireless network capabilities-based task portion distribution," 2011.

[32] C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang, "Computation offloading and resource allocation in wireless cellular networks with mobile edge computing," *IEEE Trans. Wireless. Commun.*, vol. 16, pp. 4924–4938, May 2017.

[33] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: A computation offloading framework for smartphones," in *Mobile Computing, Applications, and Services*, 2012.

[34] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," *Proc. EuroSys*, pp. 301 – 314, Apr. 2011.

[35] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. IEEE INFOCOM*, Mar. 2012.

[36] L. Yang, J. Cao, S. Tang, T. Li, and A. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," in *2012 IEEE Fifth International Conference on Cloud Computing*, Jun. 2012.

[37] C. Wang, F. R. Yu, C. Liang, Q. Chen, and L. Tang, "Joint computation offloading and interference management inwireless cellular networks with mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 66, pp. 2795–2808, Aug. 2017.

[38] D. Liang, "Spectral- and energy-efficient transmission with joint bandwidth assignment and transmit power allocation," in *Signal and Information Processing*, Washington, DC, USA, Dec. 2016.

[39] Y. K. Song, D. Kim, and J. Zander, "Pilot power adjustment for saving transmit power in pilot channel assisted ds-cdma mobile systems," *IEEE Trans. Wirel. Commun.*, vol. 9, no. 2, pp. 488–493, Feb. 2010.

[40] M. Sedighizad, H. G. Bafghi, and B. Seyfe, "Sensitivity of the secrecy capacity of a wiretap channel to the channel gains with imperfect channel information," in *Communication and Information Theory*, Tehran, Iran, May 2017, pp. 1–5.

[41] R. Zhang, "Throughput maximization in wireless powered communication networks with energy saving," *IEEE Trans. Wirel. Commun.*, vol. 2015-April, pp. 516–520, Apr. 2015.

[42] T. Enokido, A. Aikebaier, and M. Takizawa, "An extended simple power consumption model for selecting a server to perform computation type processes in digital ecosystems," *IEEE Trans Ind. Informat.*, vol. 10, no. 2, pp. 1627–1636, May 2014.

[43] Z. Baharlouei, M. Hashemi, H. Narimani, and H. Mohsenian-Rad, "Achieving optimality and fairness in autonomous demand response: Benchmarks and billing mechanisms," *IEEE Trans. Smart Grid*, vol. 4, pp. 968–975, Jun. 2013.

[44] I. Atzeni, L. G. Ordonez, D. P. Palomar, and J. R. Fonollosa, "Demand-side management via distributed energy generation and storage optimization," *IEEE Trans. Smart Grid*, vol. 4, pp. 886–876, Jun. 2013.

[45] I. Atzeni, L. G. Ordonez, G. Scutari, D. P. Palomar, and J. R. Fonollosa, "Noncooperative and cooperative optimization of distributed energy generation and storage in the demand-side of the smart grid," *IEEE Trans. Signal Process.*, vol. 61, no. 10, pp. 2454–2472, Feb. 2013.

[46] Z. Fan, "A distributed demand response algorithm and its application to phev charging in smart grids," *IEEE Trans. Smart Grid*, vol. 3, pp. 1280–1290, Sep. 2012.

[47] J. Koshal, "Distributed algorithms for networked multi-agent systems: Optimization and competition," *Dissertations & Theses - Gradworks*, 2012.

[48] F. Facchinei and J. S. Pang, *Finite-Dimensional Variational Inequalities and Complementarity Problems*. Springer, 2003.

[49] D. Liu, L. Khoukhi, and A. Hafid, "Decentralized data offloading for mobile cloud computing based on game theory," in *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, Valencia, Spain, May. 2017, pp. 20–24.

[50] D. Zhang, Y. Ma, C. Zheng, Y. Zhang, X. S. Hu, and D. Wang, "Cooperative-competitive task allocation in edge computing for delay-sensitive social sensing," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, Seattle, WA, USA, Oct. 2018, pp. 243–259.