Computation Offloading for Multi-user Sequential Tasks in Heterogeneous Mobile Edge Computing

Huanhuan Xu[1], Jingya Zhou[1,2(✉)], and Fei Gu[1]

[1] School of Computer Science and Technology, Soochow University, Suzhou 215006, China
20195227020@stu.suda.edu.cn, {jy_zhou,gufei}@suda.edu.cn
[2] State Key Laboratory of Mathematical Engineering and Advanced Computing, Wuxi 214125, China

Abstract. Currently, many computation offloading studies in Mobile Edge Computing (MEC) mainly focus on the multi-user and multi-task offloading, where the tasks are independent and inseparable. However, nowadays many mobile applications, such as Augmented Reality (AR) glasses, face recognition, *etc.*, can be classified into sequential tasks. Dependencies among tasks and resource competition among multiple users in the heterogeneous edge environment make the offloading problem very challenging. In this paper, we propose a new multi-user sequential task (MUST) framework to address the above challenge. Specifically, we present a comprehensive analysis of the time cost of the task offloading process in the MUST framework and define the multi-user sequential task offloading problem. Moreover, we prove the problem is NP-hard and propose a reMUST algorithm based on regular expression to obtain the approximate optimal solution. Numerous experiments have shown that the proposed method is superior to existing alternatives in terms of cost and system scalability.

Keywords: Mobile edge computing · Computation offloading · Sequential tasks · Regular expression

This work was supported by National Natural Science Foundation of China (Grant Nos. 61972272, U1905211 and 62072321), the Open Project Program of the State Key Laboratory of Mathematical Engineering and Advanced Computing (Grant No. 2019A04), Jiangsu Planned Projects for Postdoctoral Research Funds (Grant No. 1701173B), Jiangsu Overseas Visiting Scholar Program for University Prominent Young & Middle-aged Teachers and Presidents.

H. Gao and X. Wang (Eds.): CollaborateCom 2021, LNICST 406, pp. 713–727, 2021.
https://doi.org/10.1007/978-3-030-92635-9_41

1 Introduction

With the iterative evolution of mobile device (MD) technology, users' demand for complex mobile applications is soaring. These applications, such as mobile AR, live video, collaborative editing, *etc.*, require massive computational resources of MDs to provide low-latency services [1,2]. However, since the performance of MDs is constrained by many issues, such as physical size, battery capacity, and system architecture, their computational capacities may not be able to meet the latency and computing requirements of these applications. By deploying cloud computing services to the edge of the network, MEC effectively provides low-latency services for the users [3]. MEC uses a computation offloading technology to offload computation-intensive tasks to nearby MEC servers, which expands the computing and storage capacity of MDs. Meanwhile, MEC overcomes the disadvantages of the traditional cloud computing system, such as network congestion and long transmission delay caused by the large number of task processes and the long physical distance between the data center and the network edge.

As a key technology of MEC, computation offloading distributes the computational tasks of MDs to the edge environment, which alleviates the shortages of the MDs in resource storage, computational performance, and energy efficiency [4]. Recently, many researchers work on computation offloading in MEC. These studies can reduce energy consumption and makespan effectively, or make a balance between them, by selecting task offloading decisions flexibly. However, in real mobile application scenarios, there are still some problems to be solved:

(1) **Heterogeneity**: MEC servers are heterogeneous and deployed on a large scale. There may be multiple service providers at the edge network. Even for the same service provider, different periods of deployment and equipment updates result in heterogeneity. This situation leads to great uncertainty about the cost of executing the task, which will seriously affect the efficiency of task execution.
(2) **Multi-User**: MEC is multi-user oriented, which is different from the single-user scenario that has been widely studied. Multi-user scenario has different user requirements, heterogeneous tasks, and resource competition. Especially, the competition between MDs leads to low wireless rates in the network and long queuing time on edge servers.
(3) **Dependency**: At present, many mobile applications are not independent tasks, but consist of sequential dependent tasks. For instance, Fig. 1 shows that Google Project Glass with AR technology, including the following function modules: video capture, video parsing, target recognition, content mapping, video synthesis, and real-time display. Among them, video parsing and content mapping require massive computational resources and offloading them to edge servers can effectively reduce the processing delay of the overall task.

To solve these problems, we propose a Multi-User Sequential Tasks (MUST) framework for offloading users' sequential tasks in MEC. Our contributions are summarized as follows:

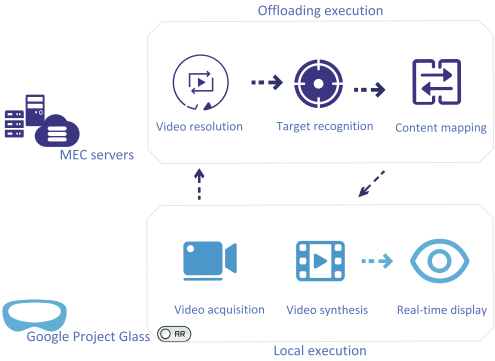


Fig. 1. The AR application execution flow

(1) We propose a framework named MUST to solve the multi-user sequential tasks offloading problem in heterogeneous MEC. When it comes to offloading the sequential tasks, we adopt the MUST framework to reduce the system cost according to the current network environment.
(2) Based on the MUST framework, we give a formal definition of the problem and further prove that it is NP-hard. By relaxing the integer limit of offloading decision and using the penalty function to consider other constraints, we propose a heuristic method based on Regular Expression (RE) to get the approximate optimal solution to the original problem.
(3) We make extensive numerical experiments to evaluate the effectiveness of the proposed scheme. The results demonstrate that the proposed scheme not only reduces the average delay but also improves the scalability of the system, compared with the conventional offloading strategies.

2 Related Work

A great deal of work has been done on the computation offloading problem and various offloading strategies have been proposed. Among them, the dependent task offloading problem in MEC is very complicated. Only a few studies pay attention to the sequential dependent constraint.

Kao *et al.* [5] designed a polynomial-time approximation algorithm that minimizes the maximum completion time when offloading dependent tasks under resource constraints. Zhao *et al.* [6] jointly considered dependent task offloading and service caching placement to minimize application completion time. Liu *et al.* [7] considered the problem of dependent task placement and scheduling with on-demand function configuration on the server, and proposed an approximate algorithm to minimize the application completion time. However, the above works did not take the real multi-user edge environment into account, where multiple users compete for marginal limited resources fiercely.

Fan *et al.* [8] proposed the offloading problem of multi-user dependent tasks to minimize the total cost of all applications within the time limit for completion

of each application. Jošilo *et al.* [9] modeled the problem of allocating wireless and computing resources to a set of autonomous wireless devices as a Stackelberg game and proposed an effective decentralized equilibrium algorithm to reduce task completion time. Chen *et al.* [10] studied the multi-user computation offloading problem of mobile edge cloud computing under a multi-channel wireless interference environment, designed a distributed computing offloading algorithm that could achieve nash equilibrium, and obtained good computation offloading performance. Whereas, the above works ignored the heterogeneous characters of the edge. Edge servers are generally heterogeneous in reality, which will further enhance the complexity of the problem.

Habak *et al.* [11] considered the model of using a MD cluster to accept offloaded tasks and proposed a general heuristic task scheduling scheme, whose goal is to maximize the effective execution of offloading tasks by the cluster. Lin *et al.* [12] considered the scheduling problem of an application composed of dependent tasks and proposed a heuristic algorithm. However, the authors assumed that only the native processors of a single MD are considered. Similarly, Sundar *et al.* [13] examined the problem of dependent task offloading in heterogeneous networks to minimize cost under application deadline. Sundar *et al.* [14] studied the scheduling of an application composed of dependent tasks in a general heterogeneous edge computing system and proposed a heuristic algorithm to obtain an effective solution.

In this work, edge servers have a limited workload, and there is latency cost associated with task execution, data communication, and task queuing on them, which leads to a unique formulation of the problem that has not been studied in the existing literature.

3 MUST Model and Problem Formulation

In this section we first introduce the system model, which includes the network model, task model, and cost model. Then we formally define the offloading problem of multi-user sequential tasks (MUST) with delay and load limits, and prove that there is no constant solution to this problem.

3.1 System Model

A. Network Model

As shown in Fig. 2, the typical MEC scenario includes two parts, *i.e.*, the user and the edge. The user side has N MDs, represented by a set $\mathbf{N} = \{1, 2, \ldots, N\}$. In the edge, we define $\mathbf{K} = \{1, 2, \ldots, K\}$ as the set of K access points with different computational capabilities, which we call computational access points (CAPs) for simplicity. CAPs are interconnected at a high speed via wired links [3], while MD transmits data with CAP via the single-carrier wireless channel of orthogonal frequency division multiple access (OFDMA). The channel gain is affected by shadow fading and path loss caused by the route. It is assumed that

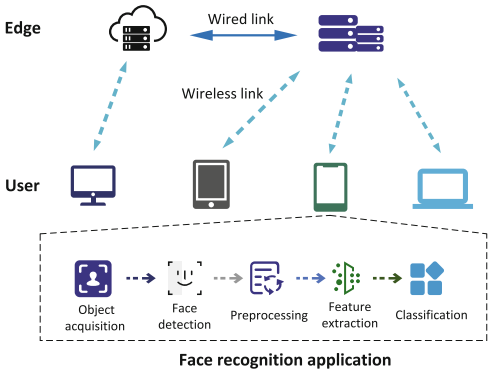


Fig. 2. An example of MEC scenario with multiple users

each CAP can cover all MDs and has an upper limit c 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 the set $\mathbf{D} = \{d_1, d_2, \ldots, d_N\}$ represents the applications on all MDs. Each application $d \in \mathbf{D}$ consists of a set $\mathbf{M} = \left\{m_1^d, m_2^d, \ldots, m_M^d\right\}$ of sequential tasks. The m-th task on the n-th MD is denoted by $\mathcal{T}_{m,n}\left(s_{m,n}, \eta_{m,n}\right)$, where $s_{m,n}$ (bits) represents the size of input data in the current task, which is also the size of output data in the previous task, and $\eta_{m,n}$ represents the number of CPU cycles spent by each bit of the task data.

Each task can be offloaded to a nearby CAP. According to the offloading decision of the previous task, the data transmission process of the current task may be different. The CAP may not process the task immediately after receiving the task data but puts it in a queue.

B. Task Dependency Model

The application of the n-th MD has a deadline T_n^{max}, and can be divided into several tasks with sequential dependent as mentioned before [15]. We use an example of face recognition to illustrate the execution flow of the application. As shown in Fig. 2, the MD needs to collect relevant face data, preprocess the collected data at the beginning, and send the processed data to the CAP, where the application service has been deployed and the model has been preloaded. The CAP can perform match detection, and return the match result to the MD. In this paper, we mainly focus on the offloading of such sequential tasks. Notations and their meanings are listed in Table 1.

We assume that the first task of the application is started from local MD and the final result must be returned to MD. Each task can be executed on one of the CAPs. We define the offloading decision variable of the task m on MD n as $x_{m,n}$, which means whether the system offloads this task to CAP k.

Further, we define $\mathbf{X} = \{\mathrm{x}_{m,n} \mid m \in \mathbf{M}, n \in \mathbf{N}\}$ as the offloading strategy of all tasks, $\mathbf{N}_k$ as the set of MDs connected to CAP k and $\mathbf{M}_k$ as the set of tasks offloaded to CAP k. For a given $\mathbf{X}$, we calculate the wireless rate as:

Table 1. Notations and their meanings

Notation	Meaning
$\mathbf{N}$	Set of mobile devices
$\mathbf{M}$	Set of tasks need to be executed
$\mathbf{K}$	Set of computational access points
$x_{m,n}$	Integer offloading variable of the n-th MD's task m
c	The upper limit on the number of accepted tasks on CAP k
$\mathcal{T}_{m,n}$	Information of the n-th MD's task m
T_n^{max}	Deadline of the n-th MD's application
T_n	Completing time of the n-th MD's application
$T_{m,n}$	Completing time of the n-th MD's task m
W	Wireless channel bandwidth of CAPs
β	Data rate of wire link between CAPs
p_n	Device power of MD n
$h_{n,k}$	Channel gain of transmitting tasks from MD n to CAP k
$r_{n,k}$	Wireless transmission rate of the n-th MD's tasks between MD n and CAP k
$t_{m,n}^{exec}$	Executing time of the n-th MD's task m
$t_{m,n}^{tran}$	Data transmission time of the n-th MD's task m
$t_{m,n}^{queu}$	Queuing time of the n-th MD's task to be executed on CAP k

$$r_{n,k}(\mathbf{X}) = W \log_2 \left(1 + \frac{p_n h_{n,k}}{\varpi + \sum_{i \in \mathbf{N}_k \setminus \{n\}} p_i h_{i,k}} \right), \tag{1}$$

where p_n is the power of the n-th MD, $h_{n,k}$ is the channel gain from MD n to CAP k based on the shadow attenuation and path loss, ϖ is the thermal noise of link between MD n and CAP k and W is the wireless channel bandwidth [16].

C. Cost Model

In our scenario, the goal of computation offloading is to minimize the average cost of the applications by offloading different sequential tasks from MDs to CAPs. Next we introduce our cost model where the process of each task consists of three parts: execution, communication, and queuing.

Task Execution: To illustrate the delay caused by execution, we define the execution time of the task m on CAP k is $t_{m,n}^{exec} = \frac{s_{m,n}\eta_{m,n}}{f_k}$.

Task Communication: Considering different communication processes, the transmission time of the n-th MD's task m can be divided into three cases:

(1) Task m is the first task and needs to be offloaded from MD n to CAP k for execution. Considering the wireless transmission process of the task, the time it consumed is $t^{tran}_{m,.n} = \frac{s_{m,n}}{r_{n,k}}$.
(2) Task m is an intermediate task and its offloading decision is different from its previous task. The task data requires to be transmitted from one CAP to another for execution. Since the fixed wired bandwidth between CAPs is β, the transmission delay is $t^{tran}_{m,n} = \frac{s_{m,n}}{\beta}$.
(3) Task m is an intermediate task and its offloading decision is identical to its previous task. It means that adjacent tasks are offloaded to the same CAP. Since there is no data transmission process, the transmission delay is $t^{tran}_{m,n} = 0$.

Since the amount of data returned by the application's final task is generally small, we ignore the transmission delay of these data like the article [10].

Task Queuing: The task potentially cannot be executed immediately when it arrives at CAP, it needs to wait until the current CAP finishes its tasks. Each CAP handles the tasks on it as an M/G/1 queue by utilizing queuing theory. It is assumed that v_k is the arrival rate of the task on the CAP k. Let θ denote the variable of computation time for any task. $E_k[\theta]$ and $E_k\left[\theta^2\right]$ represent its first and second moments. According to the Pollaczek-Khinchin formula [17], the expected queuing time for a task m of the n-th MD on CAP k is $t^{\text{queu}}_{m,n} = \frac{v_k E_k\left[\theta^2\right]}{2(1-v_k E_k[\theta])}$.

Since the application completion consists of the execution, communication and queuing of all tasks, the total delay of the n-th MD's application is:

$$T_n(\mathbf{X}_n) = \sum_{m=1}^{M} T_{m,n}(x_{m,n}) = \sum_{m=1}^{M} \left(t^{\text{exec}}_{m,n} + t^{\text{tran}}_{m,n} + t^{\text{queu}}_{m,n}\right). \tag{2}$$

3.2 Problem Formulation

Given a set of resource-constrained CAPs and a set of MDs (each MD's application consisting of several dependent tasks and has a deadline), we define the multi-user sequential dependent task offloading (MUSTO) problem. Since the goal is to minimize the average application delay in the system while satisfying the delay limit of each application and the load capacity of CAPs, the problem of MUST can be formulated as problem **P1** below:

$$\underset{\{\mathbf{X}\}}{\text{minimize}} \quad \frac{1}{N} \sum_{m=1}^{M} \sum_{n=1}^{N} T_{m,n}(x_{m,n}), \tag{3}$$

$$\begin{aligned} \text{s. t.} \quad & C_1: \quad |\mathbf{D}_k| \leq c, \forall k \in \mathbf{K}, \\ & C_2: \quad x_{m,n} \in \{1, \ldots, K\}, \forall m \in \mathbf{M}, \forall n \in \mathbf{N}, \\ & C_3: \quad \sum_{m=1}^{M} T_{m,n} \leq T_n^{\max}, \forall n \in \mathbf{N}. \end{aligned}$$

C_1 indicates that there is a limit to the number of tasks that each CAP can accept, C_2 represents that the offloading decision is an integer variable, and C_3 indicates that the overall task execution time of MD's tasks cannot exceed its deadline. Then we rephrase the offloading decision variable to the one-hot type $\mathbf{x}_{m,n} = [0, \ldots, 1, \ldots]^\top$, where $(\cdot)^\top$ represents the transpose of a matrix or a vector. Hence, the offloading decision variable can be represented as follows:

$$x_{m,n,k} = \begin{cases} 1 & \text{if MD } n' \text{ s task } m \text{ is offloaded to CAP}k, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Since each task can only be assigned to one CAP for execution, it must satisfy $\sum_{k=1}^{K} x_{m,n,k} = 1$. We define the potential cost of offloading the n-th MD's task m to CAP k as $C_{m,n,k}$. Therefore, the problem **P1** can be rewritten as **P2**:

$$\begin{aligned} & \underset{\mathbf{X}}{\text{minimize}} \sum_{m=1}^{M} \sum_{n=1}^{N} \sum_{k=1}^{K} \mathrm{C}_{m,n,k} x_{m,n,k}, \\ \text{s. t.} \quad & C_3: \quad \sum_{m=1}^{M} T_{m,n} \leq T_n^{\max}, \forall n \in \mathbf{N}, \\ & C_4: \quad \sum_{m=1}^{M} \sum_{n=1}^{N} x_{m,n,k} \leq c, \forall m \in \mathbf{M}, \forall n \in \mathbf{N}, \forall k \in \mathbf{K}, \\ & C_5: \quad x_{m,n,k} \in \{0,1\}, \forall m \in \mathbf{M}, \forall n \in \mathbf{N}, \forall k \in \mathbf{K}, \\ & C_6: \quad \sum_{k=1}^{K} x_{m,n,k} = 1, \forall m \in \mathbf{M}, \forall n \in \mathbf{N}. \end{aligned} \tag{5}$$

C_3 and C_4 indicate that the total task execution time of the MD cannot exceed its delay limit and there is a limit to tasks each CAP can undertake respectively.

Theorem 1. *The problem of MUSTO is NP-hard.*

Proof. GAP problem is NP-hard, which is described as follows [18]. The system assigns mutually independent works to the limited-resource agents. A job can only be served by one agent, an agent can serve multiple jobs, and the total amount of resources required by the agent to complete the work cannot exceed its limit. The mathematical model of GAP is expressed in the following form:

$$\begin{aligned} & \text{minimize} \quad F(x) = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{i,j} x_{i,j}, \\ \text{s. t.} \quad & \sum_{j=1}^{n} r_{i,j} x_{i,j} \leqslant b_i, \forall i \in \mathbf{I}, \\ & \sum_{i=1}^{m} x_{i,j} = 1, \forall j \in \mathbf{J}, \\ & x_{i,j} \in \{0,1\}, \forall i \in \mathbf{I}, \forall j \in \mathbf{J}, \end{aligned} \tag{6}$$

where $\mathbf{I} = \{1, 2, \ldots, m\}$ is the agent set, $\mathbf{J} = \{1, 2, \ldots, n\}$ is the working set, b_i represents the number of resources owned by the i-th agent, $r_{i,j}$ represents the number of resources consumed by the i-th agent to serve the work j and $c_{i,j}$ represents the cost of the i-th agent to serve the work j.

The objective function is to minimize the system cost. Equation (6) ensures that the total amount of resources consumed by the work j which is assigned to the i-th agent does not exceed the resource limit, each work j can only be assigned to one agent, and $x_{i,j}$ is the binary decision variable respectively.

From the above analysis, we find that when every application does not exceed its deadline, the MUSTO problem is a set of N standard GAP problems. Therefore, the MUSTO problem is NP-hard and cannot obtain a specific solution.

4 Regular Expression Based Algorithm for MUST

In this section, we present a regular expression-based algorithm for MUST, called reMUST. The workflow of reMUST is shown in Fig. 3. Specifically, reMUST offloads sequential tasks by conducting four steps: (1) *Relaxing the MUSTO Problem* to construct an ordinary least square problem (OLS) with constraints. (2) *Using penalty function* method to remove constraints for this relaxed problem. (3) *Using regular expression* for the standard OLS problem. (4) *Offloading tasks* by following the approximate optimal strategy.

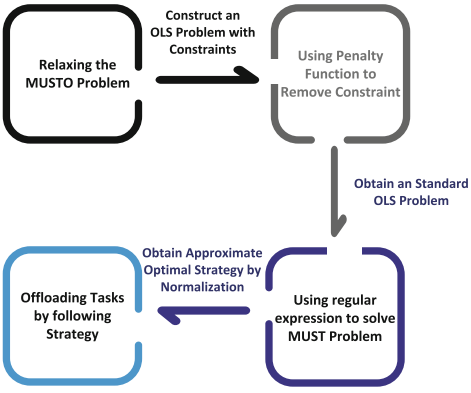


Fig. 3. The workflow of the reMUST algorithm: 1) relaxing the MUSTO problem to construct a limited OLS problem, 2) using penalty function to remove constraints, 3) using regular expression for the rewritten MUSTO problem to get the approximate optimal solution, and 4) offloading tasks according to the strategy.

Relaxing the MUSTO Problem. To reformulate problem in Eq. (5) in the OLS form, the offloading decisions must take binary values. Nevertheless, the offloading decisions may not be integer vectors under matrix operations.

Therefore, we relax the offloading decisions. By combining one-hot vector with relaxing offloading decisions, we simplify **P2** as follows:

$$\begin{aligned} \underset{\mathbf{X}}{\text{minimize}} \quad & \frac{1}{N}\sum_{m=1}^{M}\sum_{n=1}^{N}\mathbf{C}_{m,n}^{\top}\mathbf{x}_{m,n}, \\ \text{s. t. } & \mathrm{C}_3, \mathrm{C}_4. \end{aligned} \tag{7}$$

Using Penalty Function. After relaxing the variables, we use the penalty functions to replace the constraints of the problem. We make a two-layer optimization with the ordinary least square form. We further rewrite the objective function and add the formulation limitations of application delay and CAP load to **P2**. By using the definition of sums of squared deviations, the objective function of average system cost can be replaced as follows:

$$\begin{aligned} \underset{\mathbf{X}}{\text{minimize}} \quad & \frac{1}{N}\sum_{n=1}^{N}\sum_{m=1}^{M}\left(\mathbf{C}_{m,n}^{\top}\mathbf{x}_{m,n} - \overline{\mathbf{C}^{\top}\mathbf{x}}\right)^2, \\ \text{s.t.} \quad & \mathrm{C}_3, \mathrm{C}_4, \end{aligned} \tag{8}$$

where $\overline{\mathbf{C}^{\top}\mathbf{x}}$ represents the average task delay. Since the least square method cannot be applied to the optimization problem with limited conditions, we add C_3 and C_4 to the objective function by using the penalty functions and get:

$$\underset{\mathbf{X}}{\text{minimize}} \quad \mathrm{E} = \frac{1}{N}\sum_{n=1}^{N}\left(\sum_{m=1}^{M}\left(\mathbf{C}_{m,n}^{\top}\mathbf{x}_{m,n} - \overline{\mathbf{C}^{\top}\mathbf{x}}\right)^2 + \rho_1\right) + \rho_2, \tag{9}$$

where $\rho_1 = \mu\left(\max\left(0, \sum_{m=1}^{M}\mathbf{C}_{m,n,k}^{\top}\mathbf{x}_{m,n,k} - T_n^{\max}\right)\right)$ and $\rho_2 = \sum_{k=1}^{K}\phi\left(\max\left(0, \sum_{m=1}^{M}\sum_{n=1}^{N}\mathbf{x}_{m,n,k} - c\right)\right)$ are penalty functions of constraints C_3, C_4, and μ, ϕ are the penalty factors.

Using RE to Solve MUSTO Problem. After getting the form of the relaxed MUSTO problem with no constraints, we use a RE-based method to solve the MUSTO problem according to its properties. This problem is a function of mapping some vector variables to a number. By mapping offloading strategy to cost, the first-order partial derivatives of cost concerning offloading strategy can be obtained. Since the current object function is to minimize $E(\mathbf{X}_n)$, it gets optimal solution when $\nabla_{\mathbf{x}_n}E(\mathbf{X}_n) = 0$, *i.e.*, $\mathbf{X}_n = \left(\mathbf{C}_n^{\top}\mathbf{C}_n\right)^{-1}\mathbf{C}_n^{\top}\overline{\rho_1}$ according to Theorem 2. After using normalization we can finally obtain our approximate optimal offloading strategy $\mathbf{X}_n^*$ for tasks on the n-th MD.

Theorem 2. *For a least square problem, i.e.,* $\text{minimize}\sum_{i=1}^{I}\left(\mathbf{A}^{\top}x_i - b_i\right)^2$*, its solution satisfies* $\mathbf{X} = \left(\mathbf{A}^{\top}\mathbf{A}\right)^{-1}\mathbf{A}^{\top}\mathbf{B}$.

Proof. For a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, a constant vector $\mathbf{B} \in \mathbb{R}^{m\times 1}$ and the vector variable $\mathbf{X} \in \mathbb{R}^{n\times 1}$ in the ordinary least square problem, we have:

$$\begin{aligned} \nabla_{\mathbf{x}} f(\mathbf{X}) =& \nabla_{\mathbf{x}}(\mathbf{A}\mathbf{X} - \mathbf{B})^{\top}(\mathbf{A}\mathbf{X} - \mathbf{B}) \\ \Leftrightarrow \nabla_{\mathbf{x}} f(\mathbf{X}) =& \nabla_{\mathbf{x}}\left(\mathbf{X}^{\top}\mathbf{A}^{\top}\mathbf{A}\mathbf{X} - \mathbf{X}^{\top}\mathbf{A}^{\top}\mathbf{B} - \mathbf{B}^{\top}\mathbf{A}\mathbf{X} + \mathbf{B}^{\top}\mathbf{B}\right). \end{aligned}$$

Algorithm 1. Offloading Algorithm for MUST

Input:. MDs information, CAPs information, network parameters
Output:. optimal offloading strategy $\mathbf{X}^*$, average cost E^*
Initial:. Random offloading strategy $\mathbf{X}$

while there is application in $\mathbf{N}^{'}$ **do**
  Calculate current wireless rate r
  for task in $\mathbf{M}$ **do**
    Calculate execution $t_{m,n}^{exec}$ and queuing cost $t_{m,n}^{queu}$
    if normalized offloading strategy $\mathbf{x}_{m,n}$ == prior **then**
      Communication cost = 0
    else
      Calculate communication cost $t_{m,n}^{tran}$
    Compute cost vector $\mathbf{C}_n$
  Construct optimization function in (9)
  Compute partial optimal strategy $\mathbf{X}_n^*$
return $\mathbf{X}^*$, E^*

Since $\mathbf{B}^\top\mathbf{B}$ is a real number, the optimization can be further converted to:

$$\begin{aligned}
&\nabla_{\mathbf{x}} f(\mathbf{X}) = \nabla_{\mathbf{x}} \left(\mathbf{X}^\top\mathbf{A}^\top\mathbf{A}\mathbf{X} - \mathbf{X}^\top\mathbf{A}^\top\mathbf{B} - \mathbf{B}^\top\mathbf{A}\mathbf{X} + \mathbf{B}^\top\mathbf{B}\right) \\
\Leftrightarrow\ &\nabla_{\mathbf{x}} f(\mathbf{X}) = \nabla_{\mathbf{x}} tr\left(\mathbf{X}^\top\mathbf{A}^\top\mathbf{A}\mathbf{X}\right) - 2\nabla_{\mathbf{x}}\left(tr\left(\mathbf{B}^\top\mathbf{A}\mathbf{X}\right)\right. \\
\Leftrightarrow\ &\nabla_{\mathbf{x}} f(\mathbf{X}) = \nabla_{\mathbf{x}} tr\left(\mathbf{X}^\top\mathbf{A}^\top\mathbf{A}\mathbf{X}\right) - 2\mathbf{A}^\top\mathbf{B},
\end{aligned}$$

where $tr(\cdot)$ represents the trace of the matrix. According to the matrix property, we can get $\nabla_{\mathbf{x}} tr\left(\mathbf{X}^\top\mathbf{A}^\top\mathbf{A}\mathbf{X}\right)\mathbf{A}\mathbf{A}^\top\mathbf{X} + \mathbf{A}^\top\mathbf{A}\mathbf{X} = 2\mathbf{A}^\top\mathbf{A}\mathbf{X}$. Then, we have

$$\begin{aligned}
&\nabla_{\mathbf{x}} f(\mathbf{X}) = \nabla_{\mathbf{x}} \left(\mathbf{X}^\top\mathbf{A}^\top\mathbf{A}\mathbf{X} - \mathbf{X}^\top\mathbf{A}^\top\mathbf{B} - \mathbf{B}^\top\mathbf{A}\mathbf{X} + \mathbf{B}^\top\mathbf{B}\right) \\
\Leftrightarrow\ &\nabla_{\mathbf{x}} f(\mathbf{X}) = 2\mathbf{A}^\top\mathbf{A}\mathbf{X} - 2\mathbf{A}^\top\mathbf{B}.
\end{aligned}$$

Therefore, we get the optimal solution when $2\mathbf{A}^\top\mathbf{A}\mathbf{X} - 2\mathbf{A}^\top\mathbf{B} = 0$, *i.e.* , $\mathbf{X} = \left(\mathbf{A}^\top\mathbf{A}\right)^{-1}\mathbf{A}^\top\mathbf{B}$.

Offloading Tasks. We offload tasks according to the offloading strategy until all tasks are offloaded. The reMUST algorithm is formally described in Algorithm 1. It starts from a strategy profile in which all tasks are offloaded randomly. We first denote $\mathbf{N}^{'}$ by the set of MDs that have never changed their strategy, noting that at the beginning $\mathbf{N}$=$\mathbf{N}^{'}$. The reMUST algorithm for each task consists of two phases that are executed sequentially. In the first phase, it calculates execution, communication, and queuing cost by offloading tasks to potential CAPs. In the second strategy judgment phase, we calculate the solution of **P2**.

5 Performance Evaluation

In this section, we perform extensive simulation experiments to evaluae the performance of our algorithm. The simulation is carried out on a MacBook Pro with a 2 GHz Intel Core i5 quad-core processor.

5.1 Setup

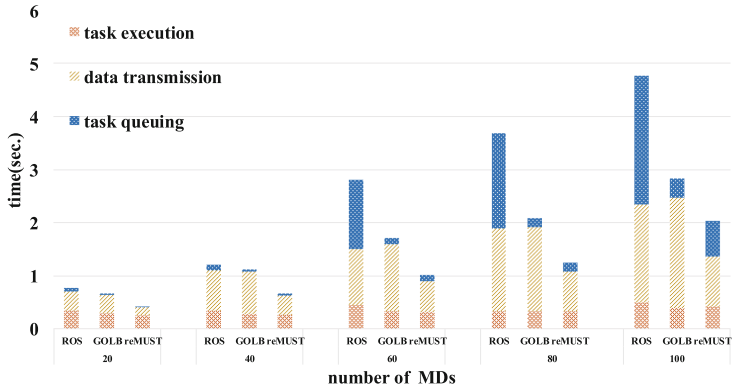
The edge environment is a 250×250 m^2 area with 50 MDs and 9 CAPs embedded edge servers. Each MD has an application that consists of 5 sequential tasks. The size of the initial computing task follows the uniform distribution of $[600, 1200]$ KB. The number of CPU cycles required for completing the task follows the random distribution of [500, 1000] Megacycles, and the computing capacity of CAP is selected from {15, 20, 30} GHz [19]. According to the channel gain model in [19], the channel loss is set as $140.\ 7 + 36.\ 7\log_{10}(d)$, where d represents the distance between the MD and CAP. The positions of MD and CAP follow a random and uniform distribution, and CAPs interconnect with each other via wired links. For the communication model, the noise power is set as ϖ = -100 dBm, $p_n = 23$ dBm, $\beta = 1$ Gbps [16,20]. The wireless channel is a single carrier channel, and the channel bandwidth is $W = 36$ MHz. We compare our reMUST algorithm 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.

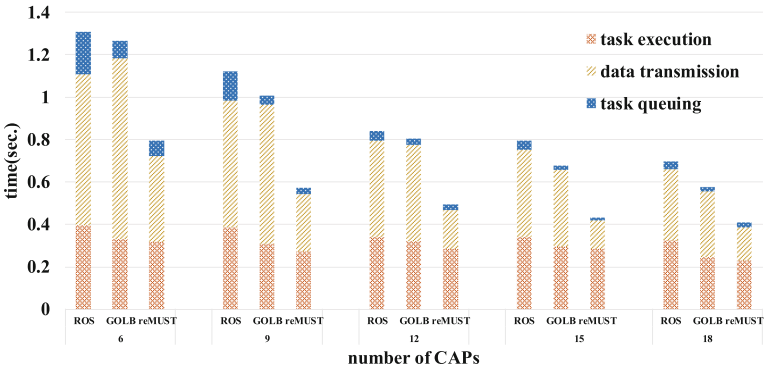
5.2 Results

In the first group of experiments, we evaluate the average application delay of all task offloading algorithms. Figure 4 (a) shows that all algorithms have a sustained increasing trend along with the growth of the number of MDs. Among them, reMUST always achieves the lowest average latency. We notice that reMUST′s queuing time is longer than GOLB when the number of MDs comes to 100. It is because GOLB tends to offload tasks to CAPs with low workloads, which leads to high transmission delay, while reMUST takes both data transmission and task queuing into account. Although reMUST has a long queuing process, its transmission delay is significantly lower than other algorithms. Thus, reMUST achieves the lowest overall delay compared with other algorithms. Figure 4 (b) illustrates that the average delays of all algorithms decrease as CAPs increase. We find that the average delay falls slowly when the number of CAPs exceeds 12. Then the queuing time decreases significantly. In this case, when the number of CAP reaches 12, the system resources are sufficient for the current task scale and the resource competition weakens. Compared to other algorithms, the advantage of reMUST is mainly reflected in the reduction of data transmission delay.

In the second group of experiments, we evaluate the impact on the completion rate of applications, where completion rate refers to the proportion of completed applications within their deadlines. Figure 5 illustrates that all algorithms perform worse while the number of MDs is larger. Among them, only reMUST has a nearly 100% completion rate when the number of MDs is 60, while the completion rate of *ROS* and *GOLB* is about 50%. When there are



(a) Impact of the number of MDs on delay



(b) Impact of the number of CAPs on delay

Fig. 4. Number of MDs & CAPs vs. average delay

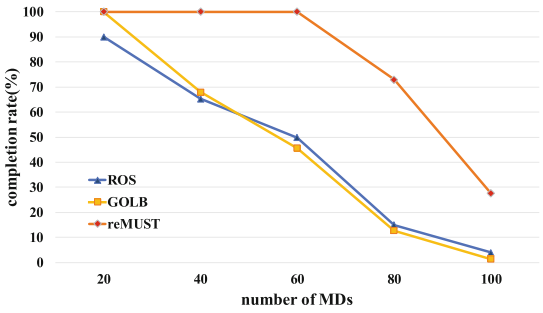


Fig. 5. Number of MDs vs. completion rate

100 MDs in the system, reMUST′s completion rate is nearly 30%, it is because the network environment is so congested that few applications could meet their deadlines, meanwhile, the completion rates of *ROS* and *GOLB* almost drop to zero.

In conclusion, the reMUST algorithm not only reduces the average delay of applications in the system, but also enhances the scalability of offloading strategy compared to the conventional schemes.

6 Conclusion

In this paper, we studied the computation offloading problem of multi-user sequential tasks in heterogeneous MEC environments. Our goal is to minimize the average delay of applications on user devices. For this fundamental issue, we proposed the MUST framework to complete multi-user sequential tasks offloading. Specifically, we formally defined the problem and further proved that it is NP-hard. Then we proposed the reMUST algorithm to obtain the approximate optimal solution based on solid theoretical analysis. Finally, we conducted extensive simulation experiments to demonstrate the effectiveness of the reMUST.

References

1. Lei, Y., Zheng, W., Ma, Y., Xia, Y., Xia, Q.: A novel probabilistic-performance-aware and evolutionary game-theoretic approach to task offloading in the hybrid cloud-edge environment. In: Gao, H., Wang, X., Iqbal, M., Yin, Y., Yin, J., Gu, N. (eds.) CollaborateCom 2020. LNICST, vol. 349, pp. 255–270. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-67537-0_16
2. Liu, F., Lv, B., Huang, J., Ali, S.: Towards mobility-aware dynamic service migration in mobile edge computing. In: Gao, H., Wang, X., Iqbal, M., Yin, Y., Yin, J., Gu, N. (eds.) CollaborateCom 2020. LNICST, vol. 349, pp. 115–131. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-67537-0_8
3. Chen, N., Yang, Y., Zhang, T., Zhou, M.T., Luo, X., Zao, J.K.: Fog as a service technology. IEEE Commun. Mag. **56**(11), 95–101 (2018)
4. Yang, B., Cao, X., Li, X., Zhang, Q., Qian, L.: Mobile-edge-computing-based hierarchical machine learning tasks distribution for IIoT. Things J. IoT-J **7**(3), 2169–2180 (2020)
5. Kao, Y.H., Krishnamachari, B., Ra, M.R., Bai, F.: Hermes: latency optimal task assignment for resource-constrained mobile computing. Mob. Comput. INFOCOM **16**(11), 3056–3069 (2017)
6. Zhao, G., Xu, H., Zhao, Y., Qiao, C., Huang, L.: Offloading dependent tasks in mobile edge computing with service caching. In: Conference on Computer Communications INFOCOM, pp. 1997–2006 (2020)
7. Liu, L., Tan, H., Jiang, S.H.C., Han, Z., Li, X.Y., Huang, H.: Dependent task placement and scheduling with function configuration in edge computing. In: IWQoS, pp. 1–10 (2019)
8. Fan, Y., Zhai, L., Wang, H.: Cost-efficient dependent task offloading for multiusers. IEEE Access **7**, 115843–115856 (2019)

9. Jošilo, S., Dán, G.: Wireless and computing resource allocation for selfish computation offloading in edge computing. In: Conference on Computer Communications INFOCOM, pp. 2467–2475 (2019)
10. Chen, X., Jiao, L., Li, W., Fu, X.: Efficient multi-user computation offloading for mobile-edge cloud computing. Netw. TON **24**(5), 2795–2808 (2016)
11. Habak, K., Ammar, M., Harras, K.A., Zegura, E.: Femto clouds: leveraging mobile devices to provide cloud service at the edge. In: International Conference on Cloud Computing CLOUD, pp. 9–16 (2015)
12. Lin, X., Wang, Y., Xie, Q., Pedram, M.: Task Scheduling with Dynamic Voltage and Frequency Scaling for Energy Minimization in the Mobile Cloud Computing Environment. IEEE Trans. Services Comput. **8**(2), 175–186 (2015)
13. Sundar, S., Liang, B.: Communication Augmented Latest Possible Scheduling for cloud computing with delay constraint and task dependency. In: Conference on Computer Communications Workshops (INFOCOM WKSHPS) INFOCOM WKSHPS, pp. 1009–1014 (2016)
14. Sundar, S., Liang, B.: Offloading dependent tasks with communication delay and deadline constraint. In: INFOCOM, pp. 37–45 (2018)
15. Yang, L., Cao, J., Cheng, H., Ji, Y.: Multi-user computation partitioning for latency sensitive mobile cloud applications. TOC **64**(8), 2253–2266 (2015)
16. Chen, X.: Decentralized computation offloading game for mobile cloud computing. IEEE Trans. Parallel Distrib. Syst. **26**(4), 974–983 (2015)
17. Chan, W.C.: Pollaczek-Khinchin formula for the M/G/1 queue in discrete time with vacations. IEE Proc. Comput. Digital Tech. **144**(4), 222–226 (2002)
18. Author, R., Review, H., Source, H., Logic, S., Url, S.: Review of Computers and Intractability. A Guide to the Theory of NP-Completeness (1983)
19. Tran, T.X., Pompili, D.: Joint task offloading and resource allocation for multi-server mobile-edge computing networks. Technol. TVT **68**(1), 856–868 (2019)
20. Guo, H., Liu, J.: Collaborative computation offloading for multiaccess edge computing over fiber-wireless networks. Technol. TVT **67**(5), 4514–4526 (2018)