This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2019.2951206, IEEE Transactions on Industrial Informatics

A Code-Oriented Partitioning Computation Offloading Strategy for Multiple Users and Multiple Mobile Edge Computing Servers

Yan Ding, *Student Member, IEEE,* Chubo Liu, Xu Zhou, Zhao Liu, and Zhuo Tang

***Abstract*—In this paper, we investigate code-oriented partitioning computation offloading strategy for multiple user equipments (UEs) and multiple mobile edge computing (MEC) servers with limited resources (i.e., limited computing power and waiting task queues with finite capacity). The paper aims to develop an offloading strategy to decide the execution location, CPU frequency, and transmission power for UE while minimizing the execution overhead (i.e., a weighted sum of energy consumption and computational time) of UE's applications, which is an NP-hard problem. To achieve the objective, first, we transform the problem into a convex optimization problem and find the optimal solution. Second, we propose a decentralized computation offloading strategy (DCOS) algorithm for UE, and define a dictionary data structure for recording the strategy of the UE to reduce the algorithm complexity. Finally, the effectiveness of DCOS, and the impact of various key parameters on the strategy and overhead are demonstrated by simulation experiments.**

***Index Terms*—Code-oriented partitioning offloading, computation offloading strategy, mobile edge computing, multiple servers with limited resources.**

I. Introduction

MBOILE edge computing (MEC) is an architecture that deploys limited computing power and storage capacity close to user equipment (UE), aiming to reduce the service delay and energy consumption of UE, and promote the quality of experience (QoE) and the quality of service (QoS) [1]. In MEC, the most relevant part to users is the computation offloading of applications [2], [3]. The computation offloading strategy research can be categorized into full offloading and partitioning offloading [4]. Full offloading focuses on indivisible applications and has been investigated in [5], [6], [7], [8], [9], [10]. Partitioning offloading investigates divisible applications. Based on the type of applications, partitioning offloading can be further classified as data-oriented partitioning offloading (DOPO), continuous-execution partitioning offloading (CEPO), and code-oriented partitioning offloading (COPO) [11]. DOPO requires that an application can be able to split into subsets of any size, and the amount of the data to be processed is known beforehand [12], [13], [14], [15]. CEPO focuses on applications that cannot know the data amount in advance and has been studied in [16], [17]. COPO corresponds to applications that can be divided into several tasks with dependencies and has been studied in [18], [19], [20], [21], [22].

However, there are some limitations in the above works. First, most of the researches assume that the resources of MEC servers are unlimited (i.e., unlimited computing power and waiting task queues with infinite capacity). The unlimited computing power means that the server can execute an unlimited number of tasks simultaneously. Although some works limit the computing power of the MEC server, they assume that the waiting queue of server has an infinite capacity, such as references [12], [19]. Second, if there are multiple MEC servers that can provide services and UE has a set of tasks with dependencies, the process of the UE making offloading strategy will be more complicated. Meanwhile, under the environment of MEC servers with limited resources, the UE's offloading strategy is affected by other UEs that simultaneously send requests to the same MEC server. The strategy made only between a certain pair of the UE and the MEC server increases the additional consumption caused by request collisions. But there are no works consider the situations. Third, the UE, the MEC server, the network environment, and the application type may all be heterogeneous in MEC. However, except for reference [10], [12], most of the above works do not consider the characteristic of MEC. Finally, although the MEC server is suggested to be deployed as densely as possible to satisfy the requests of user [23], from the perspective of the average utilization of servers and energy saving, the suggestion may not be the best option. But all above works do not study the appropriate number of MEC servers that can meet the UEs' requirements.

Based on the above discussions, we need to consider the following five questions.

- For an application that can be divided into some tasks with dependency, which tasks should be offloaded to the MEC server?
- How to allocate the resources of UE to reduce the execution overhead of the UE while meeting the latency constraint of the application?
- How does UE select the MEC server that performs the tasks when there are multiple MEC servers with limited resources?
- How to handle request conflicts and redundant calculations caused when multiple UEs simultaneously send requests to the same MEC server?
- How to determine the number of MEC servers to minimize the overhead of UEs while maximizing the average

The authors are with the College of Information Science and Engineering, Hunan University, and National Supercomputing Center in Changsha, Hunan, Changsha 410082, China.
E-mail: ding@hnu.edu.cn, liuchubo@hnu.edu.cn, zhouxu@hnu.edu.cn, lzhao@hnu.edu.cn, and ztang@hnu.edu.cn.
Chubo Liu is the author for correspondence.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2019.2951206, IEEE Transactions on Industrial Informatics

utilization of MEC servers?

In order to address the above issues, this paper investigates the COPO strategy for multiple UEs and multiple MEC servers (MUMMS) with limited resources (i.e., limited computing power and waiting task queues with finite capacity), and has the following works. First, we restrict the resources of UE and the MEC server. All UEs and MEC servers can only execute one task at a time, that is, they can only execute tasks in a serial manner. Meanwhile, all MEC servers have a queue with finite capacity for waiting tasks. Second, we consider the additional overhead generated by MEC servers collaboratively providing services. When the MEC server that is executing a UE's task can no longer accept subsequent requests from the UE, the MEC server needs to send the result of the task to other available MEC servers, thereby causing the additional waiting overhead. In addition, the tasks in the waiting queue of MEC servers also have additional waiting overhead. Third, we further consider the heterogeneous characteristics of MEC. Each UE is specified by its own CPU frequency, transmission power, application type, latency constraint, and communication channel. Each MEC server is also specified by its computing power, queue capacity, and communication channel. Finally, we propose two indicators, including MEC execution rate (MER, i.e., the proportion of the tasks that are executed remotely) and MEC utilization rate (MUR, i.e., the proportion of MEC servers participating in performing the tasks), to find the appropriate number of MEC servers that can meet the UEs' requirements. The evaluation of MER and MUR can avoid deploying too many idle MEC servers, and can also reduce the energy consumption of MEC servers.

The problem is to schedule the offloaded tasks of UE on a set of MEC servers as well as to allocate CPU resource and transmission power of the UE, such that the execution overhead of the UE is minimized. The main contributions of our work are as follows.

- We restrict the resources of UE and the MEC server, and consider the additional overhead generated by MEC servers collaboratively providing services. Moreover, we consider the heterogeneous characteristics of MEC. The scenario studied in this paper is closer to the real world.
- We formulate the problem as an execution overhead minimization problem that satisfies the resource and latency constraints. We transform the original non-convex optimization problem into a convex optimization problem through the variable substitution technique, and prove that there is an optimal solution to the problem.
- We define a dictionary data structure for recording the offloading strategies of UE to handle the request conflicts and redundant calculations under the environment of MUMMS. And then, we propose a *decentralized computation offloading strategy* (DCOS) algorithm to make the offloading policy for each UE.
- The convergence of DCOS, the effect of DCOS to reduce the execution overhead, and the impact of various key parameters on offloading decision and the overhead are demonstrated by simulation experiments. In addition, we discuss the impact of MEC servers on the execution overhead of UEs and propose two indicators to help to reduce the number of idle MEC servers.

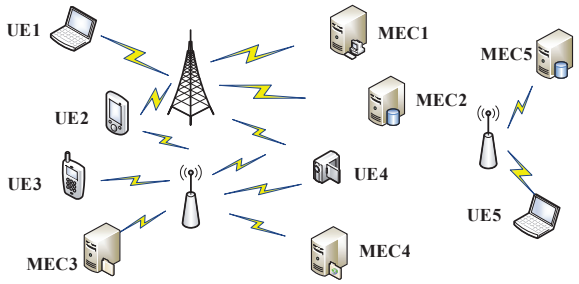


Fig. 1. An illustration of the MUMMS scenario.

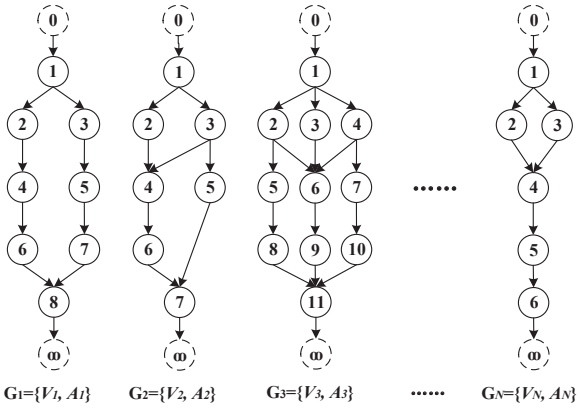


Fig. 2. Program call graph of UEs.

To the best of our knowledge, this is the first paper that studies the COPO strategy optimization for MUMMS with limited resources. And the scenario is closer to the real world. The rest paper is outlined as follows. Section II demonstrates the system model, mathematical models and formulations of the problem studied in this paper. Section III introduces the DCOS in detail. Section IV describes the simulation experiments and evaluates the performance of DCOS. Section V presents our conclusions.

II. System Model and Problem Formulation

A. System Model

Fig. 1 illustrates an example of the MUMMS scenario. We have the following assumptions. First, we assume that there are N UEs and M MEC servers. UE_i $\left(i \in \{1, 2, \ldots, N\}\right)$ offloads tasks to MEC_j $\left(j \in \{1, 2, \ldots, M\}\right)$ through wireless communication, such as 4G or 5G. Second, the wireless links are orthogonal channels so that the link will not be interfered by other links. Third, UE_i knows the workload of application and channel-side information in advance, but does not know anything about other UEs. UE_i communicates with all connectable MEC servers to determine the status of these servers, such as the computing power and the number of tasks in the waiting queue. The overhead of probe communication between MEC_j and UE_i is ignored. Fourth, once the UE starts uploading a task to the MEC server, the MEC server ensures that the expectation completion time of the task will not be

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2019.2951206, IEEE Transactions on Industrial Informatics

postponed. Fifth, UE$_i$ remains station while uploading tasks and receiving the result of tasks. Sixth, MEC$_j$ can execute the task of UE$_i$ directly. Finally, the computing power of UE and MEC servers are restricted, that is, they can only execute tasks in serial manner. Moreover, the maximum capacity of MEC$_j$'s waiting queue is denoted by Q_j ($0 < Q_j \ll +\infty$).

B. Application Model

We assume that the application has been split into a set of tasks with dependency by MAUI [24], CloneCloud [25], ThinkAir [26], mCloud [27] or ULOOF [28]. Thus, the application generated by UE can be represented as a program call graph [29]. As shown in Fig. 2, UE$_i$ can also be expressed as $G_i = \{V_i, A_i\}$, where V_i represents the set of tasks generated by UE$_i$, and A_i represents dependencies between the tasks. For example, if there are $v_a, v_{a+1} \in V_i$, and there is a directed edge from v_a to v_{a+1} in G_i, then v_{a+1} must be executed after v_a. Because UE$_i$ executes tasks serially, $(v_a, v_{a+1}) \in A_i$ is always true. Moreover, we use w_{i,v_a} to represent the workload of v_a, which indicates the number of CPU clock cycles required to complete the task. δ_{i,v_a} represents the processing density of v_a (bits per cycle) and σ_{i,v_a} indicates the ratio of the output data size to the input data size. v_0 means that the task can only be executed locally, such as the user's input operation. v_∞ is a virtual task that represents the termination of the application. Other tasks can be executed remotely or locally. In the following, if not specified, $|V_i| + 1 > a > 0$. We use a binary variable $I_{i,j,v_a} \in \{0, 1\}$ to indicate whether v_a is executed at MEC$_j$. $I_{i,0,v_a} = 1$ represents v_a is executed locally. Because a task can only be executed at one location, there is a constraint, i.e., $\sum_{j=0}^{M} I_{i,j,v_a} = 1$.

C. Communication Model

According to the Rayleigh fading channel model [30], the rate of UE$_i$ to transmit v_a to MEC$_j$ can be obtained from

$$R_{i,j,t,v_a} = W_{i,t} \log_2 \left(1 + \frac{p_{i,j,t,v_a} h_i}{d_{i,j}^{\omega_i} N_i}\right), \tag{1}$$

where $W_{i,t}, p_{i,j,t,v_a}, h_i, d_{i,j}, \omega_i, N_i$ are the transmission channel bandwidth, the transmission power of UE$_i$, the channel fading coefficient, the distance between UE$_i$ and MEC$_j$, the channel path loss exponent, and the channel white Guassian noise, respectively.

The rate of UE$_i$ to receive the result of v_a from MEC$_j$ is

$$R_{i,j,r} = W_{i,r} \log_2 \left(1 + \frac{p_{i,r} h_i}{d_{i,j}^{\omega_i} N_i}\right), \tag{2}$$

where $W_{i,r}$ is the receiving channel bandwidth and $p_{i,r}$ is the receiving power of UE$_i$.

In addition, if v_a is offloaded to MEC$_{j'}$ and v_{a+1} will be offloaded to MEC$_j$ ($j' \neq j$), the rate of MEC$_{j'}$ to transfer the result of v_a to MEC$_j$ is

$$R_{j',j} = W_{j',j} \log_2 \left(1 + \frac{p_{j',j} h_{j'}}{d_{j',j}^{\omega_{j'}} N_{j'}}\right), \tag{3}$$

where $W_{j',j}, p_{j',j}, h_{j'}, d_{j',j}, \omega_{j'}, N_{j'}$ are the channel bandwidth, the transmission power of MEC$_{j'}$, the channel fading coefficient, the distance between MEC$_{j'}$ and MEC$_j$, the channel path loss exponent, and the channel white Guassian noise, respectively.

D. Computation Model

1) Local computation model: The maximum CPU frequency of UE$_i$ is $f_{i,max}$ (cycles per second). Meanwhile, the CPU of UE$_i$ is equipped with the technique of dynamic voltage and frequency scaling (DVFS) such that can adjust the CPU frequency. Thus, the actual CPU frequency is represented as f_{i,v_a}, where $0 \leq f_{i,v_a} \leq f_{i,max}$. According to [31], the local energy consumption of v_a can be formulated as

$$E_{i,v_a}^{l} = w_{i,v_a} \kappa_i f_{i,v_a}^2, \tag{4}$$

where κ_i is the coefficient factor of UE$_i$'s chip architecture. The local computational time of v_a is

$$T_{i,v_a}^{l} = \frac{w_{i,v_a}}{f_{i,v_a}}. \tag{5}$$

2) MEC server computation model: The computing power of an MEC server is denoted by f_j (cycles per second). Due to the dependencies between the tasks, the execution scheme of v_a is affected by v_{a-1}. Therefore, the computational time of v_a executed at MEC$_j$ is

$$T_{i,v_a}^{j} = t_{i,v_a}^{j,exe} + trt_{i,j,v_a} + ret_{i,j,v_a} + t_{j,v_a} + t_{j',j,v_{a-1}}, \tag{6}$$

where $t_{i,v_a}^{j,exe} = w_{i,v_a}/f_j$, $trt_{i,j,v_a} = w_{i,v_a}\delta_{i,v_a}/R_{i,j,t,v_a}$, $t_{j',j,v_{a-1}} = w_{i,v_{a-1}}\delta_{i,v_{a-1}}\sigma_{i,v_{a-1}}/R_{j',j}$, t_{j,v_a}, $ret_{i,j,v_a} = w_{i,v_a}\delta_{i,v_a}\sigma_{i,v_a}/R_{i,j,r}$ are the computational time of v_a executed at MEC$_j$, the transmission delay of v_a, the delay to transmit the result of v_{a-1} from MEC$_{j'}$ to MEC$_j$, the waiting delay if v_a is offloaded to MEC$_j$, and the receiving delay of v_a, respectively. It should be noted that the transmitted result of v_{a-1} will not need to be executed by MEC$_j$ repeatedly. The energy consumption of v_a executed at MEC$_j$ is

$$E_{i,v_a}^{j} = p_{i,j,t,v_a} trt_{i,j,v_a} + p_{i,r} ret_{i,j,v_a}. \tag{7}$$

E. Problem Formulation

Some tasks are more sensitive to delay, and some tasks are more sensitive to energy. To balance the 2 consumptions, we draw on the work of [32] and define the execution overhead of v_a as $Cost_{i,v_a} = \gamma_i E_{i,v_a} + (1 - \gamma_i) T_{i,v_a}$, where $\gamma_i \in [0, 1]$ denotes the tradeoff parameter of UE$_i$'s energy consumption. γ_i can be obtained from the multiple attribute utility theory [33]. The execution overhead of UE$_i$ can be formulated as

$$\begin{aligned} Cost_i = \sum_{a=1}^{|V_i|} \Bigg(& \left(1 - \sum_{j=1}^{M} I_{i,j,v_a}\right) Cost_{i,v_a}^{l} \\ & + \sum_{j=1}^{M} I_{i,j,v_a} Cost_{i,v_a}^{j} \Bigg), \end{aligned} \tag{8}$$

where $Cost_{i,v_a}^{l} = \gamma_i E_{i,v_a}^{l} + (1 - \gamma_i) T_{i,v_a}^{l}$ and $Cost_{i,v_a}^{j} = \gamma_i E_{i,v_a}^{j} + (1 - \gamma_i) T_{i,v_a}^{j}$. In this paper, we minimize the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2019.2951206, IEEE Transactions on Industrial Informatics

execution overhead of the application generated by UE_i. Thus, for UE_i, the problem can be formulated as

$$\begin{aligned}
\mathrm{P1}: \quad & \min_{I_i,P_i,F_i} Cost_i, \\
s.t. \quad & C_1: 0 \le f_{i,v_a} \le f_{i,max}, \\
& C_2: 0 \le p_{i,j,t,v_a} \le p_{i,tmax}, \\
& C_3: \sum_{a=1}^{|V_i|} \left(\left(1 - \sum_{j=1}^{M} I_{i,j,v_a}\right) T^l_{i,v_a} \right. \\
& \qquad \left. + \sum_{j=1}^{M} I_{i,j,v_a} T^j_{i,v_a} \right) \le L_i, \\
& C_4: \forall (v_a, v_{a+1}) \in A_i, v_a, v_{a+1} \in V_i, \\
& C_5: \sum_{j=0}^{M} I_{i,j,v_a} = 1, I_{i,j,v_a} \in \{0,1\},
\end{aligned} \tag{9}$$

where I_i, P_i, F_i are the offloading decision set of V_i, the transmission power strategy set of V_i, and the CPU frequency strategy set of V_i. In P1, C_1, C_2 are the CPU frequency constraint and transmission power constraint, respectively. C_3 is the latency constraint of the application. C_4 represents the dependency between the tasks. C_5 indicates that a task can only be executed at one location at a time. According to P1, we can know that the problem is a mixed integer programming problem, and an NP-hard problem [22].

III. Task Scheduling and Resource Allocation Algorithm

P1 can be transformed to a standard linear programming problem through the branch and bound method. Thus, we consider using convex optimization method to solve this problem. For a task, the subproblem can be formulated as

$$\begin{aligned}
\mathrm{P2}: \quad & \min_{\boldsymbol{s_{i,v_a}}} Cost_{i,v_a}, \\
s.t. \quad & C_1, C_2, C_3, C_4, \\
& C_6: \sum_{j=0}^{M} I_{i,j,v_a} = 1, I_{i,j,v_a} \in [0,1], \\
& C_7: (1 - I_{i,j,v_a}) T^l_{i,v_a} + I_{i,j,v_a} T^j_{i,v_a} \le L_{i,r,v_a},
\end{aligned} \tag{10}$$

where $L_{i,r,v_a} = L_i - \sum_{b>a}^{|V_i|} w_{i,v_b}/f_{i,min} - L_{i,p,v_a}$ is the remaining time of v_a, and $\boldsymbol{s_{i,v_a}}$ denotes the offloading strategies $\big($i.e., $I_{i,j,v_a}, p_{i,j,t,v_a}$, and $f_{i,v_a}\big)$ of v_a. $f_{i,min} = \sum_{a=1}^{|V_i|} w_{i,v_a}/L_i$ is the minimum CPU frequency to meet the latency requirement. L_{i,p,v_a} is the predecessor tasks' computational time of v_a. As shown in P2, the problem can be further decomposed into 2 subproblems based on I_{i,j,v_a}. When $I_{i,0,v_a} = 1$, the original problem becomes a CPU resource allocation problem, which is attempting to minimize the overhead for local execution. When $I_{i,j,v_a} = 1$, the original problem is converted into a transmission power allocation problem, which is attempting to minimize the overhead for remote execution. Thus, we should first give I_{i,j,v_a} a value.

A. Offloading Decision

The optimal decision of the task can be gotten from

$$\begin{aligned}
\mathrm{P3}: \quad & \min_{I_{i,j,v_a}} Cost^l_{i,v_a} + I_{i,j,v_a}(Cost^j_{i,v_a} - Cost^l_{i,v_a}), \\
s.t. \quad & C_1, C_2, C_4, C_6, C_7.
\end{aligned} \tag{11}$$

It can be known that $Cost_{i,v_a}$ is a linear function with regard to (w.r.t.) I_{i,j,v_a}. Thus, we have the following theorem [34].

Theorem 1 P3 *is a convex optimization problem w.r.t.* I_{i,j,v_a}.

We can obtain I_{i,j,v_a} through the greedy strategy, that is, choosing the execution location with the minimal overhead. Thus, the offloading decision can be given as

$$I_{i,j,v_a} = \begin{cases} 1, & \text{if } Cost^l_{i,v_a} > Cost^j_{i,v_a}, \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

B. CPU Frequency Strategy

If $Cost^l_{i,v_a} < Cost^j_{i,v_a}$, the task will be executed locally. The optimal CPU frequency can be obtained from the problem

$$\begin{aligned}
\mathrm{P4}: \quad & \min_{f_{i,v_a}} Cost^l_{i,v_a}, \\
s.t. \quad & C_1, C_4, C_7.
\end{aligned} \tag{13}$$

The optimal CPU strategy f^*_{i,v_a} can be obtained from the following theorem.

Theorem 2 P4 *is a convex optimization problem w.r.t.* f_{i,v_a}. *And* f^*_{i,v_a} *can be gotten from*

$$f^*_{i,v_a} = \begin{cases} f_{i,max}, & \text{if } \gamma_i = 0 \text{ or } f^{op}_{i,v_a} > f_{i,max}, \\ f_{i,min}, & \text{if } \gamma_i = 1, \\ f^{op}_{i,v_a}, & \text{otherwise,} \end{cases} \tag{14}$$

where $f^{op}_{i,v_a} = \sqrt[3]{(1 - \gamma_i + \tau_{i,v_a})/2\gamma_i\kappa_i}$, *and* τ_{i,v_a} *is the nonnegative Lagrange multipliers related to* C_7.

Proof. Because $d^2 Cost^l_{i,v_a}/df^2_{i,v_a} \ge 0$, we can obtain f^{op}_{i,v_a} through the KKT conditions of P4. As shown in Equation (8), if $\gamma_i = 0$, the original problem will be converted to minimize the time consumption. Hence, the optimal CPU frequency is $f^*_{i,v_a} = f_{i,max}$. And if $\gamma_i = 1$, the original problem will be converted to minimize the energy consumption. Thus, $f^*_{i,v_a} = f_{i,min}$. Therefore, the optimal strategy of the CPU can be obtained from Equation (14). The detailed proof is omitted. □

It can be seen from Theorem 2 that the CPU strategy is only related to κ_i, γ_i and τ_{i,v_a}. Meanwhile, w_{i,v_a} and other parameters do not affect the allocation of CPU resource.

C. Transmission Power Strategy

If $Cost^l_{i,v_a} > Cost^j_{i,v_a}$, the task will be executed remotely. The optimal transmission power can be gotten from

$$\begin{aligned}
\mathrm{P5}: \quad & \min_{p_{i,j,t,v_a}} Cost^j_{i,v_a}, \\
s.t. \quad & C_2, C_4, C_7.
\end{aligned} \tag{15}$$

It is obvious that P5 is a non-convex optimization problem. However, we can adopt the variable substitution technique to make P5 to be a convex optimization problem.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2019.2951206, IEEE Transactions on Industrial Informatics

Theorem 3 *The convex optimization problem*

$$\begin{aligned} \text{P6}: \quad & \min_{x_{i,j,v_a}} \gamma_i \left(p_{i,r} ret_{i,j,v_a} + \frac{w_{i,v_a}\delta_{i,v_a}}{W_{i,t}x_{i,j,v_a}} \frac{2^{x_{i,j,v_a}}-1}{D^t(d_{i,j})} \right) \\ & +(1-\gamma_i)\left(t^{j,exe}_{i,v_a} + t_{j,v_a} + t_{j',j,v_{a-1}} + ret_{i,j,v_a} \right. \\ & \left. + \frac{w_{i,v_a}\delta_{i,v_a}}{W_{i,t}x_{i,j,v_a}} \right), \qquad (16) \\ s.t. \quad & C_7, \\ & C_8: 0 \le x_{i,j,v_a} \le \log_2\left(1+p_{i,tmax}D^t(d_{i,j})\right), \end{aligned}$$

where $x_{i,j,v_a} = \log_2\left(1+p_{i,j,t,v_a}D^t(d_{i,j})\right)$ *and* $D^t(d_{i,j}) = h_i/\left(d_{i,j}^{\omega_i}N_i\right)$*, is equivalent to* P5*. And the optimal transmission power can be obtained from*

$$p^*_{i,j,t,v_a} = \begin{cases} p_{i,j,t,max}, & \text{if } \gamma_i = 0 \text{ or } h(p_{i,tmax}) \le 0, \\ p^{op}_{i,j,t,v_a}, & \text{otherwise}, \end{cases} \qquad (17)$$

where $h(p_{i,j,t,v_a}) = \ln(1+p_{i,j,t,v_a}D^t(d_{i,j})) - 1 - \left(D^t(d_{i,j})(\tau_{i,v_a}+1-\gamma_i)-\gamma_i\right)/\gamma_i\left(1+p_{i,j,t,v_a}D^t(d_{i,j})\right)$*, and* p^{op}_{i,j,t,v_a} *is the solution of* $h(p_{i,j,t,v_a})=0$ *and can be obtained through binary search method.*

Proof. We can convert P5 to a convex optimization problem by replacing $\log_2\left(1+p_{i,j,t,v_a}D^t(d_{i,j})\right)$ with x_{i,j,v_a}. According to the KKT conditions of P6, we know that the optimal transmission power should meet $h(p^{op}_{i,j,t,v_a})=0$. Meanwhile, if $h(p_{i,tmax}) \ge 0$, p^{op}_{i,j,t,v_a} can be obtained through the binary search method [34]. Therefore, we have the conclusion. □

D. The Optimal Strategy for a Task

Algorithm 1 Offloading Strategy for One-to-One (OSOTO)

Input: $G_i, L_i, w_{i,v_a}, p_{i,r}, p_{i,j,t,v_a}, f_{i,v_a}, f^c_j, D^t(d_{i,j}), \gamma_i, \delta_{i,v_a}, \sigma_{i,v_a}, \tau_{i,v_a}, W_{i,t}, W_{i,r}$, and K_i.
Output: p_{i,j,t,v_a}, f_{i,v_a}, and I_{i,j,v_a}.

while $k < K_i$, and $|\tau_{i,v_a}(k+1) - \tau_{i,v_a}(k)| > 10^{-11}$ **do**
  Calculate $Cost^l_{i,v_a}$ and $Cost^j_{i,v_a}$ through Equation (8);
  $I_{i,0,v_a} \leftarrow 1\{Cost^l_{i,v_a} > Cost^j_{i,v_a}\}$;
  if $I_{i,0,v_a} = 1$ **then**
    Update f_{i,v_a} by Equation (14);
  else
    Update p_{i,j,t,v_a} by Equation (17);
  end if
  Update τ_{i,v_a} by Equation (19);
  $k \leftarrow k+1$;
end while
$p_{i,j,t,v_a} \leftarrow p_{i,j,t,v_a}$;
$f_{i,v_a} \leftarrow f_{i,v_a}$;
Calculate $Cost^l_{i,v_a}$ and $Cost^j_{i,v_a}$ through Equation (8);
$I_{i,j,v_a} \leftarrow 1\{Cost^l_{i,v_a} > Cost^j_{i,v_a}\}$;
return p_{i,j,t,v_a}, f_{i,v_a}, and I_{i,j,v_a}.

As shown in Equations (14) and (17), f^*_{i,v_a} and p^*_{i,j,t,v_a} are all related to τ_{i,v_a}. Thanks to Theorems 1-3, we can get Theorem 4 [34].

$$\text{dict}_{i,v_a} = \begin{cases} \text{MEC}_3: & \left[es_3, t_{3,v_a}, p^*_{i,3,t,v_a}, f^{(3)}_{i,v_a}, cost^3_{i,v_a}\right] \\ \text{MEC}_1: & \left[es_1, t_{1,v_a}, p^*_{i,1,t,v_a}, f^{(1)}_{i,v_a}, cost^1_{i,v_a}\right] \\ \text{Local}: & \left[es_0, t_{0,v_a}, p_{i,j,t,v_a}, f^*_{i,v_a}, cost^l_{i,v_a}\right] \\ & \cdots\cdots \\ \text{MEC}_4: & \left[es_4, t_{4,v_a}, p^*_{i,4,t,v_a}, f^{(4)}_{i,v_a}, cost^4_{i,v_a}\right] \end{cases}$$

Fig. 3. The structure of the strategy record dictionary.

Theorem 4 P2 *is a convex optimization problem w.r.t.* I_{i,j,v_a}, f_{i,v_a}*, and* p_{i,j,t,v_a}*.*

Theorem 4 indicates that the problem P2 has a zero duality gap and satisfies the Slater's constraint [34]. Therefore, we can get the optimal strategy through the dual problem of P2. The dual problem of P2 is

$$\begin{aligned} \text{P7}: \quad & \max_{\tau_{i,v_a}} \min_{\boldsymbol{s_{i,v_a}}} L_{i,v_a}(\tau_{i,v_a}, \boldsymbol{s_{i,v_a}}), \qquad (18) \\ s.t. \quad & C_9: \tau_{i,v_a} \ge 0, \end{aligned}$$

where $L_{i,v_a}(\tau_{i,v_a}, \boldsymbol{s_{i,v_a}}) = (1-I_{i,j,v_a})Cost^l_{i,v_a} + I_{i,j,v_a}Cost^j_{i,v_a} + \tau_{i,v_a}((1-I_{i,j,v_a})T^l_{i,v_a} + I_{i,j,v_a}T^j_{i,v_a} - L_{i,r,v_a})$. Then, we can use the sub-gradient method to update τ_{i,v_a} for a given strategy. The update function of τ_{i,v_a} is

$$\tau_{i,v_a}(k+1) = max\left\{0, \tau_{i,v_a}(k) + \theta_{i,v_a}\frac{\partial L_{i,v_a}}{\partial \tau_{i,v_a}}\right\}, \qquad (19)$$

where $k > 0$ indicates the index of the iteration, $1 > \theta_{i,v_a} > 0$ represents the update step size. Therefore, for a task and an MEC server, if we get $\boldsymbol{s_{i,v_a}}$, we can update τ_{i,v_a}. And we can update the offloading strategy after the update of τ_{i,v_a}. Repeat this process until we get the optimal strategy (i.e., $k = K_i$ or $|\tau_{i,v_a}(k+1) - \tau_{i,v_a}(k)| \le 10^{-11}$). Algorithm 1 shows the process by which the UE gets an optimal strategy of a task between the local and an MEC server. The complexity of the algorithm is O(K_iU_i), where K_i is the maximum iterations of the update of τ_{i,v_a}, U_i is the number of the binary search iterations of p_{i,j,t,v_a}.

E. The DCOS Algorithm

If there are multiple MEC servers that can provide services, the process of the UE making offloading strategy will be more complicated. For UE$_i$, some MEC servers can communicate with them, but they are not necessarily suitable for providing computing or storage resource. In order to avoid unnecessary calculations, UE$_i$ should first determine a set of available MEC servers ($\mathbf{MEC_{av}}$) within the communication range. UE$_i$ can select $\mathbf{MEC_{av}}$ according to Theorem 5.

Theorem 5 *For a UE and an MEC server, if* $D^t(d_{i,j}) \ge (2^{o_{i,j,v_a}}-1)/p_{i,tmax}$*, where* $o_{i,j,v_a} = \delta_{i,v_a}w_{i,v_a}/\left(W_{i,t}(L_{i,r,v_a} - t^{j,exe}_{i,v_a} - t_{j,v_a} - t_{j',j,v_{a-1}} - ret_{i,j,v_a})\right)$*, MEC$_j$ is an available server for UE$_i$.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2019.2951206, IEEE Transactions on Industrial Informatics

Algorithm 2 Offloading Strategy for One-to-Many (OSOTM)

Input: dict_{i,v_a}, $t = 0$.
Output: I^*_{i,j,v_a}, f^*_{i,v_a}, and p^*_{i,j,t,v_a}.

1: $key_{list} \leftarrow \text{dict}_{i,v_a}.keys()$;
2: $op \leftarrow key_{list}[0]$;
3: $t_{op,v_a}(t) \leftarrow \text{dict}_{i,v_a}[op].t_{op,v_a}$;
4: UE_i communicates with MEC_{op} to get $t_{op,v_a}(t+1)$;
5: **if** $op = 0$ or $t \geq B_i$ or $t_{op,v_a}(t+1) \leq t_{op,v_a}(t)$ **then**
6: $\quad I^*_{i,op,v_a} \leftarrow 1$;
7: $\quad f^*_{i,v_a} \leftarrow \text{dict}_{i,v_a}[op].f_{i,v_a}$;
8: $\quad p^*_{i,j,t,v_a} \leftarrow \text{dict}_{i,v_a}[op].p_{i,j,t,v_a}$;
9: $\quad$ **return** I^*_{i,op,v_a}, f^*_{i,v_a}, and p^*_{i,j,t,v_a};
10: **else**
11: $\quad$ **for** $j \in key_{list}$ **do**
12: $\quad\quad$ **if** $j = 0$ or $j = op$ **then**
13: $\quad\quad\quad$ **continue**;
14: $\quad\quad$ **else**
15: $\quad\quad\quad$ UE_i communicates with $\text{MEC}_j \in \mathbf{MEC_{av}}$ to get $t_{j,v_a}(t+1)$;
16: $\quad\quad\quad$ $\Delta t_{j,v_a} \leftarrow t_{j,v_a}(t+1) - t_{j,v_a}(t)$;
17: $\quad\quad\quad$ $\text{dict}_{i,v_a}[j].t_{j,v_a} \leftarrow t_{j,v_a}(t+1)$;
18: $\quad\quad\quad$ $\text{dict}_{i,v_a}[j].cost^j_{i,v_a} \leftarrow cost^j_{i,v_a} + (1-\gamma_i)\Delta t_{j,v_a}$;
19: $\quad\quad\quad$ $t \leftarrow t+1$;
20: $\quad\quad$ **end if**
21: $\quad$ **end for**
22: **end if**
23: $\text{dict}_{i,v_a} \leftarrow Sort(\text{dict}_{i,v_a})$;
24: Call Algorithm 2 recursively until I^*_{i,j,v_a}, f^*_{i,v_a}, and p^*_{i,j,t,v_a} are obtained.

Proof. If v_a will be offloaded to MEC_j, $T^j_{i,v_a} \leq L_{i,r,v_a}$. Then, we bring L_{i,r,v_a} into Equation (6) and get the conclusion. □

If the resource of MEC servers is not exhausted, the servers can continue to receive task requests from other UEs. Therefore, UE_i should communicate with the optimal server ($\text{MEC}_{op} \in \mathbf{MEC_{av}}$) to determine if the request can be completed as former expected before uploading the task to the server. In order to handle the request conflicts and redundant calculations, we define a dictionary data structure (dict_{i,v_a}) for recording the strategies of the UE's task. UE_i retains the optimal strategies of the task executed locally and remotely. The *key* of dict_{i,v_a} is the execution location. And the *value* of $\text{dict}_{i,v_a}[key]$ is a list that holds the optimal strategies and the overhead of the task, including the execution sequence, the waiting delay, the CPU frequency, the transmission power, and the execution overhead for executing the task. Fig. 3 is an example of dict_{i,v_a}. It should be noted that dict_{i,v_a} is sorted through the merge sort based on the overhead.

Therefore, for a task and multiple MEC servers, we can use dict_{i,v_a} to obtain the optimal strategy between local and multiple MEC servers. As shown in Algorithm 2, we assume that the initial strategy and overhead of the UE's local and remote execution (i.e., for all MEC servers belonging to $\mathbf{MEC_{av}}$) have been obtained through Algorithm 1. Since dict_{i,v_a} has been sorted according to the overhead from small to large, the first record of dict_{i,v_a} is the current optimal strategy (lines 1-3). If the task is executed locally, the UE starts processing the task directly (lines 5-10). We assume that the UE has communicated with MEC_{op} at time slot t and communicates again at time slot $t+1$ to determine whether to start uploading the task. And if the completion time of the task has not been postponed or the number of communications exceeds the maximum iteration number (B_i), the UE starts uploading the task to MEC_{op} (lines 5-10). Otherwise, the UE will update dict_{i,v_a} (lines 11-21). Repeat this process until we get the optimal strategy (lines 23-24). The complexity of the algorithm is $\text{O}(B_i H_i \log_2 H_i)$, where $H_i = |\mathbf{MEC_{av}}|$.

With the help of Algorithms 1 and 2, we propose Algorithm 3 to determine the offloading strategy for UE_i in the scenario of MUMMS. For each UE and each of its tasks, UE_i first determines $\mathbf{MEC_{av}}$, gets the optimal offloading strategies to execute the task locally and remotely through Algorithm 1, and initializes dict_{i,v_a}. Then the optimal strategy of the task can be obtained from Algorithm 2. It is not difficult to know that the complexity of the algorithm for multiple UEs is $O\big(M \sum_{i=1}^{N} |V_i| K_i U_i B_i H_i \log_2 H_i\big)$.

Algorithm 3 The DCOS algorithm for UE_i

Input: $G_i, V_i, f_{i,max}, p_{i,tmax}, p_{i,r}, L_i, f_j, W_{i,t}, W_{i,r}, h_i, N_i, d_{i,j}, \omega_i, \delta_i, \sigma_i, h_j, \omega_j, N_j, W_{j'}, p_{j',j}, h_{j'}, d_{j',j}, \omega_{j'}$, and $N_{j'}$.
Output: I^*_i, F^*_i, and P^*_i.

1: **for** $v_a \in V_i, a > 0$ **do**
2: $\quad f_{i,v_a} \leftarrow f_{i,max}$;
3: $\quad p_{i,j,t,v_a} \leftarrow p_{i,tmax}$;
4: $\quad \mathbf{MEC_{av}} \leftarrow \emptyset$;
5: $\quad$ **for** $j \in M$ **do**
6: $\quad\quad$ **if** $T^j_{i,v_a} \leq L_{i,r,v_a}$ **then**
7: $\quad\quad\quad \mathbf{MEC_{av}}$.append($\text{MEC}_j$);
8: $\quad\quad$ **end if**
9: $\quad$ **end for**
10: $\quad$ **for** $\text{MEC}_j \in \mathbf{MEC_{av}}$ **do**
11: $\quad\quad local_{cost} \leftarrow \infty$;
12: $\quad\quad \text{dict}_{i,v_a}[0] \leftarrow [1, p_{i,j,t,v_a}, f_{i,v_a}, local_{cost}]$;
13: $\quad\quad$ Obtain $Cost^l_{i,v_a}$, I^*_{i,j,v_a}, p^*_{i,j,t,v_a}, and $f^{(j)}_{i,v_a}$ through Algorithm 1;
14: $\quad\quad \text{dict}_{i,v_a}[j] \leftarrow [es_j, p^*_{i,j,t,v_a}, f^{(j)}_{i,v_a}, Cost^j_{i,v_a}]$;
15: $\quad\quad$ **if** $local_{cost} > Cost^l_{i,v_a}$ **then**
16: $\quad\quad\quad \text{dict}_{i,v_a}[0] \leftarrow [1, p^*_{i,j,t,v_a}, f^{(j)}_{i,v_a}, Cost^l_{i,v_a}]$;
17: $\quad\quad$ **else**
18: $\quad\quad\quad$ **continue**;
19: $\quad\quad$ **end if**
20: $\quad$ **end for**
21: $\quad \text{dict}_{i,v_a} \leftarrow Sort(\text{dict}_{i,v_a})$;
22: $\quad$ Obtain I^*_{i,j,v_a}, f^*_{i,v_a}, and p^*_{i,j,t,v_a} from Algorithm 2;
23: **end for**
24: **return** I^*_i, F^*_i, and P^*_i.

IV. Simulation Experiments and Results Analysis

A. The Convergence of DCOS

Referring to [12], we generate UE_i and MEC_j according to the following parameters: $f_{i,max} = (5+0.1i)10^{8+0.1i}$ cycles/s,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2019.2951206, IEEE Transactions on Industrial Informatics

$p_{i,tmax} = 3 + 0.1i$ W, $\kappa_i = 10^{-(20+0.5i)}$, $d_{i,j} = 2 + 0.1j$ m, $d_{j',j} = random(1, 10)$ m, $f_j = (5 + 0.1j)10^{9+0.1j}$ cycles/s, $w_{i,v_a} = 10 + i$ cycles, $\delta_{i,v_a} = 10 + i$ bits/cycle, $\sigma_{i,v_a} = 0.05 + 0.005i$, $|V_i| = 10$, $W_{i,t} = W_{i,r} = 10^8$ HZ, $N_i = N_j = 10^{-9}$, $h_i = h_j = 10^{-3}$, $\gamma_i = 0.5$, $\omega_i = 3$, $L_i = 1$ s and $Q_j = 10$.

Table I shows the average number of iterations required for a UE's task to obtain its optimal strategy in different scenarios. The number of iterations refers to the number of loops required to search the optimal value of parameters (i.e., I^*_{i,j,v_a}, f^*_{i,v_a}, and p^*_{i,j,t,v_a}). According to the table, we have the following three observations.

- For different (N, M), where $N \leq 50$ and $M \leq 50$, the average number of the iterations of a task is less than 220.
- It can be seen that when the number of MEC servers is fixed, the average overhead increases as the number of UEs increases due to the limitation of computing power of MEC servers.
- It can be seen that when the number of UEs and the application type are fixed, as the number of MEC servers increases, the average overhead cannot be continuously reduced.

Next, we demonstrate the above observations in detail.

B. The Effect of DCOS

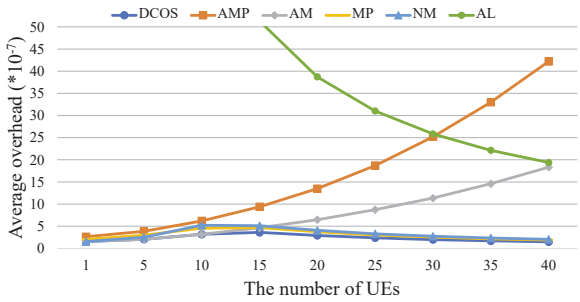
In this experiment, we use the following five schemes as baselines to evaluate the effect of DCOS.

- UE_i executes its tasks locally while using DVFS technology. We mark this scheme as AL.
- All tasks will be offloaded to the MEC server to execute. However, UE_i uploads its task using the maximum transmission power. We mark this scheme as AMP.
- All tasks will be offloaded to the MEC server to execute. Meanwhile, the transmission power is obtained from Equation (17). We mark this scheme as AM.
- All tasks can be executed locally or remotely. However, UE_i sends requests only to the most powerful servers among $\mathbf{MEC_{av}}$. The scheme is marked as MP.
- All tasks can be executed locally or remotely. However, UE_i sends requests only to the servers closest to itself among $\mathbf{MEC_{av}}$. The scheme is marked as NM.

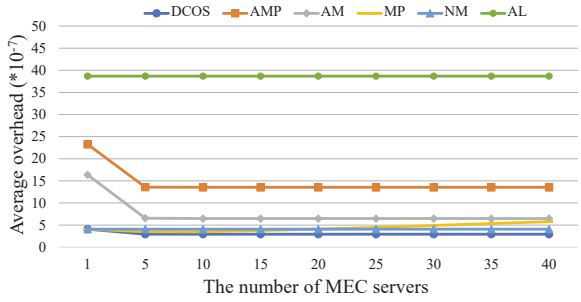
We set $\kappa_i = 10^{-(20+0.5i)}$ in the experiments, which means that as i increases, the local execution overhead of UE_i decreases. Thus, we can see that the AL curve in Fig. 4(a) shows a downward trend. As shown in Fig. 4(a), from the trend of the AMP and AM, the tasks with larger workload and longer distance are suitable to be executed locally. The MP and NM curves reflect that increasing the computing power of MEC servers will help reduce the overhead of UEs. The NM, MP, and DCOS curves indicate that when the MEC server has sufficient resource, the MEC server with more powerful are more suitable as the request server. In the figure, the NM, MP, and DCOS curves first increase and then decrease. The reason lies in that the resource of MEC servers is limited. Due to the limited resource of MEC servers, as the number of UEs increases, the waiting time of the UEs increases, so the average remote execution overhead of UEs is gradually increasing.

TABLE I
THE AVERAGE NUMBER OF ITERATIONS REQUIRED FOR A UE'S TASK TO OBTAIN ITS OPTIMAL STRATEGY IN DIFFERENT SCENARIOS

Scenario		Average iterations	Average overhead ($\times 10^{-7}$)
N	M		
1	1	23	1.55
1	3	63	1.55
3	1	23	1.9
3	3	63	1.69
30	5	40	1.99
30	10	77	1.99
50	50	219	1.2



(a)



(b)

Fig. 4. The comparison between DCOS and other schemes. (a) $M = 15$. (b) $N = 20$.

Thus, we can see that the NM, MP, and DCOS curves first increase. However, according to the definition of UE_i, as i increases, the workload of the task gradually increases, but the local execution overhead decreases, so the tasks of these UEs are executed locally while reducing the average overhead of UEs. Hence, we can see that the NM, MP, and DCOS curves then decrease. Fig. 4(b) shows the comparison between DCOS and other schemes when we set $N = 20$. We can also know that as the number of MEC servers increases, the average overhead of UEs cannot be continuously reduced. As shown in Fig. 4, the DCOS can effectively reduce the average execution overhead of UEs in the heterogeneous environment of MEC. Experiments show that compared with the five other strategies, the DCOS performs better.

Table II shows some tasks' offloading strategy and their overhead when the resource of MEC servers is limited or unlimited (i.e., the MEC servers can execute an unlimited number of tasks simultaneously without incurring additional overhead for UEs). It should be noted that the execution sequence refers to the order in the waiting queue of MEC_j at the time. As shown in the table, under the environment of MEC

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2019.2951206, IEEE Transactions on Industrial Informatics

TABLE II
THE STRATEGY AND OVERHEAD OF UE COMPARISON BETWEEN THE RESOURCE OF MEC SERVERS IS LIMITED AND UNLIMITED ($N = 10, M = 5$)

Task	Execution location		Execution sequence		p_{i,j,t,v_a}		Overhead ($\times 10^{-8}$)	
	Limit	Unlimit	Limit	Unlimit	Limit	Unlimit	Limit	Unlimit
$v_{1,1}$	MEC_4	MEC_1	1	1	0.39	0.35	1.72	1.55
$v_{3,1}$	MEC_2	MEC_1	1	1	0.36	0.35	2.05	2
$v_{5,1}$	MEC_5	MEC_1	1	1	0.4	0.35	3.05	2.72
$v_{7,1}$	MEC_4	MEC_1	2	1	0.39	0.35	3.94	3.57
$v_{10,1}$	MEC_4	MEC_1	3	1	0.39	0.35	5.76	5.11
$v_{10,2}$	MEC_4	MEC_1	2	1	0.39	0.35	5.39	5.11
$v_{10,3}$	MEC_4	MEC_1	2	1	0.39	0.35	5.39	5.11

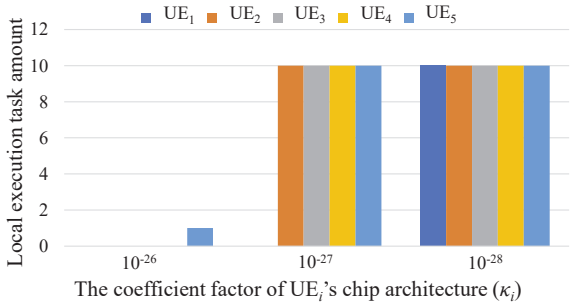


Fig. 5. The impact of κ_i on the local execution task amount.

servers with limited resources, the UE's offloading strategy and overhead are affected by other UEs. The dictionary data structure proposed in this paper can handle the conflicts when multiple UEs simultaneously send requests to the same MEC server. Compared with the related work, the scenario studied in this paper is closer to the real word.

C. The Impact of Other Parameters

To analyze the impact of κ, γ, δ, and the distance (d) between the MEC server and UE on the overhead, in this experiment, we set $N = 5, M = 2$.

1) The impact of κ: κ_i is the coefficient factor of UE_i's chip architecture, which represents the energy consumption per unit cycle of the CPU. As shown in Fig. 5, as κ_i decreases, the number of tasks executed locally increases, that is, the higher energy utilization rate. Therefore, the energy utilization of UE should be further improved to meet the growing demand of the complex applications. Increasing the energy utilization of UE will make the tasks more suitable for local execution, which can also help to improve the QoE.

2) The impact of γ: γ_i denotes the tradeoff parameter of UE_i's energy consumption. As shown in Fig. 6, as γ_i increases, the decision is more inclined to choose a strategy that consumes less energy. Conversely, as γ_i decreases, the decision is more inclined to choose a strategy with less delay. By setting γ_i, the user can set a strategy that is more biased toward delay or energy consumption according to their own needs and the QoE can also be improved.

3) The impact of δ: δ_i represents the processing density of task (bits per cycle). The increase in δ_i will directly increase the overhead required for transmitting tasks. However, when δ_i is increased to a certain extent, the overhead of local execution may be less than the overhead required for remote execution.

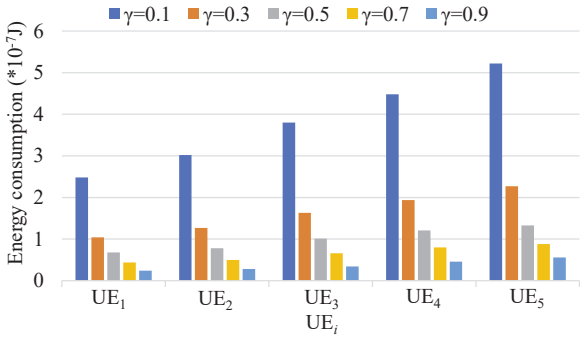


Fig. 6. The impact of γ_i on the execution overhead. (a) The impact of γ_i on the energy consumption. (b) The impact of γ_i on the computational time.

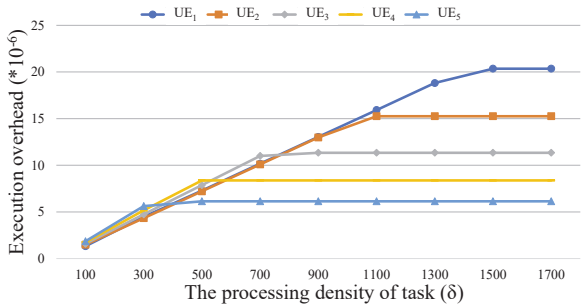


Fig. 7. The impact of δ_{i,v_a} on the execution overhead.

Therefore, we can see that the curves in Fig. 7 grows first and then stabilizes.

4) The impact of d: As can be seen from Fig. 8, $d_{i,j}$ is proportional to the overhead of UEs when the communication environment is certain. Thus, MEC servers should be deployed closer to UE, and the QoS should be further improved to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2019.2951206, IEEE Transactions on Industrial Informatics

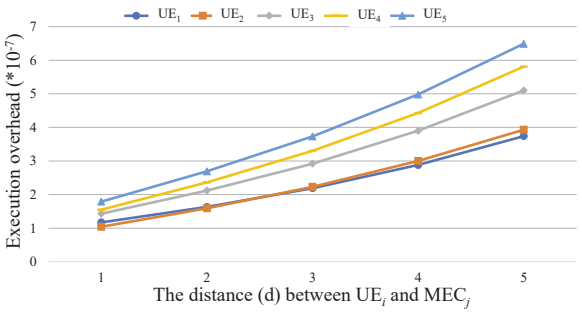


Fig. 8. The impact of $d_{i,j}$ on the execution overhead.

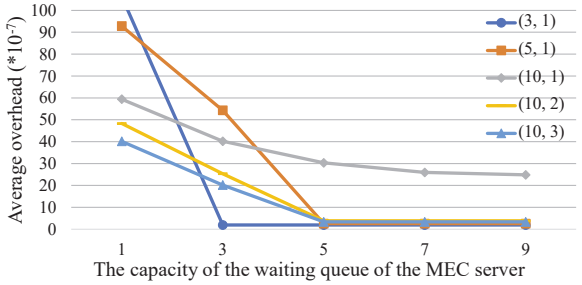


Fig. 9. The impact of the capacity of waiting queue on the average overhead of UEs.

reduce the execution overhead required for UE to perform their application remotely.

D. The Impact of MEC Server

Next, we analyze the impact of the MEC server on the average overhead of UEs, which increase the average utilization of MEC servers and save energy consumption of the MEC server.

1) The impact of the capacity of waiting queue: Fig. 9 shows the impact of the capacity of waiting queue on the average overhead of UEs under the different value of (N, M). It can be seen that increasing the capacity of waiting queue can appropriately reduce the average overhead of UEs. However, when the UE size and application type are fixed, as the resources of MEC servers increase, the overhead cannot be continuously reduced.

2) The impact of MEC server amount: As shown in Table I, when the number of MEC servers and UEs reaches a ratio, the more MEC servers can no longer reduce the overhead. In order to study the appropriate number of MEC servers, we introduce 2 indicators, including MEC execution rate (MER) and MEC utilization rate (MUR). MER represents the proportion of the tasks that are executed remotely and can be gotten from

$$\mathrm{MER} = \frac{\sum_{j=1}^{M}\sum_{i=1}^{N} I_{i,j,v_a}}{\sum_{i=1}^{N}|V_i|}. \tag{20}$$

MUR is the proportion of MEC servers participating in performing the tasks, indicating the average utilization rate of MEC servers. MUR can be obtained from

$$\mathrm{MUR} = \frac{1}{M}\sum_{j=1}^{M}\sum_{i=1}^{N} I_{i,j,v_a}, \tag{21}$$

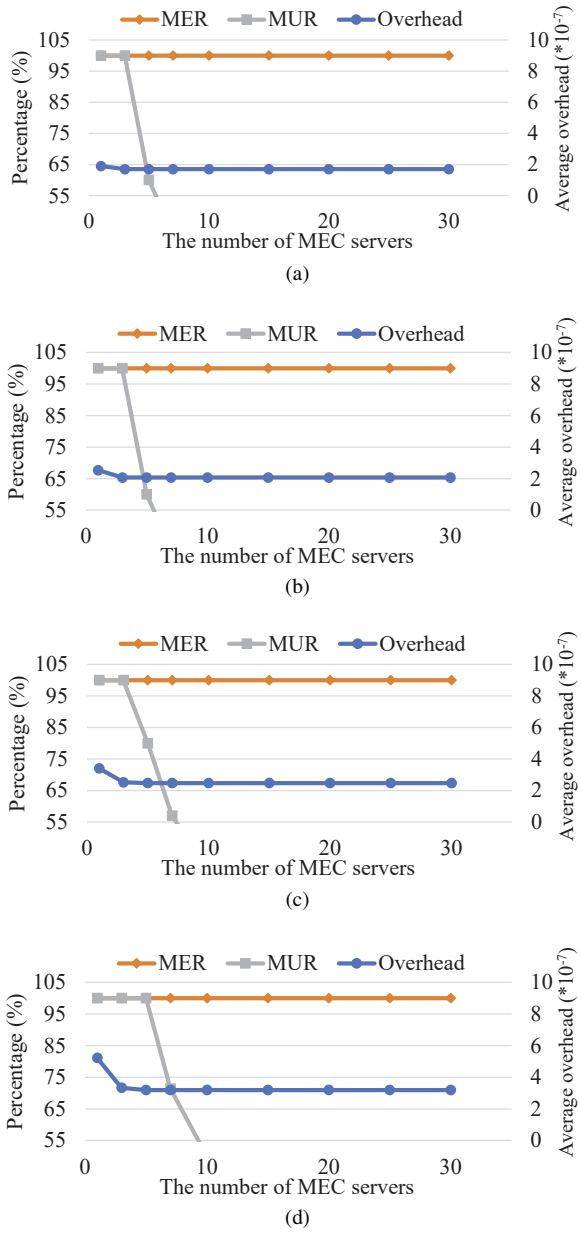


Fig. 10. The impact of MEC server amount on MER, MUR, and the overhead. (a) $N = 3$. (b) $N = 5$. (c) $N = 7$. (d) $N = 10$.

The evaluation of MER and MUR can avoid deploying too many idle MEC servers, and can also reduce the energy consumption of MEC servers. Fig. 10 shows the impact of M on MER, MUR, and the average overhead for different N. As can be seen from Fig. 10, when $M/N \geq 0.5$, there will be some MEC servers in idle state, which will increase the overhead of MEC servers. Thus, from the perspective of MER, MUR, and energy saving, the number of MEC servers needs to be determined based on the application scenario, not the more the better.

V. Conclusion

In this paper, we study the COPO strategy for MUMMS with limited resources, that is, the UEs and MEC servers can only execute tasks in a serial manner, and each of MEC

servers has a finite capacity queue for waiting tasks. We first transform the non-convex optimization problem into a convex optimization problem, define a dictionary data structure for recording the strategies of UE, and propose a decentralized algorithm DCOS to make the offloading strategy for each UE. Second, the effect of DCOS to reduce the overhead and the impact of key parameters on the overhead are demonstrated by the simulation experiments. Finally, we give some suggestions for MEC deployment through analyzing the impact of MEC servers on the overhead. In the paper, we assume that UE remains station while uploading and receiving the data, however, the UE may always be moving. Thus, as the UE keeps moving, we should further study the COPO strategy to bring the scenario closer to the real world.

ACKNOWLEDGMENTS

We would like to express our gratitude to the associate editor and anonymous reviewers for their comments which are very important to improve the quality of the manuscript. This work was partially supported by the National Key Research and Development Program of China (2018YFB1003401, 2018YFB0203804), the National Outstanding Youth Science Program of National Natural Science Foundation of China (Grant No. 61625202), the Program of National Natural Science Foundation of China (Grant Nos. 61876061, 61702170, 61772182, 61802032, 61572176), and the Fundamental Research Funds for the Central Universities.

REFERENCES

[1] ETSI, "New white paper: Etsi's mobile edge computing initiative explained," ETSI, Report, 2015. [Online]. Available: https://www.etsi.org/newsroom/news/1009-2015-09-news-new-white-paper-etsi-s-mobile-edge-computing-initiative-explained

[2] M. V. Barbera, S. Kosta, A. Mei, and J. Stefa, "To offload or not to offload? the bandwidth and energy costs of mobile cloud computing," in *2013 Proceedings IEEE INFOCOM*, April 2013, pp. 1285–1293.

[3] M. V. Barbera, S. Kosta, A. Mei, V. C. Perta, and J. Stefa, "Mobile offloading in the wild: Findings and lessons learned through a real-life experiment with a new cloud-aware system," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, April 2014, pp. 2355–2363.

[4] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1628–1656, thirdquarter 2017.

[5] Y. Mao, J. Zhang, and K. B. Letaief, "A lyapunov optimization approach for green cellular networks with hybrid energy supplies," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 12, pp. 2463–2477, Dec 2015.

[6] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 974–983, April 2015.

[7] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, Dec 2016.

[8] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 1, no. 2, pp. 89–103, June 2015.

[9] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, "Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks," *IEEE Access*, vol. 4, pp. 5896–5907, 2016.

[10] C. Liu, K. Li, J. Liang, and K. Li, "Service reliability in an hc: Considering from the perspective of scheduling with load-dependent machine reliability," *IEEE Transactions on Reliability*, vol. 68, no. 2, pp. 476–495, June 2019.

[11] O. Muñoz, A. Pascual-Iserte, and J. Vidal, "Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 10, pp. 4738–4755, Oct 2015.

[12] K. Li, "A game theoretic approach to computation offloading strategy optimization for non-cooperative users in mobile edge computing," *IEEE Transactions on Sustainable Computing*, pp. 1–1, 2018.

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

[14] J. Xu, L. Chen, and S. Ren, "Online learning for offloading and autoscaling in energy harvesting mobile edge computing," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 3, pp. 361–373, Sep. 2017.

[15] S. Cao, X. Tao, Y. Hou, and Q. Cui, "An energy-optimal offloading algorithm of mobile computing based on hetnets," in *2015 International Conference on Connected Vehicles and Expo (ICCVE)*, Oct 2015, pp. 254–258.

[16] M. Molina, O. Muñoz, A. Pascual-Iserte, and J. Vidal, "Joint scheduling of communication and computation resources in multiuser wireless application offloading," in *2014 IEEE 25th Annual International Symposium on Personal, Indoor, and Mobile Radio Communication (PIMRC)*, Sep. 2014, pp. 1093–1098.

[17] S. Wang and S. Dey, "Adaptive mobile cloud computing to enable rich mobile multimedia applications," *IEEE Transactions on Multimedia*, vol. 15, no. 4, pp. 870–883, June 2013.

[18] S. Guo, B. Xiao, Y. Yang, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, April 2016, pp. 1–9.

[19] L. Yang, J. Cao, H. Cheng, and Y. Ji, "Multi-user computation partitioning for latency sensitive mobile cloud applications," *IEEE Transactions on Computers*, vol. 64, no. 8, pp. 2253–2266, Aug 2015.

[20] Maofei Deng, Hui Tian, and Bo Fan, "Fine-granularity based application offloading policy in cloud-enhanced small cell networks," in *2016 IEEE International Conference on Communications Workshops (ICC)*, May 2016, pp. 638–643.

[21] X. Lin, Y. Wang, Q. Xie, and M. Pedram, "Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment," *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 175–186, March 2015.

[22] P. D. Lorenzo, S. Barbarossa, and S. Sardellitti, "Joint optimization of radio resources and code partitioning in mobile cloud computing," *CoRR*, vol. abs/1307.3835, 2013. [Online]. Available: http://arxiv.org/abs/1307.3835

[23] S. Wang, Y. Zhao, J. Xu, J. Yuan, and C.-H. Hsu, "Edge server placement in mobile edge computing," *Journal of Parallel and Distributed Computing*, vol. 127, pp. 160 – 168, 2019.

[24] 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 *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '10. New York, NY, USA: ACM, 2010, pp. 49–62.

[25] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proceedings of the Sixth Conference on Computer Systems*, ser. EuroSys '11. New York, NY, USA: ACM, 2011, pp. 301–314.

[26] S. Kosta, A. Aucinas, Pan Hui, R. Mortier, and Xinwen Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *2012 Proceedings IEEE INFOCOM*, March 2012, pp. 945–953.

[27] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, S. N. Srirama, and R. Buyya, "mcloud: A context-aware offloading framework for heterogeneous mobile cloud," *IEEE Transactions on Services Computing*, vol. 10, no. 5, pp. 797–810, Sep. 2017.

[28] J. L. D. Neto, S. Yu, D. F. Macedo, J. M. S. Nogueira, R. Langar, and S. Secci, "Uloof: A user level online offloading framework for mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 17, no. 11, pp. 2660–2674, Nov 2018.

[29] B. G. Ryder, "Constructing the call graph of a program," *IEEE Transactions on Software Engineering*, vol. SE-5, no. 3, pp. 216–226, May 1979.

[30] B. Sklar, "Rayleigh fading channels in mobile digital communication systems. i. characterization," *IEEE Communications Magazine*, vol. 35, no. 9, pp. 136–146, Sep. 1997.

[31] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. O. Wu, "Energy-optimal mobile cloud computing under stochastic wireless channel," *IEEE Transactions on Wireless Communications*, vol. 12, no. 9, pp. 4569–4581, Sep. 2013.
[32] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, October 2016.
[33] J. Wallenius, J. S. Dyer, P. C. Fishburn, R. E. Steuer, S. Zionts, and K. Deb, "Multiple criteria decision making, multiattribute utility theory: Recent accomplishments and what lies ahead," *Manage. Sci.*, vol. 54, no. 7, pp. 1336–1349, Jul. 2008.
[34] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge University Press, 2004.

Yan Ding received his B.S. degree in Software Engineering from North University of China in 2014, and M.S. degree in Computer Application Technology from Xinjiang University, Urumqi, China, in 2018. He is currently pursuing the Ph.D. degree with the Hunan University, China. His research interests include mobile edge computing, data analysis, machine learning, and network security. He is a student member of the IEEE and CCF.

Chubo Liu received the B.S. degree and Ph.D. degree in computer science and technology from Hunan University, China, in 2011 and 2016, respectively. He is currently an associate professor of computer science and technology at Hunan University. His research interests are mainly in game theory, approximation and randomized algorithms, cloud and edge computing. He has published over 20 papers in journals and conferences such as the IEEE Transactions on Parallel and Distributed Systems, the IEEE Transactions on Cloud Computing, the ACM Transactions on Modeling and Performance Evaluation of Computing Systems, the Theoretical Computer Science, and ICPADS.

Xu Zhou received the master's degree in the Department of Information Science and Engineering, Hunan University, in 2009. She is currently an associate professor in the Department of Information Science and Engineering, Hunan University, Changsha, China. Her research interests include parallel computing and data management.

Zhao Liu is currently working toward the Ph.D. degree from College of Information Science and Engineering in Hunan University, China. His research interest includes crowd sensing, game theory, mobil computing and cloud computing.

Zhuo Tang received the Ph.D. in computer science from Huazhong University of Science and Technology, China, in 2008. He is currently a professor of the College of Computer Science and Electronic Engineering at Hunan University, and is the associate chair of the department of computing science. His majors are distributed computing system, cloud computing, and parallel processing for big data, including distributed machine learning, security model, parallel algorithms, and resources scheduling and management in these areas. He has published almost 50 journal articles and book chapters. He is a member of ACM and CCF.