IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received August 1, 2019, accepted August 15, 2019, date of publication August 19, 2019, date of current version August 31, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2936208

Cost-Efficient Dependent Task Offloading for Multiusers

YINUO FAN[1], LINBO ZHAI[1], AND HUA WANG[2]
[1]School of Information Science and Engineering, Shandong Normal University, Jinan 250014, China
[2]School of Software, Shandong University, Jinan 250100, China

Corresponding author: Linbo Zhai (zhai@mail.sdu.edu.cn)

This work was supported in part by the Key Research and Development Program of Shandong Province, China, under Grant 2017GGX10142, and in part by the China Scholarship Fund.

ABSTRACT The extensive use of mobile intelligent devices, such as smart phones and tablets, induces new opportunity and challenge for computation offloading. Task offloading is an important issue in a system consisting of multiple types of devices, such as mobile intelligent devices, local edge hosts and a remote cloud server. In this paper, we study the offloading assignment of multiple applications, each one comprising several dependent tasks, in such a system. To evaluate the total cost in the offloading process, a new metric is introduced to take into account features of different devices. The remote server and local hosts are more concerned about their processors utilization, while mobile devices pay more attention to their energy. Therefore, this metric uses relative energy consumption to denote the cost of mobile devices, and evaluates the cost of the remote server and local hosts by the processor cycle number of task execution. We formulate the offloading problem to minimize the system cost of all applications within each application's completed time deadline. Since this problem is NP-hard, the heuristic algorithm is proposed to offload these dependent tasks. At first, our algorithm arranges all tasks from different applications in a priority queue considering both completed time deadline and task-dependency requirements. Then, based on the priority queue, all tasks are initially assigned to devices to protect mobile devices with low energy and make them survive in the assignment process as long as possible. At last, to obtain a better schedule realizing lower system cost, based on the relative remaining energy of mobile devices, we reassign tasks from high-cost devices to low-cost devices to minimize the system cost. Simulation results show that our proposed algorithm increases the successfully completed probability of whole applications and reduces the system cost effectively under time and energy constraints.

INDEX TERMS Offloading, dependent tasks, mobile, cost.

I. INTRODUCTION

In the recent few years, with the extensive use of mobile intelligent devices, such as smart phones and tablets, computationally intensive applications including image/video processing, augmented reality, face recognition and interactive gaming are becoming popular on these mobile devices. Although users enjoy the benefits of these applications, the quickly draining battery frustrates users since these complex applications consume more energy. Hence, to address this problem, computation offloading has been proposed. Computational offloading refers to application migration from a mobile device with limited computation capacity to the powerful cloud server [1]. Generally, an application consists of several tasks, and task offloading order will influence the overall energy consumption and makespan. Hence, offloading tasks rather than entire applications may be more efficient. Computation offloading is able to reduce mobile devices' energy consumption by offloading complicated tasks to the cloud. Nevertheless, offloading tasks to the remote cloud brings significant transmission delays, particularly in busy networks. Thus, local edge hosts which are closer to mobile devices, leading to low latency, may be another answer to task offloading. Besides, tasks can also be offloaded to nearby idle mobile devices [2]. Therefore, the future computation is expected to combine diverse sources of computation services in networks [3]–[7].

In existing research on offloading tasks [10], it is assumed that an application consists of multiple tasks which are

The associate editor coordinating the review of this article and approving it for publication was Sayed Chhattan Shah.

independent of each other. However, tasks in an application have internal dependent relationship. A task may have predecessor tasks which must be completed before the beginning of the task, and it may also have successor tasks which cannot be executed until the task is completed. Obviously, the precedence constraints between tasks will drastically complicate the scheduling decision. Furthermore, new challenge will be incurred since both the cost and the executing time of applications are taken into account. To facilitate tractable analysis, prior studies assume simplified processor models, such as infinite-capacity local processors [12]–[14], non-concurrent local and remote processors [11], executing time denoted by task length and negligible delay between local processors [15], [16]. Besides, the prior studies simply use the absolute value of energy and time to evaluate the offloading problem. This may not reflect the real cost of mobile devices with limited energy.

In this paper, we study the offloading assignment of multiple applications in a system, which consists of different computing devices such as mobile intelligent devices, local edge hosts and a remote cloud server. A more realistic model is designed to consider dependent tasks in an application, finite-capacity mobile devices, executing time denoted by processor cycle number and parallel execution between the local devices and the cloud. To evaluate the offloading cost, we define a new metric in the whole offloading process. The different features of all computing devices are considered in this new metric, where relative energy consumption is used to evaluate mobile devices and processor cycle number is proposed for local edge hosts and the remote cloud server. Based on the realistic model and the new metric, the offloading problem is formulated to minimize the system cost of all applications while satisfying each application's completed deadline. Since this problem is NP-hard, the heuristic algorithm is proposed to offload these dependent tasks. Considering completed time deadline and task-dependency requirements, at first, our algorithm arranges all tasks in a priority queue. Then, based on the priority queue, all tasks are sequentially assigned to computing devices and initial assignment is obtained. At last, based on the relative remaining energy of mobile devices, we reassign tasks from high-cost devices to low-cost devices to minimize the system cost. Simulation results show that our proposed algorithm reduces the system cost effectively and the successfully completed probability of all applications increases under time and energy constraints.

This paper studies the dependent task offloading problem in a system. The main contributions are summarized as follows:

- We define the system architecture consisting of multiple mobile intelligent devices, local edge hosts and a remote cloud server, and describe the data exchange among them. Moreover, a new metric, considering relative energy consumption of mobile devices and processor cycle number of local edge hosts and the remote server, is proposed to evaluate the offloading effect.
- Based on the system architecture and the metric, we formulate task offloading problem to minimize the system cost of all applications within each application's completed deadline.
- The heuristic algorithm is proposed to offload total dependent tasks. During the initial assignment, all tasks from different applications are arranged based on a priority queue. After the initial assignment, we reassign tasks based on relative remaining energy to minimize the system cost while satisfying the energy and completed time constraints.
- Simulation results show that our proposed algorithm reduces the system cost effectively and increases the successfully completed probability of all applications.

The rest of the paper is organized as follows. In Section II, a review about related works is provided. In Section III, we describe the system architecture, define a new evaluation metric and propose the system model. In Section IV, the heuristic algorithm is developed to solve the task offloading problem. In Section V, the proposed algorithm is evaluated by simulation results. Finally, conclusions are shown in Section VI.

II. RELATED WORKS

To facilitate offloading process, prior studies assume tasks of an application are independent. Nevertheless, tasks in an application have internal dependent relationship. A task may have predecessor tasks which must be finished before the beginning of the task, and it may also have successor tasks which cannot be executed until the task is completed. Considering the relationship among tasks, authors use call graph between methods of an application which implies that tasks are dependent [8]. In [9], all tasks are sequentially executed, and the output data generated by one task is the input of the next one. The internal dependent relationship among tasks will drastically complicate the scheduling decision [17], [18].

Since the offloading of dependent tasks is complex under precedence constraints, earlier works focus on a single metric, energy or makespan. In [19], authors focus on performance of the parametric partitioning and measure the energy consumption, with the assumption that the cloud and a mobile device will not run simultaneously. However, exploiting parallelism between the cloud and the mobile device can sharply improve the makespan of the application. During the wireless communications, authors propose a dynamic energy-aware model to solve energy waste problems under the dynamic networking environment [25]. In [26], authors focus on the energy-saving problem and propose a novel task assignment which reduces the energy cost to heterogeneous cores and mobile cloud. In [18], genetic algorithms are proposed for dependent tasks to reduce makespan. To solve the contradiction between optimal outputs and latency, authors propose a novel approach based on the mechanism of Reinforcement Learning to achieve optimal allocation through a self-learning process [27]. Then, several approaches using both energy and makespan as the metrics have been proposed [30]–[32].

Most recent studies have focused on offloading decisions. In [20], authors propose techniques to improve the performance for specific mobile applications. In [21], authors try to save energy of mobile devices by offloading. However, these jobs do not consider task scheduling problem of multiple applications to decrease cost. In [23], authors coordinate offloading requests of multiple applications to reduce the wireless energy cost which is caused by the tail problem. This research does not improve performance in a system comprising multiple mobile devices. Considering both energy and application completed time, authors of [22] propose new offloading algorithms. In [11] and [12], application deadline and cost minimization are jointly considered. Subject to the application deadline, authors aim to save maximized energy at a mobile device by task offloading [11]. They simply assume that tasks on cloud and the mobile device cannot run simultaneously, and formulate the offloading problem based on the assumption. In [12], authors propose a dynamic programming algorithm for deterministic and stochastic application deadlines. The aforementioned studies only consider offloading tasks from a mobile device to the remote cloud, rather than scheduling tasks in a complex system consisting of different kinds of devices, such as mobile intelligent devices, local edge hosts and a remote cloud server.

Considering multiple types of devices, the problem of offloading dependent tasks has been studied [13], [14], [16], [24]. In [13], under a resource cost constraint, authors propose a fully polynomial time approximation scheme to minimize the overall latency when they offload dependent tasks to multiple devices. It is assumed that the devices possess infinite computing capacity so that any number of tasks can be processed simultaneously, which is unrealistic. In [14], to maximize the overall useful computation, authors consider a similar offloading model for a cluster of mobile devices and design generic task scheduling heuristics. In [16], authors formulate the scheduling problem of an application comprising dependent tasks and propose a heuristic algorithm. However, only energy consumption of mobile device is considered as an objective, and it is assumed all local processors exist on a single mobile device. Subject to an application deadline, authors of [24] minimize the cost for heterogeneous local and remote processors. The cost, associated with both task execution and transmission between any two devices, is considered under the realistic assumption that the local processors possess finite capacity. However, this work is applicable to an application and multiple applications are not considered. In [11], authors divide tasks into cloud tasks and local tasks, and schedule local tasks into a randomly selected local core.

III. SYSTEM MODEL

In this section, at first, the system architecture is described. Then, a new metric is proposed to evaluate the offloading effect. After that, the offloading problem is formulated to minimize the system cost under the constraints.

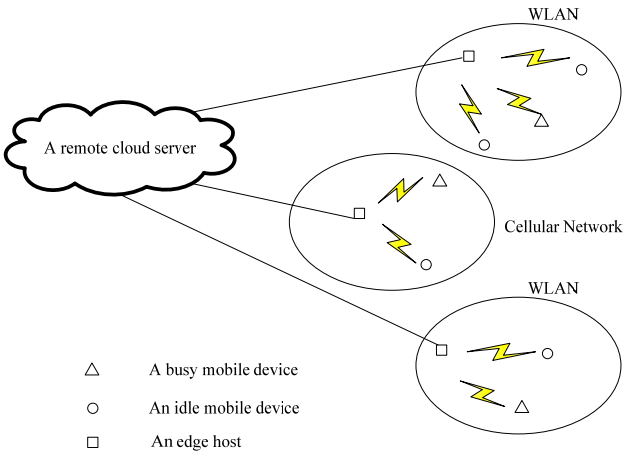


FIGURE 1. The system architecture.

A. SYSTEM ARCHITECTURE

The system is composed of multiple mobile intelligent devices, local edge hosts and a remote cloud server. There are two kinds of mobile intelligent devices, busy mobile devices launching applications and idle mobile devices without any application. Each busy device may launch several applications, and each application consisting of multiple dependent tasks has a completed time constraint. A busy mobile device with applications can compute tasks itself or offload them to idle mobile devices or local edge hosts or the remote server. Each mobile device can only execute one task at a time, while each edge host with several processors can execute more than one task and the remote server with infinite computing processors is able to execute infinite tasks at the same time. Each mobile device communicates other mobile devices or edge hosts by wireless transmission, and edge hosts have cables to communicate with the remote cloud server. Therefore, mobile devices cannot exchange data with the remote cloud server directly. Edge hosts are used as relays when the communication between a mobile device and the remote cloud server occurs. Let M denote the set of busy mobile devices launching applications, M' denote the set of idle mobile devices, Q denote the set of local edge hosts, and R denote the remote server. The system architecture is shown in Fig. 1. There are a remote cloud server, three edge hosts and several mobile devices. Edge hosts and mobile devices are connected by wireless transmission, such as WLAN and cellular network, while edge hosts have cables to communicate with the remote cloud server.

B. A NEW EVALUATION METRIC

The system consists of different kinds of devices such as mobile devices, local edge hosts and a remote cloud server. These devices focus on various aspects. The remote server and local hosts are more concerned about their processors utilization, while mobile devices pay more attention to their energy. Therefore, we design a new metric which takes the features of all devices into consideration to evaluate the offloading cost of the system.

For the remote server and local hosts, the cost is evaluated by their processor utilization. The longer the utilization time is, the higher the cost is. Previous studies assume processor utilization time is proportional to the task length, and use task length as the cost for simplicity [24]. Actually, the computation complexity depends on the task property, rather than task length. For instance, the computation complexity of a video is different from that of a text, even they have the same file length. Therefore, using task length as the cost is unrealistic. Here, the processor cycle number of task execution is used to evaluate the cost. For the remote server, we use the sum cycle number of processors during executing tasks to evaluate the cost. The cost is obtained as

$$C_R = b_1 N_R \tag{1}$$

where b_1 is a constant and N_R denotes the sum cycle number to execute the total tasks in the remote server.

For an edge host, we also use the sum cycle number of processors during executing tasks to evaluate the cost. For an edge host k, the cost is derived as

$$C_k = b_2 N_k \quad k \in Q \tag{2}$$

where b_2 is a constant and N_k is the sum cycle number to execute the total tasks in the edge host k.

For mobile devices, energy is one of the most important issues. Hence, energy consumption is introduced to denote the cost. We use the normalized value in the range (0, 1) to denote each mobile device's energy. For a mobile device, if its initial energy is less than a predefined threshold before executing a task, or the energy consumption during completing a task makes its remaining energy less than the predefined threshold after executing this task, the device will not execute this task. Otherwise, the device will execute the task. Here, we do not directly use the energy consumption as a mobile device's cost since the same energy consumption means different actual impacts on mobile devices with various initial energy. For instance, the same energy consumption value 0.1 has different impacts on two mobile devices, one device with initial energy value 0.3 and the other device with initial energy value 0.9. The device with initial energy value 0.3 is more affected because its energy falls by 33.3%, while the energy of the other device only falls by 11.1%. Therefore, the ratio between the energy consumption and the total available energy is introduced to describe the cost of a mobile device. Let Δe_b denote the energy consumption during an idle mobile device b completing tasks, e_b denote device b's initial energy, TH_b denote device b's energy threshold, e_d denote the energy consumption during a busy mobile device d completing tasks, e_d denote device d's initial energy and TH_d denote device d's energy threshold. Then, the cost of mobile device b and device d is derived as

$$\begin{cases} C_b = b_3 \dfrac{\Delta e_b}{e_b - TH_b}, b \in M' \\ C_d = b_3 \dfrac{\Delta e_d}{e_d - TH_d}, d \in M \end{cases} \tag{3}$$

where b_3 is a constant.

In (1)-(3), there are three constants (b_1, b_2 and b_3), which have dependency. If a task is executed by a mobile device, the main cost is computation, resulting in energy consumption. If the task is executed by an edge host, the task is transmitted to the edge host and then executed. Hence, we need to consider not only the transmission time but also the executing time of this task. In addition, the energy consumption of mobile devices is denoted by the relative value while the cost of the remote server and edge hosts is denoted by the absolute value. This is another issue that should be noticed when we define these three constants. Thus, the ratio between b_2 and b_3 is defined as $(T^s + T^e)/(T^e N'_k)$, where T^s denotes the transmission time of an application from a mobile device to an edge host, T^e denotes the executing time in the edge host and N'_k denotes the corresponding host cycle number to complete the task load that equals a mobile device's maximized computing ability within its energy constraint. Similarly, the ratio between b_1 and b_2 is defined as $(T^{s'} + T^{e'})/T^{e'}$, where $T^{s'}$ denotes the transmission time of the whole application from an edge host to the remote server, and $T^{e'}$ denotes the executing time in the remote server. After b_3 is given, b_1 and b_2 can be obtained.

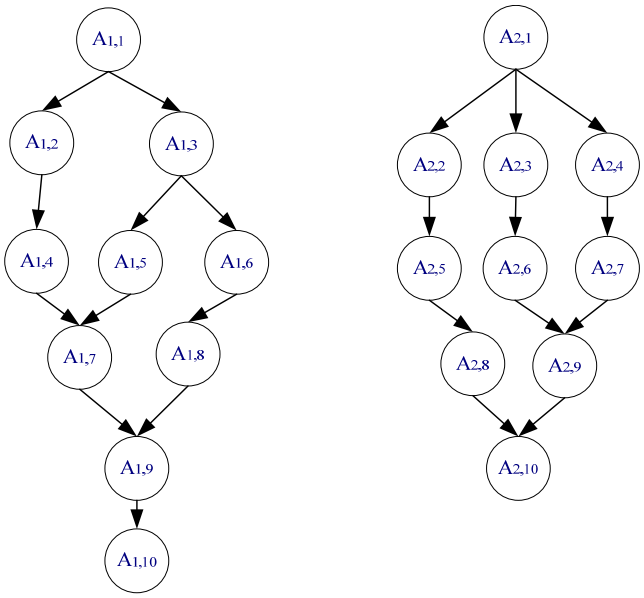
Therefore, the total cost of the system is

$$C_s = C_R + \sum_{k \in Q} C_k + \sum_{b \in M'} C_b + \sum_{d \in M} C_d \tag{4}$$

C. SYSTEM MODEL

In our system, each application, consisting of several dependent tasks, has its internal relationship. To illustrate the relationship among dependent tasks, we use a directed acyclic graph to represent the internal relationship within each application. Let A_{ji} denote task i of application j, A_{jk} denote task k of application j and the directed edge (A_{ji}, A_{jk}) denote the task-dependency relationship that A_{jk} cannot be executed before A_{ji} is completed.

As shown in Fig. 2, there are two applications (application 1 and application 2). The first application comprises location-based information finding tasks while the second application comprises content creation tasks, similar to the application types in [33]. Each of them consists of ten dependent tasks and their connections are different. In application 1, there is task-dependency relationship among ten tasks from $A_{1,1}$ to $A_{1,10}$. Task $A_{1,2}$ and $A_{1,3}$ cannot be executed before task $A_{1,1}$ is completed. Similarly, only when both task $A_{1,4}$ and $A_{1,5}$ are completed, task $A_{1,7}$ will be executed. After task $A_{1,7}$ and $A_{1,8}$ are finished, task $A_{1,9}$ will be executed. Then, task $A_{1,10}$ will not be processed until task $A_{1,9}$ is completed. Application 2 also has its task-dependency relationship, which is different from application 1. The beginning time of task $A_{2,2}$, $A_{2,3}$ and $A_{2,4}$ is determined by $A_{2,1}$. The later completed time of $A_{2,6}$ and $A_{2,7}$ indicates the beginning time of task $A_{2,9}$. Additionally, we should notice that tasks of different applications are independent of others and they can be executed in parallel.



(a) Application 1 (b) Application 2

FIGURE 2. Task-dependency relationship.

When a busy mobile device d has some applications, each one consisting of several tasks, it can executes these tasks itself or offload them to others. There are four cases below.

i) If a busy mobile device d executes task A_{ji} of application j itself, there is only executing time for this task. The completed time T_{jid} equals the executing time. Let N_{jid} denote the processor cycle number to execute task A_{ji} in mobile device d, and Td denote the length of each processor cycle. We can obtain

$$T_{jid} = N_{jid} T_d \tag{5}$$

The corresponding energy consumption of device d is $\Delta e_{jid} = TjidPd$, where P_d denotes the operating power of device d. The operating power is the same with [24].

ii) If the mobile device d offloads its task A_{ji} to an idle mobile device b, it should send task A_{ji} to the device b. Then, task A_{ji} is executed in device b. After that, device d receives the results from device b. Therefore, the completed time consists of not only executing time in the device b but also the duration of sending task A_{ji} and receiving results. Let l_{ji} denote the length of task A_{ji}, res_{ji} denote task A_{ji}'s results, and v denote the wireless transmission rate. The duration of sending task A_{ji} to device b is $T^s_{jib} = l_{ji}/v$, and duration of receiving task A_{ji}'s results from device b is $T^r_{jib} = res_{ji}/v$. Let N_{jib} denote the processor cycle number to execute task A_{ji} in mobile device b, and T_b denote the length of each processor cycle. The executing time is $N_{jib}T_b$. Thus, the completed time T_{jib} is derived as

$$T_{jib} = T^s_{jib} + N_{jib}T_b + T^r_{jib} \tag{6}$$

In this process, both device d and device b consume their energy. For device d, there are sending energy consumption and receiving energy consumption. Let P_{sd} denote device d's sending power and P_{rd} denote the receiving power. Thus, device d's energy consumption is $\Delta e_{jid} = P_{sd}T^s_{jib} + P_{rd}T^r_{jib}$. For device b, there are receiving energy consumption, computing energy consumption, and sending energy consumption. Let P_{sb} denote device b's sending power, P_{rb} denote the receiving power, and P_b denote the operating power of device b. Thus, device b's energy consumption is $\Delta e_{jib} = P_{rb}T^s_{jib} + P_bN_{jib}T_b + P_{sb}T^r_{jib}$.

iii) If the mobile device d offloads its task A_{ji} to a local edge host k, it should send task A_{ji} to edge host k. Then, task A_{ji} is executed in edge host k. After that, device d receives the results from edge host k. Therefore, the completed time is composed of executing time in edge host k and the duration of sending task A_{ji} and receiving results. The duration of sending task A_{ji} to device k is $T^s_{jik} = l_{ji}/v$, and duration of receiving task A_{ji}'s results from device k is $T^r_{jik} = res_{ji}/v$ where res_{ji} denotes task A_{ji}'s results. Let N_{jik} denote the processor cycle number to execute task A_{ji} in a local host k, and T_k denote the length of each processor cycle. The executing time is $N_{jik}T_k$. Thus, the completed time T_{jik} is derived as

$$T_{jik} = T^s_{jik} + N_{jik}T_k + T^r_{jik} \tag{7}$$

In this process, it is not necessary to consider energy consumption of device k because it has electricity supply. For device d, there are sending energy consumption and receiving energy consumption. Thus, device d's energy consumption is $\Delta e_{jid} = P_{sd}T^s_{jik} + P_{rd}T^r_{jik}$.

iiii) If the mobile device d offloads its task A_{ji} to the remote server R, it should send task A_{ji} to server R. That means it needs to send task A_{ji} to a local edge host g at first, then the local edge host g relays task A_{ji} to server R. After that, task A_{ji} is executed in server R. When task A_{ji} is completed, server R transmits its results to local edge host g and this edge host relays the results to device d. Therefore, there are the exchange time between d and g, executing time in remote server R, and exchange time between R and g. The time that device d sends task A_{ji} to edge host g is $T^s_{jig} = l_{ji}/v$, and the time that device d receives task A_{ji}'s results from edge host g is $T^r_{jig} = res_{ji}/v$. The exchange time between R and g is determined by how busy the backbone network is. Let T^s_{jiR} denote the task sending time from g to R, and T^r_{jiR} denote the result sending time from R to g. Let N_{jiR} denote the processor cycle number to execute task A_{ji} in the remote server R, and T_R denote the length of each processor cycle. The executing time is $N_{jiR}T_R$. Thus, the completed time T_{jiR} is derived as

$$T_{jiR} = T^s_{jig} + T^s_{jiR} + N_{jiR}T_R + T^r_{jiR} + T^r_{jig} \tag{8}$$

In this process, it is not necessary to consider energy consumption in g and R because they have electricity supply. For device d, there are sending energy consumption and receiving energy consumption. Thus, device d's energy consumption is $\Delta e_{jid} = P_{sd}T^s_{jig} + P_{rd}T^r_{jig}$.

Considering task-dependency relationship in application j, task A_{ji} cannot be assigned to a device x until the total predecessors of task A_{ji} have been completed. When task A_{ji}

is able to be assigned to device x, if device x is the remote cloud server R, task A_{ji} will be executed immediately since the remote server has infinite computing processors to support multitasking. If device x is a local edge host, which is able to execute several tasks at a time, task A_{ji} has to wait for the availability of a processor when all device x's processors are executing tasks. If device x is a mobile device, only executing one task at a time, task A_{ji} has to wait for the availability of device x when there is another task being executed in device x. Let TS_{jix} denote task A_{ji}'s started moment for device x, P denote the task set of all task A_{ji}'s predecessors, TC_{jp} denote the completed moment of task $A_{jp} \in P$, and TA_{jix} denote the available moment of task A_{ji} in device x. Then, we can obtain

$$TS_{jix} = \begin{cases} \max\{\max\limits_{A_{jp}\in P} TC_{jp}, TA_{jix}\}, & x \neq R \\ \max\limits_{A_{jp}\in P} TC_{jp}, & x = R \end{cases} \tag{9}$$

Based on (5)-(8), the completed time T_{jix} of task A_{ji} in device x can be obtained. Thus, task A_{ji}'s completed moment TC_{jix} for device x is derived as

$$TC_{jix} = TS_{jix} + T_{jix} \tag{10}$$

When all tasks of application j have been assigned to devices, the completed moment of application j equals the completed moment of its latest task. Let TC_{ji} denote the completed moment of task A_{ji}. If task A_{ji} is assigned to device x, TC_{ji} equals TC_{jix}. Thus, the completed moment TC_j of application j is obtained as

$$TC_j = \max_{A_{ji}} TC_{ji} \quad \text{task } Aji \in \text{ application } j \tag{11}$$

Let U_{jix} denote whether task A_{ji} is assigned to device x. We can obtain

$$U_{jix} = \begin{cases} 1, & \text{if } A_{ji} \text{ is assigned to } x \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

Since a task will be assigned to a device, we can obtain $\sum_x U_{jix} = 1$. Let V_{im} denote the executed order between task task A_{ji} and task A_{qm}. V_{im} is defined as

$$V_{im} = \begin{cases} 1, & \text{if task } A_{qm} \text{ executed before } A_{ji} \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

If task A_{ji} is assigned to mobile device x, the available moment TA_{jix} in device x is not earlier than the completed moment of task A_{qm} that device x has before task A_{ji}. This constraint is obtained as

$$TA_{jix} \geq U_{jix} U_{qmx} V_{im} TC_{qm} \; x \in M' \cup M \; for \; \forall A_{ji}, A_{qm} \tag{14}$$

Based on the description from *i*) to *iiii*), the total energy consumption Δe_d of mobile device $d \in M$ is derived as

$$\Delta e_d = \begin{cases} \sum\limits_j \sum\limits_i U_{jid} N_{jid} T_d P_d \\ \sum\limits_j \sum\limits_i U_{jib}(P_{sd} T^s_{jib} + P_{rd} T^r_{jib}), & b \in M' \\ \sum\limits_j \sum\limits_i U_{jik}(P_{sd} T^s_{jik} + P_{rd} T^r_{jik}), & k \in Q \\ \sum\limits_j \sum\limits_i U_{jiR}(P_{sd} T^s_{jig} + P_{rd} T^r_{jig}), & g \in Q \end{cases} \tag{15}$$

The total energy consumption Δe_b of mobile device $b \in M'$ is obtained as

$$\Delta e_b = \sum_j \sum_i U_{jib}(P_{rb} T^s_{jib} + P_b N_{jib} T_b + P_{sb} T^r_{jib}) \tag{16}$$

The total cycle number N_R for the remote server and the cycle number N_k for local edge host $k \in Q$ are derived as

$$\begin{cases} N_R = \sum\limits_j \sum\limits_i U_{jiR} N_{jiR} \\ N_k = \sum\limits_j \sum\limits_i U_{jik} N_{jik}, \quad k \in Q \end{cases} \tag{17}$$

Each application consisting of several tasks has completed time constraint. That means the completed moment of its latest task cannot exceed this constraint. Let Con_j denote the completed time constraint of application j. Based on (11), the completed moment TC_j of application j should be no more than Con_j.

Based on (4), (15), (16) and (17), the offloading problem can be formulated as

$$\begin{aligned} &\min b_1 N_R + \sum_{k\in Q} b_2 N_k + \sum_{b\in M'} \frac{b_3 \Delta e_b}{e_b - TH_b} + \sum_{d\in M} \frac{b_3 \Delta e_d}{e_d - TH_d} \\ &s.t. \; TC_j \leq Con_j \\ &\quad e_b - \Delta e_b \geq TH_b, \quad b \in M' \\ &\quad e_d - \Delta e_d \geq TH_d, \quad d \in M \\ &\quad Eq.(9) - (14) \end{aligned} \tag{18}$$

The problem in (18) is NP-hard because Generalized Assignment Problem (GAP), which is NP-hard, is a special case of this problem. Hence, it is difficult to find an optimal solution within polynomial time. Consequently, we propose the heuristic algorithm to solve it in next section.

IV. OFFLOADING ALGORITHM

In this section, the heuristic offloading algorithm is proposed to minimize the system cost under the constraints.

A. OFFLOADING ALGORITHM

During the assignment process, we try to assign task A_{ji} to a device x^* to realize the lowest system cost. The assignment process consists of priority arrangement, initial assignment and reassignment. In the initial assignment, we generate a schedule to obtain the system cost as low as possible under the

energy and completed time constraints. Then, we introduce relative remaining energy to reassign tasks based on the initial schedule to minimize the system cost.

At the first step, all tasks are arranged in a priority queue based on their assignment urgency.

Let Δe_{jix} denote mobile device x's energy consumption for task A_{ji}, and TH_x denote its energy threshold. To maintain the remaining energy of mobile device x higher than TH_x after executing task A_{ji}, mobile device x should have enough energy no less than the real threshold THR_{jix} before executing task A_{ji}. Thus, real threshold THR_{jix} is derived as

$$THR_{jix} = TH_x + \Delta e_{jix} \tag{19}$$

Let LC_{ji} denote the latest allowable completed moment of task A_{ji}, S denote the task set of A_{ji}'s successors,LC_{js} denote the latest allowable completed moment of task $A_{js} \in S$, X_{js} denote the set of mobile devices satisfying real threshold before executing task A_{js}, and T_{jsx} denote the completed time of task A_{js} in device x. Then, we can obtain

$$LC_{ji} = \min_{A_{js} \in S}(LC_{js} - \min_{x \in X_{js}} T_{jsx}) \tag{20}$$

For the last task of application j, the latest allowable completed moment of this task is *Conj*, which denotes the completed time constraint for application j. Otherwise, the time constraint will not be satisfied. Based on the latest allowable completed moment *Conj* of the last task in application j, each predecessor's latest allowable completed moment can be calculated following (20) until the first task in application j. To avoid the application j breaking the completed time constraint, any task A_{ji} of application j must be completed before LC_{ji}.

Let X_{ji} denote the set of mobile devices satisfying real threshold before executing task A_{ji}. Using (20), the latest allowable beginning moment LB_{ji} of task A_{ji} can be obtained as

$$LB_{ji} = LC_{ji} - \min_{x \in X_{ji}} T_{jix} \tag{21}$$

A task with a smaller value of the latest allowable beginning moment is more urgent and should be assigned earlier than others. Otherwise, the time constraint will be broken. Therefore, the smaller LB_{ji} is, the higher priority of task A_{ji} is. The latest allowable beginning moment of each task can be calculated following (19)-(21). Based on the latest allowable beginning moment of each task, the priority queue of all tasks is obtained.

At the second step, all tasks are initially assigned following the priority queue. When there is only one application, all tasks of this application are simply assigned in the order of the priority queue. When there are multiple independent applications, each consisting of several tasks, tasks of these applications can be executed in parallel. This leads to such a case. At a time point, the predecessors of a task with lower priority in one application are completed, while the predecessors of another task with higher priority in other application are not completed. In this case, the task with lower priority should be assigned first to improve efficiency. The details of initial assignment are described as follows.

Task A_{ji} with the highest priority is chosen from all unassigned tasks. If all A_{ji}'s predecessors have not been completed, this task cannot be assigned and it is deposited in a backup queue. If all A_{ji}'s predecessors have been completed, the remote server, local edge hosts and mobile devices with current energy higher than the real energy threshold in (19) are potential devices for task A_{ji}'s assignment. We calculate the completed moment and system cost of task A_{ji} in each potential device based on (4)-(17). From the devices which are able to complete task A_{ji} before its latest allowable completed moment LC_{ji}, we can select a suitable device to assign task A_{ji}. If it is the remote server or a local edge host which achieves the lowest system cost for task A_{ji}, task A_{ji} will be assigned to the device realizing the lowest system cost. If some mobile devices can achieve lower system cost than the remote server and local edge hosts, one of these mobile devices will be chosen. Task A_{ji} will be assigned to a mobile device, which has highest remaining energy after executing task A_{ji}, rather than the mobile device achieving the lowest system cost. The aim is to protect low energy mobile devices and make them survive during the assignment process as long as possible. We should notice that A_{ji}'s assignment influences the tasks in the backup queue and may lead to these tasks exceeding their latest allowable completed moment, which will break the time constraint of the corresponding applications. Therefore, we do not immediately assign A_{ji} to a device even if its predecessors have been completed. Instead, we calculate the completed moment of each task in the backup queue as if A_{ji} is actually assigned. If a task A_{nk} in the backup queue cannot be completed no later than its latest allowable completed momentLC_{nk}, task A_{ji} will not be assigned, and it is deposited into the backup queue. If each task in the backup queue can be completed no later than LC_{nk}, task A_{ji} will be actually assigned to the chosen device. After the actual assignment of a task, we will renew the current energy of mobile devices and assign the next task until the whole tasks are assigned.

At the last step, tasks assigned during the initial assignment are reassigned to decrease system cost. We try to move tasks from high-cost devices to low-cost devices under the completed time constraint. When we move a task from one device to another, there may be several choices leading to different reassignment results, and only the reassignment realizing lowest cost is chosen.

To choose the proper task reassignment, we introduce the relative remaining energy to evaluate the reassignment opportunity of tasks in a mobile device. Let RE_k denote the remaining energy of mobile device k after the initial assignment. For mobile device k, relative remaining energy RRE_k is defined as

$$RRE_k = RE_k - TH_k \tag{22}$$

where TH_k denotes device k's energy threshold, RE_k and TH_k are the normalized values in the range (0, 1).

Relative remaining energy RRE_k evaluates the relative distance between the actual remaining energy of mobile device k and its energy threshold. Thus, based on (3), the lower RRE_k is, the higher the mobile device k's cost is in initial assignment.

To move tasks from high-cost devices to low-cost devices, after the initial assignment, we choose a mobile device with the lowest relative remaining energy, and reassign the tasks in this mobile device. It is assumed m tasks are assigned to this mobile device in the initial assignment and there are n devices in the system. Each task can be moved to other $n-1$ devices leading to $m^*(n-1)$ reassignment cases. Some reassignment cases may break the time or energy constraints and they should be removed. For the reassignment cases satisfying constraints, their system cost values can be calculated based on (4), (15), (16) and (17). The reassignment case realizing the lowest system cost is chosen.

We use the current chosen reassignment to update global task distribution, leading to the change of mobile devices' remaining energy. Then, we repeat the reassignment process until no lower system cost is found.

The details of the heuristic multi-user reassignment offloading algorithm (MRO) are described in Alg. 1.

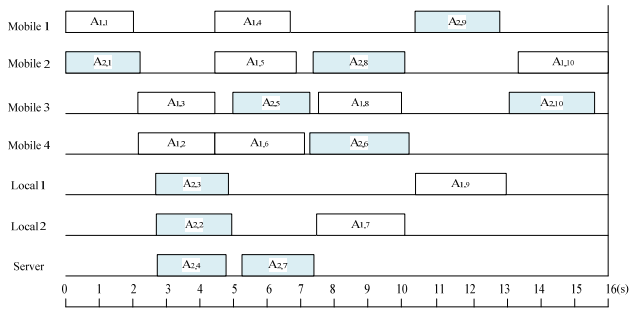
For example, we perform the proposed algorithm on two applications with the task graphs shown in Fig. 2. In the system, there are four mobile devices, two local hosts and a remote cloud server. For four mobile devices, we set the initial energy as 1.0, 0.92, 0.88 and 0.96. For simplicity, these mobile devices have the same energy threshold which equals 0.1. For tasks from $A_{1,1}$ to $A_{1,10}$, we set the task executing time as 2s, 2.2s, 2.2s, 2.4s, 2.6s, 2.8s, 2.6s, 2.4s, 2.6s, 2.7s when they are executed by mobile device 1. For tasks from $A_{2,1}$ to $A_{2,10}$, we set the task executing time as 2s, 2.2s, 2.2s, 2.4s, 2.6s, 2.8s, 2.6s, 2.4s, 2.6s, 2.7s when they are executed by mobile device 1. We use the executing time of mobile device 1 as the baseline. For the same task, the executing time of mobile device 2 is 1.1 times as long as that of mobile device 1. The executing time of mobile device 3 is 1.2 times and the executing time of mobile device 4 is 1.1 times. Two local hosts possess the same computing capacity and the executing time is 0.9 times that of mobile device 1. The executing time of remote server is 0.75 times. The transmission time is set to one third of the executing time of mobile device 1 for simplicity. We set $b_1 = 0.2$, $b_2 = 0.4$ and $b_3 = 0.3$. The completed time constraints of the two applications are 17.5s and 17s, respectively.

Based on the latest allowable beginning moment of all tasks, the priority queue is obtained as ($A_{2,1}$, $A_{1,1}$, $A_{1,2}$, $A_{1,3}$, $A_{2,2}$, $A_{2,3}$, $A_{2,4}$, $A_{1,4}$, $A_{1,5}$, $A_{1,6}$, $A_{2,5}$, $A_{2,7}$, $A_{2,6}$, $A_{2,8}$, $A_{1,7}$, $A_{1,8}$, $A_{1,9}$, $A_{2,9}$, $A_{2,10}$, $A_{1,10}$). During the initial assignment, these tasks are assigned based on the priority queue. Fig. 3(a) presents the results of initial assignment. At first, task $A_{2,1}$, which has the highest priority, is assigned to mobile device 1. Then, task $A_{1,1}$ is assigned to mobile device 4. After that, task $A_{1,2}$ and $A_{1,3}$ cannot be assigned until $A_{1,1}$ is completed. Similarly, task $A_{2,2}$, $A_{2,3}$ and $A_{2,4}$ are waiting for task $A_{2,1}$'s completion. Within one application, a task can be assigned after all its predecessors are finished. As shown in Fig. 3(a), tasks are assigned to mobile devices, local hosts and the

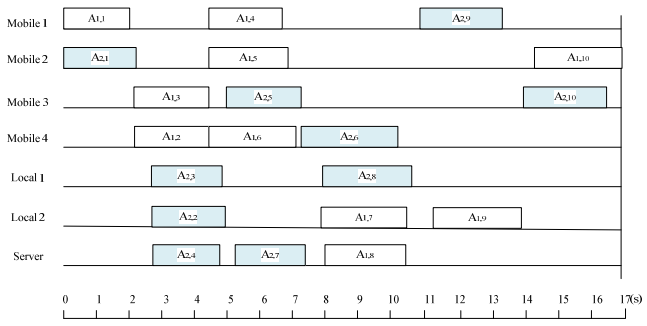
Algorithm 1 Heuristic Multi-User Reassignment Offloading

Input: Task relationship in each application Energy and time constraints
Output: The minimized cost z and task assignment $RA(z)$

priority queue PQ $\leftarrow \varphi$
backup queue $BQ \leftarrow \varphi$
for each application j
  for each task $A_{ji} \in$ application j
    $THR_{jix} \leftarrow$ Eq. (19)
    obtain X_{js} satisfying real threshold
    $LC_{ji} \leftarrow$ Eq. (20)
    $LB_{ji} \leftarrow$ Eq. (21)
    add A_{ji} to PQ based on LB_{ji}
  end for
end for
while $PQ \cup BQ \neq \varphi$
 select task A_{qm} with smallest LB_{qm} from $PQ \cup BQ$
  if A_{qm}'s predecessors are not completed
   move A_{qm} to BQ
  else
   calculate potential assignment device set S
   compute the cost of each device in S
   choose $x \in S$ realizing lowest cost
   choose $y \in Q \cup R$ realizing lowest cost
   If $x \neq y$
    obtain mobile set S' achieving lower cost than y
    $x \leftarrow$ a device with highest remaining energy in S'
   end if
   assign A_{qm} to x virtually
   If any task in BQ does not break constraints
    assign A_{qm} to x really
    If $A_{qm} \in PQ$
     $PQ \leftarrow PQ / Aqm$
    else
     $BQ \leftarrow BQ/ A_{qm}$
    end if
   else
    move A_{qm} to BQ
  end if
 end if
end while
select a mobile device k with the lowestRRE_k
for each task in k
  for each other device
    compute reassignment cost
    If this reassignment satisfying constraints
    $z \leftarrow$ minimized reassignment cost
    $RA(z) \leftarrow$ reassignment with minimized cost
    end if
  end for
end for
update mobile devices' tasks and remaining energy
repeat 38-48 until no lower cost
Return z and $RA(z)$



(a) Initial assignment



(b) Reassignment

FIGURE 3. An example of task assignment.

remote server. Based on our algorithm, the corresponding system cost is 17.39.

After the initial assignment, our algorithm will reassign tasks to decrease the system cost. Fig. 3(b) shows the results of task reassignment. Since mobile device 3 executes many tasks, leading to drastic energy decline approaching its energy threshold, task $A_{1,8}$ in mobile device 3 is reassigned to the remote server to reduce cost. Similarly, task $A_{2,8}$, located in mobile device 2 in the initial assignment, is reassigned to local host 1 because this reassignment is able to decrease system cost. Due to task $A_{2,8}$'s reassignment, task $A_{1,9}$ moves from local host 1 to local host 2 to guarantee the executing efficiency. After the reassignment, the system cost drops to 15.37 and the completed time of both applications increases slightly. This indicates task reassignment can efficiently bring down the system cost and result in the growth of completed time, which is still satisfying the time constraint. In addition, reassignment also influences the sequence of task execution. Task $A_{2,9}$ and $A_{1,9}$ are from different applications and $A_{1,9}$ has higher priority. After task reassignment, task $A_{2,9}$ is assigned before $A_{1,9}$. Although the task with lower priority is scheduled first, there is no confusion on offloading because the tasks from different applications are independent.

In conclusion, our proposed algorithm is applied to not only single application but also multiple applications. If there is only one application, all tasks of this application will be sequentially assigned based on their latest allowable beginning moments. If there are multiple applications, it is complicated. To improve the executing efficiency, tasks cannot be sequentially assigned following the latest allowable beginning moments. Each application is independent on other applications, resulting in each task of one application is independent on other tasks of other applications. Hence, as shown in the example, task $A_{2,9}$ of application 2 has lower priority than task $A_{1,9}$ of application 1 while there is no relationship between these two tasks. At a time point, the predecessors of task $A_{2,9}$ with lower priority are completed, while the predecessors of task $A_{1,9}$ with higher priority are not completed. In this case, task $A_{2,9}$ does not wait for task $A_{1,9}$. Instead, it is assigned before $A_{1,9}$ to improve efficiency. Although the task with lower priority is scheduled first, there is no confusion on offloading because the tasks from different applications are independent.

B. ANALYSIS OF COMPLEXITY

The complexity of proposed MRO algorithm is computed as follows. The MRO consists of three steps. In the first step, the computation complexity is $O(jin)$ where j denotes the number of applications, i denotes the number of tasks in each application and n denotes the number of mobile devices.

In the second step, we do not immediately assign task A_{ji} to a suitable device even if its predecessors have been completed since A_{ji}'s assignment influences the tasks in the backup queue and may lead to these tasks exceeding their latest allowable completed moment. Instead, we calculate the completed moment of each task in the backup queue as if A_{ji} is actually assigned. This process dominates the complexity of the second step. Hence, the computation complexity is $O(jin^2)$.

At last, tasks are reassigned to decrease system cost. When we move a task from one device to another, it results in a new assignment. Hence, the computation complexity is $O(ji(n-1)n^2)$.

Therefore, the computation complexity of the whole algorithm is $O((n-1)n^2ji)$.

V. SIMULATIONS

In this section, our proposed algorithm (MRO) is evaluated by extensive simulations. We use the simulator proposed in [34] to model tasks. Compared with ITAGS algorithm [24], MAUI+ECS algorithm [29] and MAUI+Random algorithm [11], our algorithm can reduce the system cost efficiently. The system has four mobile devices, two local edge hosts and a remote cloud server. Each local host can execute three tasks in parallel while a remote server is able to execute infinite tasks in parallel. For four mobile devices, we set the initial energy as 1.0, 0.92, 0.88 and 0.96. The energy threshold is 0.1 for all mobile devices. They possess the same computing capacity and their parameters are the same with Nexus 5X smartphone. Two local hosts

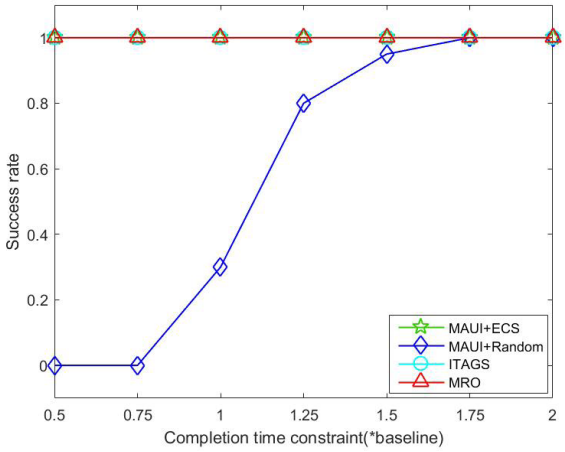
possess the same computing capacity, and the executing time of a same task is 0.9 times as long as that of a mobile device. The remote server has the highest computing capacity and the executing time is 0.75 times for the same task. We set $b_1 = 0.2$, $b_2 = 0.4$ and $b_3 = 0.3$.

The size of task load and that of task results follows Gaussian distribution N(200KB, 2500) and N(20KB, 25), respectively. The transmission rate is 1Mbps. We understand that transmission rate of wireless communication is up to 54Mbps in Wi-Fi, while the transmission rate of mobile communication network is lower than that of Wi-Fi and it is not stable, influenced by shadowing, multiple fading and other issues. Hence, we use 1 Mbps as transmission rate between mobile devices and local edge hosts. This is the general assumption which is more realistic. The processor cycle number of executing tasks also follows Gaussian distribution N(2000M, 200). Here, we use TGFF to generate the task relationship in each application [28]. The task relationship of an application determines completed time constraint, and the default value of time constraint equals the total executing time of an application in a mobile device. Here, each application consists of the content creation tasks, and all parameters are the same with [33].

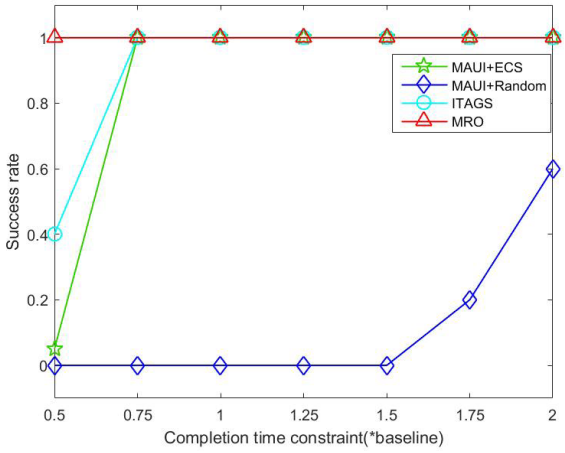
We use two metrics, success rate and system cost, to evaluate MRO and other algorithms. Success rate is defined as the percentage that the whole applications are completed within their completion time constraints. System cost is defined as in Equation (4).

A. SUCCESS RATE

Fig. 4 shows the success rate of different algorithms under different completion time constraints. The default baseline is set as the total executing time of an application in a mobile device. The completion time constraint is represented by multiplying a factor (i.e., [0.5, 2.0]) to the baseline. Each application consists of 20 tasks. As shown in Fig. 4(a), with only one application, the success rate reaches 100% for ITAGS, MAUI+ECS and MRO, while MAUI+Random cannot reach. This is because MAUI+Random randomly assigns tasks to mobile devices without considering completion time constraints, which results in timeout and the lower success rate. ITAGS, MAUI+ECS and MRO consider the completion time constraint during assigning tasks, leading to their higher success rate. As the completion time constraint increases, it is easier to assign total tasks within completion time constraint. Therefore, the success rate of MAUI+Random also increases to 100% in the final. When the number of applications increases, two applications in Fig. 4(b), several tasks in different applications can be launched at the same time. Since the number of devices does not change, tasks need to compete the devices to satisfy their time constraints, leading to lower success rate for MAUI+ECS, ITAGS and MAUI+Random. However, our algorithm (MRO) takes the time constraint into consideration during both task initial assignment and task reassignment. Thus, our algorithm always has the higher success rate.



(a) Number of applications = 1



(b) Number of applications = 2

FIGURE 4. Success rate with different number of applications.

B. SYSTEM COST

For one application with fixed number of tasks, different task connections will result in various system cost even if each task does not change. Fig. 5 shows six kinds of task connections for an application consisting of ten tasks. As shown in Fig. 5, each task does not change while the task relationship changes. Fig. 6 illustrates the change of system cost corresponding to different task connections. MRO realizes lower system cost than other algorithms. As the task connection changes, the system cost also changes since task-dependency relationship influences the executed sequence of tasks. For the sixth connection, less tasks can be executed in parallel and most of them must wait for the completion of predecessors. Hence, the executing efficiency is lower than other connections, which results in a longer completed time. To satisfy the time constraint, some tasks have to be assigned to high-cost devices.

Fig.7 shows two applications (app 1 and app 2) consisting of 20 tasks, and their task connections. There are only different task connections between these two applications. As the

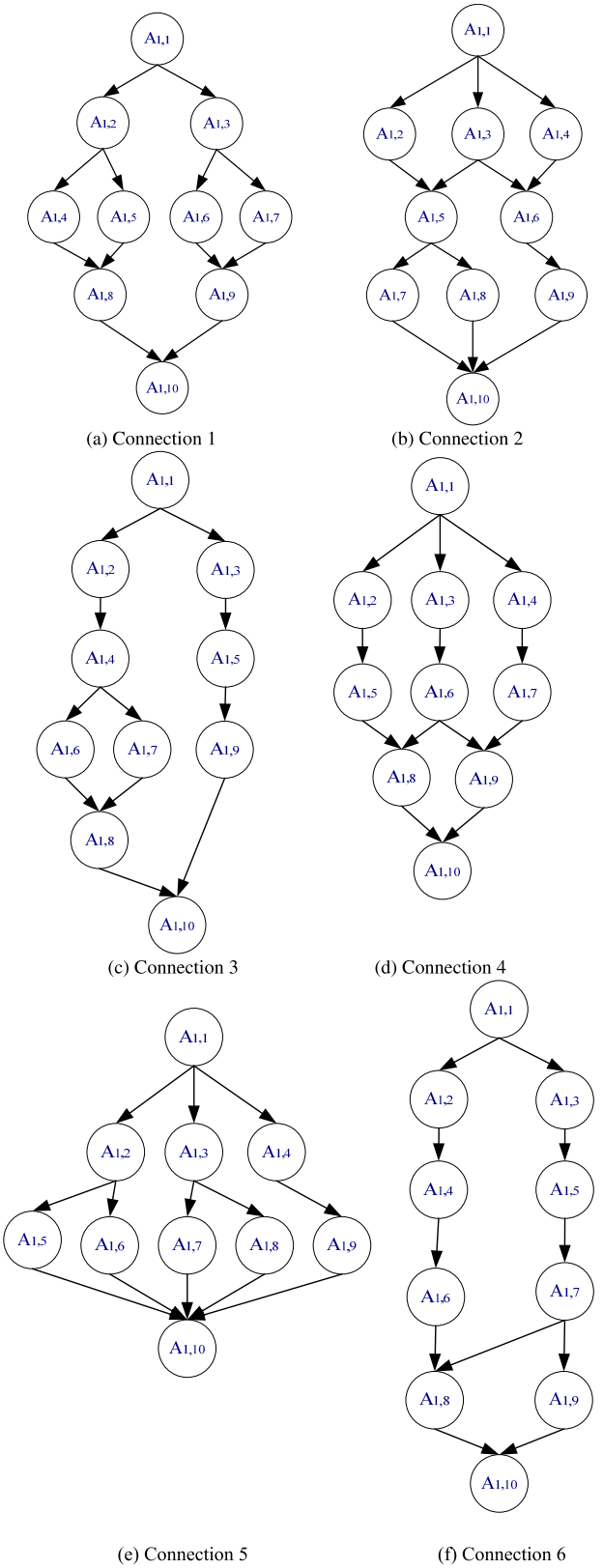


FIGURE 5. Six kinds of task connections.

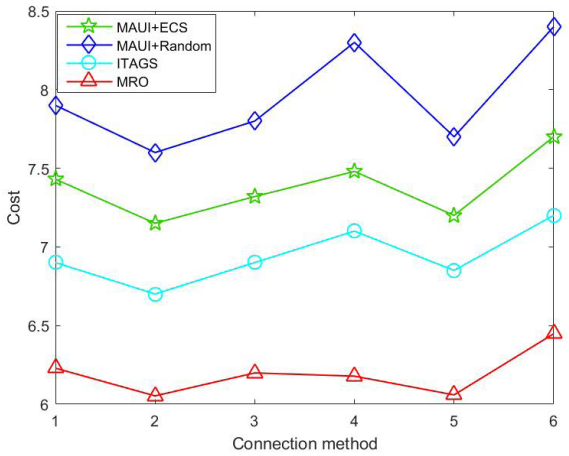


FIGURE 6. System cost with different connections.

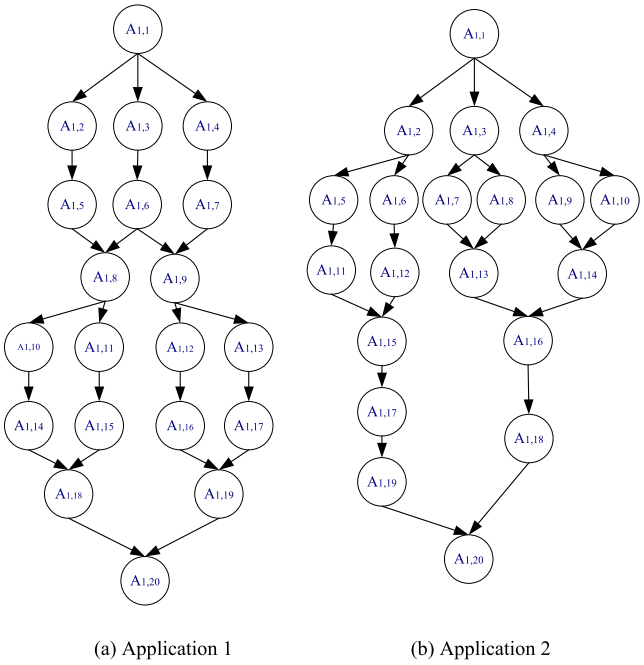


FIGURE 7. Two kinds of task connections.

number of mobile devices changes from 4 to 8, Fig. 8 illustrates the system cost for app 1 and app 2, respectively. As shown in Fig. 8, the system cost gradually decreases when the number of mobile devices increases. As the number of mobile devices increases, tasks have more choices and can be assigned to those low-cost devices which decrease the total system cost. In addition, the cost of app 2 is higher than app 1 no matter which algorithm is used. The reason is that the relationship of app 2 results in less tasks that can be executed in parallel. Hence, the executing time is longer than that of app 1, which may break the time constraint. To satisfy the time constraint, some tasks have to be assigned to high-cost devices, leading to a higher cost of app 2.

For the application with task connection shown in Fig. 5(a), Fig. 9 shows the change of the system cost when the initial energy of mobile devices changes. Since other algorithms do not consider the impact of initial energy, their system cost will not change with the increment of initial energy.

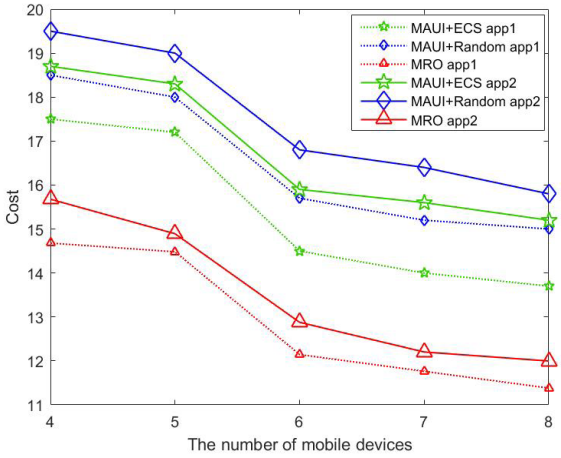


FIGURE 8. System cost with changed mobile devices.

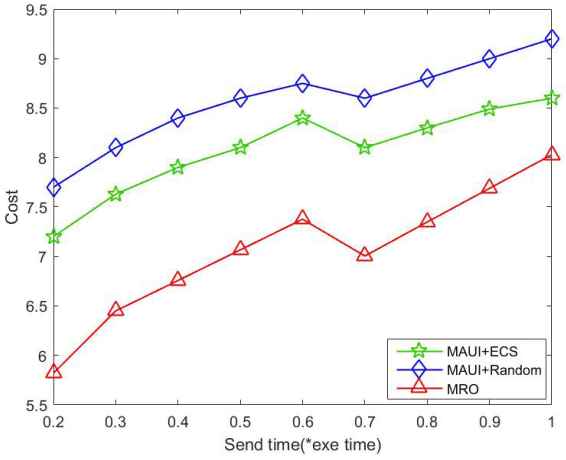


FIGURE 10. System cost with different sending time

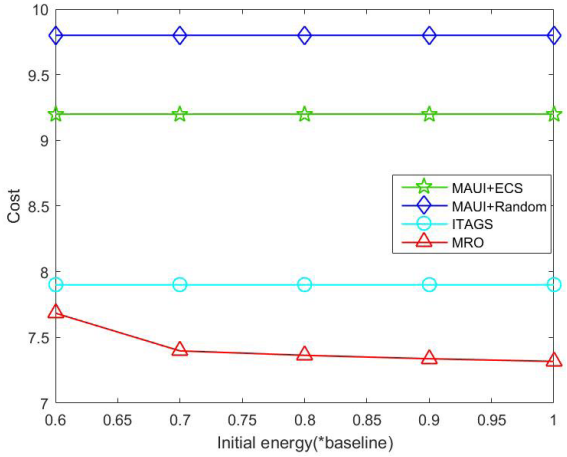


FIGURE 9. System cost with various initial energy.

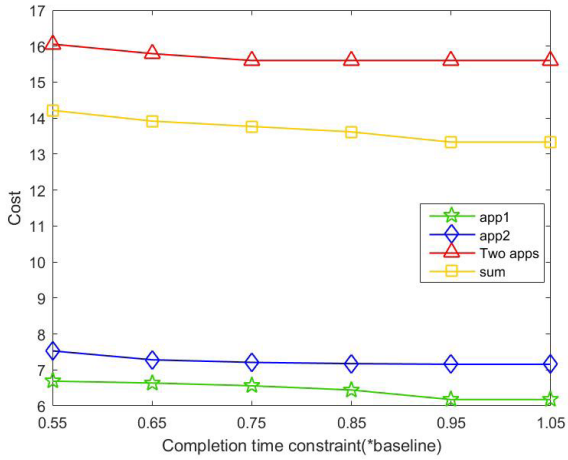


FIGURE 11. System cost with the completion time constraint.

In MRO, the system cost is related to both energy consumption and the initial energy. For the four mobile devices, we use the energy values, such as 1.0, 0.92, 0.88 and 0.96, as the initial baseline. To evaluate the influence of initial energy, we analyze the system cost as initial baseline of four mobile devices multiplies a factor (i.e., [0.6, 1.0]). As shown in Fig. 9, the higher initial energy is, the lower system cost is. Even if there is energy consumption during task execution, a mobile device with higher initial energy will not drop to energy threshold. Therefore, the total cost, which is related to the ratio between the energy consumption and the total available energy, is lower when the initial energy is higher.

For one application shown in Fig. 7(a), Fig. 10 shows the change of system cost when the sending time increases. The sending time is the duration of sending a task, which is determined by network load. When there is so much data transmitted in the network, the network load is high, leading to large sending time. For a task, we use its executing time to evaluate the sending time, which is expressed as the execution time multiplied by a factor (i.e., [0.2, 1.0]). As the sending time increases, the mobile devices will consume more energy and the system cost increases.

.

The MRO algorithm adopts the method of reassignment to reduce the system cost while other two algorithms do not reassign tasks. When the sending time is small, it is easier to reassign tasks to other devices and the sending cost is small. When the sending time is large, the sending process consumes more energy. Therefore, compared to other algorithms, MRO algorithm has a great advantage when the sending time is small, while its advantage will decline when the sending time is large. Moreover, the curve does not go up all the time. There is a drop around the abscissa value 0.6 for MRO. Within this range of sending time, the energy consumption of sending a task to another device is higher than that of executing the task in a busy mobile device which launches the task. Therefore, the busy device does not assign the task to other devices. Instead, it will execute the task itself and achieve the low cost. Although there is some fluctuation, we can observe the overall upward trend.

For two applications shown in Fig. 2, Fig. 11 shows the system cost with the completion time constraint. For each application, the default value of completion time constraint is the sum of the executing time on a mobile device. As shown in Fig.11, the completion time constraint is expressed as the default value multiplied by a factor (i.e., [0.55, 1.05]). Fig. 11 illustrates the system cost for only application 1, only application 2 and both of them. Besides, based on application 1's cost and application 2' cost, the sum cost is also described. As shown in Fig. 11, the system cost decreases as the completion time increases. When the completion time constraint becomes large, each task is more likely to be assigned to a low-cost device. Thus, the total cost decreases. After the completion time constraint exceeds certain range, the system cost is stable, which means no device with lower cost can be found.

In addition, the system cost for executing both applications at the same time is bigger than the sum of application 1's cost and application 2' cost. When two applications are executed at the same time, there are more tasks and the time is not enough. Some tasks have to choose high-cost devices to satisfy the time constraint. That is why the cost of executing both applications at the same time is higher than the sum cost.

VI. CONCLUSION

In this paper, offloading assignment is studied in a system consisting of mobile intelligent devices, local edge hosts and a remote cloud server. We define a more realistic system architecture and introduce relative energy consumption as a metric to reflect the real cost of mobile devices. Based on the system architecture, the offloading problem is formulated to minimize the system cost within each application completed deadline. To solve this NP-hard problem, the heuristic algorithm is proposed to offload total dependent tasks. At first, our algorithm arranges all tasks from different applications in a priority queue considering both completed time deadline and task-dependency requirements. Then, based on the priority queue, tasks are initially assigned to devices with higher energy under the time constraint. To obtain a better schedule realizing lower system cost, based on the relative remaining energy of mobile devices, we reassign tasks from high-cost devices to low-cost devices to minimize the system cost. Simulation results show that our proposed algorithm reduces the system cost effectively.

REFERENCES

[1] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Mar. 2012, pp. 945–953.

[2] A. Mtibaa, K. A. Harras, K. Habak, M. Ammar, and E. W. Zegura, "Towards mobile opportunistic computing," in *Proc. IEEE 8th Int. Conf. Cloud Comput. (CLOUD)*, Jun./Jul. 2015, pp. 1111–1114.

[3] Y. Qin, H. Wang, F. Zhu, and L. Zhai, "A multi-objective ant colony system algorithm for virtual machine placement in traffic intense data centers," *IEEE Access*, vol. 6, pp. 58912–58923, 2018.

[4] X. Li, H. Wang, S. Yi, X. Yao, F. Zhu, and L. Zhai, "Redundancy-guaranteed and receiving-constrained disaster backup in cloud data center," *IEEE Access*, vol. 6, pp. 47666–47681, 2018.

[5] D. Lu, X. Huang, G. Zhang, X. Zheng, and H. Liu, "Trusted device-to-device based heterogeneous cellular networks: A new framework for connectivity optimization," *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 11219–11233, Nov. 2018.

[6] L. Zhai, H. Wang, and C. Liu, "Distributed schemes for crowdsourcing-based sensing task assignment in cognitive radio networks," *Wireless Commun. Mobile Comput.*, Dec. 2017, Art. no. 5017653.

[7] X. Li, H. Wang, S. Yi, S. Liu, L. Zhai, and C. Jiang, "Disaster-and-evacuation-aware backup datacenter placement based on multi-objective optimization," *IEEE Access*, vol. 7, pp. 48196–48208, 2019.

[8] Y. Zhang, H. Liu, L. Jiao, and X. Fu, "To offload or not to offload: An efficient code partition algorithm for mobile cloud computing," in *Proc. IEEE 1st Int. Conf. Cloud Netw. (CLOUDNET)*, Nov. 2012, pp. 80–86.

[9] W. Zhang, Y. Wen, and D. O. Wu, "Energy-efficient scheduling policy for collaborative execution in mobile cloud computing," in *Proc. IEEE Conf. Conf. Comput. Commun. (INFOCOM)*, Apr. 2013, pp. 190–194.

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

[11] E. Cuervo, A. Balasubramanian, D.-K. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in *Proc. 8th Int. Conf. Mobile Syst., Appl., Services (MobiSys)*, Jun. 2010, pp. 49–62.

[12] Y.-H. Kao and B. Krishnamachari, "Optimizing mobile computational offloading with delay constraints," in *Proc. IEEE Global Commun. Conf. (Globecom)*, Dec. 2014, pp. 2289–2294.

[13] Y.-H. Kao, B. Krishnamachari, M.-R. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr./May 2015, pp. 1894–1902.

[14] K. Habak, M. Ammar, K. A. Harras, and E. Zegura, "Femto clouds: Leveraging mobile devices to provide cloud service at the edge," in *Proc. IEEE 8th Int. Conf. Cloud Comput. (CLOUD)*, Jun./Jul. 2015, pp. 9–16.

[15] S. Sundar and B. Liang, "Communication augmented latest possible scheduling for cloud computing with delay constraint and task dependency," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2016, pp. 1009–1014.

[16] 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 Trans. Serv. Comput.*, vol. 8, no. 2, pp. 175–186, Mar./Apr. 2015.

[17] M. Jia, J. Cao, and L. Yang, "Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr./May 2014, pp. 352–357.

[18] M.-A. H. Abdel-Jabbar, I. Kacem, and S. Martin, "Unrelated parallel machines with precedence constraints: Application to cloud computing," in *Proc. IEEE 3rd Int. Conf. Cloud Netw. (CloudNet)*, Oct. 2014, pp. 438–442.

[19] C. Wang and Z. Li, "Parametric analysis for adaptive computation offloading," *ACM SIGPLAN Notices*, vol. 39, no. 6, pp. 119–130, 2004.

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

[21] Y. Geng, W. Hu, Y. Yang, W. Gao, and G. Cao, "Energy-efficient computation offloading in cellular networks," in *Proc. IEEE 23rd Int. Conf. Netw. Protocols (ICNP)*, Nov. 2015, pp. 145–155.

[22] M.-H. Chen, B. Liang, and M. Dong, "Joint offloading and resource allocation for computation and communication in mobile cloud with computing access point," in *Proc. IEEE Conf. Conf. Comput. Commun. (INFOCOM)*, May 2017, pp. 1–9.

[23] L. Tong and W. Gao, "Application-aware traffic scheduling for workload offloading in mobile clouds," in *Proc. 35th Annu. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.

[24] S. Sundar and B. Liang, "Offloading dependent tasks with communication delay and deadline constraint," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 37–45.

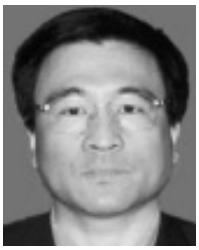
[25] K. Gai, M. Qiu, H. Zhao, L. Tao, and Z. Zong, "Dynamic energy-aware cloudlet-based mobile cloud computing model for green computing," *J. Netw. Comput. Appl.*, vol. 59, pp. 46–54, Jan. 2016.

[26] K. Gai, M. Qiu, and H. Zhao, "Energy-aware task assignment for mobile cyber-enabled applications in heterogeneous cloud computing," *J. Parallel Distrib. Comput.*, vol. 111, pp. 126–135, Jan. 2018.

[27] K. Gai and M. Qiu, "Reinforcement learning-based content-centric services in mobile sensing," *IEEE Netw.*, vol. 32, no. 4, pp. 34–39, Jul./Aug. 2018. doi: 10.1109/MNET.2018.1700407.
[28] *Task Graphs For Free*. Accessed: Apr. 1998. [Online]. Available: http://ziyang.eecs.umich.edu/dickrp/tgff/
[29] Y. C. Lee and A. Y. Zomaya, "Energy conscious scheduling for distributed computing systems under different operating conditions," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 8, pp. 1374–1381, Aug. 2011.
[30] 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 Trans. Serv. Comput.*, vol. 10, no. 5, pp. 797–810, Sep./Oct. 2015.
[31] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: Elastic execution between mobile device and cloud," in *Proc. 6th Conf. Comput. Syst.*, Apr. 2011, pp. 301–314.
[32] E. Ilavarasan and R. Manoharan, "High performance and energy efficient task scheduling algorithm for heterogeneous mobile computing system," *Int. J. Comput. Sci. Inf. Technol.*, vol. 2, no. 2, pp. 10–27, Apr. 2010. doi: 10.5121/ijcsit.2010.2202.
[33] L. Pu, X. Chen, J. Xu, and X. Fu, "Crowdlet: Optimal worker recruitment for self-organized mobile crowdsourcing," in *Proc. IEEE 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.
[34] M. Hirsch, C. Mateos, J. M. Rodriguez, and A. Zunino, "DewSim: A trace-driven toolkit for simulating mobile device clusters in Dew computing environments," *Softw. Pract. Exper.*, to be published. doi: 10.1002/spe.2696.

YINUO FAN is currently pursuing the master's degree with the School of Information Science and Engineering, Shandong Normal University. Her current research interests include offloading algorithm, antenna array, and distributed network optimization.

LINBO ZHAI received the B.S. and M.S. degrees from the School of Information Science and Engineering, Shandong University, in 2004 and 2007, respectively, and the Ph.D. degree from the School of Electronic Engineering, Beijing University of Posts and Telecommunications, in 2010. He is currently a Teacher with Shandong Normal University. His current research interests include cognitive radio, crowdsourcing, and distributed network optimization.



HUA WANG is currently a Professor with the School of Software, Shandong University. His research interests include network optimization, network algorithms, network measurement, network architecture and protocol, and network simulation. His research has been supported by China Next Generation Internet Project and NSF of China.