TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 09/18 pp93–104
DOI: 10.26599/TST.2021.9010087
Volume 28, Number 1, February 2023

Multiuser Computation Offloading for Long-Term Sequential Tasks in Mobile Edge Computing Environments

Huanhuan Xu, Jingya Zhou*, Wenqi Wei, and Baolei Cheng

Abstract: Mobile edge computing has shown its potential in serving emerging latency-sensitive mobile applications in ultra-dense 5G networks via offloading computation workloads from the remote cloud data center to the nearby network edge. However, current computation offloading studies in the heterogeneous edge environment face multifaceted challenges: Dependencies among computational tasks, resource competition among multiple users, and diverse long-term objectives. Mobile applications typically consist of several functionalities, and one huge category of the applications can be viewed as a series of sequential tasks. In this study, we first proposed a novel multiuser computation offloading framework for long-term sequential tasks. Then, we presented a comprehensive analysis of the task offloading process in the framework and formally defined the multiuser sequential task offloading problem. Moreover, we decoupled the long-term offloading problem into multiple single time slot offloading problems and proposed a novel adaptive method to solve them. We further showed the substantial performance advantage of our proposed method on the basis of extensive experiments.

Key words: mobile edge computing; sequential tasks; computation offloading; dependency

1 Introduction

The rapid growth of mobile devices (MDs) has brought us to the era of network interconnections of all things, and the demand for low-latency mobile applications is soaring. However, the computational capabilities of MDs are insufficient to support the real-time requirements of these applications because of MDs' inherent limitations, such as limited battery capacity, limited context awareness, and system architecture[1, 2]. For example, to ensure the quality of service and prevent the users from feeling dizzy and sick, the virtual reality system not only needs to transmit high-resolution images with a frame rate of more than 120 frames per second but also needs to complete computation-intensive tasks, such as target recognition and virtual holographic projection modeling[3].

As a promising trend, mobile edge computing (MEC) is capable of handling the challenging requirements of these emerging applications. To provide low-latency services for users, MEC allows users to offload their tasks to nearby MDs through a wireless access network.

By making full use of idle resources around users, MEC builds a carrier-grade service environment that enables users to enjoy an elastic, high-quality network experience[4]. However, compared with centralized data centers, edge clouds usually cannot provide powerful computation capacity because MDs are lightweight, and the wireless resources are limited and can be jammed when many users try to offload their tasks simultaneously.

• Huanhuan Xu, Jingya Zhou, and Baolei Cheng are with the School of Computer Science and Technology, Soochow University, Suzhou 215006, China. E-mail: 20195227020@stu.suda.edu.cn; jy_zhou@suda.edu.cn; chengbaolei@suda.edu.cn.
• Jingya Zhou is also with the State Key Laboratory of Mathematical Engineering and Advanced Computing, Wuxi 214125, China, and the Provincial Key Laboratory for Computer Information Processing Technology, Soochow University, Suzhou 215006, China.
• Wenqi Wei is with the School of Computer Science, Georgia Institute of Technology, Atlanta, GA 30332, USA. E-mail: wenqiwei@gatech.edu.
* To whom correspondence should be addressed.
Manuscript received: 2021-10-25; accepted: 2021-11-08

Therefore, the limited resources must be jointly managed to exploit the potential of MEC.

As a key component in MEC, computational offloading allocates computational tasks from MDs to edge servers. Computation offloading has a significant impact on system performance and can alleviate MDs' deficiencies in energy efficiency, resource storage, and computational performance[5–7]. By looking for a nearby MEC server for task offloading, mobile users with limited resources can save energy and optimize their experience[5].

Numerous efforts have been exerted in developing efficient computation offloading strategies, aiming to improve the system performance of MEC. They have reduced energy consumption and processing delay effectively or made a trade-off between them by selecting task offloading decisions flexibly[8–10]. For instance, by adopting a graph to model the dependencies among tasks, the task offloading strategy proposed in Ref. [6] optimizes the application delay. The task scheduling strategy is put forward for local and edge offloading in Ref. [11] by utilizing the Markov decision process method and achieves a short average execution delay.

To utilize computation offloading efficiently in MEC, many issues remain in real mobile application scenarios:

(1) **Heterogeneity:** MEC servers are typically lightweight and heterogeneous, which means they are along with different data formats, latency requirements, computation resources, and management. These factors explain why MEC servers are massively deployed on edge networks and often come from multiple service providers. Thus, they have different performance in accomplishing data storage, computation, and transmission. The heterogeneity of MEC servers leads to high uncertainties in task execution cost and task execution efficiency and becomes a bottleneck for computational tasks.

(2) **Multiuser:** Most existing studies focus on single-user task offloading and ignore the resource competition problem in multiuser scenarios. Specifically, the competition among MDs leads to low wireless rates in the network and long queuing times on edge servers. Therefore, developing an effective offloading strategy is necessary to meet the user demand and improve the utilization of the computational resources of MEC servers.

(3) **Dependency:** Many mobile applications, such as online games, consist of a set of consecutive dependent tasks. However, existing works pay attention to the computational offloading of independent tasks rather than dependent tasks and fail to address the task-dependency challenge. Moreover, multiuser scenarios make the offloading problem more challenging than usual. For example, Google Project Glass is equipped with augmented reality technology that includes the following functional modules: Video capture, video parsing, target recognition, content mapping, video synthesis, and real-time display. These functions must be executed in strictly sequential order. Among them, video parsing and content mapping put high demand on computational resources, and offloading them to edge servers can effectively reduce the processing latency of the whole application.

(4) **Long-term execution:** Most existing studies are about the static offloading scenario where the optimized offloading strategy is determined before the task execution and ignores the existence of the long-term execution of sequentially dependent tasks. Dynamic events, such as task departure, subsequent task arrival, and computing resource changing, make it even difficult to deal with long-term execution.

To address the above issues, we proposed a multiuser computation offloading framework for long-term sequential tasks (MCO-LST) in MEC environments. Specifically, we focused on adaptive offloading for multiple users' long-term sequential tasks. The offloading model is based on multiple time slots, and in each time slot, the local offloading strategy is optimized adaptively. Note that the sequential tasks are interdependent, and their execution spans multiple time slots so that the optimization of each time slot is not independent of others. Our main contributions are summarized as follows:

(1) We proposed a novel framework MCO-LST for multiple users to optimize the long-term sequential tasks offloading problem in heterogeneous MEC environments. In this framework, we reduced the system delay by optimizing the offloading strategy for multiple time slots. To the best of our knowledge, this study is the first to consider several challenging issues in MEC, including heterogeneity, multiuser, sequential dependency, and long-term execution.

(2) We proposed the method with multiple time slots for further system delay reduction. We converted long-term computational offloading into a multi-time slot optimization problem and optimized the current optimal offloading strategy for each time slot adaptively.

(3) We conducted extensive numerical experiments

to evaluate the effectiveness of our proposed scheme. The results show that it not only reduces the average delay but also improves the system scalability, compared with the conventional offloading strategies and improved Hungarian algorithm without updating a strategy adaptively.

The rest of this paper is summarized below. Section 2 presents the related work. Section 3 introduces the MCO-LST model and the problem formulation. Section 4 describes the adaptive multiuser sequential task (AMUST) offloading algorithm. Section 5 provides the performance evaluations. Finally, Section 6 concludes the paper.

2 Related Work

Many efforts have been devoted to computation offloading problems, and many solutions have been proposed. However, the problem of dependent task offloading in MEC is more complex, and only a few studies have paid attention to the dependency constraints imposed by sequential tasks.

Deng et al.[12] considered a real-world application that comprises a task set and has a delay constraint, and they also proposed a fine-granularity offloading strategy that aims to reduce energy power. Kao et al.[13] studied how to offload dependent tasks under resource constraints and designed an algorithm with a polynomial-time approximation to minimize the maximum completion time. Al-Habob et al.[14] considered parallel and sequential task offloading to multiple MEC servers, and also utilized the algorithms based on genetic algorithms and conflict graph models to minimize offloading latency and failure probability. Zhao et al.[15] jointly considered dependent task offloading and service caching placement to minimize application completion time. Liu et al.[16] considered the problem of dependent task placement and scheduling with on-demand function configuration on a server, and they also proposed an approximate algorithm to minimize the application completion time. However, the above works did not consider the real multiuser edge environment where multiple users fiercely compete for limited marginal resources. The following studies considered the multiuser offloading problem in MEC.

Dai et al.[17] explored the task offloading problem in a data-driven manner and utilized the solution based on an asynchronous deep Q-learning and convex theory to reduce average service time and cost. Fan et al.[18] proposed the offloading problem of multiuser dependent tasks to reduce the total application cost in the system without exceeding the limit for completion of each application. Jošilo and Dán[19] modeled the resource allocation problem as a Stackelberg game in an MEC scenario with wireless devices and proposed an effective decentralized equilibrium algorithm to reduce task completion delay. Bi et al.[20] investigated how to find an optimal offloading strategy to improve the system utility and also analyzed the expectation of time complexity on the basis of Karush-Kuhn-Tucker to derive the optimal scheme. Chen et al.[21] explored the multiuser task offloading problem in an MEC scenario with channel competition and designed a computing offloading algorithm to obtain Nash equilibrium, and they also obtained good computation offloading performance. However, the above works ignore the long-term dynamic characteristic of edge networks, which further enhances the complexity of the offloading problem.

Zhou et al.[22] presented a model to capture the execution of sensing tasks and leveraged the Lyapunov optimization method to obtain the low average cost of the system for the long term. Huang et al.[23] utilized Lyapunov optimization to design a dynamic offloading algorithm, which saves energy and satisfies the given application execution time requirement. Yetim and Martonosi[24] examined the problem of estimating the delay tolerance of applications and used four low-overhead and effective methods to infer the delay tolerance of applications. Mao et al.[25] concentrated on achieving green computing via energy harvesting (EH) technologies, studied a green MEC system equipped with EH devices, and designed a Lyapunov optimization-based task offloading scheme to save energy. However, they assumed that the latency among local processors is negligible. Sundar and Liang[26] investigated the problem of application offloading in a general heterogeneous edge computing scenario where each application is composed of multiple tasks with interdependency, they also proposed a heuristic algorithm to obtain an effective solution.

In this study, each MEC server has a limited workload, and a long-term application delay concerned with data communication, task queuing, and task execution exists, resulting in a particular formulation of the problem that has not been investigated in existing studies.

3 MCO-LST Framework and Problem Formulation

In this section, we first introduce the MCO-LST framework, including the network model, task model,

and task offloading model. Then, we formally define the offloading problem.

3.1 Network model

Figure 1 shows a demonstration of typical sequential tasks offloading in MEC environments where N MDs are represented by set $\mathcal{B} = \{b_1, b_2, \ldots, b_N\}$, and $\mathcal{C} = \{c_1, c_2, \ldots, c_K\}$ is the set of K access points (CAPs) with different computational capabilities. CAPs are interconnected with high-speed networks[4], whereas MD uses CAPs to transfer data via the single-carrier wireless channel of orthogonal frequency division multiple access. The channel gain is affected by shadow fading and path loss. Each CAP can cover all MDs and has an upper limit e on the number of accepted tasks, which is similar to the heterogeneous edge environment with dense MDs.

Each MD has an application to be processed, and set $\mathcal{D} = \{d_1, d_2, \ldots, d_N\}$ represents the mobile applications on all MDs. Each application $d \in \mathcal{D}$ consists of a set $\mathcal{A} = \{a_1, a_2, \ldots, a_M\}$ of M sequential tasks. We adopt a widely used task model to describe the m-th task on the n-th MD as $\pi_{m,n}(c_{m,n}, h_{m,n})$, where $c_{m,n}$ (in unit of bit) represents the size of the input data of the current task, which is also the size of the data output from the previous task, and $h_{m,n}$ represents the number of CPU cycles required for task execution[27, 28]. For example, the core tasks of virtual reality (VR) games require complex calculation, whereas normal calculation tasks, such as general video monitoring, are relatively simple[29]. To be more specific, face recognition may require 2339 (in unit of cycle/bit), whereas video transcoding requires 200–1200 (in unit of cycle/bit)[30].

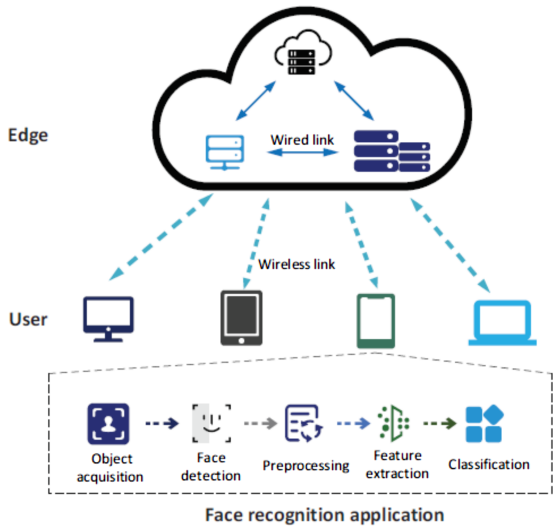

Fig. 1 Demonstration of typical sequential tasks offloading in MEC environments.

In this study, we adopt a system-managed MEC offloading model. This model runs in discrete time slots $\mathcal{T} = \{1, 2, \ldots, T\}$, and the length of each time slot is determined by the service provider. At the beginning of each time slot, the system manager is aware of the current task information and device information in the system and will offload each task to a nearby CAP for execution. Depending on the offloading decision of the previous task, the transmission process of the current task may be different. The CAP may not process the offloaded task after receiving it immediately and instead just place it in a linear queue which follows the first-come, first-serviced rule.

3.2 Task model

We considered that each application on an MD can be partitioned into several tasks with sequential dependencies, which is similar to Ref. [31]. The application is launched from the MD, and results must be returned to it. To model this requirement, we use a one-way chain table to represent the sequential task dependencies of the application, which does not lose the generality of sequential tasks.

At each time slot, each task can be offloaded to a nearby CAP for execution, and we defined the task offloading variable as

$$x_{m,n} \in \{1, 2, \ldots, K\}, m \in \mathcal{A}, \quad n \in \mathcal{B} \tag{1}$$

Specifically, $x_{m,n} = k$ means that the m-th task on the n-th MD is offloaded to the k-th CAP for execution. For ease of presentation, we defined the offloading strategies for all tasks as $\boldsymbol{x} = (x_{m,n})_{m \in \mathcal{A}, n \in \mathcal{B}}$ and the offloading strategy at the t-th time slot as $\boldsymbol{x}(t)$. We defined the set of tasks to be offloaded at the t-th time slot as $\mathcal{S}(t)$. We used the offloading strategy to determine $\mathcal{S}_k(t) \triangleq \{m, n | x_{m,n}(t) = k\}$, which is the set of tasks offloaded to the k-th CAP at the t-th time slot. Specifically, we defined $s_k(t) \triangleq |\mathcal{S}_k(t)|$ as the number of tasks currently offloaded to the k-th CAP. In addition, we defined the CPU cycles of the set of all offloaded tasks to the k-th CAP at the current t-th time slot as $U_k(t)$. Considering that the tasks on CAPs may not be completed immediately, the task queue on the k-th CAP at the t-th time slot is defined as $\mathcal{Q}_k(t)$, which contains all the tasks to be executed.

Based on the task model mentioned above, we formalized the delay costs of the task offloading process, which are coupled with the long-term offloading

strategies.

3.3 Task offloading model

The objective of task offloading in our scenario is to minimize the total latency cost of the applications by offloading sequential tasks on different CAPs with limited computational resources. Next, we present our cost model where each task completion flow comprises three components: Execution, communication, and queuing. To facilitate the discussions, we present notations in Table 1.

Task execution: To analyze the delay caused by the execution, we defined f_k as the computational capacity of the k-th CAP. The execution time of a task on the k-th CAP is not dependent on the time slots but on the offloading strategy, and is calculated as follows:

$$l^{ex}_{m,n}(x_{m,n}) = \frac{h_{m,n}}{f_k}(x_{m,n}) \tag{2}$$

Task communication: At each time slot, the required input data of the offloaded task are transferred depending on the offloading decisions of the precursor task, the current task, and the current local network environment, from which we can calculate the communication delay. The delay is also irrelevant to the time slots. Therefore, according to the communication process, the task transmission delay on the n-th MD time at the t-th time slot is divided into the following two cases:

(1) The input data for task m are already on the k-th CAP, that is, $x_{m-1,n} = x_{m,n}$. In this situation, no data transfer is required, and the communication delay is

$$l^{tr}_{m,n}(x_{m,n}) = 0 \tag{3}$$

Table 1 Notations used in this paper.

Notation	Meaning
$\mathcal{B}$	The set of mobile devices, $\mathcal{B} = \{b_1, b_2, \ldots, b_N\}$
$\mathcal{A}$	The set of tasks to be executed, $\mathcal{A} = \{a_1, a_2, \ldots, a_M\}$
$\mathcal{C}$	The set of computational access points, $\mathcal{C} = \{c_1, c_2, \ldots, c_K\}$
$\mathcal{T}$	The set of time slots of the system, $\mathcal{T} = \{1, 2, \ldots, T\}$
$\pi_{m,n}$	Information of the n-th MD's m-th task
$x_{m,n}$	Integer offloading variable of the n-th MD's m-th task
$L_{m,n}$	Completion time of the n-th MD's m-th task
r	Data rate of wired link among CAPs
$\mathcal{S}_k(t)$	Task set to be offloaded on the k-th CAP at the t-th time slot
$\mathcal{Q}_k(t)$	Task queue of the k-th CAP at the t-th time slot
$\boldsymbol{p}(t)$	Load vector of CAPs at the t-th time slot
$\boldsymbol{q}(t)$	Task priority vector at the t-th time slot

(2) The input data of a task must be transmitted to a CAP for execution, that is, $x_{m-1,n} \neq x_{m,n}$. Given that the fixed wired bandwidth between CAPs is r, the communication delay is

$$l^{tr}_{m,n}(x_{m,n}) = \frac{c_{m,n}}{r}(x_{m,n}) \tag{4}$$

Specifically, considering that the amount of data returned as a result of the final application task is generally small, we ignored the communication delay of the result returned by the last task of the application, similar to Ref. [21].

Task queuing: Considering that tasks may not be executed immediately when they are offloaded to a CAP, we needed to wait for the CAP to finish the previously arrived tasks. Hence, we must consider the task waiting process on the CAP, which is impacted by offloading strategies of all time slots. Let $V_k(t)$ be the CPU cycles of the task queue on the k-th CAP at the t-th time slot, then we had

$$V_k(t+1) = V_k(t) - f_k + U_k(t) \tag{5}$$

The average queuing time of a task on the k-th CAP at the t-th time slot can be expressed as

$$l^{qu}_{m,n}(\boldsymbol{x}) = \frac{V_k(t)}{f_k}(\boldsymbol{x}(t)) + \frac{U_k(t)}{s_k(t) \cdot f_k}(\boldsymbol{x}(t)) \tag{6}$$

Given that the application completion process consists of the execution, communication, and queuing of all tasks, the application completion delay of the n-th MD's m-th task is expressed as

$$L_{m,n}(\boldsymbol{x}) = l^{ex}_{m,n} + l^{tr}_{m,n} + l^{qu}_{m,n} \tag{7}$$

From the above task completion process and formulation, aggressively minimizing delays from one aspect will inevitably bring changes to the delay in other aspects, and this impact will span time slots due to the resource competition and dependencies among tasks. As a result, a natural question is how to adaptively optimize the delay impact of a real-time offload strategy efficiently.

3.4 Problem formulation

We have explained the AMUST offloading problem from the system perspective. Our goal is to minimize the average application completion delay and meanwhile satisfy the real-time requirement. The problem takes changing tasks, wireless access, and CAP resources as inputs to obtain an adaptive offloading strategy for multi-device sequential tasks. Thus, the time is divided into multiple time slots during task offloading. To achieve this goal, the system should optimize the corresponding offloading strategy for each time slot. Then, we gave

the formal definition of the task offloading problem as follows:

$$\min L = \sum_{n=1}^{N} \sum_{m=1}^{M} L_{m,n}(\boldsymbol{x})$$
$$\text{s.t. } x_{m,n} \in \{1, 2, \ldots, K\}, \quad \forall m \in \mathcal{A}, \quad \forall n \in \mathcal{B} \tag{8}$$

Our optimization goal and constraints explicitly represent the interplay among task execution, wire transmission, and server queue processing aspects at each time slot. They also reflect the interdependence among sequential tasks and drive the offloading strategies to optimize the application completion delay over a long period. Finally, they optimize the adaptive offloading goal for multi-device applications with sequential tasks.

The offloading strategies for discrete time slots are tightly tied to task dependencies and optimization objectives. On account of that, this nonlinear integer programming problem includes the generalized assignment problem at each time slot as a special case (for a network without exceeding the CAP load upper limit), and it is NP-hard[32]. Hence, the exact solution cannot be obtained in polynomial time. Moreover, the optimal solution of the formal problem requires complete system information at all time slots, including task set and server queue information at every time slot, which is dynamically changing and extremely difficult to obtain in advance. Thus, designing efficient adaptive offloading strategies is highly desirable. In this way, the strategy can use only the current system information to continuously adjust to changes in the system. Therefore, we proposed an adaptive task offloading algorithm that can efficiently accomplish task offloading at a low-latency cost and does not need global system information.

4 Adaptive Multiuser Sequential Task Offloading

We now present the proposed framework on the basis of task urgency and server load balancing. During the real-time offloading, the complete set of network information and the tasks of the system in a given time slot is unpredictable. To address this challenge, we transformed the global task offloading problem into multiple single time slot adaptive task offloading problems to derive locally optimal offloading strategies. Then, we propose an auxiliary approach on the basis of task prioritization and server load balancing. Accordingly, the system obtains more explicit offloading preferences from the exploration containing sequential task information and updates the current locally optimal strategy to obtain an improved and globally scoped one.

4.1 Local optimal offloading strategy

At each time slot in the system, there may be tasks to be offloaded, and the system manager has to assign appropriate CAP resources to them, which is reflected in the offloading decisions of these tasks. Before the next time slot arrives, only the current system information is available. The delays of task offloading processes and their impacts on the system are available at the end of the current time slot, but the complete information of all the following time slots is unknown until the corresponding offloading decisions are made.

To well represent the variation of system conditions and resource availability, we redefined $\boldsymbol{x}_i(t) = [0, 1, \ldots, 0]^{\mathrm{T}}$ as the offloading variable of the i-th task to be offloaded at a given t-th time slot, where $[\cdot]^{\mathrm{T}}$ denotes the transpose of a vector. $x_{i,j}(t) = 1$ means that the i-th task in the set of tasks is offloaded to the j-th CAP at the t-th time slot; otherwise, $x_{i,j}(t) = 0$.

Considering that a task is served by only one CAP for a given time slot, we had the following constraints:

$$\sum_{j \in \mathcal{C}} x_{i,j}(t) = 1, \forall t \in \mathcal{T}, \forall i \in \mathcal{S}(t) \tag{9}$$
$$x_{i,j}(t) \in \{0, 1\}, \forall i \in \mathcal{S}(t), \forall j \in \mathcal{C} \tag{10}$$

Therefore, the original problem can be rewritten as follows:

$$\min \sum_{t=1}^{T} \sum_{i=1}^{s(t)} L_i(\boldsymbol{x})$$
$$\text{s.t. Eqs. (9) and (10)} \tag{11}$$

where $L_i(\boldsymbol{x})$ is the completion delay of the i-th task, and its calculation is similar to Eq. (7), i.e.,

$$L_i(\boldsymbol{x}) = l_i^{ex} + l_i^{tr} + l_i^{qu} \tag{12}$$

The above problem is a 0-1 integer programming with multiple time slots. Specifically, the offloading decision and optimization objective have a long-term nonlinear relationship. To achieve the goals of real-time adaptive offloading and long-term system cost optimization, we must determine the optimal offloading strategy on the basis of the set of offloading tasks and the network environment at the current time slot. Therefore, at the t-th given time slot, we needed to obtain the current optimization objective:

$$\min \sum_{i}^{s(t)} L_i(\boldsymbol{x})$$
$$\text{s.t. Eqs. (9) and (10)} \tag{13}$$

This task offloading problem is a special 0–1 integer programming that can be converted to an assignment problem. The Hungarian algorithm is always an appropriate solution devoted to these issues, and it can solve the assignment problem simply and effectively[33]. At a given t-th time slot, the number of tasks $s(t)$ and the number of CAP K are probably not equal, and a CAP can accept multiple tasks at the same time. Therefore, the benefit matrix of this assignment problem may not be a square matrix. CAPs can accept multiple tasks at the same time; thus, we introduce virtual tasks to match the requirement of the Hungarian algorithm[34]. The improved Hungarian algorithm for the local offloading strategy has the following steps:

(1) For a local offloading problem at time slot t, we created its cost matrix $\boldsymbol{O}_0(t)(n(t)\cdot n(t))$.

(2) For each row of cost matrix $\boldsymbol{O}_0(t)$, we subtracted the smallest value and obtain $\boldsymbol{O}_1(t)$.

(3) For each row of cost matrix $\boldsymbol{O}_1(t)$, we subtracted the smallest value and obtain $\boldsymbol{O}_2(t)$.

(4) From utilizing the least straight lines to covering all zeros in $\boldsymbol{O}_2(t)$, we obtained $\boldsymbol{O}_3(t)$.

(5) We established a new matrix $\boldsymbol{O}_3(t)$ where we subtract each number that is not covered by lines from the smallest value. Meanwhile, we added the smallest value to the line intersection point, and then we obtained $\boldsymbol{O}_4(t)$.

(6) We gained the local offloading decision from the row or column with the least zeros and obtained the optimal task offloading strategy $\boldsymbol{x}(t)$.

We used Algorithm 1 to depict the detailed procedure. The offloading variables are globally related to the optimization objective. Hence, we must update the system task set on the basis of the current task set $\mathcal{S}(t)$ and offloading variable information $\boldsymbol{x}(t)$. After determining the local optimal offloading strategy at the current time slot, we can enable task offloading at the next time slot, as shown in Algorithm 2. By combining with the task set update process, Algorithm 3 shows the detailed AMUST offloading algorithm.

Algorithm 1 Improved Hungarian based task offloading algorithm

Input: Network environment paramenters, $\mathcal{S}(t)$ and $\mathcal{Q}(t)$;
Output: $\boldsymbol{x}^*(t)$;

if $\mathcal{S}(t)=\varnothing$ **then**
  $\boldsymbol{x}(t)\leftarrow\varnothing$;
else if $\mathcal{S}(t)\neq\varnothing$ **then**
  $s(t)\leftarrow|\mathcal{S}(t)|$;
  $n(t)\leftarrow s(t)\cdot K$;
  if $n(t)>K$ **then**
    add virtual CAPs;
  update cost matrix $\boldsymbol{O}(t)$ according to the improved Hungarian algorithm;
  get optimal $\boldsymbol{x}^*(t)$;
return $\boldsymbol{x}^*(t)$;

Algorithm 2 Task set update algorithm

Input: Network environment paramenters, $\mathcal{S},\pi,\mathcal{Q}(t)$, and $x(t)$;
Output: $\mathcal{S}'$;

if $t=1$ **then**
  $\mathcal{S}(1)\leftarrow\varnothing$;
  For each $i\leqslant M$ **do**
    $\mathcal{S}(1)\leftarrow\mathcal{S}(1)\cup\{\pi_{1,i}\}$;
    calculate $L_{1,i}$ according to Eq. (7);
    $L_{1,i}\leftarrow L_{1,i}$;
    $\mathcal{S}(1+L_{1,i})\leftarrow\mathcal{S}(1+L_{1,i})\cup\{\pi_{2,i}\}$;
else if $t\geqslant 2$ **then**
  if $\mathcal{S}(t)=\varnothing$ **then**
    break;
  else if $\mathcal{S}(t)\neq\varnothing$ **then**
    $s(t)\leftarrow|\mathcal{S}(t)|$;
    For each $i\leqslant s(t)$ **do**
      determine task $\pi_{m,n}$ according to $\mathcal{S}(t)$;
      calculate delay $L_{m,n}$;
      $L_{m,n}\leftarrow\lceil L_{m,n}\rceil$;
      $\mathcal{S}(1+L_{m,n})\leftarrow\mathcal{S}(1+L_{m,n})\cup\{\pi_{m+1,i}\}$;
 $\mathcal{S}'\leftarrow\mathcal{S}$;
return $\mathcal{S}'$;

Algorithm 3 AMUST offloading algorithm

Input: $\mathcal{S},\mathcal{Q}_k(t),\boldsymbol{q}(t)$, and $\boldsymbol{p}(t)$;
Output: $\boldsymbol{x}^*(t)$;

if $\mathcal{S}(t)=\varnothing$ **then**
  $\boldsymbol{x}(t)\leftarrow\varnothing$;
else if $\mathcal{S}(t)\neq\varnothing$ **then**
  $s(t)\leftarrow|\mathcal{S}(t)|$;
  get $\boldsymbol{x}(t)$ according to Algorithm 1;
  calculate $\boldsymbol{q}(t)$ and $\boldsymbol{p}(t)$;
  sort $\mathcal{S}(t)$ according to $\boldsymbol{p}(t)$;
  For each $i\leqslant z$ **do**
    find the smallest $q_k(t)$;
    if $x_i(t)=k$ **then**
      break;
    else if $x_i(t)\neq k$ **then**
      if $L((\boldsymbol{x})'(t))\geqslant L((\boldsymbol{x})(t))$ **then**
        break;
      else if $L((\boldsymbol{x})'(t))<L((\boldsymbol{x})(t))$ **then**
        $x_i(t)\leftarrow k$;
        update the overall cost matrix $\boldsymbol{O}(t)$;
  update $\mathcal{S}$ according to Algorithm 2;
$\boldsymbol{x}^*(t)\leftarrow\boldsymbol{x}(t)$;
return $\boldsymbol{x}^*(t)$;

4.2 Adaptive offloading strategy update

In this section, we design effective auxiliary methods on the basis of task prioritization and server load balancing for the adaptive task offloading of multiuser sequential tasks. The key idea is to properly construct an offloading strategy updating mechanism that fits the scenario and captures the aspects of the task offloading process that will affect the overall goal optimization to a large extent. On the basis of the characteristics of the current model, we can abide by the following steps to update the local optimal offloading strategy for simulating the effect of global information on the overall offloading strategy preference.

First, from the system perspective, we maintained a CAP load vector in each time slot as follows:

$$\boldsymbol{q}(t)=\left[\frac{V_1(t)}{f_1},\frac{V_2(t)}{f_2},\ldots,\frac{V_K(t)}{f_K}\right]^{\mathrm{T}} \tag{14}$$

where we adapt $q_k(t)=\dfrac{V_k(t)}{f_k}$ to represent the load situation of the k-th CAP at the current time slot, which can be updated according to the current network environment and long-term information. f_k does not vary, but $V_k(t)$ will fluctuate with the change of the system state, and its update process abides by Eq. (5).

The load vector provides a good reflection of the computational resources of CAPs in the current system, thus avoiding the phenomenon of blindly offloading tasks to partially overloaded CAPs. This phenomenon can lead to the degradation of CAP performance in the foreseeable future.

From the task perspective, we should obtain the priority of the currently scheduled task at the current time slot, which is closely related to the remaining CPU cycles required by the application in our model. Therefore, we constructed a priority vector over the tasks to be offloaded at time slot t and sorted the tasks in the set $\mathcal{S}(t)$ accordingly:

$$\boldsymbol{p}(t)=[w_1,w_2,\ldots,w_{s(t)}]^{\mathrm{T}} \tag{15}$$

where w_i denotes the CPU cycle sum of the subsequent tasks of the i-th task at the t-th time slot. The scale of this vector varies as the time slot advances, reflecting the nature of the impacts of different offloading strategies on the overall goal optimization in discrete time slots.

To sum up, by combining the above two features that can reveal the optimization aspects affecting the global objective, we can develop an auxiliary method on the basis of task priority and server load balancing to update the obtained locally optimal offloading strategy. After each acquisition of the locally optimal offload strategy, the system changes to traverse former z tasks on the basis of task priority and query the impact on the total system goal by offloading the current tasks with higher priority to the CAP with less load to complete the update once optimized, as shown in Algorithm 3.

In conclusion, in our adaptive task offloading algorithm, the system first evaluates a locally optimal offloading strategy on the basis of task information and the network environment for each time slot. Then, the system updates the current local strategy by utilizing the auxiliary methods on the basis of server load balance and task priority to obtain an improved global optimization goal.

4.3 Feasibility and complexity analysis

Given that our task offloading problem defined in Eq. (8) is NP-hard, the feasibility guarantee cannot be determined in polynomial time. Therefore, we considered a fallback scheme that updates the offloading decisions of all tasks belonging to applications at each time slot. Using this scheme is feasible under the premise of the situation that partial tasks at the same time slot have no significant impact on the costs of other tasks. Hence, the offloading decisions can meet the low-cost requirements for overall tasks. The computational complexity of the fallback scheme is $O(s(t)^3K^4)$. By concluding the time to obtain lower bound offloading strategies, the computational complexity for the AMUST algorithm is $O((z+1)s(t)^3K^3)$, which is polynomial concerning the size of the applications.

5 Experiment

We conducted simulation experiments to evaluate and compare the performance of AMUST with other benchmark methods. The experiments are conducted on a MacBook Pro configured with a 2-GHz Intel Core i5 quad-core processor.

5.1 Experiment settings

For most scenarios, the following parameter settings are used. We assumed that the area is a coverage area of 250 m × 250 m, which contains 60 MDs and 9 CAPs with embedded edge servers. Each MD has an application and each application has five tasks to be executed, the task size follows a uniform distribution of [600, 1800] Kbit, the number of required CPU cycles follows a random distribution of [300, 1500] megacycles, and the computational capacities of the CAPs are

selected from {2, 3, 6} GHz[35]. The locations of the CAPs follow a uniform distribution, and the CAPs are interconnected by wired links. For the communication model, the noise power was set to $r = 6$ Gbps[36].

Specifically, we compared the performance of AMUST with the following benchmark schemes:

Random offloading scheme (ROS). It randomly assigns tasks to arbitrary CAPs for execution.

Greedy offloading with load balance (GOLB). It always assigns tasks to the CAP with the lowest workload.

Task offloading without adaptive updating (TOw/AU). It does not update adaptively after utilizing the improved Hungarian algorithm at each time slot.

5.2 Results

We evaluated the impact of the number of MDs by changing it from 20 to 100 as shown in Fig. 2. The average delay of all algorithms shows a growing trend. Among them, AMUST always achieves the best performance. With the growth of MDs, the average delay increases gradually. The difference in delay between GOLB and AMUST decreases from 20.2% at 20 MDs to 7.8% at 100 MDs. This is because AMUST gives preference to the task queuing process, which is the main reference item considered in GOLB. By contrast, the delay involves additional queuing time due to the fierce competition for the limited CAP resources when the number of MDs increases from 80 to 100.

We noticed that AMUST's data transmission time was longer than TOw/AU's when the number of MDs reaches 100. The reason was that TOw/AU does not update the offloading strategies for time slots, leading to less transmission delay. Meanwhile, AMUST considers data transmission and task queuing. Although AMUST has a long data transmission process, its queuing delay is significantly lower than other algorithms. Thus, compared with other algorithms, AMUST achieves the lowest overall delay.

In Fig. 3, we compare the performance in the aspect of average delay when the number of CAPs increases from 3 to 15. The application delay gradually decreases as the number of CAPs increases. The difference in the performance between ROS and AMUST increases from 26.5% at 3 CAPs to 57.8% at 15 CAPs. AMUST can choose the number CAP with a low workload more deftly, thus avoiding offloading tasks to overburdened CAPs. Moreover, when the number of CAPs reaches 9, the addition of CAPs in the system reduces the delay by a slight margin. In this case, the system resources are sufficient for the current task scale, and the resource competition weakens. Compared with other algorithms, the advantage of AMUST is mainly reflected in the reduction of the combination of data transmission and task execution delay.

Figure 4 shows the comparison of average delay by changing the number of tasks at each mobile application. The large number of tasks reflects the complexity of applications in the long-term system. This comparison covers the range from simple to complex applications. The average delay increases as tasks augment. The performance difference between ROS and AMUST increases from 22.3% at three tasks to 43.5% at seven tasks. The reason is that the growth of the task number of each application leads to an increase in offloading tasks for the long term, leading to a more dynamic system. However, AMUST considers CAP load balance and communication process and avoids unnecessary data transfer and unbearably long queuing time.

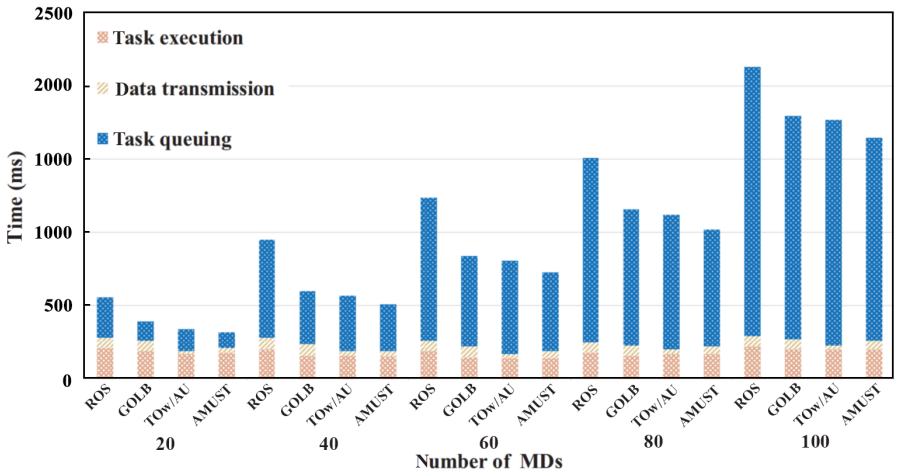

Fig. 2 Impact of the number of MDs on delay.

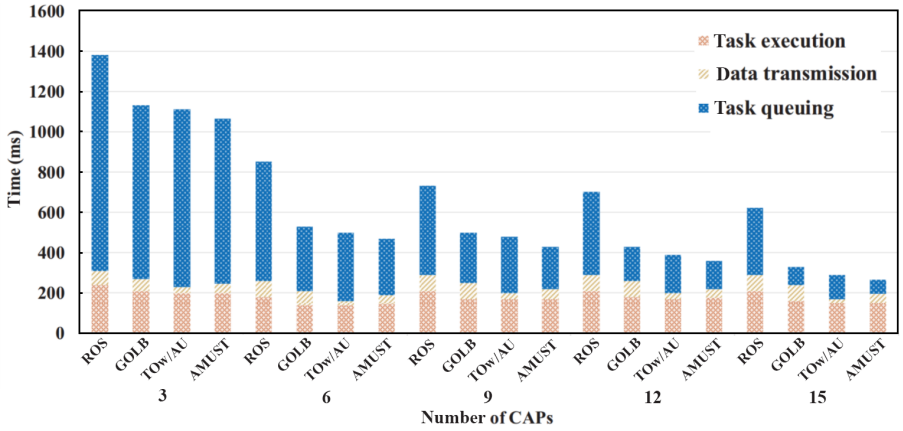

Fig. 3 Impact of the number of CAPs on delay.

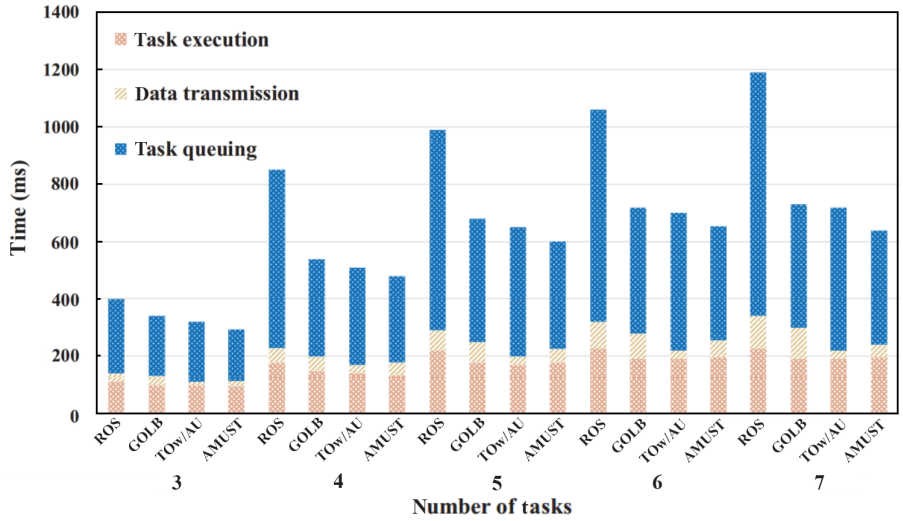

Fig. 4 Impact of the number of tasks on delay.

6 Conclusion

This study investigated the multiuser sequential task offloading problem in heterogeneous MEC environments. Our goal was to minimize the average application delay. To this end, we proposed the MCO-LST framework to deal with offloading for long-term sequential tasks. Specifically, we formally defined the problem. Then, we proposed an effective algorithm AMUST to obtain the optimal offloading strategy. Finally, we conducted extensive experiments to demonstrate the effectiveness of AMUST.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (Nos. 61972272, 62172291, 62072321, and U1905211), the Natural Science Foundation of the Jiangsu Higher Education Institutions of China (No. 21KJA520008), the Open Project Program of the State Key Laboratory of Mathematical Engineering and Advanced Computing (No. 2019A04), the Postgraduate Research & Practice Innovation Program of Jiangsu Province (No. SJCX21_1344), and the Provincial Key Laboratory for Computer Information Processing Technology (No. KJS1740).

References

[1] E. Cuervo, A. Balasubramanian, D. K. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, MAUI: Making smartphones last longer with code offload, in *Proc. 8th Int. Conf. Mobile Systems, Applications, and Services*, San Francisco, CA, USA, 2010, pp. 49–62.

[2] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. W. Zhang, ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading, presented at the 2012 Proc. IEEE INFOCOM, Orlando, FL, USA, 2012, pp. 945–953.

[3] E. Bastug, M. Bennis, M. Medard, and M. Debbah, Toward interconnected virtual reality: Opportunities, challenges, and enablers, *IEEE Commun. Mag.*, vol. 55, no. 6, pp. 110–117, 2017.

[4] N. X. Chen, Y. Yang, T. Zhang, M. T. Zhou, X. L. Luo, and J. K. Zao, Fog as a service technology, *IEEE Commun. Mag.*, vol. 56, no. 11, pp. 95–101, 2018.

[5] B. Yang, X. L. Cao, X. F. Li, Q. Q. Zhang, and L. J. Qian, Mobile-edge-computing-based hierarchical machine learning tasks distribution for IIoT, *IEEE Internet Things J.* vol. 7, no. 3, pp. 2169–2180, 2020.

[6] M. K. Jia, J. N. Cao, and L. Yang, Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing, presented at the 2014 IEEE Conf. Computer Communications Workshops (INFOCOM WKSHPS), Toronto, Canada, 2014, pp. 352–357.

[7] W. W. Zhang, Y. G. Wen, and D. O. Wu, Energy-efficient scheduling policy for collaborative execution in mobile cloud computing, presented at the 2013 Proc. IEEE INFOCOM, Turin, Italy, 2013, pp. 190–194.

[8] J. Wang, J. Hu, G. Y. Min, A. Y. Zomaya, and N. Georgalas, Fast adaptive task offloading in edge computing based on meta reinforcement learning, *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 242–253, 2021.

[9] J. P. Champati and B. Liang, Semi-online algorithms for computational task offloading with communication delay, *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1189–1201, 2017.

[10] B. D. Shang, L. J. Liu, and Z. Tian, Deep learning-assisted energy-efficient task offloading in vehicular edge computing systems, *IEEE Trans. Veh. Technol.*, vol. 70, no. 9, pp. 9619–9624, 2021.

[11] J. Liu, Y. Y. Mao, J. Zhang, and K. B. Letaief, Delay-optimal computation task scheduling for mobile-edge computing systems, presented at the 2016 IEEE Int. Symp. Information Theory, Barcelona, Spain, 2016, pp. 1451–1455.

[12] M. F. Deng, H. Tian, and B. Fan, Fine-granularity based application offloading policy in cloud-enhanced small cell networks, presented at the 2016 IEEE Int. Conf. Communications Workshops, Kuala Lumpur, Malaysia, 2016, pp. 638–643.

[13] Y. H. Kao, B. Krishnamachari, M. R. Ra, and F. Bai, Hermes: Latency optimal task assignment for resource-constrained mobile computing, *IEEE Trans. Mobile Comput.*, vol. 16, no. 11, pp. 3056–3069, 2017.

[14] A. A. Al-Habob, O. A. Dobre, A. G. Armada, and S. Muhaidat, Task scheduling for mobile edge computing using genetic algorithm and conflict graphs, *IEEE Trans. Veh. Technol.*, vol. 69, no. 8, pp. 8805–8819, 2020.

[15] G. M. Zhao, H. L. Xu, Y. M. Zhao, C. M. Qiao, and L. S. Huang, Offloading dependent tasks in mobile edge computing with service caching, presented at the IEEE Conf. Computer Communications, Toronto, Canada, 2020, pp. 1997–2006.

[16] L. Y. Liu, H. S. Tan, S. H. C. Jiang, Z. H. Han, X. Y. Li, and H. Huang, Dependent task placement and scheduling with function configuration in edge computing, presented at the 2019 IEEE/ACM 27$^{\text{th}}$ Int. Symp. Quality of Service, Phoenix, AZ, USA, 2019, pp. 1–10.

[17] P. L. Dai, K. W. Hu, X. Wu, H. L. Xing, and Z. F. Yu, Asynchronous deep reinforcement learning for data-driven task offloading in MEC-empowered vehicular networks, presented at the IEEE Conf. Computer Communications, Vancouver, Canada, 2021, pp. 1–10.

[18] Y. N. Fan, L. B. Zhai, and H. Wang, Cost-efficient dependent task offloading for multiusers, *IEEE Access*, vol. 7, pp. 115843–115856, 2019.

[19] S. Jošilo and G. Dán, Wireless and computing resource allocation for selfish computation offloading in edge computing, presented at the IEEE Conf. Computer Communications, Paris, France, 2019, pp. 2467–2475.

[20] R. Bi, Q. Liu, J. K. Ren, and G. Z. Tan, Utility aware offloading for mobile-edge computing, *Tsinghua Science and Technology*, vol. 26, no. 2, pp. 239–250, 2021.

[21] X. Chen, L. Jiao, W. Z. Li, and X. M. Fu, Efficient multi-user computation offloading for mobile-edge cloud computing, *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, 2016.

[22] J. Y. Zhou, J. X. Fan, and J. Wang, Task scheduling for mobile edge computing enabled crowd sensing applications, *Int. J. Sensor Networks*, vol. 35, no. 2, pp. 88–98, 2021.

[23] D. Huang, P. Wang, and D. Niyato, A dynamic offloading algorithm for mobile computing, *IEEE Trans. Wirel. Commun.*, vol. 11, no. 6, pp. 1991–1995, 2012.

[24] O. B. Yetim and M. Martonosi, Dynamic adaptive techniques for learning application delay tolerance for mobile data offloading, presented at the 2015 IEEE Conf. Computer Communications, Hong Kong, China, 2015, pp. 1885–1893.

[25] Y. Y. Mao, J. Zhang, and K. B. Letaief, Dynamic computation offloading for mobile-edge computing with energy harvesting devices, *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, 2016.

[26] S. Sundar and B. Liang, Offloading dependent tasks with communication delay and deadline constraint, presented at the IEEE Conf. Computer Communications, Honolulu, HI, USA, 2018, pp. 37–45.

[27] S. Zhang, N. Zhang, S. Zhou, J. Gong, Z. S. Niu, and X. M. Shen, Energy-aware traffic offloading for green heterogeneous networks, *IEEE J. Sel. Areas Commun.*, vol. 34, no. 5, pp. 1116–1129, 2016.

[28] Y. Y. Mao, C. S. You, J. Zhang, K. B. Huang, and K. B. Letaief, A survey on mobile edge computing: The communication perspective, *IEEE Commun. Surv. Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[29] L. Q. Zhang, J. J. Luo, L. Gao, and F. C. Zheng, Learning-based computation offloading for edge networks with heterogeneous resources, presented at the 2020 IEEE Int. Conf. Communication, Dublin, Ireland, 2020, pp. 1–6.

[30] J. Kwak, Y. Kim, J. Lee, and S. Chong, DREAM: Dynamic resource and task allocation for energy minimization in mobile cloud systems, *IEEE J. Sel. Areas Commun.*, vol. 33, no. 12, pp. 2510–2523, 2015.

[31] L. Yang, J. N. Cao, H. Cheng, and Y. S. Ji, Multi-user computation partitioning for latency sensitive mobile cloud applications, *IEEE Trans. Comput.*, vol. 64, no. 8, pp. 2253–2266, 2015.

[32] Y. Sahni, J. N. Cao, L. Yang, and Y. S. Ji, Multi-hop multi-task partial computation offloading in collaborative edge

computing, *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 5, pp. 1133–1145, 2021.

[33] H. W. Kuhn, The Hungarian method for the assignment problem, *Naval Res. Logist. Quart.*, vol. 2, nos. 1&2, pp. 83–97, 1955.

[34] S. Aslam, A. Shahid, and K. G. Lee, IMS: Interference minimization scheme for cognitive radio networks using Hungarian algorithm, presented at the 1st Int. Conf. Future Generation Communication Technologies, London, UK, 2012, pp. 17–21.

[35] T. X. Tran and D. Pompili, Joint task offloading and resource allocation for multi-server mobile-edge computing networks, *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, 2019.

[36] H. Z. Guo and J. J. Liu, Collaborative computation offloading for multiaccess edge computing over fiber-wireless networks, *IEEE Trans. Veh. Technol.*, vol. 67, no. 5, pp. 4514–4526, 2018.

Huanhuan Xu received the BEng degree in electronic information engineering from Jinling Institute of Technology, Nanjing, China, in 2018. He is currently pursuing the PhD degree in computer science at the School of Computer Science and Technology, Soochow University, Suzhou, China. His research interests include mobile edge computing and task offloading.

Jingya Zhou received the BEng degree in computer science from Anhui Normal University, Wuhu, China, in 2005, and the PhD degree in computer science from Southeast University, Nanjing, China, in 2013. He is currently an associate professor with the School of Computer Science and Technology, Soochow University, Suzhou, China. He has co-authored more than 60 papers in these areas, many of which have been published in top journals and conferences such as ACM CSUR, INFOCOM, ICDE, ICDCS, and ICPP. His research interests include cloud and edge computing, network embedding, online social networks, and data center networking.

Wenqi Wei received the BEng degree from Huazhong University of Science and Technology, Wuhan, China. He is currently pursuing the PhD degree at the School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA. His research interests include data analytics, data privacy, AI security, and machine learning.

Baolei Cheng received the BS, MS, and PhD degrees in computer science from Soochow University, Suzhou, China, in 2001, 2004, and 2014, respectively. He is currently an associate professor in computer science at the School of Computer Science and Technology, Soochow University, Suzhou, China. His research interests include parallel and distributed systems, algorithms, interconnection architectures, and software testing.