Journal of Systems Architecture 137 (2023) 102847

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Stackelberg game-based task offloading and pricing with computing capacity constraint in mobile edge computing

Zhao Tong [a,*], Xin Deng [a], Jing Mei [a], Longbao Dai [a], Kenli Li [b], Keqin Li [c]

[a] College of Information Science and Engineering, Hunan Normal University, Changsha, 410012, China
[b] College of Information Science and Engineering, Hunan University, and National Supercomputing Center in Changsha, 410082, China
[c] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

ARTICLE INFO

Keywords:
Mobile edge computing
Pricing
PSO
Stackelberg game
Task offloading

ABSTRACT

The task offloading technology in mobile edge computing can improve the benefits of each device. Moreover, designing reasonable prices for computing resources is critical to balancing the interests of end servers and devices. However, unbalanced information availability complicates this issue. To address this problem, the Stackelberg game is introduced to describe the relationship between the MEC server and end users (EUs). Then, the task offloading decision problem of EUs is simplified and proved by derivation. Last, uniform pricing and differentiated pricing algorithms are proposed. The differentiated pricing strategy is proposed by the PSO-based pricing optimization algorithm (DPOA), which can precisely constrain the offloading of each EU. Simulation results show the DPOA effectively improve profit of the server, and reduce latency of EUs.

1. Introduction

With the booming development of the Internet of Things (IoT) and the widespread popularity of Fifth-Generation (5G) communication technology, the era of Internet of Everything is accelerating. IoT technology has been widely used in the domains of smart transportation, smart medical care, smart education and intelligent security, etc. [1,2]. According to estimates from the Cisco Internet Annual Report (2018–2023) white paper, the number of connected devices worldwide will increase to 29.3 billion by 2023, up from 18.4 billion in 2018 [3]. Simultaneous high-speed operation of hundreds of millions of devices is bound to generate enormous amounts of data and information [4]. Cloud computing has virtually unlimited capacity in terms of storage and processing power, which opens up possibilities for the implementation of IoT [5]. IoT devices request services from the cloud platform according to needs and pay for them on demand [6]. Moreover, end users (EUs, end devices in IoT such as smart watches, smart cars, real-time monitoring and UAVs, etc.) that implement the collection of data and send it to the MEC server, aim to obtain a relatively desirable Quality of Experience (QoE).

However, with the explosive growth of IoT devices and data, centralized cloud-based services are gradually revealing shortcomings in real-time, privacy protection, resource expense and reliability [7]. To mitigate these shortcomings of cloud computing, mobile edge computing (MEC) has been proposed as a new paradigm. MEC is a novel architecture that provides intelligent services at the edge of the network close to devices or data source [8]. Compared to cloud computing, MEC has the following merits: (1) It processes EUs' tasks on the edge side of the network, which relieves stress on remote cloud computing centers and network bandwidth. (2) The combination of MEC and 5G technology provides localized services to diverse end devices, improving the response time of enhanced services [9]. Nevertheless, the MEC system cannot possess as abundant computational capacity as the cloud computing system. A large number of EUs offload computation tasks to the MEC server, putting tremendous pressure on the limited wireless bandwidth and computing resources. The quality of task offloading directly affects the execution efficiency of the MEC system [10]. At the same time, it is a challenge for the MEC server to develop a reasonable pricing strategy and for EUs to determine the optimal amount of tasks to offload. It must take into account the heterogeneity of EUs and their task properties. For example, EUs may differ in computing abilities, the size of the generated tasks and their latency requirements. Additionally, EUs in the system may be selfish individuals, so they are only interested in optimizing their own performance. Thus, the development of task offloading and pricing in MEC systems should be carried out in accordance with individual interests and task requirements of heterogeneous devices. In this paper, we focus on the problem of task offloading and pricing in the MEC system where multiple EUs compete for resources of a single MEC server.

* Corresponding author.
E-mail addresses: tongzhao@hunnu.edu.cn (Z. Tong), 202120293782@hunnu.edu.cn (X. Deng), Meijing@hunnu.edu.cn (J. Mei), 202120293781@hunnu.edu.cn (L. Dai), lkl@hnu.edu.cn (K. Li), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.sysarc.2023.102847
Received 22 November 2022; Received in revised form 22 January 2023; Accepted 15 February 2023
Available online 20 February 2023

We adopt a Stackelberg game scheme to overcome such a challenge. Game theory is a useful approach for achieving satisfactory solutions for all devices in scenarios of a competitive nature. The Stackelberg game, as a leader–follower game, is appropriate for analyzing competitive relationship between the MEC server and multiple EUs, all of whom are only concerned with their own interests. Specifically, as the leader, the MEC server prices its own computational resources and sells them to EUs that need to offload tasks. The EUs with task latency constraints then act as followers to develop a reasonable task offloading strategy based on the price given by the leader. The MEC server seeks to maximize profits, and EUs pursue the smallest cost. The cost is the weighted sum of fees paid to the MEC server, energy consumption and latency. We build relationship between the price set by the MEC server and task offloading ratio decided by EUs. Then, a uniform pricing strategy and a differentiated pricing strategy are proposed to hunt for the optimal price made by the MEC server, separately.

In summary, the contributions of this paper are listed as follows:

- The task offloading problem of multiple heterogeneous EUs competing for limited computational resources from a single server is considered for a MEC scenario. Under the proposed two-layer resource management model, the system benefit is maximized by regulating the price to constrain task offloading.
- The principle between the MEC server and EUs is modeled as a Stackelberg game. In the task attributes, the response time tolerance as a delay constraint is added to influence the EUs' decision. The MEC server acts as a leader to decide unit price of the resource to maximize its revenue. Each EU that acts as a follower decides the size of tasks offloaded to the edge to minimize its cost.
- Through a series of derivations, the relationship between the pricing strategy of the MEC server and task offloading decisions of EUs is built. The PSO-based pricing optimization algorithm (DPOA) is proposed to resolve differentiated pricing scheme. The DPOA filters out pricing schemes that are highly profitable for the server.
- Extensive experiments are conducted to verify efficiency and performance of the proposed algorithm. Experimental results demonstrate superiority of the proposed DPOA in terms of increasing profitability of the MEC server, reducing latency of EUs, and increasing task offloading ratio.

The remainder of this paper is organized as follows. We present the related work of this paper in Section 2, then we introduce Stackelberg game theory and provide the problem definition of the model in Section 3. Section 4 gives the formulaic description of the problem. Section 5 gives solutions of the proposed Stackelberg game model. In Section 6, we provide the experimental parameter settings and performance evaluation of the experimental results. Finally, the conclusion is discussed in Section 7.

2. Related work

In this section, the objectives of task offloading and its current state of research in cloud computing and MEC are discussed. The innovations of the paper are demonstrated subsequently. A comparison of related work for task offloading is listed in Table 1.

2.1. Task offloading objectives

As a key technology of MEC, task offloading has become a prevalent subject among researchers. A great deal of research has focused on reducing system latency with the aim of improving the QoE and enhancing the user experience [11–14]. For example, Hu et al. [14] presented a deep reinforcement learning-based computation offloading method to mitigate task latency and energy consumption in dynamic MECs with large-scale EUs. In [15], Duan et al. considered mobile awareness and proposed an online task offloading strategy with adaptive load balancing in MEC systems. This approach is effective in reducing computing costs for users and achieving load balancing. Wu et al. [16] proposed an offloading strategy for vehicular networks under MEC systems. The greatest innovation is to consider the practical limitations of imperfect channels due to the high mobility of vehicles. Mao et al. [17] tackled the computational offloading problem caused by limited battery of the device through energy harvesting techniques. They proposed a dynamic computational offloading algorithm based on Lyapunov optimization, which not only lowered the execution cost but also reduced the task failures. Using the same method, Tong et al. [18] proposed Lyapunov online energy consumption optimization algorithm to balance the queue backlog and energy consumption of the system. Besides, from the economic point of view, Li et al. [19] proposed a partial offloading algorithm based on minimizing the cost of vehicles in vehicular networks. Du et al. [20] studied the maximization problem of the long-term average payoff of miners in an MEC blockchain system. They proposed a low-complexity algorithm based on the A3C deep reinforcement learning algorithm that boosts the total reward of miners.

The aforementioned investigations have good performance in solving task offloading and resource allocation problems. However, they do not consider either the impact of resource pricing or simultaneous optimization of the purposes of the MEC server and EUs.

2.2. Cloud-based task offloading strategies

To improve the flexibility of cloud resources, resource management in cloud networks requires robust pricing design to address many issues [31]. As a result, much research has been conducted on the issue of resource allocation and pricing strategies for cloud resources. Auction theory is a state-of-the-art approach focused on pricing problems [21,22,32]. To facilitate resource transactions between resource owners and mobile users, Wang et al. [22] constructed an auction model and subsequently proposed an effective payment evaluation mechanism to prevent the dishonest behavior of sellers in transactions. Kantere et al. [23] proposed a price-demand model and a dynamic pricing model to maximize profits and guarantee user satisfaction. Game theory is a popular method for solving resource allocation and pricing problems in cloud computing (CC) [24,25]. Liu et al. [24] formulated the request migration strategy between multiple servers as a noncooperative game between multiple servers in a distributed, noncooperative and competitive environment. The goal of the study was to minimize the disutility of servers. In [25], a multi-cloud intermediary framework for streaming big data computing was proposed. The pricing-repurchasing problem was modeled as a two-stage Stackelberg game to maximize the revenue of intermediaries.

2.3. MEC-based task offloading strategies

By comparison, resource allocation and pricing in mobile edge cloud environments is still in the nascent stage. Most studies, such as [11–14], assume that edge servers provide services for free. However, costs are needed for the deployment and maintenance of MEC servers. In addition, considering pricing of MEC servers is consistent with realistic scenarios. The distributed nature of game theory makes it a good match for multi-EU offloading situation [33]. In particular, the Stackelberg game has become a widespread method for resource allocation and pricing in wireless networks. To solve the resource allocation problem between multiple heterogeneous mobile edge clouds and EUs, Chen et al. [26] proposed a Stackelberg game framework to reconcile the interests of both sides of the problem, and then decomposed the problem into a set of subproblems. They concluded that an EU with spare resources can play the role of MEC. Liu et al. [27] investigated how to ensure effective power allocation during the task offloading under increasing intelligent jamming attacks. A Stackelberg defence mechanism based on deep neural networks is proposed. Simulation experiments demonstrate its effectiveness. Wang et al. [28] investigated a tiered dynamic pricing mechanism for cloud–edge-client cooperation with

Table 1
Overview of recent researches.

Ref.	Tech.	Obj.	Arch.	Adv.	Disad.
[11]	Pso, genetic algorithm, greedy algorithm	Average task response time	MEC	Achieving a lower average task response time	Not considering energy consumption
[12]	Block Coordinate Descent (BCD) method	Latency	MEC	Minimizing the average latency, linear computational complexity	Ignoring economic benefits
[13]	KKT conditions	End-to-end delay	MEC	Improving the QoE	Not considering monetary costs
[14]	Reinforcement learning	Latency, energy consumption	MEC	Suitable for large-scale user devices	Ignoring the limited computing capacity of MEC server
[15]	Reinforcement learning	Total computation costs	MEC	Considering the mobility of users	Ignoring complexity analysis
[16]	Deep reinforcement learning	The system cost	MEC	Considering the imperfect channel estimation	Weak energy model
[17]	Lyapunov optimization	The execution cost	MEC	Low-complexity, requiring little prior knowledge	Ignoring privacy assessments
[18]	Lyapunov optimization	Energy, delay	MEC	balance the queue backlog and energy consumption	Ignoring the cost of server
[19]	Mathematical derivation	Cost for vehicles	MEC	Enhancing revenue	Lack of time complexity analysis
[20]	Blockchain, A3C deep reinforcement learning	Rational total profit of miners	MEC	Converging fast	No privacy evaluation
[21]	Auction theory	Revenue of server	CC	Real test environment, near optimal profit	Ignoring network-related costs
[22]	Auction theory	Budget balance	MCC	Improving auction fairness and individual rationality	Ignoring latency requirements
[23]	MINLP	The cloud profit	CC	Simultaneously considering QoS and profit	Weak energy model
[24]	Non-cooperative game, variational inequality	The expected response times of all servers	CC	Converging to a Nash equilibrium quickly	Not considering energy consumption of servers
[25]	Stackelberg game	Revenue of the multiple cloud	CC	Analyzing the game equilibrium, low time complexity	Ignoring the time tolerance
[26]	Stackelberg game, KKT conditions	Revenue of both MECs and EUs	MEC	Demonstrating that an EU with idle resources can play the role of an MEC	Not considering the energy efficiency of EUs
[27]	Stackelberg game	Power allocation	MEC	Considering intelligent jamming attacks	Ignoring latency requirements
[28]	Stackelberg game	Utility of all participants	MEC	Improving QoE, reaching the maximum benefit equilibrium for servers	Ignoring latency requirements and time complexity analysis
[29]	Stackelberg game	Revenue, latency	MEC	Limited server computation resources	Ignoring edge cost model
[30]	Stackelberg game, differential evolution algorithm	Revenue, QoE	MEC	Considering cost of server, improving the profit of the MEC server	Not considering time tolerance

incomplete information. A two-tier Stackelberg game was introduced to describe cloud–edge-client collaboration. The approach improved the service quality of users and the benefits of service providers compared to traditional pricing schemes. However, the study did not consider system energy consumption. To address the problem of server pricing and user offloading for a single MEC server and multi-user cases, Liu et al. [29] investigated a price-based distributed approach where the edge server with limited computational power set prices to maximize its income, whereas each user made offloading decisions locally to minimize its own expenses. Nevertheless, they did not consider the cost in the expression of utility function of the MEC server. On the basis of [29], Tao et al. [30] proposed a pricing and offloading strategy based on the Stackelberg game. The optimal policy was then solved by a differential evolutionary algorithm. However, even though this work considered the latency of task local computation and offloading computation, they ignored the response time tolerance of each EU, which is given as a condition to constrain the task offloading of the EUs.

Based on the above related works, this paper models the resource pricing and task offloading problems under resource-limited edge servers as a Stackelberg game model. Compared with peer studies, this paper adds a delay constraint in task attributes to ensure QoE. The energy consumption of the MEC server and the integration overhead of the EU side are also considered comprehensively. Then, a PSO-based differentiated pricing algorithm is proposed to optimize the pricing of the server.

3. Stackelberg game and MEC model

In this section, the principles of the Stackelberg game are described first. Then, the scenario of the proposed model is defined.

3.1. Stackelberg game

Game theory is a powerful tool for studying distributed mechanisms that enables all EUs in a system to achieve mutually satisfactory solutions [34]. A complete game contains ingredients such as game players, game actions, game information, game order and outcomes. Among them, game players are rational participants that act in the game to maximize their own interests. Game actions present a set of executable action plans for players to choose [35]. Game information refers to the knowledge that players possess about the game, and game order describes the order in which players make their decisions. Outcomes

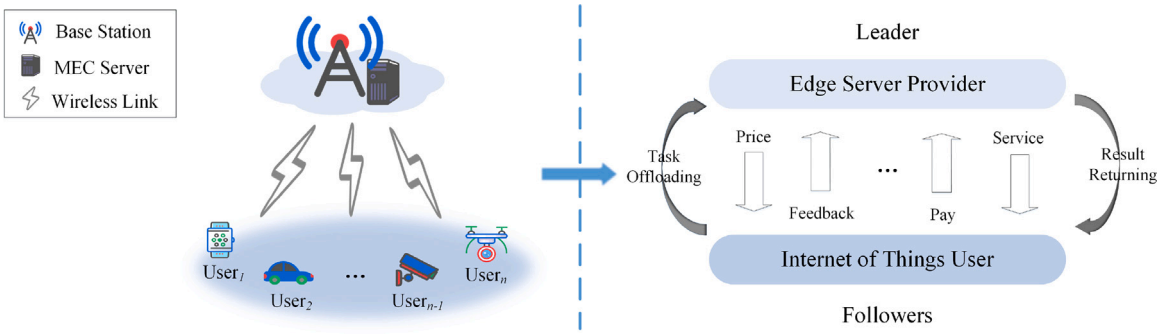


Fig. 1. Interaction architecture between the server and EUs in MEC system. Multiple EUs offload tasks to the server for computation and the results are returned to the EUs, where both sides play to maximize their benefits.

represent the gain or loss of each player at the end of the game. According to the relationship between players, games can be divided into cooperative games and non-cooperative games. Moreover, games can be grouped into static games and dynamic games with respect to the time series of the game. Because dynamic games carry a more complex theory than general static games, researchers have paid more attention to dynamic games.

The Stackelberg game is a typical dynamic game. Two types of players are involved: the leader, defines his strategy first, and followers, act after the leader's decision [36]. Taking the market as an example, position between competing manufacturers is asymmetric, leading to different decision sequences. The large manufacturer has more market information and makes decisions first, then small manufacturers make their own decisions by taking the behavior of the large manufacturer into account. Based on this qualification, we construct a Stackelberg game model for the single-server multi-user system such that all devices in the MEC system are in a normal game state.

3.2. MEC model

The overview of the single-server, multi-EU MEC system model is shown in Fig. 1. Considering that during one offloading cycle, a large number of IoT EUs request task offloading over a wireless link. The MEC server in the area accepts the requested computation task, and returns the results to the EUs after completing the computation. The task offloading style for EUs is the partial offloading, which means that the task can be arbitrarily split into two parts for execution. Part tasks are left locally to be calculated, another part tasks are transmitted to the MEC server.

The total channel bandwidth between the MEC server and EUs is B, and it can be shared evenly among EUs. Each EU has mutually non-overlapping computing frequencies to simultaneously offload task to the MEC server. The quasi-static channel model is considered for task offloading, i.e., the channel remains unchanged during an offloading cycle but may change over different offloading cycles. The frequency division multiple access (FDMA) technique is applied to divide the channel into multiple equally frequency sub-channels. It achieves channel sharing among EUs. Assume that the task offloading can be performed in one cycle. To illustrate, the symbols used in this paper are listed in Table 2.

Table 2
Definition of notations.

Symbol	Definition
B	Total transmission bandwidth
C_{edge}	Energy cost of the MEC server
E^i_{loc}	Energy consumption for EU i local computing
E^i_{off}	Energy consumption from sending task of EU i
F_{edge}	CPU cycles owned by the MEC server
N_0	Noise power spectrum density
f_{edge}	CPU computing frequency of the MEC server
f^i_{edge}	CPU computation frequency allocated to EU i
f^i_{loc}	Local CPU computing frequency of EU i
h_i	Channel gain between EU i and the MEC server
k_m	Power consumption factor in local computing
p_i	Transmitting power of EU i for offloading tasks
q_i	Data amount owned by the EU i
r_i	Transmission rate of EU i offloading tasks
t^i_{exe}	Execution energy consumption of the EU i
t^i_{loc}	Local computational latency of EU i
t^i_{off}	Uplink transmission delay of EU i
u_i	Price given to EU i by the MEC server
α_i	Task offloading strategy of EU i
τ_i	Task response time tolerance of EU i

4. Problem formulation

Under the above scenario, this section models the latency and energy consumption of local computation and task offloading to the MEC server separately. Then, the Stackelberg-based offloading model is proposed.

4.1. Local computing model

Assume that there are n EUs in this model, denoted as $N = \{1, 2, \ldots, n\}$. The task of EU i is characterized by the triple $A_i = (c_i, q_i, \tau_i)$, where c_i is the CPU cycles needed to compute 1 bit of data for EU i. Each EU is expected to execute q_i bits of data, and the latency tolerance of EU i for its own task execution is τ_i. The expense of EU i in local computing is the task local computation delay and the electricity consumed in this process. The CPU processing capacity (i.e., the number of CPU revolutions per unit time of the device) of EU i is f^i_{loc}.

Each EU i arbitrarily selects the amount of data to be offloaded to the MEC server according to the needs, and we note it as α_i, where $0 \le \alpha_i \le 1$. Therefore, the amount of data that EU i has to execute locally is $q_i(1-\alpha_i)$. In that case, the time expense that EU i spends in the local device is

$$t^i_{loc} = \frac{(1-\alpha_i)q_i c_i}{f^i_{loc}}. \tag{1}$$

The energy consumption generated in the local computing is expressed as

$$E^i_{loc} = k_m {f^i_{loc}}^2 (1-\alpha_i) q_i c_i, \tag{2}$$

where $k_m {f^i_{loc}}^2$ is the energy consumption generated by each CPU cycle of EU i and k_m is a power consumption factor determined by the chip structure, which is a constant.

4.2. Task offloading model

When an EU offloads its computational task to the MEC server, the cost (e.g., latency cost and energy cost) normally includes uplink transmission cost, offloading process cost and downlink transmission cost. The downlink transmission delay is negligible because result data is relatively small.

4.2.1. Uplink transmission process

EUs deliver their tasks to the MEC server for execution in proportion according to the task offloading strategy. The EU first transmits the task to the base station through the wireless channel. The base station further transmits them to the MEC server via a high-speed fiber. The total transmission bandwidth between the server and EUs is denoted as B, which can be equally divided among the EUs. Then, the task transmission rate for EU i is

$$r_i = \frac{B}{n}\log_2\left(1+\frac{p_i h_i}{BN_0}\right), \tag{3}$$

where p_i denotes the transmission power of EU i, h_i is the channel gain between EU i and the MEC server, and N_0 is the noise power spectrum density.

The uplink transmission delay can be expressed as

$$t^i_{off} = \frac{\alpha_i q_i}{r_i}. \tag{4}$$

Therefore, the energy consumption generated by EU i during uplink transmitting is obtained, defined as

$$E^i_{off} = \frac{\alpha_i q_i}{r_i} p_i. \tag{5}$$

4.2.2. Task execution process

The MEC server processes the computation tasks submitted by EUs after receiving them. The total computing resource held by the MEC server is denoted by f_{edge}, here the computing resource refers to the CPU computing frequency. For all EUs, the frequency of each EU i cannot be greater than the total frequency of the server, i.e. $\sum_{i=1}^{n} f^i_{edge} \le f_{edge}$. Note that f^i_{edge} denotes the computational frequency assigned by the MEC server to EU i. To simplify the problem, this paper considers an equal allocation of f_{edge}[11,29], and we have

$$f^i_{edge} = \frac{f_{edge}}{n}. \tag{6}$$

The latency of EU i during the execution of tasks at the edge server can be obtained, denoted as

$$t^i_{exe} = \frac{\alpha_i c_i q_i}{f^i_{edge}}. \tag{7}$$

Meanwhile, we consider a realistic condition that there is an upper bound on the number of CPU cycles available for the MEC server to compute in an offloading period. The sum of the data offloaded to the server by all EUs cannot exceed this threshold. It can be expressed as

$$\sum_{i=1}^{n} \alpha_i c_i q_i \le F_{edge}, \tag{8}$$

where F_{edge} is the CPU quantity of the MEC server for task offloading. Note that F_{edge} and f_{edge} indicate different values, and f_{edge} represents the CPU operating speed of the MEC server.

4.3. Two-stage Stackelberg game model

In the considered model, EUs purchase the computing resources of the MEC server to execute their tasks. The MEC server makes a profit with the assurance that the amount of data that the CPU needs to process does not exceed its computational quantity. To adjust this relationship between the supply and demand of resources, the MEC server adopts a price-control means to charge for the number of CPU cycles $\alpha_i q_i c_i$ required for the computational tasks of EU i.

The interactive decision process between the MEC server and EUs can be modeled as a two-stage Stackelberg game. The game model involves four components: the leader, followers, action strategy and utility function. The MEC server acts as the leader and EUs act as the followers. In the first stage, the MEC server (leader) announces the price of CPU cycles to each EU. In the second stage, EUs (followers) independently divide the task into two parts, local computation part and offloading computation part, based on the published prices. The action strategy consists of the price set and the task offloading strategy set. Assume that the pricing set of the MEC server is represented by $u = \{u_1, u_2, \dots, u_n\}$, and the task offloading strategy set of EUs is represented by $\alpha = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$. The utility functions of the MEC server and EUs reveal the satisfaction level of participants with their current action strategy and are defined as follows [37].

4.3.1. The utility function of the MEC server

The utility function of the MEC server is defined as the difference between the charge of the resources purchased by EUs and the cost of energy consumed by the server to execute the computation. We denote it as

$$U_{edge} = \sum_{i=1}^{n} u_i \alpha_i c_i q_i - \varepsilon \sum_{i=1}^{n} \eta t^i_{exe}, \tag{9}$$

where ε is the coefficient factor, which assigns the appropriate weight to the computation energy consumption. η is the impact factor of CPU cycles in the MEC system.

The MEC server wants to maximize its revenue by setting reasonable prices for selling the computation resources. Meanwhile, we consider the practical constraint of Eq. (8). Therefore, the optimal pricing u^* of MEC should satisfy

$$\begin{aligned} (\text{P1}): \quad & u^* = \arg\max U_{edge}(\alpha^*) \\ & \text{s.t.} \quad \sum_{i=1}^{n} \alpha_i c_i q_i \le F_{edge}, \end{aligned} \tag{10}$$

where α^* denotes the optimal task offloading strategy that is satisfied by all EUs.

4.3.2. The utility function of EUs

For EU i, the task offloading strategy α_i is actually the function of the price u_i. This is because the willingness of the EUs to offload the data to the edge depends on the prices assigned by the MEC server. In this model, we consider the minimum comprehensive costs of the EUs. The main costs of EU i during computational task offloading include latency cost, energy consumption cost and payment to the edge. The local execution of computational tasks and the MEC server processing of computational tasks can be performed simultaneously, so the latency cost for EU i is

$$t_i = \max\{t^i_{loc}, t^i_{off} + t^i_{exe}\}. \tag{11}$$

After normalizing and weighing the individual costs, the comprehensive cost of the EU i can be expressed as

$$U_i = \omega^t_i \eta^t_i t_i + \omega^e_i \eta^e_i (E^i_{loc} + E^i_{off}) + \omega^p_i \eta^p_i u_i \alpha_i c_i q_i, \tag{12}$$

where $0 < \eta^t_i, \eta^e_i, \eta^p_i, \omega^t_i, \omega^e_i, \omega^p_i < 1$. η^t_i, η^e_i and η^p_i are the normalized coefficients of delay cost, energy consumption cost and resource cost for each EU, respectively. ω^t_i and ω^e_i denote the adjustment factors of delay and energy consumption for EU i, and ω^p_i denotes the discount factor of resource pricing for each EU.

The adjustment of ω^t_i, ω^e_i and ω^p_i is flexible and can be adjusted according to the actual conditions of the equipment required by the application. For example, if the device has higher requirements for user experience, it may pay more attention to the latency cost and will adjust ω^t_i higher. If the device is at a low battery, it may pay more attention to energy consumption when developing the task offloading strategy. Therefore, a higher ω^e_i will be chosen. Additionally, if the device is under budgeted for resource costs, a higher ω^p_i will be picked.

Due to $t_i = \max\{t^i_{loc}, t^i_{off} + t^i_{exe}\}$, we can obtain

$$t_i = \begin{cases} \dfrac{(1-\alpha_i)q_i c_i}{f^i_{loc}}, & 0 \le \alpha_i \le \phi_i, \\ \dfrac{\alpha_i q_i}{r_i} + \dfrac{\alpha_i c_i q_i}{f^i_{edge}}, & \phi_i < \alpha_i \le 1, \end{cases} \tag{13}$$

where ϕ_i is expressed as

$$\phi_i = \frac{1}{1 + \frac{f_{loc}}{c_i r_i} + \frac{f^i_{loc}}{f^i_{edge}}}. \tag{14}$$

Substituting coefficients and the Eq. (13) into the Eq. (12), it can be equivalently expressed as

$$U_i = \begin{cases} \omega^t_i \eta^t_i \left(\dfrac{(1-\alpha_i)q_i c_i}{f^i_{loc}} \right) + \omega^p_i \eta^p_i u_i \alpha_i c_i q_i + \\ \omega^e_i \eta^e_i \left(k_m (f^i_{loc})^2 (1-\alpha_i) q_i c_i + \dfrac{\alpha_i q_i p_i}{r_i} \right), \\ \qquad\qquad 0 \le \alpha_i \le \phi_i, \\ \omega^t_i \eta^t_i \left(\dfrac{\alpha_i q_i}{r_i} + \dfrac{\alpha_i c_i q_i}{f^i_{edge}} \right) + \omega^p_i \eta^p_i u_i \alpha_i c_i q_i + \\ \omega^e_i \eta^e_i \left(k_m (f^i_{loc})^2 (1-\alpha_i) q_i c_i + \dfrac{\alpha_i q_i p_i}{r_i} \right), \\ \qquad\qquad \phi_i < \alpha_i \le 1. \end{cases} \tag{15}$$

Following the prices given by the MEC server, EUs prefer to minimize their comprehensive costs within the constraints of task latency through their own strategies. We denote it as

$$\begin{aligned} (\text{P2}): \quad & \alpha^*_i = \arg\min U_i(\alpha_i, u^*_i) \\ & \text{s.t.} \quad t_i \le \tau_i. \end{aligned} \tag{16}$$

It is worth noting that the resource cost terms in P1 and P2 are mutually exclusive. P1 and P2 in the model are coupled together in a complex way. That is, the pricing strategy of the MEC server affects the EUs' task offloading strategies. In turn, the EUs' offloading strategies also have an impact on the pricing strategy of the MEC server.

5. Solutions for two-stage stackelberg game

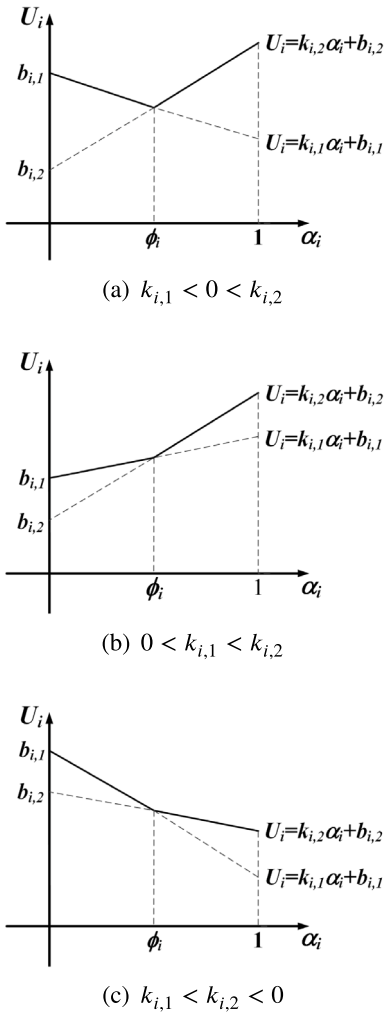
In this section, the range of optimal pricing for the MEC server is firstly derived. Then, the uniform pricing scheme and the differentiated pricing scheme are considered, separately.

5.1. Optimal pricing range

For a given price set, the optimal task offloading strategy α^* of the EUs can be obtained by solving P2. After the server receives the optimal response α^* from the EUs, it adjusts its strategy by solving P1 to obtain the optimal pricing scheme to maximize its profit. The process is called backward induction.

After the MEC server determines the resource price of EU i, the problem P2 of EU i can be deduced as a segmentation function related only to its own task offloading strategy α_i, i.e.

$$U_i = \begin{cases} \left(\dfrac{\omega^e_i \eta^e_i q_i p_i}{r_i} + \omega^p_i \eta^p_i u_i c_i q_i - \omega^e_i \eta^e_i k_m (f^i_{loc})^2 q_i c_i - \right. \\ \left. \dfrac{\omega^t_i \eta^t_i q_i c_i}{f^i_{loc}} \right) \alpha_i + \dfrac{\omega^t_i \eta^t_i q_i c_i}{f^i_{loc}} + \omega^e_i \eta^e_i k_m (f^i_{loc})^2 q_i c_i, \\ \qquad\qquad \max\{0, \beta_{i,1}\} \le \alpha_i \le \phi_i, \\ \left(\dfrac{\omega^e_i \eta^e_i q_i p_i}{r_i} + \omega^p_i \eta^p_i u_i c_i q_i - \omega^e_i \eta^e_i k_m (f^i_{loc})^2 q_i c_i + \right. \\ \left. \omega^t_i \eta^t_i \dfrac{q_i}{c^i} + \omega^t_i \eta^t_i \dfrac{c_i q_i}{f^i_{edge}} \right) \alpha_i + \omega^e_i \eta^e_i k_m (f^i_{loc})^2 q_i c_i, \\ \qquad\qquad \phi_i < \alpha_i \le \min\{\beta_{i,2}, 1\}, \end{cases} \tag{17}$$



(a) $k_{i,1} < 0 < k_{i,2}$

(b) $0 < k_{i,1} < k_{i,2}$

(c) $k_{i,1} < k_{i,2} < 0$

Fig. 2. U_i with respect to slope.

where $\beta_{i,1}$ and $\beta_{i,2}$ are expressed as

$$\beta_{i,1} = 1 - \frac{\tau_i f^i_{loc}}{q_i c_i} \tag{18}$$

and

$$\beta_{i,2} = \frac{\tau_i r_i f^i_{edge}}{q_i (f_{edge} + c_i r_i)}. \tag{19}$$

Based on the oblique equation of the line, Eq. (17) can be simplified as

$$U_i = \begin{cases} k_{i,1}\alpha_i + b_{i,1}, & \max\{0, \beta_{i,1}\} \le \alpha_i \le \phi_i, \\ k_{i,2}\alpha_i + b_{i,2}, & \phi_i < \alpha_i \le \min\{\beta_{i,2}, 1\}, \end{cases} \tag{20}$$

where $k_{i,1}$, $k_{i,2}$, $b_{i,1}$, $b_{i,2}$ are respectively expressed as the corresponding parts in Eq. (17) and satisfy $k_{i,2} > k_{i,1}$ and $b_{i,1} > b_{i,2}$.

Considering the resource price of EU i has been given, we classify U_i according to the slope of it and then discuss the optimal task offloading strategy for EU i. The diagram of U_i with respect to the slope is shown in Fig. 2.

As shown in Fig. 2(a), when $k_{i,1} < 0 < k_{i,2}$, i.e., the pricing of the MEC server satisfies

$$\mu_1 - \mu_2 - \mu_3 \left(\frac{1}{r_i c_i} + \frac{1}{f^i_{edge}} \right) \le u_i \le \mu_1 + \frac{\mu_3}{f^i_{loc}} - \mu_2, \tag{21}$$

where μ_1, μ_2 and μ_3 are expressed as

$$\mu_1 = \frac{w_i^e \eta_i^e k_m (f_{loc}^i)^2}{\omega_i^p \eta_i^p}, \tag{22}$$

$$\mu_2 = \frac{\omega_i^e \eta_i^e p_i}{r_i \omega_i^p \eta_i^p c_i}, \tag{23}$$

$$\mu_3 = \frac{\omega_i^t \eta_i^t}{\omega_i^p \eta_i^p}. \tag{24}$$

Respectively, let

$$\lambda_i^1 \triangleq \mu_1 - \mu_2 - \mu_3 \left(\frac{1}{r_i c_i} + \frac{1}{f_{edge}^i} \right), \tag{25}$$

$$\lambda_i^2 \triangleq \mu_1 + \frac{\mu_3}{f_{loc}^i} - \mu_2 \tag{26}$$

be the upper and lower thresholds of the price u_i. Eq. (21) can be expressed as $\lambda_i^1 \le u_i \le \lambda_i^2$. In this case, the optimal task offloading strategy α_i^* for each EU to minimize the comprehensive cost is ϕ_i. This situation shows that when the MEC server prices between the upper and lower thresholds, EUs will consider both local and edge computing capabilities. Then, each EU chooses the appropriate task offloading ratio to minimize its comprehensive cost.

As shown in Fig. 2(b), when $0 < k_{i,1} < k_{i,2}$, i.e., $u_i > \lambda_i^2$, EUs prefer to compute all data locally. Since the task response time tolerance is considered, the optimal task offloading policy for EU i is

$$\alpha_i^* = \max \left\{ 0, 1 - \frac{\tau_i f_{loc}^i}{q_i c_i} \right\}. \tag{27}$$

This indicates that when the published price of the MEC server is higher than a threshold, EUs are inclined to local computing considering the comprehensive cost. The revenue of MEC server will be greatly reduced in this case. While the server wants to maximize its own benefits, it will avoid setting prices smaller than this threshold.

As shown in Fig. 2(c), when $k_{i,1} < k_{i,2} < 0$, i.e., $u_i < \lambda_i^1$, EUs tend to offload all tasks to the edge side for execution. However, due to the response time tolerance, its task offloading strategy is

$$\min \left\{ \frac{\tau_i r_i f_{edge}^i}{q_i (f_{edge} + c_i r_i)}, 1 \right\}. \tag{28}$$

This case suggests that when the price of the MEC server is below a threshold, EUs will be attracted to offload all tasks to the edge for execution. Then a large number of EUs request computation simultaneously, making a slowdown of execution at the edge. EUs may put some of their tasks to be executed locally considering the execution latency. In addition, the MEC server sells all the computing resources at a low price. Such pricing is not cost-effective.

In particular, without considering the execution delay, when $k_{i,1} < 0 \le k_{i,2}$, we have $u_i = \lambda_i^1$, $\alpha_i^* \in (\phi_i^*, 1)$. Instead, when $k_{i,1} \le 0 < k_{i,2}$, we have $u_i = \lambda_i^2$, $\alpha_i^* \in (0, \phi_i^*)$. EUs have a selfish nature. When the cost is the same within a range of the amount of data offloaded to the server, they will offload as many tasks as possible to the MEC server for execution, i.e., take the right endpoint.

Based on the above analysis, the relationship between the optimal task offloading strategy of the EU i and the pricing strategy of the MEC server can be expressed as

$$\alpha_i = \begin{cases} \min \left\{ \dfrac{\tau_i r_i f_{edge}^i}{q_i (f_{edge} + c_i r_i)}, 1 \right\}, & u_i \le \lambda_i^1, \\ \phi_i, & \lambda_i^1 < u_i \le \lambda_i^2, \\ \max \left\{ 0, 1 - \dfrac{\tau_i f_{loc}^i}{q_i c_i} \right\}, & u_i > \lambda_i^2. \end{cases} \tag{29}$$

We simulate three cases of the server pricing. The result data show that the case $\lambda_i^1 < u_i \le \lambda_i^2$ has good superiority against the other two cases in terms of reducing latency, and improving the server's profit and offloading rates of EUs. Thus, the MEC server tends to explore uniform and differentiated pricing schemes based on this case.

5.2. Uniform pricing scheme

The uniform pricing scheme means the MEC server prices per unit of resource the same for each EU, denoted as u'. Obviously, $u_1 = u_2 = \cdots = u_n = u'$.

The MEC server sets the price at a middle value to keep the profit at a steady state. First, discard the values less than 0 in the set $R = \{\lambda_1^1, \lambda_2^1, \ldots, \lambda_n^1, \lambda_1^2, \lambda_2^2, \ldots, \lambda_n^2\}$ and only retain the values greater than 0, denoted as the set R^+. Then, the server obtains the average value of the elements in the set R^+, denoted as u^*, which is the server's strategy.

In practice, sellers usually give priority to buyers with high purchasing quantity. The MEC server receives EUs' task offloading strategies and sorts the EUs according to the size of the offloading radio. Then, the server sells resources to the EUs in descending order of the offloading ratio until all resources are sold out. The remaining EUs have an task offloading strategy of 0.

The interaction process between the MEC server and the EUs during uniform pricing can be expressed as the following steps. Firstly, with the basic information about the EUs, the MEC server obtains the set R^+ based on the expressions of λ_i^1 and λ_i^2. Secondly, the MEC server obtains the average value of the elements of the set R^+ and then broadcasts the price u^* to the EUs. Thirdly, the EU i decides the task offloading ratio α_i based on the given price u^* and conveys the set α to the MEC server. Then, the server receives the best responses from EUs and then sorts them in descending order based on the size of their task offloading ratios. Finally, the MEC server sells resources to EUs according to the descending order of the task offloading ratios and judges whether $\sum_{i=1}^{k} \alpha_i^* c_i q_i \le F_{edge}$ is satisfied. Here, k is the EU to whom the server is currently selling resources. If not satisfied, the MEC server ends the transaction. The task offloading ratio for the latter EUs is 0. The pseudocode for the entire process is presented in Algorithm 1.

From the procedure, lines 2 to 5 are numerical calculations. Lines 6 to 12 are performing loop statements to determine the offloading strategy for EUs, and n iterations are executed. Thus, the time complexity of the uniform pricing strategy is $O(n)$. The execution time of the algorithm is linearly related to the number of EUs.

Algorithm 1: Uniform Pricing Strategy

```
Input: Gaming environment parameters
Output: u* and α*
1  for episode = 1 to E do
2      Initialize A_i, ω_i^t, ω_i^e, ω_i^p, α and u;
3      Obtain set R = {λ_1^1, λ_2^1, ..., λ_n^1, λ_1^2, λ_2^2, ..., λ_n^2} from calculating
         n triples and environment parameters;
4      Get R+ by discard the values less than 0 in R, and derive u*;
5      Obtain task offloading strategy and sort EUs in descending
         order;
6      for i = 1 to n do
7          if Σ_{k=1}^{i} α_k^* c_k q_k ≤ F_edge then
8              Update task offloading strategy α_i^* = α_i;
9          else
10             Update task offloading strategy α_i^* = 0;
11         end
12     end
13 end
14 return u* and α*
```

5.3. Differentiated pricing scheme

Differentiated pricing means that the MEC server charges different fees for CPU resources for different EUs. PSO is an evolutionary search algorithm in which the particles maintain only two characteristics, position and velocity [38]. In PSO, position represents the direction of

motion. In the jth iteration, current position and velocity of the ith particle can be represented by X_i^j and V_i^j, respectively. Suppose the particle population size is $popSize$. In this model, position represents the optimal price of EU i found during the current iteration. The velocity indicates step length that particles move toward the optimal price.

The standard PSO is used to solve this problem. The position and velocity are adjusted based on the following principles

$$x_i^{j+1} = x_i^j + v_i^j, \tag{30}$$

$$v_i^{j+1} = \omega v_i^j + c_1 r_i^j (pbest_i^j - x_i^j) + c_2 r_2^j (gbest^j - x_i^j), \tag{31}$$

where ω is the inertia weight that regulates the searching range of the solution space. c_1 and c_2 are acceleration factors that regulate the maximum step size of learning. In addition, r_1 and r_2 are two random numbers, generated in the range of 0 to 1, increasing randomness of the search. All particles determine fitness value based on the fitness function to evaluate if the current position is good or bad. In this experiment, Eq. (9) works as the fitness function. The $pbest_i^j$ represents the best position experienced by particle i, and $pbest^j$ represents the best position experienced by all particles.

By the PSO, the differentiated pricing process by the MEC server for EUs can be summarized in the following steps. Firstly, randomly initialize the positions and velocities of the particle swarm. Secondly, calculate the fitness value of each particle based on the utility function of the MEC server. Thirdly, record the individual optimal position of each particle in each vector $pBest^{iter}$, and mark the position with the optimal adaptation value in $pBest^{iter}$ as $gBest^{iter}$. Then, update the position and velocity of each particle according to Eqs. (30) and (31). Finally, if iteration count is not reached, return to the second step. Otherwise, end the algorithm, at which point the vector $pbest^{iter}$ is the optimal pricing of the MEC server. The pseudocode for the above process is shown in Algorithm 2.

Next, we analyze the time complexity of the DPOA. In each time slice, all EUs perform the actions in parallel in lines 3–9 of Algorithm 2, which are mainly basic numerical computations. From line 10, the iteration of the algorithm starts. The number of iterations is $iterNum$ and the problem size is $sizePop$. The time complexity of the DPOA is $O(iterNum \times sizePop)$. Thus, the larger the $iterNum$ and $sizePop$, the larger the time complexity. In terms of complexity, the time complexity of the DPOA is greater than that of the uniform pricing strategy. This means that the accuracy of the pricing affects the time spent. Overall, the execution time of the DPOA is reasonable.

Algorithm 2: DPOA Pricing Strategy

Input: Gaming environment parameters; Particle population parameters
Output: $u*$ and $\alpha*$

1 Initialize $\omega_i^t, \omega_i^e, \omega_i^p$;
2 **for** *episode = 1 to e* **do**
3 Initialize A_i and pop^0;
4 Initialize α and u;
5 Adjust α under the constraint $\sum_{i=1}^{k} \alpha_i^* c_i q_i \le F_{edge}$;
6 Obtain the thresholds of velocity $popv^0$ and position $popx^0$ according to λ_i^1 and λ_i^2;
7 Set the initial position as the $pBest^0$ for each particle;
8 Calculate $pBest_i.Fitness$ of each particle;
9 Obtain the optimal position $gBest^0$;
10 **for** *iter = 1 to iterNum* **do**
11 **for** *i = 1 to sizePop* **do**
12 Dynamically update c_1 and c_2;
13 Calculate $popv^{iter}$ and $popx^{iter}$;
14 Judge and perform boundary processing;
15 **end**
16 Calculate $pop_i^{iter}.Fitness$ of each particle;
17 **for** *i = 1 to sizePop* **do**
18 **if** $pop_i^{iter}.Fitness > pBestvalue_i^{iter}$ **then**
19 Set $pBest_i^{iter} = popx_i^{iter}$;
20 **end**
21 **if** $pop_i^{iter}.Fitness > gBestvalue^{iter}$ **then**
22 Set $gBest = popx_i^{iter}$;
23 **end**
24 **end**
25 Calculate task offloading ratio and average task latency;
26 **end**
27 **end**
28 **return** $u^* = gBest$ and $\alpha^* = \alpha$

6. Performance evaluation

In this section, experiments are conducted to simulate the proposed DPOA. We evaluate the performance of the algorithm in terms of improving latency, enhancing server efficiency and improving task offloading rate, and make a comparison with the uniform pricing algorithm.

6.1. Experimental setup

The entire simulation is performed in MATLAB to emulate the task offloading process in the IoT scenario. The parameter values for the simulation experiments are mainly referred to [39,40]. There are $N = 60$ EUs with various computing tasks. The local CPU computing capability of EU i is selected uniformly from the set [0.1, 1] GHz, and the number of CPU cycles needed to execute 1 bit of data for EU i is uniformly distributed by the set [500, 1500] cycles/slot. The maximum CPU that the server can perform computation is 6×10^9 cycles/slot. Factors influencing the weight of the resource cost paid to the server, latency and energy consumption on the EUs' utilities can be adjusted according to practical needs. Specifically, the three weighting factors are assumed to be equal to 1/3. If not otherwise specified, the critical simulation parameters are listed in Table 3.

Table 3
Configurations of simulation parameters.

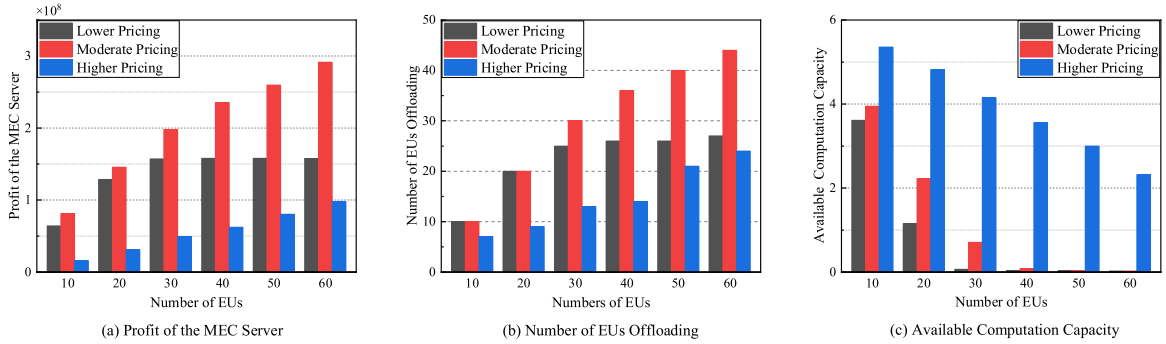
Parameters	Value
Iteration of e	200
Iteration of iterNum	70
The computing power of the server	100 GHz
The task size of EU i	[100, 500] KB
The total Channel Bandwidth	30 MHz
The electricity consumption factor	10^{-12}
The transmission power of EU i	0.1 W
The maximum tolerance time of EU i	[0.2, 1] s
The noise power spectrum density	10^{-8}
Unit energy consumption of the server	4 J/GHz

6.2. Result analysis

6.2.1. Experiments on pricing range selection

In the previous derivation, three pricing intervals are obtained, as presented in Eq. (29). To demonstrate that the pricing of the MEC server needs to be in a moderate range, a set of comparative experiments have been performed. As shown in Fig. 3, we define the three pricing ranges as "Lower Pricing", "Moderate Pricing" and "Higher Pricing". With a fixed number of EUs, the impact of the three pricing ranges on profit of the server, and the number of EUs offloading task is explored. Form the curves, growth trend of the MEC server's profit and EUs' offloading rate is generally consistent.

The effects of pricing ranges on the profitability of the MEC server is shown in Fig. 3(a). At the higher pricing scenario, the MEC server is the least profitable. As the number of EUs increases, the profit increases in an approximately linear trend. This is because the server has sufficient



(a) Profit of the MEC Server (b) Number of EUs Offloading (c) Available Computation Capacity

Fig. 3. Comparison of different pricing range.

computing capability to meet the demand of EUs that choose to offload, as shown in Fig. 3(c). For the case of lower pricing, the profit of the server rises initially as the number of EUs increases. However, the highest point is reached relatively quickly. As low prices attract EUs to offload all tasks, the CPU capacity of the MEC server is used up when EUs grow to 30. Profit does not increase with the number of EUs. With moderate pricing, profits for 10, 20 and 30 EUs increase faster than for 40, 50 and 60 EUs. This is because in the former case, the server's computing resources are sufficient. The demand for computing resources exceeds the supply for 40, 50 and 60 EUs. At the time, fierce competition makes profit still in a rising trend.

The changes in the number of EUs that choose to offload for different numbers of EUs in the three pricing scenarios are plotted in Fig. 3(b). All EUs choose to offload tasks at both low pricing and moderate pricing when the number of EUs is 10 and 20. At low pricing, the number of EUs performing offloading remains stable from 30 onward. This is because the EUs that arrive first have filled up the computing capacity, as shown in Fig. 3(c). Therefore, the later EUs can only execute tasks wholly locally. In the case of moderate pricing, EUs generally weigh the cost against latency and energy consumption, and perform a portion of the tasks locally. Thus more EUs can share computing resources of the MEC server.

In summary, if the MEC server sets the price too high, EUs are reluctant to offload tasks due to their limited budget. Considering the weak competition among EUs, the server also avoids selling resources at low prices. Therefore, it would stabilize the prices in a moderate range.

6.2.2. Experiments on PSO parameter selection

In the standard PSO algorithm, the learning factors are usually taken as a constant 2. To choose the optimal learning factors, we fix the inertia weight $\omega = 0.7298$, and explore the effect of dynamic changes of c_1 and c_2 on the algorithm. The dynamic selection strategies for both learning factors c_1 and c_2 are: increasing strategy of $c = 1 + \sin(t\pi/\max DT)$ and decreasing strategy of $c = 2 - \sin(t\pi/\max DT)$, where t represents the current number of iterations and $\max DT$ is the maximum number of iterations. A classical set of values with $c_1 = c_2 = 1.4962$ is taken with reference to the experiments.

The experiment is implemented in five groups. The detailed parameter settings and convergence trends of the curves are presented in Fig. 4. When c_1 is increasing with a fixed ω, the optimization-seeking accuracy of PSO improves. When other conditions are kept static and c_2 is ascending, the speed of PSO seeking improves. In addition, the curve of $c_1 = c_2 = 1.4962$ puts profit of the server in an intermediate state. The gray curve in Fig. 4 achieves convergence first and converges with high accuracy. This is because it has increasing c_1 and c_2. Therefore, in the later experiments, we will choose the parameters of the gray curve to attain better results.

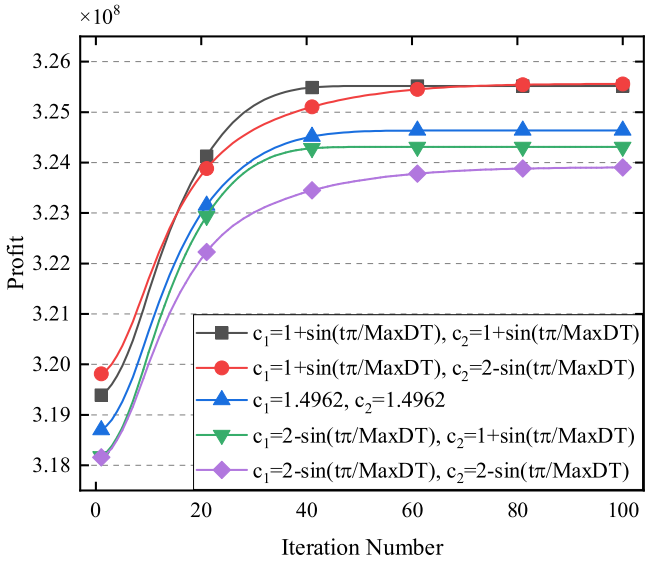


Fig. 4. The convergence of PSO.

6.2.3. Comparative experiments on the capacity of MEC

We perform comparison experiments on the proposed two pricing schemes regarding the computational capacity of the MEC server. The performance is compared with respect to profit of the server, average latency, task offloading rate of tasks and capacity utilization of the server, as shown in Fig. 5. The number of EUs in this experiment is fixed at 20.

Both uniform and differentiated pricing improve as the computing capacity of the server increases, as shown in Fig. 5(a). Differentiated pricing is more advantageous in terms of profit due to more precise resource allocation. The changes in average latency of the two pricing approaches as the server's computing capacity increases are illustrated in Fig. 5(b). Both curves first decrease and then gradually stabilize. The computational capacity of the MEC server is used up when the computational capacity is 1 and 2, as shown in Fig. 5(d). At this point, the computational capacity of the server is in short supply, so local computational delay is the main delay overhead. As the computational capacity increases, the server gradually meets EUs' demand. When the computational capacity is 4, there is a cliff-like drop in the latency of differentiated pricing. This is because the EUs have experienced a change in the resources from tight to sufficient. The average latency performance of the blue curve representing the uniform pricing strategy is inferior to that of the red curve. This is because uniform pricing fails to make full use of the server's resources. For comparison, we also considered the fully local computing scheme. Compared to the two proposed pricing schemes, fully local computing scheme has the worst performance, as the latency does not vary with the computational

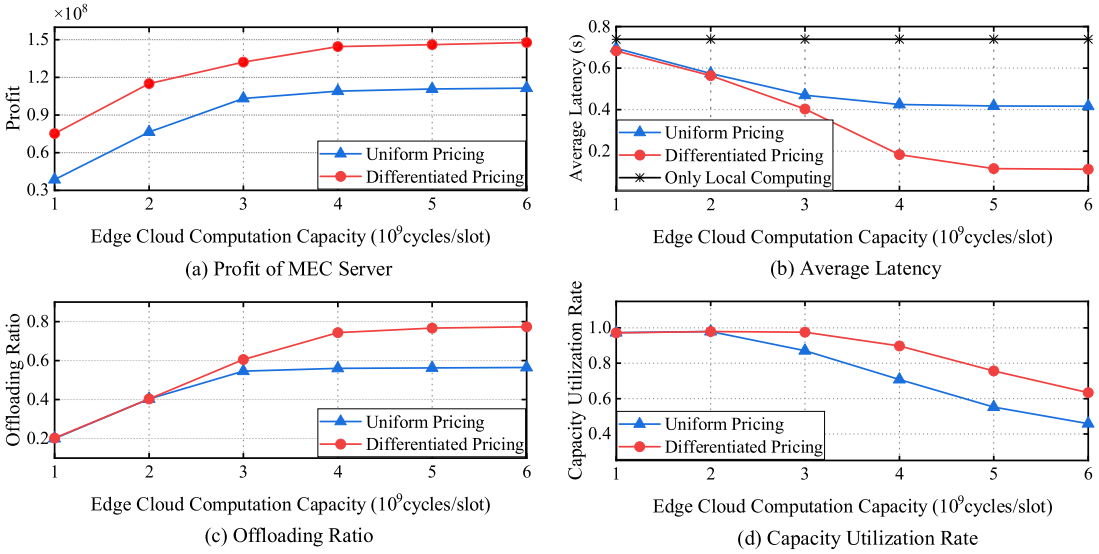


Fig. 5. Comparison with capacity.

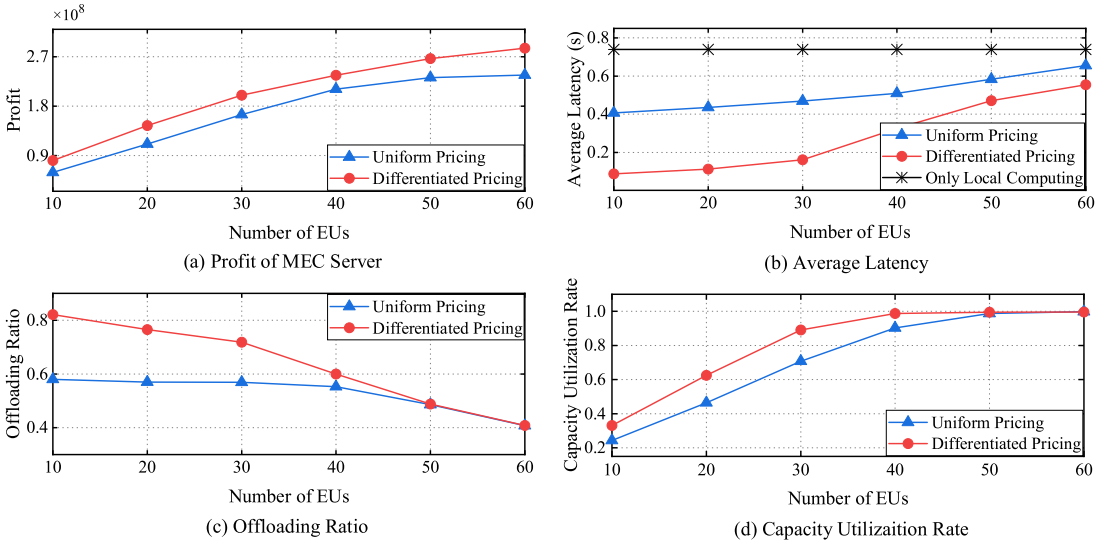


Fig. 6. Comparison with numbers of EUs.

capacity. As shown in Fig. 5(c), the task offloading ratio for both pricing strategies increases and then stabilizes as the server's computing capacity increases. Eventually, the EUs' task offloading rate is remarkably higher at differentiated pricing than at uniform pricing. Combined with Fig. 5(d), the EUs' task offloading rate increases as capacity increases when the MEC capacity utilization rate approaches 1. Conversely, when the capacity utilization rate is less than 1, it indicates that demand of EUs has been satisfied and the task offloading rate slowly plateaus.

6.2.4. Comparative experiments on the numbers of EUs

Performance between the uniform pricing scheme and the differentiated pricing scheme regarding the change in the number of EUs are compared in Fig. 6. In this experiment, the computational capacity of the MEC server is fixed at 6×10^9 cycles/slot. The number of EUs is incremented from 10 to 60.

A similar conclusion as in Fig. 5 can be obtained regarding comparison of the two proposed pricing schemes and the fully local computing scenario. As shown in Fig. 6(a), competition among EUs becomes fiercer as the number of EUs increases, resulting in higher resource pricing and higher profit for the MEC server. The trend in average latency is plotted in Fig. 6(b). The red and blue lines are steeper at the EUs count of 40 to 60 than at 10 to 30. The MEC capacity is sufficient when the number of EUs is 10 to 30. As the number increases, the frequency of computation allocated to each EU i decreases, resulting in slower execution of tasks and increased latency. As the number of EUs changes from 40 to 60, the computational capacity is almost exhausted, as shown in Fig. 6(d). At this point, the rising number of EUs leads to an increase in the percentage of latency generated by local computation, so the average latency rises faster. Consistent with the analysis above, in the curves of Fig. 6(c), EUs have the opportunity to offload more tasks to edge when the number of EUs is low. The task offloading rate is almost equal for both pricing schemes as the number of EUs increases to 50. At this point, the server could gain higher profit due to the precise pricing of differentiated pricing scheme.

6.2.5. Comparative experiment on channel bandwidth

Channel bandwidth affects efficiency of task offloading. For example, when the bandwidth is high, the tasks are transferred quickly and the overall latency will be relatively low for the EUs. It also has an impact on the task offloading ratio and profit of the MEC server. The number of EUs in this experiment is fixed at 60. The effect of channel bandwidth on performance of the DPOA is shown in Fig. 7.

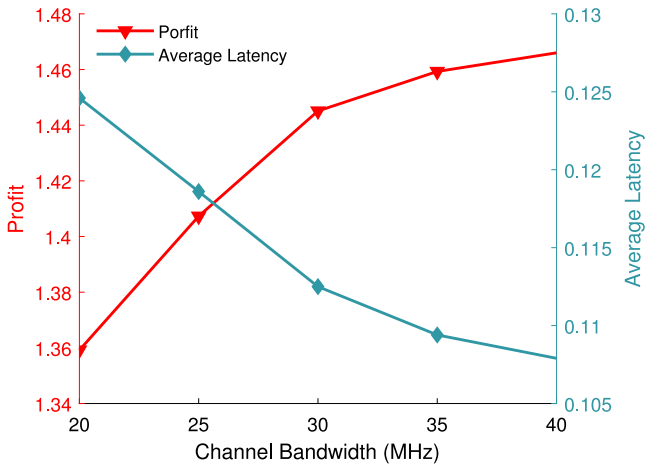


Fig. 7. The influence of channel bandwidth.

Both the average latency and the profit of the MEC server have better performance as the bandwidth increases. Trends of the curves reveal that profit of the server and average latency conflict with each other. When the profit is low, EUs purchase fewer computing resources. The corresponding latency is higher. Conversely, when the profit of the server is high, the task execution is efficient, and therefore the latency is low. When the channel bandwidth increases, EUs can be distributed at a higher computation frequency and perform tasks faster. Thus, EUs tend to offload tasks to the MEC server to improve QoE. This also leads to lower average latency and higher profit of the server. An intermediate value $B = 30$ was chosen in the experiment to ensure the performance of the edge and ends.

7. Conclusions and future work

The advent of IoT enables end devices to pay for resources on demand to edge devices. In this paper, we study the problem of task offloading and resource allocation in the MEC system with limited computational resources. The two-stage Stackelberg game is applied to model the EU-server interaction problem. In the first stage, the server sets the resource prices based on EUs' task attributes. In the second stage, EUs determine their requirements based on the prices decided by the server. The simulation results show that the DPOA based on the Stackelberg game can effectively improve the task processing capacity and resource utilization efficiency of the system through differentiated pricing.

In future work, we will introduce more complex multi-EU multi-server MEC scenarios, where both the EU side and the MEC side execute tasks using energy harvested from nature. In addition, algorithms with lower time complexity will be proposed to save energy.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The authors do not have permission to share data.

Acknowledgments

The authors thank the editors and reviewers for their insightful comment and valuable suggestions. This work was supported by the Program of National Natural Science Foundation of China (grant No. 62072174, 61502165), National Natural Science Foundation of Hunan Province, China (grant No. 2022JJ40278, 2020JJ5370), Scientific Research Fund of Hunan Provincial Education Department, China (Grant No. 22A0026).

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: Vision and challenges, IEEE Internet Things J. 3 (5) (2016) 637–646.
[2] S. Duan, D. Wang, J. Ren, F. Lyu, Y. Zhang, H. Wu, X. Shen, Distributed artificial intelligence empowered by end-edge-cloud computing: A survey, IEEE Commun. Surv. Tutor. (2022) http://dx.doi.org/10.1109/COMST.2022.3218527.
[3] Cisco Annual Internet Report (2018–2023) White Paper, Cisco, San Jose, CA, USA, 2020.
[4] F. Lyu, J. Ren, N. Cheng, P. Yang, M. Li, Y. Zhang, X.S. Shen, LEAD: Large-scale edge cache deployment based on spatio-temporal WiFi traffic statistics, IEEE Trans. Mob. Comput. 20 (8) (2020) 2607–2623.
[5] M. Díaz, C. Martín, B. Rubio, State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing, J. Netw. Comput. Appl. 67 (C) (2016) 99–117.
[6] T. Wang, Y. Mei, X. Liu, J. Wang, H.-N. Dai, Z. Wang, Edge-based auditing method for data security in resource-constrained internet of things, J. Syst. Archit. 114 (5) (2021) 101971.
[7] I.A. Elgendy, W. Zhang, Y.-C. Tian, K. Li, Resource allocation and computation offloading with data security for mobile edge computing, Future Gener. Comput. Syst. 100 (2019) 531–541.
[8] Y. Liu, M. Peng, G. Shou, Y. Chen, S. Chen, Toward edge intelligence: Multiaccess edge computing for 5G and Internet of Things, IEEE Internet Things J. 7 (8) (2020) 6722–6747.
[9] A. Zhu, Y. Wen, Computing offloading strategy using improved genetic algorithm in mobile edge computing system, J. Grid Comput. 19 (3) (2021) 1–12.
[10] Z. Tong, H. Chen, X. Deng, K. Li, K. Li, A scheduling scheme in the cloud computing environment using deep Q-learning, Inform. Sci. 512 (2020) 1170–1191.
[11] Z. Chen, H. Zheng, J. Zhang, X. Zheng, C. Rong, Joint computation offloading and deployment optimization in multi-UAV-enabled MEC systems, Peer-To-Peer Netw. Appl. 15 (1) (2022) 194–205.
[12] W. Feng, H. Liu, Y. Yao, D. Cao, M. Zhao, Latency-aware offloading for mobile edge computing networks, IEEE Commun. Lett. 25 (8) (2021) 2673–2677.
[13] J. Ren, G. Yu, Y. Cai, Y. He, Latency optimization for resource allocation in mobile-edge computation offloading, IEEE Trans. Wireless Commun. 17 (8) (2018) 5506–5519.
[14] Z. Hu, J. Niu, T. Ren, B. Dai, Q. Li, M. Xu, S.K. Das, An efficient online computation offloading approach for large-scale mobile edge computing via deep reinforcement learning, IEEE Trans. Serv. Comput. 15 (2) (2021) 669–683.
[15] S. Duan, F. Lyu, H. Wu, W. Chen, H. Lu, Z. Dong, X. Shen, Moto: Mobility-aware online task offloading with adaptive load balancing in small-cell mec, IEEE Trans. Mob. Comput. (2022) http://dx.doi.org/10.1109/TMC.2022.3220720.
[16] Y. Wu, J. Xia, C. Gao, J. Ou, C. Fan, J. Ou, D. Fan, Task offloading for vehicular edge computing with imperfect CSI: A deep reinforcement approach, Phys. Commun. 55 (2022) 101867.
[17] Y. Mao, J. Zhang, K.B. Letaief, Dynamic computation offloading for mobile-edge computing with energy harvesting devices, IEEE J. Sel. Areas Commun. 34 (12) (2016) 3590–3605.
[18] Z. Tong, J. Cai, J. Mei, K. Li, K. Li, Dynamic energy-saving offloading strategy guided by Lyapunov optimization for IoT devices, IEEE Internet Things J. (2022) http://dx.doi.org/10.1109/JIOT.2022.3168968.
[19] L. Li, T. Lv, P. Huang, P.T. Mathiopoulos, Cost optimization of partial computation offloading and pricing in vehicular networks, J. Signal Process. Syst. 92 (12) (2020) 1421–1435.
[20] J. Du, W. Cheng, G. Lu, H. Cao, X. Chu, Z. Zhang, J. Wang, Resource pricing and allocation in MEC enabled blockchain systems: An A3C deep reinforcement learning approach, IEEE Trans. Netw. Sci. Eng. 9 (1) (2021) 33–44.
[21] A.N. Toosi, K. Vanmechelen, F. Khodadadi, R. Buyya, An auction mechanism for cloud spot markets, ACM Trans. Auton. Adapt. Syst. (TAAS) 11 (1) (2016) 1–33.
[22] X. Wang, Y. Sui, J. Wang, C. Yuen, W. Wu, A distributed truthful auction mechanism for task allocation in mobile cloud computing, IEEE Trans. Serv. Comput. 14 (3) (2018) 628–638.
[23] V. Kantere, D. Dash, G. Francois, S. Kyriakopoulou, A. Ailamaki, Optimal service pricing for a cloud cache, IEEE Trans. Knowl. Data Eng. 23 (9) (2011) 1345–1358.

[24] C. Liu, K. Li, K. Li, A game approach to multi-servers load balancing with load-dependent server availability consideration, IEEE Trans. Cloud Comput. 9 (1) (2018) 1–13.
[25] H. Li, M. Dong, K. Ota, M. Guo, Pricing and repurchasing for big data processing in multi-clouds, IEEE Trans. Emerg. Top. Comput. 4 (2) (2016) 266–277.
[26] Y. Chen, Z. Li, B. Yang, K. Nai, K. Li, A stackelberg game approach to multiple resources allocation and pricing in mobile edge computing, Future Gener. Comput. Syst. 108 (2020) 273–287.
[27] J. Liu, X. Wang, S. Shen, Z. Fang, S. Yu, G. Yue, M. Li, Intelligent jamming defense using DNN Stackelberg game in sensor edge cloud, IEEE Internet Things J. 9 (6) (2021) 4356–4370.
[28] T. Wang, Y. Lu, J. Wang, H.-N. Dai, X. Zheng, W. Jia, EIHDP: Edge-intelligent hierarchical dynamic pricing based on cloud-edge-client collaboration for IoT systems, IEEE Trans. Comput. 70 (8) (2021) 1285–1298.
[29] M. Liu, Y. Liu, Price-based distributed offloading for mobile-edge computing with computation capacity constraints, IEEE Wirel. Commun. Lett. 7 (3) (2017) 420–423.
[30] M. Tao, K. Ota, M. Dong, H. Yuan, Stackelberg game-based pricing and offloading in mobile edge computing, IEEE Wirel. Commun. Lett. 11 (5) (2021) 883–887.
[31] K. Kang, S.X. Xu, R.Y. Zhong, B.Q. Tan, G.Q. Huang, Double auction-based manufacturing cloud service allocation in an industrial park, IEEE Trans. Autom. Sci. Eng. 19 (1) (2022) 295–307.
[32] N.C. Luong, P. Wang, D. Niyato, Y. Wen, Z. Han, Resource management in cloud networking using economic analysis and pricing models: A survey, IEEE Commun. Surv. Tutor. 19 (2) (2017) 954–1001.
[33] H. Lin, S. Zeadally, Z. Chen, H. Labiod, L. Wang, A survey on computation offloading modeling for edge computing, J. Netw. Comput. Appl. 169 (2020) 102781.
[34] X. Chen, L. Jiao, W. Li, X. Fu, Efficient multi-user computation offloading for mobile-edge cloud computing, IEEE/ACM Trans. Netw. 24 (5) (2015) 2795–2808.
[35] S. Tadelis, Game Theory: An Introduction, Princeton University Press, 2013.
[36] A. Jakóbik, F. Palmieri, J. Kołodziej, Stackelberg games for modeling defense scenarios against cloud security threats, J. Netw. Comput. Appl. 110 (15) (2018) 99–107.
[37] B. Yang, D. Wu, H. Wang, Y. Gao, R. Wang, Two-layer Stackelberg game-based offloading strategy for mobile edge computing enhanced FiWi access networks, IEEE Trans. Green Commun. Netw. 5 (1) (2020) 457–470.
[38] F. Guo, H. Zhang, H. Ji, X. Li, V.C. Leung, An efficient computation offloading management scheme in the densely deployed small cell networks with mobile edge computing, IEEE/ACM Trans. Netw. 26 (6) (2018) 2651–2664.
[39] F. Li, H. Yao, J. Du, C. Jiang, Y. Qian, Stackelberg game-based computation offloading in social and cognitive industrial Internet of Things, IEEE Trans. Ind. Inform. 16 (8) (2019) 5444–5455.
[40] Z. Liu, J. Fu, Y. Zhang, Computation offloading and pricing in mobile edge computing based on Stackelberg game, Wirel. Netw. 27 (7) (2021) 4795–4806.

Zhao Tong received the Ph.D degree in computer science from Hunan University, Changsha, China in 2014. He was a visiting scholar at the Georgia State University from 2017 to 2018. He is currently an associate professor at the College of Information Science and Engineering of Hunan Normal University, the young backbone teacher of Hunan Province, China. His research interests include parallel and distributed computing systems, resource management, big data and machine learning algorithm. He has published more than 25 research papers in international conferences and journals, such as IEEE-TPDS, Information Sciences, FGCS, NCA, and JPDC, PDCAT, etc. He is a senior member of the China Computer Federation (CCF) and a Member of the IEEE.

Xin Deng received the B.S. degree in computer science and technology from Hengyang Normal University, Hengyang, China, in 2020. She is currently working toward the M.S. degree at the College of Information Science and Engineering, Hunan Normal University, Changsha, China. Her research interests focus on distributed parallel computing, modeling and resource pricing and allocation in mobile edge computing systems, and game theory.

Jing Mei received the Ph.D. degree in computer science from Hunan University, Changsha, China, in 2015. She is currently a Lecturer with the College of Information Science and Engineering, Hunan Normal University, Changsha. She has published more than 15 research articles in international conferences and journals, such as IEEE TRANSACTIONS ON COMPUTERS, IEEE-SC, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, Cluster Computing, Journal of Gastric Cancer, The Journal of Supercomputing, and ICPP. Her research interests include parallel and distributed computing, cloud computing, and combinatorial optimization. Dr. Mei is a member of the China Computer Federation.

Longbao Dai received the B.S. degree in computer science and technology from Hunan University of Science and Engineering, Yongzhou, China, in 2020. He is currently working toward the M.S. degree at the College of Information Science and Engineering, Hunan Normal University, Changsha, China. His research interests focus on distributed parallel computing, modeling and resource pricing and allocation in mobile edge computing systems, and game theory.

Kenli Li received the Ph.D. degree in computer science from Huazhong University of Science and Technology, China, in 2003. He was a visiting scholar at University of Illinois at Urbana-Champaign from 2004 to 2005. He is currently the dean and a full professor of computer science and technology with Hunan University and deputy director of National Supercomputing Center in Changsha. His major research areas include parallel computing, high-performance computing, grid and cloud computing. He has published more than 150 research papers in international conferences and journals such as the IEEE Transactions on Computers, the IEEE Transactions on Parallel and Distributed Systems, the IEEE Transactions on Signal Processing, the Journal of Parallel and Distributed Systems, ICPP, and CCGrid. He serves on the editorial board of the IEEE Transactions on Computers. He is an outstanding member of CCF. He is a senior member of the IEEE.

Keqin Li is a SUNY Distinguished Professor of computer science with the State University of New York. He is also a Distinguished Professor at Hunan University, China. His current research interests include cloud computing, fog computing and mobile edge computing, energy-efficient computing and communication, embedded systems and cyber–physical systems, heterogeneous computing systems, big data computing, high-performance computing, CPU–GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, intelligent and soft computing. He has authored or coauthored over 850 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He currently serves or has served on the editorial boards of the IEEE Transactions on Parallel and Distributed Systems, the IEEE Transactions on Computers, the IEEE Transactions on Cloud Computing, the IEEE Transactions on Services Computing, and the IEEE Transactions on Sustainable Computing. He is an IEEE Fellow.