Journal of Grid Computing (2022) 20:11
https://doi.org/10.1007/s10723-022-09599-x

Mobility-Aware and Code-Oriented Partitioning Computation Offloading in Multi-Access Edge Computing

Yaqin Liu · Chubo Liu · Jing Liu · Yikun Hu · Kenli Li · Keqin Li

Received: 15 March 2021 / Accepted: 20 January 2022

Abstract Multi-Access edge computing (MEC) enables less resourceful smart mobile devices (SMDs) to use computation and memory intensive applications by offloading them to edge servers with tolerant latency, significantly improving the computing paradigm of SMDs. But, when involving mobility in MEC, offloading strategy and overhead can be significantly influenced by the movements of SMDs. What′s more, the movements of SMDs make it harder to deal with precedence among subtasks. However, to the best of our knowledge, few articles have studied mobility management in code-oriented partitioning offloading. To give an efficient solution, we propose the cost-saving offloading policy with mobility prediction using convex optimization and Lagrangian approach. Our scheme can help moving SMDs in MEC like driverless vehicles efficiently complete their tasks and reveal the impact of task dependency on completion time. The experimental results show that our algorithm can achieve at least 12% performance improvement on average than other three common methods.

Keywords Computation offloading · Multi-access edge computing · Mobility management · Resource allocation · Task dependency

Y. Liu · C. Liu (✉) · Y. Hu · K. Li · K. Li
Department of Information Science and Engineering, Hunan University, Changsha, China
e-mail: liuchubo@hnu.edu.cn

C. Liu
e-mail: liuyaqin@hnu.edu.cn

Y. Hu
e-mail: yikunhu@hnu.edu.cn

K. Li
e-mail: lkl@hnu.edu.cn

K. Li
e-mail: lik@newpaltz.edu

Y. Liu · C. Liu · K. Li · K. Li
National Supercomputing Center in Changsha, Changsha 410082, China

K. Li
Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

J. Lui
Department of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan 430065, China
e-mail: Idealer@126.com

1 Introduction

1.1 Motivation

Recently, smart mobile devices (SMDs) including phones, tablets and IoT or driveless vehicles have become more and more popular because of their convenience and increased capability, driving many computation-intensive and delay-sensitive applications (e.g., interactive online games [1], image and video processing [2], cloud-like services such as m-health-care [3], and especially some frequent applications like object detection in autonomous driving [2].) However, due to the

Published online: 19 March 2022

limited resources of SMDs (battery energy, storage size, and computing capability), users cannot often get satisfied with the quality of service (QoS). To handle the explosive computation demands and the ever-increasing QoS requirements, multi-access edge computing (MEC) was proposed as one of the most promising solutions [4].

MEC aims to provide nearby users some cloud-like services with low latency by deploying limited computing and storage resources close to them. Resource-hungry devices can offload computation-intensive jobs to edge servers via wireless access to release resource tension. Compared with traditional cloud computing, MEC is closer to users in both network and geographical distances, and thus can greatly reduce network delay and bandwidth consumption to improve performance. But, designing a cost-saving offloading scheme with expected delay is challenging, especially when end devices move across coverage areas of multiple MEC centers carrying directed acyclic graph (DAG, consists of subtasks from a divisible application and their dependency relationships.)

The reason lies in that movement makes the device′s surrouding keep chaging, and parameters of MEC servers and network will be changing with it, which makes SMDs trigger frequent service migration, need remote comunication, or take temporarily optimal policies. It also makes task dependency harder to handle. As shown in Fig. 1, task 6 need task 4′s result to start execution according to task dependency, while unpredictable service migrations make where to get the result unknown, e.g., when Car 1 moves from Location 1 to 2, Server 4 is also in touch. Car 1 may suffer a disconnection with server 1 or it may migrate task 4 to the new server. Task 6′s execution time is unpredictable for making offloading strategy, until its completion. Moreover, Car 1 may find that it should offload task 4 to Server 3 at the beginning because the uploading rate to Server 3 increases fast when it is approaching and the whole cost is thus lower than that of Server 2.

1.2 Related Work

Computation offloading is critical in MEC. It determines computation efficiency and achievable performance [5]. Various computation offloading strategies have been developed in the literature, and they can be generally categorized as: delay based offloading [6–9], energy based offloading [10–13], and offloading based on trade-off between execution delay and energy consumption [3, 14, 15].

Delay based offloading policies first considered either network delay [6] or execution delay [7], they finally aggregated both [8, 9] but failed to be energy-saving. Energy based offloading policies aimed to lower energy consumption [10] or charge SMDs by acquiring energy from other devices [11]. They further studied the multi-user scenario which involves channel interference, request collision, and queue model [12, 13]. More and more researches now target at minimizing execution overhead (a weighted sum of both execution delay and energy consumption). In [14], Mao et al. reduced overhead by adjusting

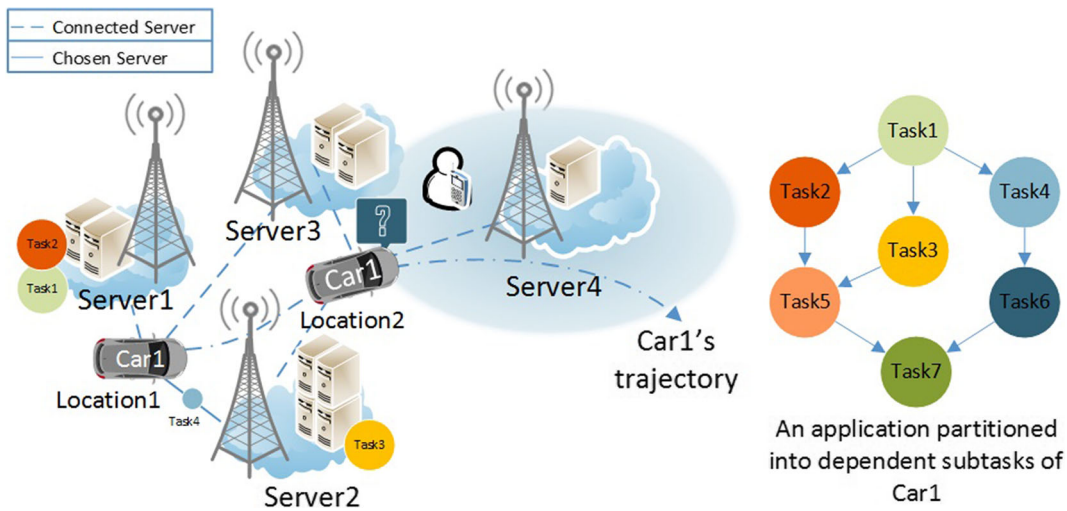


Fig. 1 Mobility management in highly overlapped server coverage areas

Table 1 Comparisons between proposed algorithm and other offloading schemes

Schemes	Environmental characteristic(s)	Task dependency	Mobility management	Objective(s)
Sun et al. [8]	Multiple SMDs and Multiple Resource-limited MEC servers	NO	NO	Minimize average delay
Sardellitti et al. [12]	Multiple SMDs and Multiple MEC servers	NO	NO	Minimize energy, Satisfy delay constraint
Mao et al. [14]	One SMD and One MEC server	NO	NO	tradeoff between delay and energy
Wang et al. [22]	One SMD and Multiple MEC servers	NO	YES	Minimize delay
Xu et al. [23]	One SMD and Multiple Heterogeneous MEC servers	NO	YES	tradeoff between delay and energy
Ding et al. [24]	One SMD and Multiple MEC servers	NO	YES	Minimize delay while satisfy energy constraint / Minimize energy while satisfy delay constraint
MCPO	One SMD and Multiple Resource-limited Heterogeneous MEC servers	**YES**	**YES**	tradeoff between delay and energy, Satisfy delay constraint

transmission power and task offloading strategies. In [3] and [15], like Mao, applications were divided into a collection of consecutive or independent subtasks, but the authors studied scenarios of multiple edge servers, which makes offloading problem more complex because of server selection and switch.

Although the last two algorithms partitioned the applications, they did not investigate code-oriented partitioning offloading (COPO), i.e., investigate applications that can be split into subtasks with dependencies. COPO has been studied in [16–18]. In [16], Yang et al. considered both the partitioning of computations for multi-users and the scheduling of offloaded computations on cloud resources. While lin et al. scheduled tasks on not only cloud resources but also local cores [17], but they did not partition the cloud servers into several virtual machines like Yang. Deng et al. in [18] offered solution for partitioning offloading in cloud-enhanced small cell networks, an environment more similar to MEC. All of them studied COPO with resourceful servers and targeted either delay or energy. Some even disregard data transferring time or data receiving time. Ding et al. involved COPO with MEC, but they also once optimized only one target [19]. What′s more, these researches failed to reveal that SMDs always weight completion time of some predecessors more because of task dependency requirements.

Moreover, none of the aforementioned researches have taken SMDs′ mobility into account. When involving mobility in MEC, making computation offloading strategy becomes even harder. There are emerging some work studying this issue [20–24]. Tarik et al. in [20] raised service migration policy to cope with mobility. In [21], Lordan et al. offered a approach to deal with both mobility and trade-off between performance and energy. But they are designed for mobile cloud computing, and can not be well applied in MEC. Wang et al. delivered a Q-learning based mobility management policy with partial server-side information [22]. In [23], Xu et al. also studied the partial information scenario, and they introduced service migration cost. Ding et al. in [24] developed different service migration strategies depending on mobility types. But neither did they consider the challenges mobility brings to COPO (the unpredictable completion time caused by mobility makes static scheduling policies not applicable for COPO to reduce cost.) nor did they solve the impact of movements on offloading during data transmission process, i.e., they ignore

the influence of distances with servers on data uploading rates. They simply use the instant uploading rates when the task is ready to make offloading policies (Table 1).

1.3 Our Contributions

Although computation offloading and mobility management have been two popular research directions in edge computing, few articles have considered user mobility in computation offloading, let alone COPO. This article aims to provide a cost-saving offloading strategy for SMDs represented by driverless vehicles moving in multi-MEC enabled networks, and show how the strategy help them efficiently complete their computation-intensive and delay-sensitive tasks with MEC. The optimal offloading policy is made by comparing the execution cost of the tasks executed on different entities. And our policy minimizes execution overhead of SMDs by controlling the CPU clock frequency and managing users′ mobility.

The main contributions and differences of this paper are listed as follows:

- We consider SMDs′ mobility in computation offloading stage including data uploading phase. Using mobility prediction method to avoid disconnection and considering the changes of data uploading rate caused by movements of SMD.
- We formulate the problem into an execution overhead minimization problem under constraints of completion deadline and task dependency, prove there exists an optimal solution to the problem, and reveal how they influence computation completion time.
- We characterize the performance of our algorithm, and demonstrate the impact of mobility and various parameters on offloading strategy by extensive experiments. Moreover, the experiments adopt resource-constraint heterogeneous MEC servers.

The rest of the paper is organized as follows. We first introduce the system model and formulates the problems mentioned above in Section 2. Then we propose and describe the corresponding algorithm, a mobility-aware and code-oriented partitioning offloading algorithm (MCPO) in Section 3. In Section 4, some simulation experiments are done to evaluate the algorithm in this paper. Finally, we carry on a summary to this article in Section 5.

2 System Model and Problem Formulation

In this section, we formally formulate our system model and corresponding optimization problem.

2.1 Network and Task Model

Figure 1 shows our mobile edge environment with deployment of edge servers owning heterogeneous network and computing resources. Let M be the number of MEC servers and $\mathscr{M} = \{1, \ldots, M\}$ be the corresponding server set. Especially, 0 represents the local SMD. The maximum CPU frequency and coverage radius of a server $m \in \mathscr{M}$ are represented respectively by $F_{\max}$ and $Occ(m)$.

SMDs move in the mobile edge environment with a computing application to be executed locally or remotely. Each application can be described as a four-tuple $\langle D, L, \gamma, T_{\max}\rangle$, where D is the size of input data, $D_{\max}$ is the maximum possible size, and L is the computation workload, that is, how many CPU cycles are needed to execute the task. γ represents the output of a task, and $\gamma_{k,n}$ indicates the data size that task n need to receive form its predecessor k's output, and $T_{\max}$ represents the completion deadline. Computing applications are dynamically divided into N ordered subtasks, denoted by $\mathscr{N} = \{1, \ldots, N\}$. The dynamic partitioning method in [25] is adopted in this paper, and other partitioning methods can also be used [26, 27]. Denote D_n and L_n respectively as the data size and the computation workload of subtask n. Use $\chi = \{\chi_{n,m} | n \in \mathscr{N}, m \in \mathscr{M} \cup \{0\}\} \in \{0, 1\}^{N\times(M+1)}$ to represent task offloading strategy matrix, and $\chi_{n,m} = 1$ to indicate that task n is unloaded to server m for execution.

Dependencies between subtasks can be represented by a DAG $G = (V, E)$. Each node $i \in V$ in G represents a subtask and a directed edge $e(i, j)$ indicates the precedence constraint between tasks i and j such that task j cannot start execution until its precedent task i completes. Taking face recognition as an example. Face recognition application is executed in two separate phase: face detection and face recognition. In detection phase, taking algorithm to narrow the initial set of face location candidates to the final set of detected face locations. Then the detected face will be compared with template image in a database separately using other algorithm.

$f_{m,n}$ indicates CPU frequency that server m can allocate to task n. Especially, $f_{0,n}$ represents local CPU frequency dealing with task n. Distance between entity (local SMD or MEC servers) m' and m when task n is ready is denoted by $\mathscr{D}=\left\{d_{n,m',m}|n\in\mathscr{N},m',m\in\mathscr{M}\cup\{0\}\right\}$. Since no prior knowledge about the user's trajectory is needed, the trajectory can be generated by any mobility model such as a random path-point model. For a task $n\in\mathscr{N}$, when $d_{n,0,m}<Occ(m)$, it can be offloaded to server m for remote execution. In our work, we denote $\mathscr{A}(n)\subseteq\mathscr{M}$ as the set of available MEC servers for task n.

2.2 Communication Model

The input data of subtasks should be transmitted to the chosen edge server for remote execution. Denote server $m\in\mathscr{A}(n)$ as the subtask n offloaded to, and p_n as the transmission power. Using the set $\mathscr{V}=\{v_{n,m}|n\in\mathscr{N},m\in\mathscr{M}\}$ to represent the velocity the SMD relative to server m when executing task n. According to the Rayleigh fading channel model [28], the achievable rate of task n transferred from entity m' to entity m can be given as

$$R\left(n,m',m\right)=W_{m',m}\log_2\left(1+\frac{G_0\left(d_0/dis\right)^{\theta}p_n}{N_0W_{m',m}}\right),\tag{1}$$

where G_0 is the path loss constant, θ is the path loss index, d_0 is reference distance, N_0 is the noise power spectral density of the MEC server receiver, $W_{m',m}$ is the channel bandwidth. dis represents the distance of the SMD m' to edge server m, so, given the speed $v_{n,m}$, $dis=d_{n,m',m}+v_{n,m}t$.

2.3 Computation Model

We use CT_n^l, $CT_{n,m}^c$ to represent the completion delay of task n executed locally or remotely (running on the server m), and use ET_n^l, $ET_{n,m}^l$, $TT_{n,m}$ to represent the local execution delay, remote execution delay and the transfer time respectively. Next we define the ready time and Weighted Cost of Energy and Delay (WCED) for task n.

Definition 1 (Ready Time) The ready time of a task is defined as the earliest time when all its immediate predecessors have been completed. The formula is defined as

$$RT_n=\max_{k\in pred(n)}\{CT_k\},\tag{2}$$

where $pred(n)$ represents the set of all predecessors of task n.

Definition 2 (Weighted Cost of Energy and Delay) Weighted Cost of Energy and Delay (WCED) is defined as a weighted sum of energy consumption and the corresponding completion delay. When the WCED for performing a task is low, we call it cost-saving. The WCED of task n can be expressed as

$$Z_n=\omega_n^E E_n+\omega_n^T CT_n,\tag{3}$$

where $0\le\omega_n^E E_n\le 1$ and $0\le\omega_n^T CT_n\le 1$ represent the weights of energy consumption and completion time respectively when making offloading strategy of task n, and $\omega_n^E+\omega_n^T=1$. Devices can set different weight values to satisfy different types of applications such as delay-sensitive applications and delay-tolerate applications.

2.3.1 Local Computing

Denote $\xi=\left[f_{0,n},n\in\mathscr{N}\right]^T$ as the local CPU clock frequency (GHz) vector. Then the local execution delay of task n can be represented as

$$ET_n^l=L_n/f_{0,n}.\tag{4}$$

The energy consumed by the local processing of the corresponding task is

$$E_n^l=\alpha L_n f_{0,n}^2,\tag{5}$$

where α is a constant, depending on the effective optional capacitance of the chip structure. In order to make the energy consumption of the local processing consistent with that in [29] $\left(\alpha f^3=0.8w\right)$, $\alpha=10^{-5}$ in this paper. We can adjust the clock frequency of CPU chips by using DVFS technology to achieve the optimal computation time and energy consumption on mobile devices.

Obviously, the local completion time of task n is

$$CT_n^l=RT_n+rt_{m',m}^k+ET_n^l,\tag{6}$$

where $rt_{m',m}^k$ represents delay for receiving partial result of predecessor k from m'. And $rt_{m',m}^k=\gamma_{k,m}/R\left(n,m',m\right)$, especially, $rt_{m',m}^k=0$ if $m'=m$. Then the local WCED of task n is

$$Z_{n,0}=\omega_n^E E_n^l+\omega_n^T CT_n^l.\tag{7}$$

2.3.2 Remote Computing

With data transmission rate $R\left(n, m', m\right)$, data transfer delay can be given as

$$TT_{n,m} = D_n / R\left(n, m', m\right). \tag{8}$$

Then the corresponding energy consumption is

$$E^c_{n,m} = TT_{n,m}\, p_n + E^{const}_n, \tag{9}$$

where E^{const}_n represents the energy consumption that the SMD contends access channel for uploading task n (determined by MAC protocol channel contention time and cloud acceptance control) plus the tail energy that can keep the channel for some time after data transmission.

As defined above, the CPU frequency that server m can allocate to task n is $f_{m,n}$, so the execution delay of task n on server m is

$$ET^c_{n,m} = L_n / f_{m,n}. \tag{10}$$

Then the completion time for task n on server m is

$$CT^c_{n,m} = RT_n + TT_{n,m} + rt^k_{m',m} + ET^c_{n,m}. \tag{11}$$

Completion time consists of task ready time, data uploading time, execution delay, and predecessor results receiving time. Note that because completion times of n's predecessors are unpredictable, data transmission of task n must begin after their completion.

From (9) and (11), it can be concluded that the WCED of task n on server m is

$$Z_{n,m} = \omega^E_n E^c_{n,m} + \omega^T_n CT^c_{n,m}. \tag{12}$$

We can observe from formula (8) and (9) that low data transmission rate $R\left(n, m', m\right)$ will lead to long transmission time for offloading input data to the server and high energy consumption for wireless access. In this case, computing tasks locally is much better for mobile devices.

2.4 Problem Formulation

For a given task sequence set $\mathcal{N}$ of the application, the WCED of the application can be calculated as

$$\begin{aligned} Z &= \sum_{n=1}^{N} Z_n \\ &= \sum_{n=1}^{N}\sum_{m=0}^{M} \chi_{n,m} Z_{n,m} \\ &= \sum_{n=1}^{N}\Bigg[\chi_{n,0}\left(\omega^E_n E^l_n + \omega^T_n CT^l_n\right) \\ &\quad + \sum_{m=1}^{M} \chi_{n,m}\left(\omega^E_n E^c_{n,m} + \omega^T_n CT^c_{n,m}\right)\Bigg], \end{aligned} \tag{13}$$

where, both CT^l_n and E^l_n are functions of local clock frequency, while the others are constants. In this paper, the optimal offloading decision and the optimal CPU frequency are obtained to reduce the WCED of the application. Then, the problem that the application has the lowest WCED and meets the application delay limit can be expressed as

$$P: \quad \min_{\chi,\xi} \sum_{n=1}^{N} Z_n, \tag{14}$$

$$\text{s.t.} \quad \max_{n\in\mathcal{N}}\{CT_n\} \le T_{\max}, \tag{15}$$

$$0 \le f_{m,n} \le F_{\max}, \tag{16}$$

$$\max_{k\in pred(n)} CT_k \le RT_n, \tag{17}$$

$$\forall \chi_{n,m} \in \{0, 1\}, \tag{18}$$

$$\text{if } \chi_{n,m} = 1, m \in \{0\} \cup \mathscr{A}(n), \tag{19}$$

where χ is the task offloading strategy matrix, and ξ is the local CPU clock frequency vector. The constraint (15) indicates that the maximum completion time of all subtasks is limited by the specified application completion delay $T_{\max}$. The setting of $T_{\max}$ depends on the application's delay requirement, i.e., the delay-tolerant application can have a high maximum completion time; otherwise, it is best to set a low maximum completion time. The (16) is the CPU frequency constraint. The constraint (17) indicates that the start time of the task cannot be earlier than the completion time of all its predecessors. The constraint (18) indicates that task n can and must be offloaded to edge servers for execution or executed locally. The constraint (19) indicates that task n can only be offloaded to the servers with which the SMD has connection or executed locally. While with mobility, $\mathscr{A}$ of the SMD is changing along time, so solutions should be taken to avoid disconnection.

The key challenge to solve the optimization problem P in (14) is that integer constraint $\chi_{n,m} \in \{0, 1\}$ makes problem P a mixed integer nonlinear programming problem, which is non-convex and NP hard in common [30]. To solve the problem, the binary offloading decision variable is relaxed to a real

number between 0 and 1, as in literature [31, 32]. To satisfy the constraint (15), we obtain a Lagrangian function of problem (14),

$$L(\kappa, \xi, \chi) = \sum_{n=1}^{N} \sum_{m=0}^{M} \chi_{n,m} Z_{n,m} + \kappa (CT - T_{\max}), \tag{20}$$

where CT represents the completion time of the latest completed subtask, and κ is the penalty factor on the objective function, which makes the function tend to be optimal under the constraints of time conditions (15). It is the cost that the algorithm must pay in order to satisfy the constraints. According to [33], the dual problem of the original problem (14) is

$$\max_{\kappa} \min_{\chi,\xi} L(\kappa, \xi, \chi). \tag{21}$$

Using layering as Optimization Decomposition (LOD) method to decompose the dual problem (21) into two-layer structures [33]. LOD can yield an optimal solution for the dual problem (21) and its original problem (14), which has no duality gap with (21). The first layer is the internal minimization of (21). The second layer, external maximization in the equation.

3 MCPO Algorithm

We can see from problem P, each application owns a completion deadline and its subtasks have dependency constraints. The system aims to lower the execution overhead by allocating the subtasks to best performance entity. When involving mobility management, we adopt a mobility prediction method to eliminate its impact on COPO and deliver offloading decisions which is more likely not temporally optimal as shown in algorithm 1.

3.1 Local Clock Frequency Control

When task n is processed locally, $\chi_{n,0} = 1, \chi_{n,m} = 0, m \neq 0$. The local clock frequency $f_{0,n}$ should satisfy the following WCED minimization formula,

$$Z_n = \sum_{n=1}^{N} \left[\chi_{n,0} \left(\omega_n^E E_n^l + \omega_n^T CT_n^l \right) + \sum_{m=1}^{M} \chi_{n,m} \left(\omega_n^E E_{n,m}^c + \omega_n^T CT_{n,m}^c \right) \right], \tag{22}$$

and be bound by the condition (16). (22) is a convex function of $f_{0,n}$, and we could rewrite this as

$$\min_{f_{0,n}} \left(\omega_n^t CT_n^l + \omega_n^E E_n^l + \kappa CT_n^l \right). \tag{23}$$

Algorithm 1 MCPO algorithm.

Require: $\mathcal{N}$:the set of N ordered tasks; $\mathcal{A}(n)$:the set of available severs for task n; $pred(n)$:the set of immediate predecessors of task n; $\mathcal{D}$:the set of distances; $\mathcal{V}$:the set of velocity of the SMD; ϵ:an infinitesimal number;

Ensure: $\{\chi, \xi\}$:optimal clock frequency control and offloading decision;

Initialize:$D_n, L_n, \omega_n^E, \omega_n^T, step(x), \kappa, \{f_{0,n}\}$and iteration index $x \leftarrow 1$;
for $n = 1 \rightarrow N$ **do**
  /*Calculation of ready time*/
  if $pred(n) == \emptyset$ **then**
    $RT_n \leftarrow 0$
  else
    $RT_n = \max_{k \in pred(n)} \{CT_k\}$
  end if
  /*Clock frequency control*/
  $x \leftarrow 1$
  repeat
    Compute $ET_n^l, E_n^l, CT_n^l, Z_{n,0}^l$by using (4)–(7)
    Calculate $C_n^l = z_n^l + \kappa CT_n^l$
    Compute the clock frequency $f_{0,n}$ by using (25)
    Update Lagarangian multipliers $\kappa(x+1)$ by using (31)
    $x \leftarrow x + 1$
  until $|\kappa(x+1) - \kappa(x)| < \epsilon$
  /*Accurate distance computing*/
  $x \leftarrow 1$
  for $m = 1 \rightarrow M$ **do**
    if $m \notin \mathcal{A}(n)$ **then**
      $C_{n,m}^c \leftarrow +\infty$
    end if
    Compute $TT_{n,m}$ by using (8) and (27)
    Compute $E_{n,m}^c, ET_{n,m}^c, CT_{n,m}^c$
    if $d_{n,m',m} + v_{n,m} TT_{n,m} > Occ(m)$ **then**
      $CT_{n,m}^c \leftarrow CT_{n,m}^c + \tau$
    end if
    Compute $Z_{n,m}$ by using (12)
    Calculate $C_{n,m}^c = Z_{n,m} + \kappa CT_{n,m}^c$
  end for
  $C_n^c \leftarrow min\{C_{n,m}^c\}$
  /*Computation offloading selection*/
  if $C_n^c > C_n^l$ **then**
    $\chi_{n,0} \leftarrow 1$
  else
    $\chi_{n,arg_m \min\{C_{n,m}^c\}} \leftarrow 1$
  end if
end for

It′s not difficult to see that the function is consecutive except for the point $f_{0,n} = 0$. RT_n is a constant associated only with the completion delay of precursor task, which is known here. Taking the derivative function, and then setting the derivative to be 0, that is

$$\min_{f_{0,n}} \left[\left(\omega_n^t + \kappa \right) \left(L_n f_{0,n}^{-1} + RT_n \right) + \omega_n^E \alpha L_n f_{0,n}^2 \right]. \tag{24}$$

The value of $f_{0,n}$ that minimizes (23) can be obtained,

$$f_{0,n} = \sqrt[3]{\frac{\omega_n^t + \kappa}{2\omega_n^E \alpha}}. \tag{25}$$

As you can see, the optimal clock frequency for processing task n depends on the weight of the completion time ω_n^T, the weight of the energy consumptionω_n^E, and the lagrangian multiplier κ. When ω_n^T and κ increases, i.e., when the system attaches more importance to the completion time and the completion deadline constrain, the local frequency increases. When ω_n^E increases, the system will pay more attention to a low energy consumption, so the frequency decreases.

3.2 More Accurate Data Transmission Delay

We adopt a mobility prediction method to manage movements of SMDs. During the data uploading period, if SMD moves at a high speed, it may move out of the coverage area of some servers, resulting in disconnection with them. While SMD is near the center of a server, a small shift can also cause a big change in the data uploading rate. Therefore, we consider SMDs′ mobile tendency, i.e., close to or away from the servers, and capture the variation of the distances to edge servers during the uploading period. The speed of SMD relative to the server is predicted by the mobility of the SMD before (the movement of SMDs has inertia, and its mobility generally changes little in a short time). To get a more accurate uploading delay mainly includes two parts: one is considering the changes of data uploading rate caused by displacements of SMDs during data uploading time; the other is reducing the possible disconnection caused by movements of SMDs.

If task n is processed in the cloud, problem P seems to be a fixed value, which does not change with the behavior of the SMD, but in fact, the data upload time is a function of the distance between the SMD and the server. The upload data size is the integral of upload rate $R\left(n, m', m\right)$ to time t, then the data upload time $TT_{n,m}$ can be calculated by the following formula,

$$\int_0^{TT_{n,m}} W_{m',m} \log_2 \left(1 + \frac{G_0 \left(d_0 / dis \right)^\theta p_n}{N_0 W_{m',m}} \right) dt = D_n, \tag{26}$$

where $dis = d_{n,m',m} + v_{n,m} t$, and it captures the distance variation of the SMD during the uploading time to help calculate a more accurate uploading time, while $d_{n,m',m}$ does not. Suppose that the SMD′s distances to servers are recorded at set intervals. Given the set interval $inter$ and distances dis, we can get $v_{n,m}^t = \left(dis^t - dis^{t-1} \right) / inter$. In this way, we can avoid analysing the concrete mobility model of the SMD for managing its mobility. However, it is hard to get accurate $TT_{n,m}$ directly through (26) via Intergration by Substitution and Part or get an approximation via various interpolation methods. So the distance dis is calculated by a number of iterations, the result of which can be approximately regarded as the average distance between the SMD and the server during data uploading time. The calculation steps are as follows:

$$\begin{aligned} eL &= dn, m', m + \frac{D_n V}{2R(n,m',m)}, \\ R\left(n, m', m\right) &= W_{m',m} \log_2 \left(1 + \frac{G0 (d_0 / eL)^\theta p_n}{N_0 W_{m',m}} \right), \end{aligned} \tag{27}$$

among them, eL is an intermediate variable used to hold the updated distance of each iteration. After several iterations, eL obtained approaches the actual average distance, which is more accurate than the one when the task is ready, i.e., dn, m', m.

Method to relieve the disconnection problem is to predict which servers the SMD may lose connection with. If $d_{n,m',m} + v_{n,m} TT_{n,m} > Occ(m)$, we think the SMD will lose connection with the server m. Add extra time costs τ to completion times of the task on those servers, indicating that, in practice, the task needs to transfer data through core network or perform service migration, which causes additional overhead. Then, the completion delay can be compared with that of other servers.

3.3 Selection Policy

The most important function of computation offloading is to help the SMD decide whether the current

task should be uploaded to the cloud, and if so, which server should be selected for offloading. The selection strategy for computation offloading in this paper is based on the goal of lowering WCED. Local WCED can be expressed as, $C_n^l = Z_n^l + \kappa C T_n^l$. And $C_{n,m}^c = Z_{n,m}^c + \kappa C T_{n,m}^c$ represents WCED at each edge server. So the WCED of remote execution, i.e., the lowest WCED at all edge servers is, $C_n^c = \min\left\{C_{n,m}^c\right\}, m \in \mathscr{A}(n)$. As a result,

$$\chi_{n,0} = \begin{cases} 0, & C_n^c < C_n^l; \\ 1, & \text{otherwise.} \end{cases} \tag{28}$$

When $\chi_{n,0} = 1$, $\chi_{n,m} = 0, m \in \mathscr{A}(n)$. While $\chi_{n,0} = 0$, i.e., $C_n^c < C_n^l$, the optimal decision can be gotten from,

$$\begin{aligned} &\min_{\chi_{n,m}} \textstyle\sum_{m=1}^{M} \chi_{n,m} C_{n,m}^c, \\ &\quad \text{s.t.} \quad (18) \text{ and } (19). \end{aligned} \tag{29}$$

If $C_{n,j}^c = \min\left\{C_{n,m}^c\right\}$, $\chi_{n,j} = 1$, and $\chi_{n,m} = 0, m \neq j$.

As can be seen from the above formulas of WCED, the final offloading strategy is related to the local clock frequency, data transmission power, data upload speed and the completion delay of precursor tasks. When the WCED of local execution at the optimal CPU frequency is lower than the minimum WCED of all remote execution, the data will be executed locally. The optimal CPU frequency is determined by the Lagrange penalty factor, the weight of energy consumption and the weight of execution delay.

3.4 Update the Lagarangian Multiplier

The outer layer maximization problem in (21) can be solved by subgradient method. The partial derivative of (20) with respect to Lagrange multiplier κ is,

$$\frac{\partial L(\kappa, \xi, \chi)}{\partial \kappa} = CT - T_{\max}, \tag{30}$$

So for a given set of local clock frequencies, we can through the following formula to update the Lagrange multiplier,

$$\begin{aligned} \kappa(x+1) &= \left[\kappa(x) - step(x) \tfrac{\partial L(\kappa,\xi,\chi)}{\partial \kappa}\right]^+ \\ &= \left[\kappa(x) - step(x)\left(T_{\max} - CT\right)\right]^+, \end{aligned} \tag{31}$$

where the $step(x)$ function is a function of the number of iterations x. In order to ensure that the final increment of κ is less than an infinite number ϵ, i.e., it converges to a certain local maximum value, the value of $step(x)$ cannot be too large, which will cause the function to oscillate back and forth around the maximum value and cannot converge.

4 Experiment Analysis

In this section, a simulation study is carried out to verify our analysis and evaluate the performance of the proposed mobility management scheme. To the best of the authors′ knowledge, there is little work studying the problem of code-oriented and cost-saving offloading when mobile devices move in the highly overlapped coverage of edge servers. The solution to this problem requires a combination of computation offloading strategy and mobility management, which means considering the mobility of devices in computation offloading phase to avoid frequent service migration. So, in this case, we′re going to use E^2M^2 [23], another offloading algorithm with mobility management, the delay-optimal algorithm and the energy-optimal algorithm as the comparison algorithms.

E^2M^2: the whole trip duration is divided into I frames, and each frame contains J periods, i.e., $T = IJ$. Set the weight of energy consumption zero and reset the weight of delay per frame. In each period, the weight of energy consumption is calculated according to the energy deficit queue. The server with a weighted minimum of time delay and energy consumption in each period is selected for offloading. E^2M^2 considers SMDs′ mobility but don′t consider the task dependency.

Delay-optimal: divides the application into a fine-grained set of tasks. The task will be offloaded to the server with the lowest latency or be executed locally when the local latency is less then the edge cloud.

Energy-optimal: the application is divided into a fine-grained set of tasks. The task will be offloaded to the server with the lowest energy consumption or be executed locally when the energy consumption is less then the edge cloud.

4.1 Parameter Set Up

Consider a typical mobile user with one SMD moving T time periods (T is unknown, while the value of each time period is fixed and same) in the network. This article simulated an 800 m × 800 m square gird

area with 25 servers. Properties such as server distribution, server performance and coverage radius can be set arbitrarily according to the real situation. In this paper, for the sake of simplicity, it is assumed that all the servers are uniformly distributed in the square grid region, i.e., each server is located in one small cell′s center, and the grid size is 160 m. Coverage diameter of each server is 250 m. Each time period is 2 minutes. User′s application data size is $D \in [0, 5]$ MB and workload is $L \in [0, 5000]$ Mega Cycles per period. The Maximum CPU frequency of all servers is $F_m = 20$ GHz, and background load $\mu_m \in [0, 16]$ GHz [23]. For wireless access points, set $G0 = -40$ db, $d0 = 1$ m, $\theta = 4$, $W = 1$ MHz, $N0 = -174$ dbm/Hz, $p = 1$ w [14].

4.2 Single Time Period and Multiple Time Period Results

In this subsection, we will compare the proposed algorithm with E^2M^2, the delay-optimal algorithm, and the energy-optimal algorithm (the last two also adopt application partitioning technique), to show that our algorithm is a long-term cost-saving strategy. For a single time period, we adopted the face recognition application with a data volume of 2 MB in [34], and divided the application into a set of 10 task with certain dependencies, as shown in Fig. 2. The data size and the CPU Cycles needed of the 10 tasks follow the gaussian distribution of $CN(\mu_1, \sigma_1^2)$ and $CN(\mu_2, \sigma_2^2)$, respectively, where constant $\mu_1 = 200$ KB, constant $\mu_2 = 200$ Mega Cycles [32], $\sigma_1 = 50, \sigma_2 = 20$. The energy weight ω^E and the time delay weight ω^T were both 0.5. The computation workloads of multi-period is processed the same as the single-period one, which divides the application into a set of multiple tasks with dependencies in Fig. 2. This paper takes fifteen time periods, namely $T = 15$, a certain time span that can reflect the performance of our algorithm in a long period of time without too much experimental calculation.

Figure 3 shows the energy consumption and computation delay of the four algorithms in a single time period. It is not difficult to see from the figure that the delay-optimal algorithm has the best performance in computation delay, but its energy consumption is much higher than other algorithms, which is about three times of the energy consumption of our algorithm. In contrast, the energy consumption of the energy-optimal algorithm is the lowest, whose computation delay is much higher and just slightly lower than E^2M^2 algorithm. The performance of our algorithm is second only to the energy-optimal algorithm in terms of energy consumption, and second only to the delay-optimal algorithm in terms of computation delay. Our algorithm receives the lowest WCED. In addition, the poor performance of E^2M^2 in a single period results from that it focus on the system performance in a long period and did not adopt partitioning method to use the parellel processing capability of multiple MEC servers.

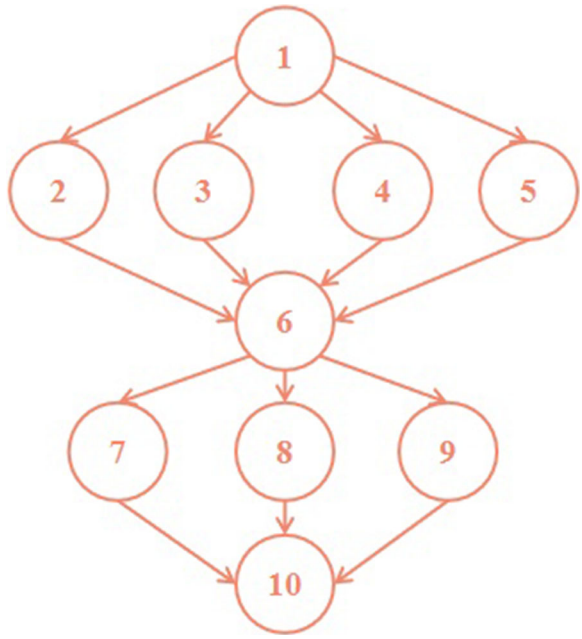


Fig. 2 An example of relationships among the subtasks

Figure 4 shows the energy consumption and computation delay of each algorithm in fifteen time periods. It can be seen that the delay-optimal algorithm still performs best in terms of computation delay and that of our algorithm is slightly higher than the delay-optimal algorithm, but much smaller than the other two. As for energy consumption, the energy-optimal algorithm is always the best. Energy consumptions of our algorithm and the E^2M^2 algorithm are approaching it sometimes but much higher than it at most time. It is not difficult to find that our algorithm owns the lowest WCED in most conditions. The computation delay of E^2M^2 algorithm is still very large in multi-periods, because its algorithm only considers to meet

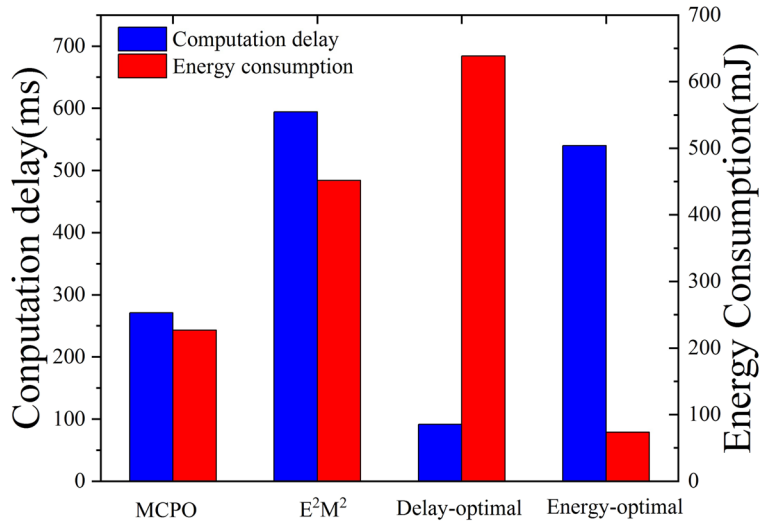
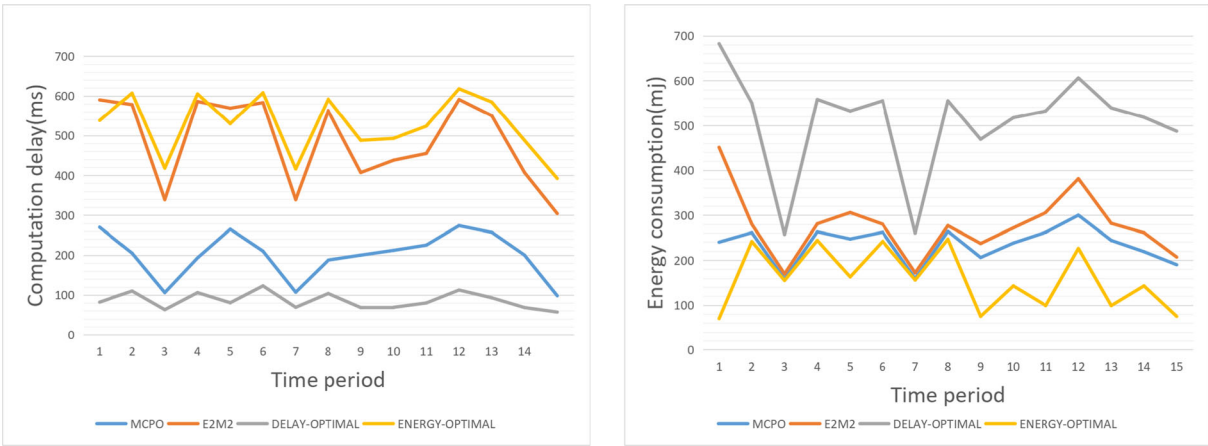


Fig. 3 Comparison of delay and energy consumption for different algorithms in a single time period

the energy consumption limit in a long time, and also because it does not divide the application.

4.3 Impact of Weights ω^E and ω^T

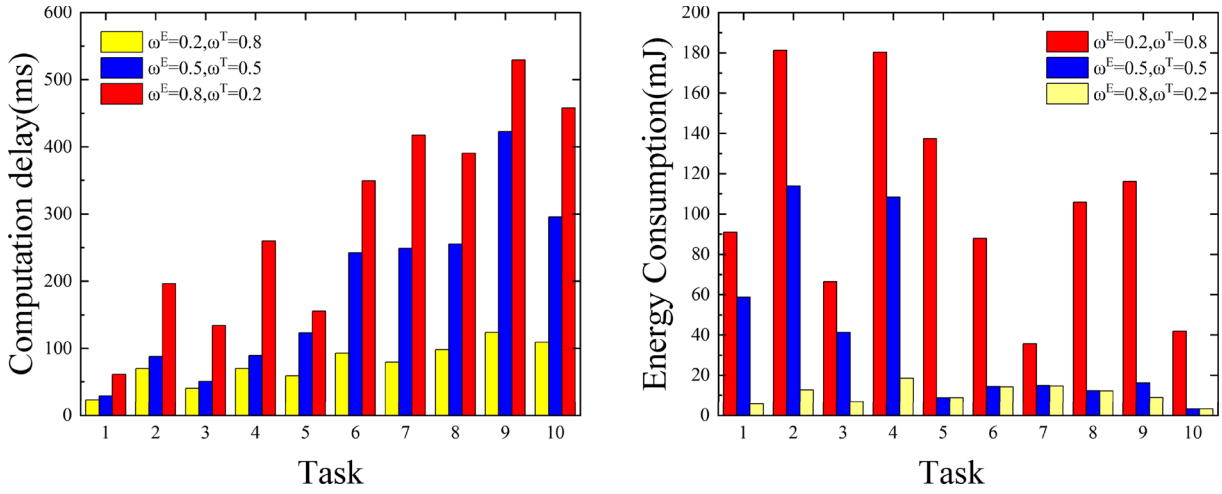
As mentioned before, the dynamic change of the weights ensure that SMDs can make more cost-saving offloading decisions which are more in line with the user′s needs and improve the user′s satisfaction. In this subsection, we will discuss the impact of ω^E and ω^T on the offloading strategy by comparing the computation delay and energy consumption at different values of ω^E and ω^T. As can be seen from Fig. 5, for certain tasks, its completion time decreases with the increase of ω^T, while its energy consumption is just the opposite. Its energy consumption went down with the increase in ω^E, and the delay was just the opposite. This is because according to (25), the increase of ω^E will lead to the decrease of the local clock frequency f, which will result in the reduction of energy consumption. A lower clock frequency means that the computation delay of local execution increases, resulting in an increase in overall application completion time. Moreover, the value of WCED is directly related



(a) Computation delay

(b) Energy Consumption

Fig. 4 Comparison of delay and energy consumption for different algorithms in four time periods



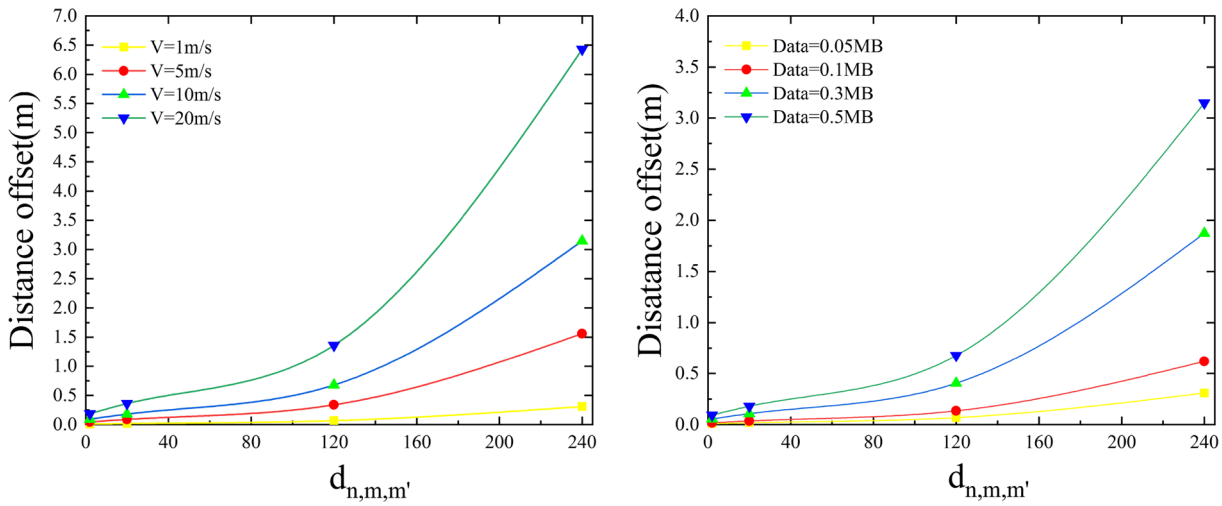
(a) Computation delay (b) Energy Consumption

Fig. 5 Comparison of delay and energy consumption under different values of weights

to ω^E and ω^T, and WCED is the basic index that determines the offloading strategy. When ω^E increases, the system will pay more attention to energy consumption cost. And for some other tasks, energy consumption and delay seems not to have changed significantly with each change of ω^E and ω^T. This is because the task scheduling decision will not change with the weights′ change if both the two consecutive offloading decision have chosen to offload tasks to edge servers. And the transmission power is a constant, when the channel state and the server side state changed little, the task processing delay and energy consumption don′t change.

4.4 The Influence of Mobility

In this subsection, we will show the relationship between data size, smart mobile devices' velocity and uploading rate to prove our mobility management is needed and helpful for making a cost-saving



(a) different speeds (b) different data size

Fig. 6 Displacements during upload time at different initial distances to MEC servers when the task is ready. **a** data size= 0.25MB, the displacement of the SMD under different speeds. **b** v=5m/s, the displacement of the SMD under different data volumes

offloading selection, based on experimental results. We can see form Fig. 6 that when SMDs' velocities or the datasizes of their subtasks are large, SMDs can move several meters away from original place (device′s distance offset) during the uploading time. They might move out of the connected server′s coverage, disconnecting with the current server, which causes service migration or remote communication to degrade offloading performance. We can also see from Fig. 6 that, device′s distance offset grows faster and faster as $d_{n,m',m}$ (distance between local SMD m' and MEC server m when task n is ready) grows when velocity and datasize are settled. It means that uploading rate is changing with the distance between local SMD and MEC server during uploading time, while this distance is changing with SMDs' movement. So we designed more complicated calculations to get an average uploading rate over uploading period rather than an uploading rate at a time, and then get a more accurate transmission delay. As you know, transmission delay is important in selecting the most cost-saving offloading scheme.

4.5 Impact of Completion Constraint on WCED

In this subsection, we will show the effect of the application completion constraint on the delay and energy consumption, and illustrate one of the considerations of our algorithm from the side. As can be seen from Fig. 7, as the completion limit gradually increases, the overall WCED of the application further decreases, and then starts to level off when the completion constraint reaches 1 second. Because devices can choose between the cloud and the local to do more cost-saving offloading options when the completion limit grows. In other words, tight completion constraint of the delay-sensitive application forces the SMDs to cost more, while our algorithm can release this problem to a certain extent.

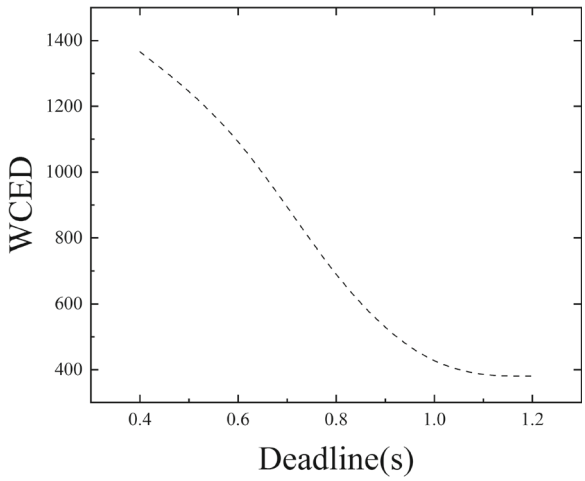


Fig. 7 Experimental result of deadline′s impact on WCED

5 Conclusion

In this paper, the problem of how to realize cost-saving computation offloading in MEC is studied when SMDs carrying DAG move across coverage areas of multiple MEC centers. We propose a novel algorithm to help moving SMDs like driverless vehicles in MEC efficiently achieve their tasks. The algorithm with mobility prediction, MCPO, is composed of three parts: local clock frequency control, accurate upload delay calculating, and computation offloading selection, to make the offloading strategy with the lowest WCED. We find that the computation offloading selection is determined not only by the computation workload of the task, but also by the maximum completion time of its predecessors, the local CPU frequency, and the mobility of the SMD. In addition, it can be observed that the optimal clock frequency for the SMD to perform the task (making the local WCED lowest) depends on the balance between completion time and energy consumption, as well as the penalty factor for the required application completion time. Finally, through the simulation of actual scenarios, the experimental results show that our MCPO algorithm can effectively reduce energy consumption, application computation delay and service migration frequency by adjusting the local CPU clock frequency and managing mobility compared with the existing offloading strategies. This paper is an online algorithm that uses the current data in reality. We will further study computation offloading mainly for driverless vehicles in MEC, offering more efficient offloading policies based on their particular features like known trajectory.

Acknowledgements This work was partially supported by the Programs of National Natural Science Foundation of China (Grant Nos. 62072165, U19A2058), Open Research Projects of Zhejiang Lab (No. 2020KE0AB01), and the Fundamental Research Funds for the Central Universities.

Data Availability The data analysed during the current study are available from the first author onup reasonable request.

References

1. Carrasco, R., Waycott, J., Baker, S., Vetere, F.: Designing the lost self: Older adults'. In: Conference on Designing Interactive Systems, pp. 441–452 (2018)
2. Al-Ars, Z., Vlugt, S., Jskelinen, P., Linden, F.: ALMARVI system solution for image and video processing in healthcare, surveillance and mobile applications. J. Signal Process. Syst. **91**(1), 1–7 (2019)
3. Dinh, T., Tang, J., La, Q., Quek, T.: Offloading in mobile edge computing: Task allocation and computational frequency scaling. IEEE Trans. Commun. **65**(8), 3571–3584 (2017)
4. Ahmed, A., Ahmed, E.: A survey on mobile edge computing. In: 10th International Conference on Intelligent Systems and Control (ISCO), pp. 1–8 (2016)
5. Wang, X., Ning, Z., Guo, S.: Multi-agent imitation learning for pervasive edge computing: A decentralized computation offloading algorithm. IEEE Trans. Parallel Distrib. Syst. **32**(2), 411–425 (2020)
6. Sun, X., Ansari, N.: PRIMAL: PRofIt maximization avatar placement for mobile edge computing. In: IEEE International Conference on Communications (ICC), pp. 1–6 (2016)
7. Jia, M., Liang, W., Xu, Z., Huang, M.: Cloudlet load balancing in wireless metropolitan area networks. In: IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, pp. 1–9 (2016)
8. Sun, X., Ansari, N.: Latency aware workload offloading in the cloudlet network. IEEE Commun. Lett. **21**(7), 1481–1484 (2017)
9. Liu, J., Mao, Y., Zhang, J., Letaief, B.K.: Delay-optimal computation task scheduling for mobile-edge computing systems. In: IEEE International Symposium on Information Theory (ISIT), pp. 1451–1455 (2016)
10. Munoz, O., Pascual-Iserte, A., Vidal, J.: Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading. IEEE Trans. Veh. Technol. **64**(10), 4738–4755 (2015)
11. You, C., Huang, K., Chae, H.: Energy efficient mobile cloud computing powered by wireless energy transfer. IEEE J. Selected Areas Commun. **34**(5), 1757–1771 (2016)
12. Sardellitti, S., Scutari, G., Barbarossa, S.: Joint optimization of radio and computational resources for multicell mobile-edge computing. IEEE Trans. Signal Inform. Process. Netw. **1**(2), 89–103 (2015)
13. Chen, X., Jiao, L., Li, W., Fu, X.: Efficient multi-user computation offloading for mobile-edge cloud computing. IEEE/ACM Trans. Network. **24**(5), 2795–2808 (2016)
14. Mao, Y., Zhang, J., Letaief, B.K.: Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems. In: IEEE Wireless Communications and Networking Conference (WCNC), pp. 1–6 (2017)
15. Li, Y., Chen, Y., Lan, T., Venkataramani, G.: MobiQoR: Pushing the envelope of mobile edge computing via quality-of-result optimization. In: IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pp. 1261–1270 (2017)
16. Yang, L., Cao, J., Cheng, H., Ji, Y.: Multi-user computation partitioning for latency sensitive mobile cloud applications. IEEE Trans. Comput. **64**(8), 2253–2266 (2015)
17. 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. Serv. Comput. **8**(2), 175–186 (2015)
18. Deng, M., Tian, H., Fan, B.: Fine-granularity based application offloading policy in cloud-enhanced small cell networks. In: Proceedings IEEE Int. Conf. Commun. Workshops, pp. 638–643 (2016)
19. Ding, Y., Liu, C., Zhou, X., Liu, Z., Tang, Z.: A code-oriented partitioning computation offloading strategy for multiple users and multiple mobile edge computing servers. IEEE Trans. Industr. Inform. **16**(7), 4800–4810 (2020)
20. Lordan, F., Badia, R.M.: COMPSs-Mobile: Parallel programming for mobile-cloud computing. In: 16th IEEE/ACM International Symposium on Cluster Cloud and Grid Computing (CCGrid) (2016)
21. Taleb, T., Ksentini, A., Frangoudis, P.A.: Follow-me cloud: When cloud services follow mobile users. IEEE Trans. Cloud Comput. **7**(2), 369–382 (2019)
22. Wang, J., Liu, K., Li, M., Pan, J.: Learning based mobility management under uncertainties for mobile edge computing. IEEE Global Communications Conference IEEE, pp. 1–6 (2018)
23. Xu, J., Sun, Y., Chen, L., Zhou, S.: E2M2: Energy efficient mobility management in dense small cells with mobile edge computing. In: IEEE International Conference on Communications (ICC), pp. 1–6 (2017)
24. Ding, Y., Liu, C., Li, K., Tang, Z., Li, K.: Task offloading and service migration strategies for user equipments with mobility consideration in mobile edge computing. In: 17th IEEE intl conf on parallel and distributed processing with applications. IEEE (2020)
25. Yang, L., Cao, J., Cheng, H., Ji, Y.: Multi-user computation partitioning for latency sensitive mobile cloud applications. IEEE Trans. Comput. **64**(8), 2253–2266 (2015)
26. Kosta, S., Aucinas, A., Hui, P., Mortier, R., Zhang, X.: ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In: Proceedings IEEE Infocom, pp. 945–953 (2012)
27. Zhou, B., Dastjerdi, V.A., Calheiros, R., Srirama, S., Buyya, R.: mCloud: A context-aware offloading framework for heterogeneous mobile cloud. IEEE Trans. Serv. Comput. **10**(5), 797–810 (2017)
28. Sklar, B.: Rayleigh fading channels in mobile digital communication systems. I. Characterization. IEEE Commun. Mag. **35**(9), 136–146 (1997)
29. Miettinen, A., Nurminen, J.: Energy efficiency of mobile clients in cloud computing. In: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud'10), pp. 1–7 (2012)
30. Liu, L., Zhang, R., Chua, K.: Wireless information and power transfer: A dynamic power splitting approach. IEEE Trans. Commun. **61**(9), 3990–4001 (2013)

31. Zhao, M., Yang, Y.: Optimization-based distributed algorithms for mobile data gathering in wireless sensor networks. IEEE Trans. Mob. Comput. **11**(10), 1464–1477 (2012)
32. Guo, S., Liu, J., Yang, Y., Xiao, B., Li, Z.: Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing. IEEE Trans. Mob. Comput. **18**(2), 319–333 (2019)
33. Chiang, M., Low, S., Calderbank, A., Doyle, J.: Layering as optimization decomposition: A mathematical theory of network architectures. Proc. IEEE. **95**(1), 255–312 (2007)
34. Praseetha, V., Vadivel, S.: Face extraction using skin color and PCA face recognition in a mobile cloudlet environment. In: 2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), pp. 41–45 (2016)