SCIENCE CHINA
Information Sciences

April 2020, Vol. 63 140313:1–140313:16
https://doi.org/10.1007/s11432-019-2780-0

• RESEARCH PAPER •
Special Focus on Internet of Minds

Joint time delay and energy optimization with intelligent overclocking in edge computing

Kehao WANG[1,2*], Zhenhua XIONG[1], Lin CHEN[3], Pan ZHOU[4] & Hyundong SHIN[5]

[1]*School of Information Engineering, Wuhan University of Technology, Wuhan* 430070*, China;*
[2]*Hubei Key Laboratory of Inland Shipping Technology, Wuhan University of Technology, Wuhan* 430063*, China;*
[3]*School of Data and Computer Science, Sun Yat-Sen University, Guangzhou* 510006*, China;*
[4]*School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan* 430000*, China;*
[5]*Department of Electronic Engineering, Kyung Hee University, Yongin-si* 17014*, South Korea*

Received 19 October 2019/Revised 13 December 2019/Accepted 4 February 2020/Published online 9 March 2020

Abstract With the rapid growth of user equipment (UE), the amount of data transmitted over networks has become enormous, exerting immense pressure on backbone networks and central cloud infrastructures. Simultaneously, corresponding applications requiring high energy consumption and low latency have multiplied the requirements for UE. Mobile edge computing (MEC) has been proposed to support the offloading of UE tasks to edge clouds for execution. The implementation of MEC requires fast data transmission between UE and edge servers, and the emerging 5G network appears to render this technology possible. In this paper, considering a large number of UE, a fixed MEC server, and an advanced intelligent network, we suggest an intelligent overclocking mechanism for the MEC server that operates for an intelligently calculated period to allow it to leverage more computing power without introducing additional hardware resources for a certain period of time. We jointly manage task offloading, server resource allocation, and overclocking to minimize the system-wide computation overhead and other risks. The proposed optimization problem is a mixed-integer nonlinear programming problem that is divided into three subproblems: offloading decision, resource allocation, and overclocking decision. We solve these subproblems using non-convex techniques and provide an iterative algorithm to obtain a heuristic solution for the original problem. Finally, simulation results show that the overclocked MEC server has lower system-wide computation overhead, faster task processing, and more offloaded UE as compared with the case without overclocking.

Keywords overclocking, intelligent network, mobile edge computing, computation offloading, resource allocation

Citation Wang K H, Xiong Z H, Chen L, et al. Joint time delay and energy optimization with intelligent overclocking in edge computing. Sci China Inf Sci, 2020, 63(4): 140313, https://doi.org/10.1007/s11432-019-2780-0

1 Introduction

The explosively increasing number of Internet-of-Things (IoT) devices, which is anticipated to reach 20 billion by 2020, imposes greater challenges for network communications. The popularity of complex distributed applications and the emerging 5G network has brought a slew of new requirements to IoT devices [1] (e.g., smart mobile devices, vehicle-mounted intelligent terminals [2], smart televisions, and virtual-reality devices). A growing amount of user equipment (UE) and their corresponding emerging applications inevitably increase network node traffic and data handling. Managing and processing these network data are key to any intelligent network [3, 4]. However, processing data from a large number

* Corresponding author (email: kehao.wang@whut.edu.cn)

of networks is a huge problem. Emerging applications are being developed with characteristics of time-delay sensitivity and high energy consumption. However, existing IoT devices can hardly support those conditions owing to limited power and resources [5]. Under this context, edge computing has been proposed as a promising IoT paradigm.

Fog-computing integration with IoT has been proposed, such that devices can be deployed at the edge of networks [6, 7]. Mobile edge computing (MEC)-enabled IoT devices have also been promulgated [8–10]. The core of this type of IoT involves the implementation of MEC to support the large number of UE that offloads data and computation-requiring tasks to edge networks for execution. These are known as MEC servers. Considering the emerging 5G network with its fast data transmission capability and the development of new MEC servers with even more powerful computing resources, MEC-enabled IoT should well-accommodate the emerging demands and their related quality-of-service (QoS) requirements while reducing the pressure on UE, backbone networks, and central cloud infrastructures [11].

MEC deploys cloud resources to the edge of Radio access networks to improve UE computing efficiency and storage capacity [12], and they can be applied for specific cases of mobile cloud computing (MCC). Compared with UE, MECs have abundant computing resources and powerful processing capabilities. MECs exist closer to the UE as compared with MCCs. This greatly reduces interaction time between UE and cloud services. Computation offloading involves the transmittal of UE data to the MEC server for execution and is key to MEC [13]. However, computation offloading brings extra overhead related to latency and energy consumption and is mainly divided into three cases: local execution, complete offloading, and partial offloading. If a task is executed locally, only the delay and energy consumption of the task is considered. If it is executed on MEC servers, transmission delays, data processing, and reporting are additionally considered. Therefore, minimizing cumulative overhead and satisfying the QoS of tasks are a worthy issue of study [14].

Generally, MEC resources are sufficient for UE offloading. In some practical applications, however, the number of UE constantly changes. Sometimes the number is very large, such as near marketplaces during rush hours. In these situations, if the MEC server can obtain more computation resources, it will reduce system pressures and improve the overall QoS. For this, an overclocking mechanism is proposed to gain additional computing ability from a given component by increasing its backbone operating frequency for calculated periods. Overclocking incurs some extra overhead and can lead to component damage and overheating. With an intelligent network, we can collect context data from the network to agilely and intelligently control MEC servers [15] to achieve quality and timely data processing. In this paper, we mainly consider minimizing delays, energy consumption, and overclocking overhead for a single MEC server connected to multiple UE.

This paper specifically considers the model in which MEC server resources can change dynamically with the number of UE. We further design a computation offloading framework that accounts for UE offloading decisions, server resource allocation, and overclocking decisions to minimize the overall computational overhead. The main contributions of this study can be summarized as follows.

• We propose a new overclocking concept for MEC servers that jointly considers task offloading, computation resource allocation, and server overclocking. To minimize the computational overhead of the system, an optimization problem is formulated as a mixed-integer nonlinear programming (MINLP) problem.

• For the proposed MINLP problem, we decompose the original problem into three subproblems: (i) offloading decision, (ii) MEC server resource allocation, and (iii) MEC server overclocking decision. The first subproblem is a convex optimization that can be easily solved with existing technologies. The second is still an MINLP problem, and we can obtain its optimal solution under known conditions. The third is a simple comparison problem that can be easily solved. Then, we propose iterative joint optimization for offloading decisions, overclocking decisions, and computation resource allocation algorithm (i.e., JOOC) to obtain a heuristic policy to solve the original problem.

• Our simulation results show that the proposed algorithm performs better than random offloading under two benchmark schemes, and the performance in the case of overclocking is superior to that without overclocking in terms of computational overhead, QoS, and offloading rate.

We introduce the background knowledge of MEC and overclocking in Section 2. In Section 3, we introduce a model of the MEC system with the overclocking capability, and we formulate an optimization problem. Section 4 provides the algorithm for solving the optimization problem in detail, and we analyze the algorithm's computing complexity. The experimental simulation results, analysis, and summary are presented in Section 5. Finally, the paper is concluded in Section 6.

2 Related work

2.1 Mobile edge computing

The emergence of 5G networks and the dawn of the IoT age have brought new opportunities and challenges to the transmission and processing of network data. As a key technology for solving these challenges, MEC has attracted a lot of research. Mach and Becvar [13] detailed the MEC system and presented two of the most commonly used metrics to evaluate performance (i.e., energy consumption and latency).

In terms of time delay, Alameddine et al. [16] jointly tackled task offloading and order of execution by considering different requirements and task latency. They mainly considered five delays: uploading, edge-to-edge, waiting, processing, and downloading. Ning et al. [17] comprehensively considered both cloud computing and MEC, using the ARkit framework to consider the partial offloading of sensitive time-delayed tasks. They mainly computed the transmission delay of the block task between local, MEC, and cloud servers. Wang et al. [18] investigated the problem of how to realize effective federated offloading for moving vehicles with a goal of minimizing total latency.

In terms of optimizing energy consumption, Zhang et al. [19] combined radio-frequency-based wireless power transfer and MEC technologies to deliver energy to energy-constrained wireless devices to improve energy efficiency. Li et al. [20] specifically noted that each terminal device could access multiple edge servers and distributed heterogeneous computational resources among multiple devices in the network to achieve optimal network-level energy efficiency while meeting QoS.

For joint optimization of latency and energy consumption, Cui et al. [21] used the mechanism of small cells [22] to improve spectrum and energy efficiencies. Pham et al. [23] introduced a novel framework for joint computation offloading and resource allocation in MEC networks with wireless backhaul [24, 25]. They proposed a model framework for wireless backhaul to achieve dense small-cell deployment and efficient data transmission. Tran et al. [26] considered a multi-cell ultra-dense network in which each base station (BS) was equipped with an MEC server to provide computation offloading services to mobile users. According to the context information of the network, they proposed a mechanism for resource allocation coordination of multiple neighboring BSs to reduce interference and resource contention among users. Zhang et al. [27] proposed an energy-aware offloading scheme and used the energy of surface-mounted diodes as a constraint to jointly optimize system overhead.

Most of the above MEC models only consider the optimization of computational overhead from user characteristics, task-offloading methods, and network access points but do not optimize system overhead from changes in the MEC server.

2.2 Overclocking

High power consumption has become one of the most important problems for supercomputer systems. Therefore, improving the energy efficiency of a computer node is our concern. It is very important to achieve a tradeoff between performance and energy consumption for better energy efficiency; overclocking the central processing unit (CPU) is a simple method for realizing this purpose.

There are many ways to overclock a CPU [28–32]. Wu et al. [29] utilized processor overclocking and memory-frequency scaling to achieve better performance and lower power consumption, bringing about better energy efficiencies for their benchmarks. Jang et al. [30] adopted the F-overclocking technique, which increased clock frequency without changing supply voltage. They then proposed an adaptive overclocking controller that dynamically managed the F-overclocking technique based on dynamic application

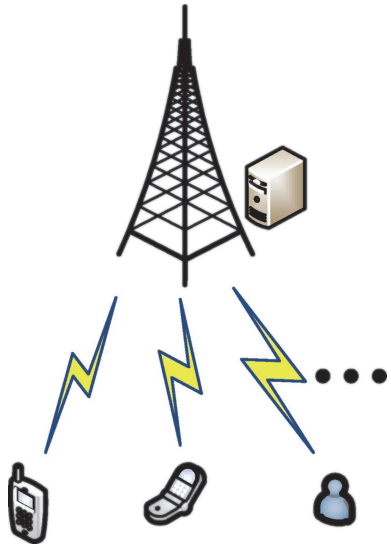

Figure 1 (Color online) System model.

characteristics. Short et al. [31] considered overclocking a controller area network to improve information throughput. Zhao et al. [32] proposed low-density parity-check-based overclocked solid-state drives.

Although there are many useful overclocking applications, they have not been used for emerging MEC technologies. Based on intelligent networks and the emergence of 5G, we propose an intelligent overclocking mechanism for an MEC server.

3 System model and problem formulation

3.1 System model

We consider an MEC system comprising multiple UE and a BS equipped with a smart overclocking server as shown in Figure 1.

We assume that there are $\mathcal{N} = \{1, 2, 3, \ldots, N\}$ UE, and each UE, n, has a task $\boldsymbol{I}_n$ to be offloaded.

$$\boldsymbol{I}_n = \{C_n, D_n, T_n^{\max}\}, \quad \forall n \in \mathcal{N}, \tag{1}$$

where C_n denotes the total number of the CPU cycles required to accomplish task $\boldsymbol{I}_n$. D_n is the computed input data size, and $T_n^{\max}$ is the QoS requirement of task $\boldsymbol{I}_n$. Each computation task can be executed either locally or on an MEC server. Consider that when the MEC server is not overclocked, its maximum computation resource is F. When the MEC server is overclocked, more computing resources (φF, φ (constant), and $0 < \varphi < 1$) will be allocated for tasks. At this time, however, an overclocking loss $L(t)$ will be generated because of the additional heat and possible hardware damage. The overclocking working time of an MEC server cannot exceed time T_0 to ensure that the MEC server works stably. The loss function $L(t)$ is given by

$$L(t) = \begin{cases} \alpha t, & 0 \leqslant t \leqslant T_0, \\ \infty, & T_0 < t < T^{\mathrm{cyc}}, \end{cases} \tag{2}$$

where $\alpha > 0$ is a fixed value representing the growth rate of the loss function $L(t)$ with time t, and the T^{cyc} represents the period of the loss function.

We define the offloading decision model as $\boldsymbol{x} = \{x_n \mid \forall n \in \mathcal{N}\}$, where $x_n \in \{0, 1\}$. When $x_n = 0$, task $\boldsymbol{I}_n$ will be executed locally. When $x_n = 1$, task $\boldsymbol{I}_n$ will be offloaded to the MEC server for execution.

We define the overclocking decision model as $a \in \{0, 1\}$, where $a = 0$ indicates that the MEC server is not overclocked, whereas $a = 1$ means that the MEC server is overclocked.

3.2 Executed locally

We denote f_n^l as the computational capability of UE n. Let t_n^l represent the computation time required for the local execution of task $\boldsymbol{I}_n$, which can be computed as

$$t_n^l = C_n / f_n^l, \quad \forall n \in \mathcal{N}. \tag{3}$$

The energy consumption E_n^l (in Joule) of UE n executing a task can be modeled as

$$E_n^l = \kappa_n C_n (f_n^l)^2, \quad \forall n \in \mathcal{N}, \tag{4}$$

where κ_n is the coefficient related to the chip-hardware architecture according to the measurements in [33]. We set $\kappa_n = 5 \times 10^{(-27)}$ in this paper.

The computational overhead of the local execution of task $\boldsymbol{I}_n$ is a function of the task's execution time and energy consumption defined as

$$U_n^l = \lambda_n^t t_n^l + \lambda_n^e E_n^l, \quad \forall n \in \mathcal{N}, \tag{5}$$

where $\lambda_n^t \in [0,1]$ and $\lambda_n^e \in [0,1]$ ($\lambda_n^t + \lambda_n^e = 1$) are weight parameters for the computational time and energy consumption of task $\boldsymbol{I}_n$. Those two weighting factors can be set by the user according to the battery condition of the UE n and the delay requirement of the computation task.

3.3 Executed on MEC

Let P_n and H_n represent the transmitting power of UE n and the channel gain between UE n and the MEC. Accordingly, the transmission rate r_n of date D_n is

$$r_n = \frac{W}{N_{\text{Off}}} \log_2 \left(1 + \frac{P_n H_n}{N_0}\right), \quad \forall n \in \mathcal{N}_{\text{Off}}, \tag{6}$$

where $\mathcal{N}_{\text{Off}}$ represents the set of tasks that are offloaded to the MEC server. $\mathcal{N}_{\text{Off}} = \{n \mid x_n = 1, n \in \mathcal{N}\}$, N_{Off} is the number of offloading UE, $N_{\text{Off}} = \sum_{n=1}^{N} x_n$, W is the system bandwidth, and N_0 is the additive white Gaussian noise power.

Let t_n^p be the time for uploading data D_n to the edge server given by

$$t_n^p = D_n / r_n, \quad \forall n \in \mathcal{N}_{\text{Off}}. \tag{7}$$

Let E_n^p denote the transmission energy consumption of task $\boldsymbol{I}_n$. Then, we get

$$E_n^p = P_n t_n^p, \quad \forall n \in \mathcal{N}_{\text{Off}}. \tag{8}$$

Let t_n^r be the execution time of task $\boldsymbol{I}_n$ executed at the MEC server. t_n^r can be expressed as

$$t_n^r = C_n / f_n^r, \quad \forall n \in \mathcal{N}_{\text{Off}}, \tag{9}$$

where f_n^r is the computing resource provided by the MEC server, and the computation resource vector is defined as $\boldsymbol{f} = \{f_n^r \mid \forall n \in \mathcal{N}\}$.

Similar to the computational overhead of local execution, the computation overhead under remote execution can be computed as

$$U_n^r = \lambda_n^t (t_n^p + t_n^r) + \lambda_n^e E_n^p, \quad \forall n \in \mathcal{N}_{\text{Off}}. \tag{10}$$

3.4 Problem formulation

To minimize the system's computational overhead, the objective function can be defined as

$$U(\boldsymbol{x}, a, \boldsymbol{f}) = \sum_{n\in\mathcal{N}} U_n(x_n, a, f_n^r) + aL(t),$$

where

$$U_n(x_n, a, f_n^r) = (1 - x_n)U_n^l + x_n U_n^r, \quad \forall n \in \mathcal{N}. \tag{11}$$

Then this problem can be formulated as follows:

$$\begin{aligned}
&\min_{\boldsymbol{x},a,\boldsymbol{f}} \left\{ \sum_{n\in\mathcal{N}} U_n(x_n, a, f_n^r) + aL(t) \right\} \\
&\text{s.t.} \quad \text{C1} : x_n = \{0, 1\}, \quad \forall n \in \mathcal{N}, \\
&\qquad \text{C2} : a \in \{0, 1\}, \\
&\qquad \text{C3} : f_n^r > 0, \quad \forall n \in \mathcal{N}_{\text{Off}}, \\
&\qquad \text{C4} : \sum_{n\in\mathcal{N}_{\text{Off}}} f_n^r \leqslant (F + a\varphi F), \\
&\qquad \text{C5} : a \max_n\{t_n^r\} \leqslant T_0, \quad \forall n \in \mathcal{N}_{\text{Off}}, \\
&\qquad \text{C6} : (1 - x_n)t_n^l + x_n(t_n^p + t_n^r) \leqslant T_n^{\max}, \quad \forall n \in \mathcal{N}.
\end{aligned} \tag{12}$$

In this formulation, C1 represents the offloading decision, C2 represents the overclocking decision, C3 indicates that the computing resource for each task offloaded on the MEC server is positive, C4 implies that the total computing resource used is limited by the MEC server's maximum resource, and C5 means that when the MEC server is overclocked. Thus, it cannot work longer than T_0. C6 means the execution time of task $\boldsymbol{I}_n$ should satisfy its QoS.

We see that $\boldsymbol{x}$ and a are all binary integers, whereas $\boldsymbol{f}$ is continuous. Therefore the optimization problem is an MINLP problem, which is NP-hard.

4 Proposed algorithm and analysis

Considering the huge computing complexity of the problem (12), we decompose it into three subproblems: offloading decisions, computation resource allocation, and overclocking decisions. Specifically, we propose an iterative algorithm (i.e., JOOC) to solve the first two subproblems of overclocking and non-overclocking states of the MEC server. Then, from the obtained solution, we can determine the overclocking decision. Because the original problem is difficult to solve directly, and it is difficult to find the optimal solution, our decomposition method cannot guarantee the optimality of the original problem. Figure 2 is the framework of our proposed scheme for the original problem (12).

4.1 Offloading decision

We assume the computation resource $\boldsymbol{f}$ and the overclocking decision a are determined. Then, the offloading decision subproblem can be rewritten as

$$\begin{aligned}
&\min_{\boldsymbol{x}} \sum_{n\in\mathcal{N}} U_n(x_n) \\
&\text{s.t.} \quad x_n = \{0, 1\}, \quad \forall n \in \mathcal{N}, \\
&\qquad (1 - x_n)t_n^l + x_n(t_n^p + t_n^r) \leqslant T_n^{\max}, \quad \forall n \in \mathcal{N}.
\end{aligned} \tag{13}$$

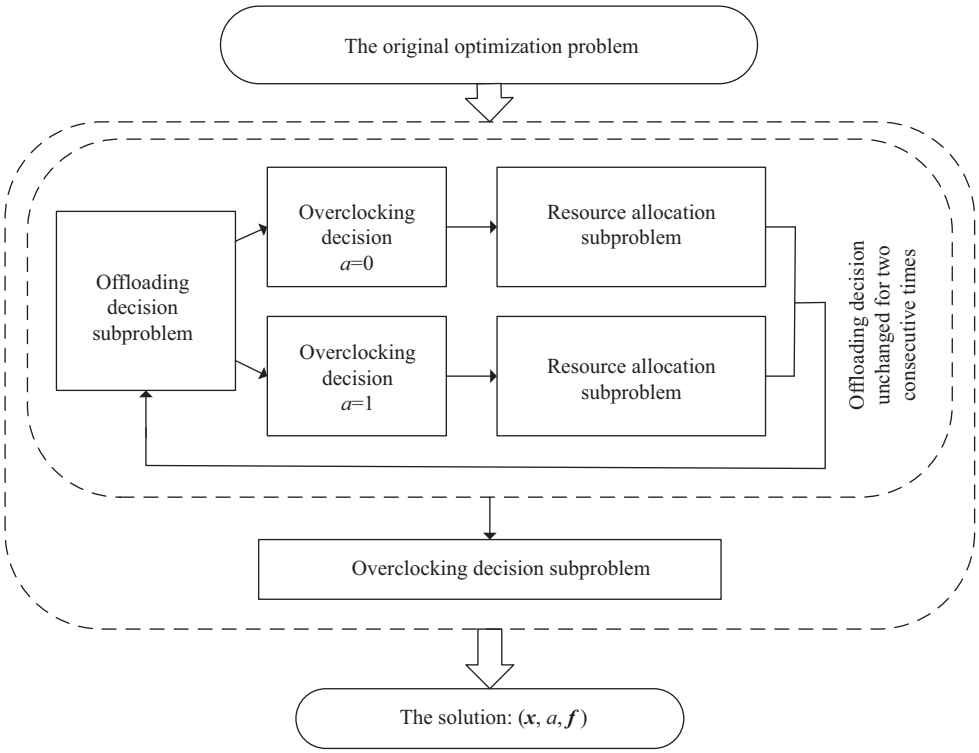

Figure 2 Proposed framework for solving the problem (12).

The problem (13) can be rewritten as

$$\sum_{n\in\mathcal{N}} U_n(x_n) = \sum_{n\in\mathcal{N}} x_n U_n^r + \sum_{n\in\mathcal{N}} (1-x_n)U_n^l = \sum_{n\in\mathcal{N}} x_n(U_n^r - U_n^l) + \sum_{n\in\mathcal{N}} U_n^l.$$

According to Subsection 3.2, $\sum_{n\in\mathcal{N}} U_n^l$ can be calculated in advance, and the problem (13) can be turned into

$$\min_{\boldsymbol{x}} \sum_{n\in\mathcal{N}} x_n(U_n^r - U_n^l). \tag{14}$$

Step 1. We assume that all tasks $\boldsymbol{I}_n$ are offloaded to the MEC server (i.e., $\mathcal{N} = \mathcal{N}_{\text{Off}}$). We then get U_n^r ($n \in \mathcal{N}_{\text{Off}}$), as calculated in Subsections 3.2 and 4.2. If $U_n^r > U_n^l$ or the execution time of task $\boldsymbol{I}_n$ cannot satisfy the condition, i.e., $(t_n^p + t_n^r) \leqslant T_n^{\max}$, this UE will not offload its computation task $\boldsymbol{I}_n$ to the MEC server, and $x_n = 0$. Otherwise, the execution time of task $\boldsymbol{I}_n$ cannot satisfy the condition $(t_n^p + t_n^r) \leqslant T_n^{\max}$. This UE will not offload its computation task $\boldsymbol{I}_n$ to the MEC server, and $x_n = 0$. However, task $\boldsymbol{I}_n$ will be chosen to offload to the MEC server (i.e., $x_n = 1$). Then we get the offloading decision $\boldsymbol{x}$ of the first iteration.

Step 2. In each iteration τ, we first pick one task whose $(U_n^r - U_n^l)$ is the largest of all rejected tasks (i.e., $x_n(\tau) = 0$, $(\forall n \in \mathcal{N}_{\text{Off}})$). Then, we cause $x_n(\tau)$ of the remaining tasks to return to 1. We can then get a new offloading decision $\boldsymbol{x}(\tau)$ to update $\mathcal{N}_{\text{Off}}$ for the next iteration until $\forall n \in \mathcal{N}_{\text{Off}}$, $U_n^r \leqslant U_n^l$ or $\mathcal{N}_{\text{Off}} = \phi$.

Step 3. As in Step 2, in each iteration τ, we remove the task having the largest $(t_n^p + t_n^r) - T_n^{\max}$ value from set $\mathcal{N}_{\text{Off}}$ until $\forall n \in \mathcal{N}_{\text{Off}}$, $(t_n^p + t_n^r) \leqslant T_n^{\max}$ or $\mathcal{N}_{\text{Off}} = \phi$. Then, we can get the final offloading decision $\boldsymbol{x}$.

4.2 Computation resource allocation

When the offloading decision $\boldsymbol{x}$ and the overclocking decision a are both known, the subproblem of resource allocation can be rewritten as

$$\begin{aligned}
\mathcal{P}: \quad & \min_{\boldsymbol{f}} \left\{ y(\boldsymbol{f}) = \sum_{n\in\mathcal{N}_{\mathrm{Off}}} \frac{\lambda_n^t C_n}{f_n^r} + a\alpha \max_n \left\{ \frac{C_n}{f_n^r} \right\} \right\} \\
\text{s.t.} \quad & D1: \sum_{n\in\mathcal{N}_{\mathrm{Off}}} f_n^r \leqslant (F + a\varphi F), \\
& D2: f_n^r > 0, \quad \forall n \in \mathcal{N}_{\mathrm{Off}}, \\
& D3: a \max_n \left\{ \frac{C_n}{f_n^r} \right\} \leqslant T_0, \quad \forall n \in \mathcal{N}_{\mathrm{Off}},
\end{aligned} \tag{15}$$

where $\max_n\{\frac{C_n}{f_n^r}\}$ indicates the maximum working time of the MEC server.

Lemma 1. (1) If the MEC server is not overclocked, i.e., $a = 0$, the optimal resource allocation strategy is

$$f_n^r = \frac{F\sqrt{\lambda_n^t C_n}}{\sum_{n\in\mathcal{N}_{\mathrm{Off}}} \sqrt{\lambda_n^t C_n}}. \tag{16}$$

(2) If the MEC server is overclocked, i.e., $a = 1$, when $\frac{C_i}{\lambda_i^t+\alpha} \geqslant \max_j\{\frac{C_j}{\lambda_j^t}\}$, $j \in \mathcal{N}_{\mathrm{Off}}, j \neq i$ and the maximum overclocking time of the MEC server does not exceed T_0, we get the optimal resource allocation strategy:

$$f_n^r = \frac{(F + \varphi F)\sqrt{\lambda_n^{t'} C_n}}{\sum_{n\in\mathcal{N}_{\mathrm{Off}}} \sqrt{\lambda_n^{t'} C_n}},$$

where $\lambda_i^{t'} = \lambda_i^t + \alpha$ and $\lambda_j^{t'} = \lambda_j^t$, $j \in \mathcal{N}_{\mathrm{Off}}, j \neq i$.

If the maximum overclocking time of the MEC server exceeds T_0, we have

$$f_n^r = \begin{cases} \dfrac{(F+\varphi F)\sqrt{\lambda_n^{t'} C_n}}{\sum_{n\in\mathcal{N}_{\mathrm{Off}}} \sqrt{\lambda_n^{t'} C_n}}, & 0 \leqslant t \leqslant T_0, \\ \dfrac{(F+\varphi F)\sqrt{\lambda_n^{t} C_n}}{\sum_{n\in\mathcal{N}_{\mathrm{Off}}} \sqrt{\lambda_n^{t} C_n}}, & T_0 < t. \end{cases} \tag{17}$$

Proof. When $a = 0$, the problem $\mathcal{P}$ can be rewritten as

$$\begin{aligned}
\mathcal{P}': \quad & \min_{\boldsymbol{f}} \left\{ y(\boldsymbol{f}) = \sum_{n\in\mathcal{N}_{\mathrm{Off}}} \frac{\lambda_n^t C_n}{f_n^r} \right\} \\
\text{s.t.} \quad & D1: \sum_{n\in\mathcal{N}_{\mathrm{Off}}} f_n^r \leqslant F, \\
& D2: f_n^r > 0, \quad \forall n \in \mathcal{N}_{\mathrm{Off}},
\end{aligned} \tag{18}$$

and we can determine that $\partial y/\partial f_n^r = -(\lambda_n^t C_n/f_n^r) < 0$ and $\partial^2 y/\partial {f_n^r}^2 = 2C_n/{f_n^r}^3 > 0$. Thus, the objective in (18) is a convex monotonic decreasing function. Because D1 and D2 are both linear constraints, $\mathcal{P}'$ is a Lagrangian duality problem.

For constraint D1, the original problem can be slacked into a single constraint optimization problem by using the Lagrange multiplier:

$$\min_{\boldsymbol{f}>0} L(\boldsymbol{f}, \beta) = \sum_{n\in\mathcal{N}_{\mathrm{Off}}} \frac{\lambda_n^t C_n}{f_n^r} + \beta \left(\sum_{n\in\mathcal{N}_{\mathrm{Off}}} f_n^r - F \right) \tag{19}$$

with a dual factor $\beta > 0$ and $f_n^r \leqslant F$.

We now have $\min_{\boldsymbol{f}>0} L(\boldsymbol{f},\beta) \leqslant \sum_{n\in\mathcal{N}_{\rm Off}} \lambda_n^t C_n/f_n^r$. To make $\min_{\boldsymbol{f}>0} L(\boldsymbol{f},\beta)$ to approach $y(\boldsymbol{f})$, the problem (19) is going to be $\min_{\boldsymbol{f}} y(\boldsymbol{f}) = \max_{\beta>0}\min_{\boldsymbol{f}>0} L(\boldsymbol{f},\beta)$. By the first derivative of $L(\boldsymbol{f},\beta)$ with respect to f_n^r, then we can get the optimal solution:

$$f_n^r = \sqrt{\lambda_n^t C_n/\beta}. \tag{20}$$

We defined $\mathcal{J}(\beta) = \min_{\boldsymbol{f}>0} L(\boldsymbol{f},\beta)$ and substituted (20) into (19). We now have $\mathcal{J}(\beta) = 2\sum_{n\in\mathcal{N}_{\rm Off}} \sqrt{\lambda_n^t C_n\beta} - \beta F$. We then obtain the first derivative of $\mathcal{J}(\beta)$ with respect to β and set its result to 0. We have

$$\sqrt{\beta} = \sum_{n\in\mathcal{N}_{\rm Off}} \sqrt{\lambda_n^t C_n}/F. \tag{21}$$

Combining (20) and (21), we get the optimal solution:

$$f_n^{r\ell} = \frac{F\sqrt{\lambda_n^t C_n}}{\sum_{n\in\mathcal{N}_{\rm Off}}\sqrt{\lambda_n^t C_n}}. \tag{22}$$

When $a=1$, it becomes more complicated to solve the original problem $\mathcal{P}$ directly. Thus, we rewrite the original problem according to the solution of $\mathcal{P}'$.

Combining (9) and (22), we know the execution time t_n^r of each task is

$$t_n^r = C_n/f_n^r = F_0 R\sqrt{C_n/\lambda_n^t}, \quad n\in\mathcal{N}_{\rm Off}, \tag{23}$$

where $F_0 = 1/(F+\varphi F)$ and $R = \sum_{n\in\mathcal{N}_{\rm Off}}\sqrt{\lambda_n^t C_n}$.

We rank the execution time of tasks from small to large. Then, we have

$$R\sqrt{C_1/\lambda_1^t} \leqslant R\sqrt{C_2/\lambda_2^t} \leqslant \cdots \leqslant R\sqrt{C_k/\lambda_k^t}, \tag{24}$$

where $k = \sum_{n=1}^{N} x_n = N_{\rm Off}$. Then, we define $\mathcal{K} = \{1,2,\ldots,k\}$.

Thus, we now know the maximum time $t^{\max}$ required by the MEC server to execute tasks:

$$t^{\max} = F_0 R\sqrt{C_k/\lambda_k^t}. \tag{25}$$

Thus problem $\mathcal{P}$ becomes

$$\begin{aligned}
\mathcal{P}'': \quad & \min_{\boldsymbol{f}} \left\{ y(\boldsymbol{f}) = \sum_{n\in\mathcal{N}_{\rm Off}} \frac{\lambda_n^t C_n}{f_n^r} + \alpha\frac{C_k}{f_k^r} \right\} \\
\text{s.t.} \quad & \text{D1}: \sum_{n\in\mathcal{N}_{\rm Off}} f_n^r \leqslant (F+\varphi F), \\
& \text{D2}: f_n^r > 0, \quad \forall n\in\mathcal{N}_{\rm Off}, \\
& \text{D3}: \frac{C_k}{f_k^r} = \max_n \left\{ \frac{C_n}{f_n^r} \right\}, \quad \forall n\in\mathcal{N}_{\rm Off}, \\
& \text{D4}: \frac{C_k}{f_k^r} \leqslant T_0.
\end{aligned} \tag{26}$$

Rewriting the time weight λ_k^t to $\lambda_k^t + \alpha$, we get a new time weight sequence $\lambda_n^{t'}$. Then, problem (26) can be rewritten as

$$\mathcal{P}'': \quad \min_{\boldsymbol{f}} \left\{ y(\boldsymbol{f}) = \sum_{n\in\mathcal{N}_{\rm Off}} \frac{\lambda_n^{t'} C_n}{f_n^r} \right\}. \tag{27}$$

We know from (18) to (23) that the execution time t_n^r of each task is

$$t_n^r = F_0 R'\sqrt{C_n/\lambda_n^{t'}}, \quad n\in\mathcal{N}_{\rm Off}, \tag{28}$$

where $R' = \sum_{n\in\mathcal{N}_{\text{Off}}} \sqrt{\lambda_n^{t'} C_n}$.

If $R'\sqrt{C_k/(\lambda_k^t+\alpha)} \geqslant R'\sqrt{C_{k-1}/\lambda_{k-1}^t}$ (i.e., $t_k^r \geqslant \max_n\{\frac{C_n}{f_n^r}\}, n \in \mathcal{N}_{\text{Off}}$), the result will satisfy condition D3, and we can find the best allocation of resources. However, there are two more cases caused by condition D4.

(a) If $t_k^r \leqslant T_0$, the MEC server can execute all tasks meeting the specified overclocking constraint. Thus, the optimal solution of problem $\mathcal{P}''$ is

$$f_n^{r'} = \frac{(F+\varphi F)\sqrt{\lambda_n^{t'} C_n}}{\sum_{n\in\mathcal{N}_{\text{Off}}} \sqrt{\lambda_n^{t'} C_n}}. \tag{29}$$

(b) If $t_k^r > T_0$, when the working time of the MEC server reaches T_0, the remaining unprocessed tasks must be processed with the MEC server in a non-overclocked state. Its resource allocation method is based on (22).

If $R'\sqrt{C_k/(\lambda_k^t+\alpha)} < R'\sqrt{C_{k-1}/\lambda_{k-1}^t}$, the maximum time of task executed on the MEC server is

$$t^{\max} = C_{k-1}^{k-1}/f_{k-1}^r = F_0 R'\sqrt{C_{k-1}/\lambda_{k-1}^t} > C_k^k/f_k^r. \tag{30}$$

Because Eq. (30) does not satisfy condition D3, we cannot directly choose this resource allocation strategy. For this case, it is difficult to find the optimal strategy. We thus use a heuristic algorithm to find a sub-optimal solution. We record the computational overhead $y_k(\boldsymbol{f})$ under the current resource allocation strategy to prepare for the comparison as follows:

$$y_k(\boldsymbol{f}) = \sum_{n\in\mathcal{N}_{\text{Off}}} \frac{\lambda_n^t C_n}{f_n^{r'}} + \alpha F_0 R'\sqrt{C_{k-1}/\lambda_{k-1}^t}. \tag{31}$$

We set $i \in \mathcal{K}, i \neq k$ and rewrite the problem (26) as

$$\mathcal{P}'': \quad \min_{\boldsymbol{f}} \left\{ y(\boldsymbol{f}) = \sum_{n\in\mathcal{N}_{\text{Off}}} \frac{\lambda_n^t C_n}{f_n^r} + \alpha\frac{C_i}{f_i^r} \right\}. \tag{32}$$

As Eq. (26) evolved into (27), we have

$$\mathcal{P}'': \quad \min_{\boldsymbol{f}} \left\{ y(\boldsymbol{f}) = \sum_{n\in\mathcal{N}_{\text{Off}}} \frac{{\lambda_n^t}^* C_n}{f_n^r} \right\}. \tag{33}$$

In this case, the execution time t_n^r of each task $\boldsymbol{I}_n$ will be

$$t_n^r = F_0 R^*\sqrt{C_n/{\lambda_n^t}^*}, \quad n \in \mathcal{N}_{\text{Off}}, \tag{34}$$

where $R^* = \sum_{n\in\mathcal{N}_{\text{Off}}} \sqrt{{\lambda_n^t}^* C_n}$.

We still have $R^*\sqrt{C_i/\lambda_i^t} \leqslant R^*\sqrt{C_k/\lambda_k^t}$, $i \in \mathcal{K}$, $i \neq k$. For task $\boldsymbol{I}_i$, we have $F_0 R^*\sqrt{C_i/\lambda_i^t} > F_0 R^*\sqrt{C_i/(\lambda_i^t+\alpha)}$. Combing the two cases, we get $R^*\sqrt{C_k/\lambda_k^t} > R^*\sqrt{C_i/(\lambda_i^t+\alpha)}$.

There is no doubt that the maximum execution time is

$$t^{\max} = C_k^k/f_k^r = F_0 R^*\sqrt{C_k/\lambda_k^t}. \tag{35}$$

In this case, the result of resource allocation is

$$f_n^{r*} = \frac{(F+\varphi F)\sqrt{\lambda_n^{t*} C_n}}{\sum_{n\in\mathcal{N}_{\text{Off}}} \sqrt{\lambda_n^{t*} C_n}}. \tag{36}$$

Thus we have

$$y_i(\boldsymbol{f}) = \sum_{n\in\mathcal{N}_{\text{Off}}} \frac{\lambda_n^t C_n}{f_n^{r*}} + \alpha F_0 R^*\sqrt{C_k/\lambda_k^t}, \quad i \in \mathcal{K}, i \neq k.$$

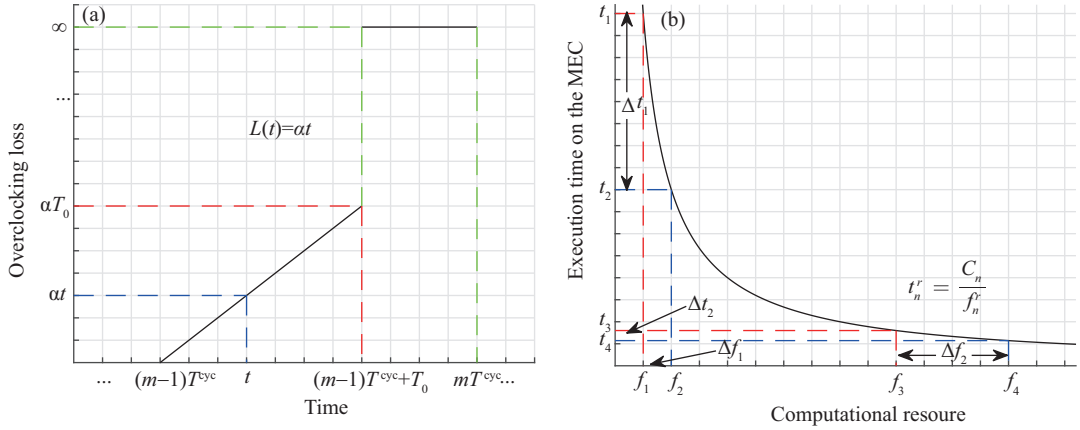

Figure 3 (Color online) (a) Loss function $L(t)$ vs. time t; (b) task allocation resource f_n^r vs. processing time t_n^r.

We then rewrite R^* and $y_i(\boldsymbol{f})$ into the following forms:

$$R^* = R + \sqrt{(\lambda_i^t + \alpha)C_i} - \sqrt{\lambda_i^t C_i}, \quad i \in \mathcal{K}, i \neq k, \tag{37}$$

$$y_i(\boldsymbol{f}) = R^* F_0 \left(R^* - \frac{\alpha\sqrt{C_i}}{\sqrt{\lambda_t^i + \alpha}} \right) + \alpha F_0 R^* \sqrt{C_k/\lambda_k^t}, \quad i \in \mathcal{K}, i \neq k. \tag{38}$$

Generally, $(\sqrt{(\lambda_i^t + \alpha)C_i} - \sqrt{\lambda_i^t C_i}) \ll R^*, i \in \mathcal{K}, i \neq k$, and $y_i(\boldsymbol{f})$ is monotonically decreasing for the variable $\sqrt{C_i/\lambda_i^t}, i \in \mathcal{K}, i \neq k$. Thus, we propose the following algorithm to ensure the superiority of the resource allocation strategy to a certain extent. We set a threshold value of s. If $s \geqslant k-1$, we figure out the computational overhead $y_i(\boldsymbol{f}), i \in \mathcal{K}, i \neq k$ of problem $\mathcal{P}$ in all cases. We form a set $\mathcal{Y} = \{y_1(\boldsymbol{f}), y_2(\boldsymbol{f}), \ldots, y_{k-1}(\boldsymbol{f})\}$. If $s < k-1$, we figure out the computational overhead $y_i(\boldsymbol{f}), i \in \{k-s, k-s+1, \ldots, k-1\}$ of problem $\mathcal{P}$, and form a set $\mathcal{Y} = \{y_{k-s}(\boldsymbol{f}), y_{k-s+1}(\boldsymbol{f}), \ldots, y_{k-1}(\boldsymbol{f}), y_k(\boldsymbol{f})\}$. We then compare the computational overhead of the MEC server in all cases and find the smallest $y_i(\boldsymbol{f})$ (i.e., $i = \text{argmin}_{i \in \mathcal{M}} \mathcal{Y}$. And where $s \geqslant k-1$, $\mathcal{M} = \mathcal{K}$. Otherwise, $\mathcal{M} = \{k-s, k-s+1, \ldots, k-1\}$). We update the i of (36), and the result is the sub-optimal solution of the case.

Just like case (2) in Lemma 1, because of condition D4, there are still two usable cases. Thus, the final resource allocation strategy is

$$f_n^r = \begin{cases} \dfrac{(F+\varphi F)\sqrt{\lambda_n^{t*} C_n}}{\sum_{n \in \mathcal{N}_{\text{Off}}} \sqrt{\lambda_n^{t*} C_n}}, & 0 \leqslant t \leqslant T_0, \\ \dfrac{(F+\varphi F)\sqrt{\lambda_n^{t} C_n}}{\sum_{n \in \mathcal{N}_{\text{Off}}} \sqrt{\lambda_n^{t} C_n}}, & T_0 < t. \end{cases} \tag{39}$$

4.3 Overclocking decision

For this decision, we need to judge whether overclocking is beneficial. When

$$\min_{\boldsymbol{x},\boldsymbol{f}} \sum_{n \in \mathcal{N}} U_n(x_n, 0, f_n^r) > \min_{\boldsymbol{x},\boldsymbol{f}} \left\{ \sum_{n \in \mathcal{N}} U_n(x_n, 1, f_n^r) + L(t) \right\}, \tag{40}$$

we will decide to start overclocking.

Because the MEC server is almost always working, in Figure 3(a), only when $mT^{\text{cyc}} \leqslant t \leqslant (T_0 + mT^{\text{cyc}})$, $m = \{0, 1, \ldots\}$, the MEC server can work in the overclocked state. The working time of the server in the overclocking state cannot exceed time $T_0 + mT^{\text{cyc}} - t$. When $(T_0 + mT^{\text{cyc}}) < t < (m+1)T^{\text{cyc}}$, the MEC service cannot be overclocked, and this period becomes the recovery period of the MEC server.

In this system, when the number of UE is relatively small, and the computing resources of the MEC server are rich, each task is assigned with more resources. If we begin overclocking, each task will get more

resources, as shown in Figure 3(b), just as f_3 to f_4. However, the benefit Δt_2 is not obvious. Meanwhile, the overclocking loss is relatively large compared to the total computational overhead of the system, even bringing negative benefits. Thus, in this case, MEC servers should not be overclocked. When the number of UE increases, the MEC server's computational resources will be short. If we start overclocking, each task will receive more resources, as shown in Figure 3(b), just as f_1 to f_2. Then, we can see that the benefit Δt_1 is quite obvious. We also see that the overclocking loss is relatively small compared to the total computational overhead of the system. Thus, the MEC server should be overclocked. Overclocking is required only when there is a large number of tasks, which concurs with the actual situation. Therefore, the value of overclocking decision a must be determined by combining the current server state and the overclocking overhead.

4.4 Algorithm and complexity analysis

The details of the proposed algorithm are summarized in Algorithm 1.

Algorithm 1 Joint optimization for offloading decision, overclocking decision, and computation resource allocation (JOOC)

1: **Initialization:**
2: Make $\boldsymbol{x} = \{x_n = 1 \mid \forall n \in \mathcal{N}\}$, $\mathcal{N}_{\text{Off}} = \mathcal{N}$.
3: **Iteration:**
4: **for** $a \leftarrow 0$ to 1 **do**
5: Get $\boldsymbol{f}$ according to (16), (17) or (39).
6: **if** $\exists n \in \mathcal{N}_{\text{Off}}$, $U_n^r > U_n^l$ **then**
7: **Repeat**
8: $i^{\#} \leftarrow \arg\min_{n \in \mathcal{N}_{\text{Off}}} \{x_n(U_n^l - U_n^r)\}$;
9: $\mathcal{N}_{\text{Off}} \leftarrow \mathcal{N}_{\text{Off}} - \{i^{\#}\}$;
10: **Update** $\boldsymbol{x}$ by setting $x_{i^{\#}} \leftarrow 0$;
11: **Update** $\boldsymbol{f}$ according to (16), (17) or (39);
12: **Until** $U_n^r < U_n^l$, $\forall n \in \mathcal{N}_{\text{Off}}$ or $\mathcal{N}_{\text{Off}} = \phi$;
13: **end if**
14: **if** $\exists n \in \mathcal{N}_{\text{Off}}$, $(t_n^p + t_n^r) > T_n^{\max}$ **then**
15: **Repeat**
16: $i^{\#} \leftarrow \arg\min_{n \in \mathcal{N}_{\text{Off}}} \{x_n(T_n^{\max} - (t_n^p + t_n^r))\}$;
17: $\mathcal{N}_{\text{Off}} \leftarrow \mathcal{N}_{\text{Off}} - \{i^{\#}\}$;
18: **Update** $\boldsymbol{x}$ by setting $x_{i^{\#}} \leftarrow 0$;
19: **Update** $\boldsymbol{f}$ according to (16), (17) or (39);
20: **Until** $(t_n^p + t_n^r) < T_n^{\max}$, $\forall n \in \mathcal{N}_{\text{Off}}$ or $\mathcal{N}_{\text{Off}} = \phi$;
21: **end if**
22: **end for**
23: **Update** a according to (40);
24: **Output:** the optimal solution $\{\boldsymbol{x}, a, \boldsymbol{f}\}$.

The main loop parts of Algorithm 1 are lines 6–13 and 14–21. The role of these two loops is to delete the elements in the set $\mathcal{N}_{\text{Off}}$, which do not meet the corresponding conditions. Currently, there are three cases for the set $\mathcal{N}_{\text{Off}}$. All elements in $\mathcal{N}_{\text{Off}}$ meet the conditions; only some elements in $\mathcal{N}_{\text{Off}}$ meet the conditions; and all elements in $\mathcal{N}_{\text{Off}}$ do not meet the conditions. Line 6 (14) and line 12 (20) in Algorithm 1 correspond to the judgments of these three cases. Thus, Algorithm 1 is convergent.

In JOOC, every iteration aims to get a better offloading decision and resource allocation strategy. The complexity of optimizing the part of the offloading decision is $\mathcal{O}(N)$. Finding a better resource allocation strategy is divided into three cases. In (18), we simply use the duality method to obtain a result. Thus, the complexity is $\mathcal{O}(1)$ in (27). We must calculate a sequence $\sqrt{\frac{C_n}{\lambda_n^t}}$ ranging from large to small, and then use the dual model to solve the problem. Thus, the complexity is $\mathcal{O}(N \log_2 N)$. In problem (33), we must find the minimum value of $y_i(\boldsymbol{f})$ $(y_i(\boldsymbol{f}) \in \mathcal{Y})$, so that the complexity is $\max\{\mathcal{O}(N \log_2 N), \mathcal{O}(N)\}$. On the whole, the complexity of each iteration is $\mathcal{O}(N \log_2 N)$. To obtain the final offloading decision and feasible resource allocation strategy, we must iterate N times at most. Then, the complexity of overclocking decision is $\mathcal{O}(1)$. Thus, the complexity of the whole algorithm is $\mathcal{O}(N^2 \log_2 N)$. If we want

Table 1 The simulation parameters

Parameter name	Parameter value	Unit
Bandwidth W	2.5×10^7	Hz
Transmission power P_n	20	dBm
Nose power N_0	-85	dBm
CPU frequency of UEs f_n^l	0.5	GHz
Total CPU frequency F	20	GHz
Task CPU cycles C_n	[0.5, 0.7]	Gigacycle
Task date size D_n	$[20, 350]\times 10^3$	kB
Task QoS $T^{\max}$	[1.0, 1.1]	s
Time weighting factor λ_n^t	0.5	–
Energy weighting factor λ_n^e	0.5	–
Growth rate of loss function α	0.3	–

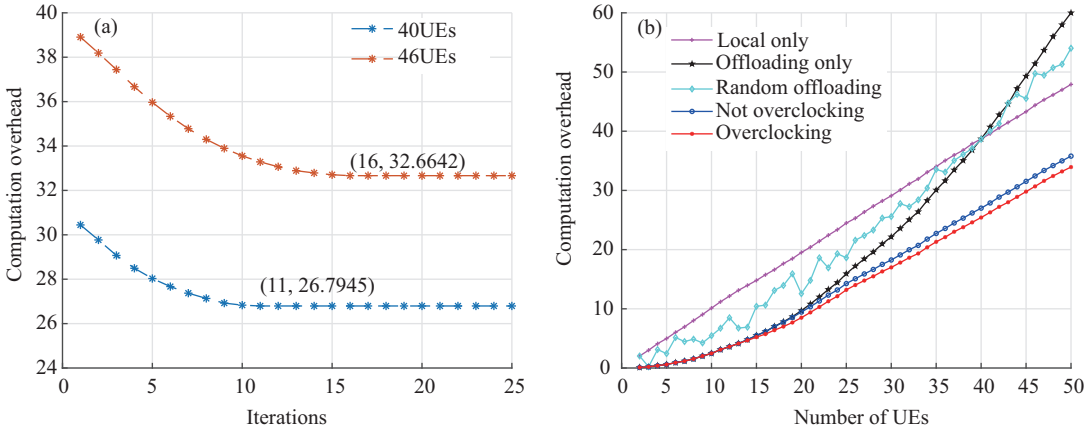

Figure 4 (Color online) (a) Computation overhead vs. the number of iterations; (b) computational overhead vs. the number of UEs.

to get an optimal solution of the original problem, we must use a traversal algorithm, in which the complexity of the algorithm is at least $\mathcal{O}(2^N)$.

5 Simulation results

In this section, we demonstrate the advantages of our algorithm with offloading tasks via experiment simulation. Table 1 shows the parameter settings of our experiment.

Figure 4(a) shows that, under different numbers of UE, the computation overhead of the system decreases with the increasing number of algorithm iterations and finally stabilizes. It is clear that the system computation overhead is minimized after 11 and 16 iterations when the number of UE is 40 and 46, respectively. This shows that the algorithm is convergent and can get better results after a finite number of iterations.

In the second experiment, we set the number of UE from 3 to 50, and we compare the computation overhead of system in different offloading decisions and two states of the MEC servers. As observed from Figure 4(b), when all tasks are executed locally, and their computational overhead is relatively large. If all tasks are executed at the MEC server, the computational overhead is small in the case of a small number of UE, because the MEC server resources and bandwidth resource are abundant. However, when the number of UE increases, considering the limited resources, the processing time of tasks will sharply increase, leading to a sharp increase in the computational overhead of the system. When the offloading decision is made randomly, the computational overhead of the system falls between the computation overhead executed locally and that executed remotely. For the JOOC algorithm, it is clear that, even

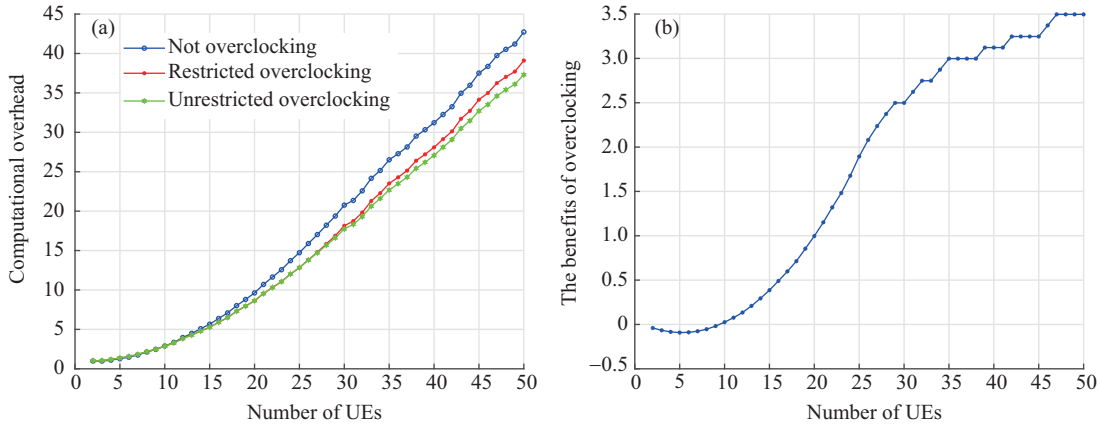

Figure 5 (Color online) (a) Computational overhead in different states; (b) benefit difference between overclocking and non-overclocking.

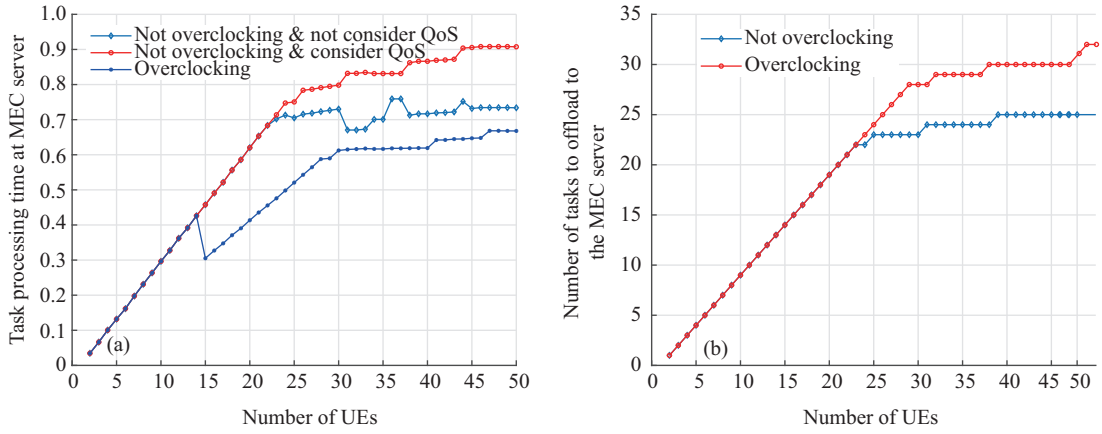

Figure 6 (Color online) (a) Processing time of MEC servers in two states; (b) computational overhead in different states.

when the number of UE is very large, the computational overhead can be accepted. When the number of users is relatively small, the MEC server will choose to offload all tasks to the servers. In this situation, the MEC server having an overclocking capability will not choose overclocking (i.e., $a = 0$), because there is no profit to be obtained. When the number of UE increases, the advantage of an overclocked MEC server gradually increases (i.e., the number of users increases to 14).

From Figure 5, we see that when the number of UE is relatively small, it is not worthwhile for the server to overclock. As the number of UE increases, the benefit becomes apparent when tasks are executed in an overclocked manner. The computational overhead of the system then becomes smaller with the number of users. Overclocking a server incurs extra costs to the server. Thus, to avoid too much cost, the overclocking working time cannot exceed T_0. Comparing the red curve and green one in Figure 5(a), we see that they overlap at the beginning. The red curve becomes slightly higher than the green one. This is because the MEC server working time is short when the task number is small, meaning that the MEC server can work stably in an overclocked state. The MEC server working time increases with the increasing number of tasks. When the working time is greater than T_0, the MEC server cannot always overclock. Thus, the computational overhead of the system will increase slightly, but will do so less if it has no overclocking.

In Figure 6(a), when the MEC server begins overclocking, the server's processing time decreases (i.e., the number of users equals 14). Additionally, the MEC server without overclocking reaches the saturated state when the number of users reaches 24. However, that with the overclocking capability can serve 28. We also can see that, as the number of UE continues increasing, the processing time of the MEC server without overclocking capability can hardly satisfy QoS requirements. Although the processing

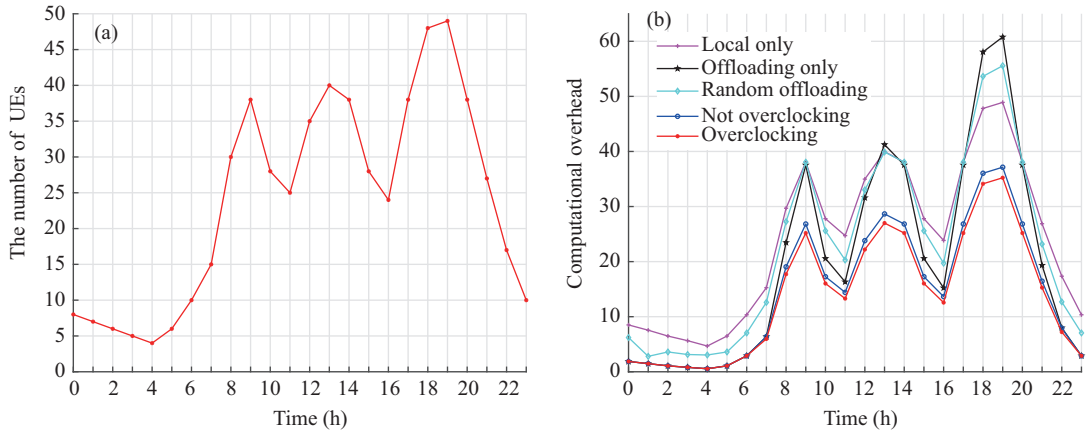

Figure 7 (Color online) (a) The number of UEs changes with time; (b) computational resource overhead vs. time.

time of tasks on the overclocking state is relatively short, QoS can be basically satisfied. As observed from Figure 6(b), the MEC server with overclocking can accommodate more tasks with less computation overhead. It can also be seen from this figure that the MEC server with overclocking reaches its resource saturation state later. Furthermore, the MEC server without overclocking has a low task-offloading rate because of the limitation of computing resources and the requirement of task QoS.

We simulate the number of UE fluctuating at different times in different situations to reflect real life. As shown in Figure 7, from 8:00 to 9:00, from 12:00 to 13:00, or from 18:00 to 19:00, there will be many UE requiring service. However, in the middle of the night or during the early morning, there will be very few. The MEC server with overclocking capability has the best performance and the minimum computing-resource overhead in these scenarios. In the middle of the night, the server chooses not to overclock. The computational overhead of the system in our algorithm is obviously smaller than other methods. With a greater number of users, the advantages of the overclocked MEC server become clear.

6 Conclusion

This paper proposed an intelligent overclocking concept for MEC servers based on the overclocking capabilities of CPUs. For this new MEC model, we proposed an optimization problem for joint UE offloading decisions, computational resource allocations, and overclocking decisions for the MEC server leveraging an MINLP problem. Thus, we have promulgated an iterative algorithm JOOC to obtain a feasible solution. Our simulation results show that our proposed algorithm has lower computational overhead and faster processing task efficiency.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant Nos. 61672395, 61972448, 61911540481), Fund of Hubei Key Laboratory of Inland Shipping Technology (Grant No. NHHY2019004), and National Research Foundation of Korea (NRF) Grant Funded by the Korea Government (MSIT) (Grant No. 2019K2A9A2A060-24389).

References

1 Atat R, Liu L, Chen H, et al. Enabling cyber-physical communication in 5G cellular networks: challenges, spatial spectrum sensing, and cyber-security. IET Cyber-Phys Syst Theor Appl, 2017, 2: 49–54
2 Ning Z, Huang J, Wang X. Vehicular fog computing: enabling real-time traffic management for smart cities. IEEE Wirel Commun, 2019, 26: 87–93
3 Reddy K Y, Gandhi N V D, Balachander K. Simulation and analysis of performance models of broadband intelligent mobile networks. In: Proceedings of 2007 International Conference on Signal Processing, Communications and Networking, 2007. 573–578
4 Jiang W, Strufe M, Schotten H D. Intelligent network management for 5G systems: the selfnet approach. In: Proceedings of 2017 European Conference on Networks and Communications (EuCNC), 2017. 1–5
5 Liu F M, Shu P, Jin H, et al. Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications. IEEE Wirel Commun, 2013, 20: 14–22

6 Chiang M, Zhang T. Fog and IoT: an overview of research opportunities. IEEE Internet Things J, 2016, 3: 854–864
7 Lin J, Yu W, Zhang N, et al. A survey on Internet of Things: architecture, enabling technologies, security and privacy, and applications. IEEE Internet Things J, 2017, 4: 1125–1142
8 Sabella D, Vaillant A, Kuure P, et al. Mobile-edge computing architecture: the role of MEC in the Internet of Things. IEEE Consumer Electron Mag, 2016, 5: 84–91
9 Corcoran P, Datta S K. Mobile-edge computing and the Internet of Things for consumers: extending cloud computing and services to the edge of the network. IEEE Consumer Electron Mag, 2016, 5: 73–74
10 Sun X, Ansari N. EdgeIoT: mobile edge computing for the Internet of Things. IEEE Commun Mag, 2016, 54: 22–29
11 Zhang G, Chen Y, Shen Z, et al. Distributed energy management for multiuser mobile-edge computing systems with energy harvesting devices and QoS constraints. IEEE Internet Things J, 2019, 6: 4035–4048
12 Abbas N, Zhang Y, Taherkordi A, et al. Mobile edge computing: a survey. IEEE Internet Things J, 2018, 5: 450–465
13 Mach P, Becvar Z. Mobile edge computing: a survey on architecture and computation offloading. IEEE Commun Surv Tutorials, 2017, 19: 1628–1656
14 Shan X, Zhi H, Li P, et al. A survey on computation offloading for mobile edge computing information. In: Proceedings of 2018 IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS), 2018. 248–251
15 Kosmides P, Lambrinos L. Intelligent routing in mobile opportunistic networks. In: Proceedings of 2018 Global Information Infrastructure and Networking Symposium (GIIS), 2018. 1–4
16 Alameddine H A, Sharafeddine S, Sebbah S, et al. Dynamic task offloading and scheduling for low-latency IOT services in multi-access edge computing. IEEE J Sel Areas Commun, 2019, 37: 668–682
17 Ning Z, Dong P, Kong X, et al. A cooperative partial computation offloading scheme for mobile edge computing enabled Internet of Things. IEEE Internet Things J, 2019, 6: 4804–4814
18 Wang H, Li X, Ji H, et al. Federated offloading scheme to minimize latency in mec-enabled vehicular networks. In: Proceedings of 2018 IEEE Globecom Workshops (GC Wkshps), 2018. 1–6
19 Bi S, Zhang Y J. Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading. IEEE Trans Wireless Commun, 2018, 17: 4177–4190
20 Li S, Tao Y, Qin X, et al. Energy-aware mobile edge computation offloading for IOT over heterogenous networks. IEEE Access, 2019, 7: 13092–13105
21 Cui L, Xu C, Yang S, et al. Joint optimization of energy consumption and latency in mobile edge computing for Internet of Things. IEEE Internet Things J, 2019, 6: 4791–4803
22 Bu S, Yu F R. Green cognitive mobile networks with small cells for multimedia communications in the smart grid environment. IEEE Trans Veh Technol, 2014, 63: 2115–2126
23 Pham Q V, Le L B, Chung S H, et al. Mobile edge computing with wireless backhaul: joint task offloading and resource allocation. IEEE Access, 2019, 7: 16444–16459
24 Siddique U, Tabassum H, Hossain E, et al. Wireless backhauling of 5G small cells: challenges and solution approaches. IEEE Wirel Commun, 2015, 22: 22–31
25 Ge X, Cheng H, Guizani M, et al. 5G wireless backhaul networks: challenges and research advances. IEEE Netw, 2014, 28: 6–11
26 Tran T X, Pompili D. Joint task offloading and resource allocation for multi-server mobile-edge computing networks. IEEE Trans Veh Technol, 2019, 68: 856–868
27 Zhang J, Hu X, Ning Z, et al. Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks. IEEE Internet Things J, 2018, 5: 2633–2645
28 Thomas D, Shanmugasundaram M. A survey on different overclocking methods. In: Proceedings of the 2nd International Conference on Electronics, Communication and Aerospace Technology (ICECA), 2018. 1588–1592
29 Wu F, Chen J, Dong Y, et al. Improve energy efficiency by processor overclocking and memory frequency scaling. In: Proceedings of 2018 IEEE 20th International Conference on High Performance Computing and Communications, IEEE 16th International Conference on Smart City, IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2018. 960–967
30 Jang H B, Lee J, Kong J, et al. Leveraging process variation for performance and energy: in the perspective of overclocking. IEEE Trans Comput, 2014, 63: 1316–1322
31 Short M, Sheikh I. Dual-rate overclocking in can networks: a soft-core controller prototype. In: Proceedings of 2010 Seventh International Conference on Networked Sensing Systems (INSS), 2010. 314–317
32 Zhao K, Li J P, Ma J, et al. Overclocking NAND flash memory I/O link in LDPC-Based SSDs. IEEE Trans Circ Syst II, 2014, 61: 885–889
33 Wang C, Yu F R, Liang C, et al. Joint computation offloading and interference management in wireless cellular networks with mobile edge computing. IEEE Trans Veh Technol, 2017, 66: 7432–7445