Computation Offloading over a Shared Communication Channel for Mobile Cloud Computing

Kai Guo, Mingcong Yang, and Yongbing Zhang
Graduate School of Systems and Information Engineering, University of Tsukuba, Japan
Email: s1730141@s.tsukuba.ac.jp, s1730144@s.tsukuba.ac.jp, ybzhang@sk.tsukuba.ac.jp

***Abstract*—In this paper, we focus on the problem of offloading computation intensive tasks of mobile applications from resource-scarce mobile devices to the servers located at the edge networks in order to minimize the average response time of the applications. We consider a number of mobile devices connected by a shared communication channel to a server located in the edge network and therefore transmission collision occurs if more than one mobile device attempts to transmit data simultaneously. We first formulate the offloading problem as a mixed integer programming (MIP) problem. Since the problem is NP-hard, we design a heuristic algorithm that considers possible transmission collision over the shared channel and offloads efficiently tasks of a mobile application to the server. We demonstrated that our proposed algorithm outperforms previous offloading algorithms significantly in terms of the average response time. Furthermore, we showed that our proposed algorithm yields less energy consumption than previous algorithms in realistic system scenarios.**

***Index Terms*—Mobile Cloud Computing, computation offloading, shared communication channel**

I. Introduction

Mobile Cloud Computing (MCC) as a new paradigm for mobile applications extends the computation and storage capabilities of mobile devices and furthermore reduces the power consumptions at mobile devices [1], [2]. The limitations of computing power and storage capacity along with battery lifetime of a mobile device can be alleviated by offloading computation intensive tasks from the mobile device to the servers allocated in the edge networks for remote processing [3], [4]. However, it is quite challenging to determine how to offload tasks of a mobile application since the offloading performance depends heavily on the characteristics of both the application and mobile device and also on the bandwidth of the communication network connecting the mobile device to the servers. A number of mobile devices are usually connected to a server located in the edge network via a shared access network, called *shared channel* in this paper, such as a wireless LAN or a cellular network. Therefore, more than one data transmission may collide with each other over the shared channel.

In our previous work [5], we proposed an algorithm that attempts to offload tasks of mobile applications in order to minimize the total consumption energy of mobile devices. We expressed a mobile application as a task flow graph and partitioned the graph into two separate parts, one consisting of the tasks being executed locally at the mobile device and the other consisting of the tasks executed remotely at the cloud server. The authors [6] proposed an offloading algorithm that attempts to minimize the response time of an application. However, they considered only one application in the system and furthermore the offloading decision is made based on the function call graph, i.e., the call relationship between the tasks. More importantly, transmission collision is not taken into account in the previous works. In this paper, we consider multiple mobile devices in the system each of which may execute a distinct application and transmission collisions may occur if more than one device transmits data simultaneously. We focus on how to minimize the average response time of all the mobile applications. We represent the execution of a mobile application as a task flow graph and attempt to minimize the path length, i.e., time required to pass through a path on the task flow graph from the beginning task to the ending task.

The contributions of this paper can be summarized as follows. We consider the problem of minimizing the average application response time on condition that transmission collisions over the access network may occur. The *response time* of an application means the time period starting from the instant the application is executed to the instant the user receives the output result. We express the execution flow of an application by a task flow graph. We formulate the problem of how to offload tasks of a mobile application to the server via a shared communication channel as a Mixed Integer Programming (MIP) problem. Since the offloading problem is NP-hard, we propose an efficient offloading algorithm. We evaluate our proposed algorithm compared with previous algorithms and show that our proposed algorithm outperforms previous algorithms significantly both in the application response time and the energy consumption.

II. Related Work

Deploying small-scale servers at the edge of the Internet is a promising way for mobile cloud computing to provide powerful computing resources to mobile devices with low latency. Computation-intensive and time-sensitive applications such as augmented reality [7] and face recognition [6] implemented on mobile devices can offload computation intensive tasks to the edge servers to reduce both the application response time and the energy consumption. Computation offloading approaches from an application viewpoint are proposed by the authors [8]–[10] in which an application is considered as an inseparable job and offloaded to the cloud server for remote execution. They formulated the problem of how to reduce the energy consumption as a noncooperative game and showed that there exists a Nash equilibrium for mobile users to offload or not. However, an application is usually comprised of a number of dependent tasks each of which can be considered as an

independent entity for offloading, leading to shorter application response time.

More recently, researchers [4], [11], [12] focused on the problem of how to partition and offload computation tasks of an application in order to speed up the execution of the application. Ra *et al.* [4] proposed a greedy offloading algorithm for mobile applications each of which consists of multiple tasks that are processed sequentially. M. Jia *et al.* [11] proposed to express an application as either a parallel or a sequential task flow graph and designed a heuristic offloading algorithm that tries to balance the load between mobile devices and the cloud server. However, the relations between computation tasks of an application are generally complicated and can not be simply expressed by only a parallel or chain flow graph. Yang *et al.* [12] proposed a heuristic offloading algorithm that aims to minimize the average response time of applications each of which consists of a simple task chain. However, there is no work in the literature taking into account of the collision between data transmissions over the access network connecting mobile devices to the server.

III. System Model and Problem Formulation

A. Mobile Cloud Computing System Model

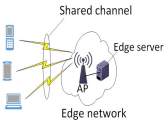

Fig. 1. MCC System model.

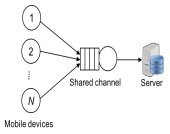

Fig. 2. Network model.

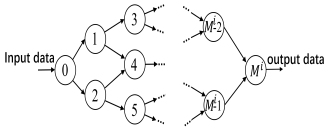

Fig. 3. Application model.

We consider a mobile cloud computing (MCC) system model as shown in Fig.1 where a set of mobile devices, denoted by $\mathcal{N} = \{1, 2, \cdots, N\}$, are connected to an *edge server*, also simply called a server, that is deployed at the edge of the Internet near mobile devices and works as a small-scale cloud server for the mobile devices. The server has more powerful computing resources than any mobile device and a number of mobile devices are connected to an access point (AP) via a shared radio communication channel, e.g., a wireless network. The AP is connected to the edge server via a high speed link and the transmission delay between AP and edge server can be neglected.

The network model considered in this paper is shown in Fig.2. We assume that two or more than two mobile devices may compete with one another for the shared communication channel for data transmission if they transfer data to the server simultaneously. We assume that the data transmission over the shared channel is performed based on the first-in-first-out (FIFO) principle and an ongoing data transmission cannot be interrupted by any other transmission.

B. Application Model

We assume that each user i executes only one application at his/her mobile device and the application of user i consists of a set of inseparable tasks $\mathcal{M}^i = \{0, 1, 2, \cdots, M^i\}$ which are all eligible for offloading except tasks 0 and M^i. Hereafter, we use a *user*, an *application*, or a *mobile device* interchangeably. We represent the execution of application i by a directed task flow graph as shown in Fig.3 in which the set of nodes indicate the computation tasks of application i. Furthermore, the arc between two nodes, denoted by $e^i_{j,k} \in \mathcal{E}^i$, indicates that task j sends its output data to task j. The data size transmitted from task j to k is denoted by $d^i_{j,k}$ and we assume that the transmission delay from task j to k can be neglected if tasks j and k are executed both at the same location, either at a mobile device or at the server. We assume that both the server and a mobile device have enough CPU resources so that the tasks can be processed in parallel if necessary. Furthermore, we assume that a task can start its computation only if it receives all the input data from its previous tasks.

C. Computation Offloading Problem

Our objective in this paper is to offload computation tasks of mobile applications so as to minimize the average application response time. When a mobile application is executed at a mobile device, the information on the application such as the computation workload of each task and data size transmitted between tasks is sent to the edge server. The server then determines which tasks should be offloaded and then sends the offloading decision back to the mobile device. Since the information about the applications and the offloading decisions is negligibly smaller than the data size sent between tasks, only the delays of data transmission between tasks are taken into account.

The execution times for a task j of application i locally at mobile device i and remotely at the edge server are denoted by m^i_j and c^i_j, respectively, and we assume that $m^i_j \geq c^i_j$. The beginning and ending times of the executions of task j are denoted by τ^i_j and T^i_j, respectively. Furthermore, the beginning and ending times of the data transmission from task j to k are denoted by $y^i_{j,k}$ and $W^i_{j,k}$, respectively. We use a binary integer variable x^i_j as the decision variable to show whether to offload task j or not. The value of x^i_j is 1 if task j of application i is determined to offload to the server and 0 otherwise. We let $x^i = \{x^i_j\}, j \in \mathcal{M}^i$ and $x = \{x^i_j, j \in \mathcal{M}^i, i \in \mathcal{N}\}$. We use a binary integer variable $\alpha^i_{j,k}$ to indicate whether two consecutive tasks j and k are processed at different locations either at mobile device or at the server. The value of $\alpha^i_{j,k}$ is 1 if tasks j and k are processed at different locations and 0 otherwise. Furthermore, we use a binary integer variable $\beta^{i,i'}_{j,k,j',k'}$ to indicate the order of two data transmissions from task j to k for application i and from task j' to k' for application i'. The value of $\beta^{i,i'}_{j,k,j',k'}$ is given by the following relation.

$$\beta^{i,i'}_{j,k,j',k'} = \begin{cases} 1, & \forall y^i_{j,k} \leq y^{i'}_{j',k'} \\ 0, & \forall y^i_{j,k} > y^{i'}_{j',k'} \end{cases} \quad (1)$$

The schedule for executing a mobile application i is denoted by $S^i = \{x^i_j, \tau^i_j, y^i_{j,k}\}, j \in \mathcal{M}^i, e^i_{j,k} \in \mathcal{E}^i, i \in \mathcal{N}$. The main notation used in this paper is shown in Table I.

Since a data transmission between two tasks may collide with another data transmission at the shared channel, we need to avoid a possible collision. For an application i, we have the following relations for tasks j and k, and data transmissions between tasks j and k.

TABLE I. Notation used in this paper.

Symbol	Meaning
$\mathcal{N}$	set of applications
$\mathcal{M}^i$	set of tasks of application i
$\mathcal{E}^i$	set of directed link connecting two tasks of application i
m_j^i	execution time of task j of application i at mobile device
c_j^i	execution time of task j of application i at cloud server
τ_j^i	beginning time of execution of task j of application i
T_j^i	ending time of execution of task j of application i
$e_{j,k}^i$	link from task k to j on the task flow graph of application i
$y_{j,k}^i$	beginning time for data transmission from task j to k of application i
$W_{j,k}^i$	ending time for data transmission from j to k of application i
x_j^i	decision variable indicating whether to offload task j of application i to cloud
$\alpha_{j,k}^i$	a binary integer variable indicating whether two consecutive tasks j and k of application i are executed at different sides, either at mobile device or at the server
$d_{j,k}^i$	data size transmitted from task j to k of application i
$\beta_{j,k,j',k'}^{i,i'}$	a binary integer variable indicating whether the transmission from tasks j to k of application i precedes another transmission from tasks j' to k' of application i'
s	channel transmission rate
S^i	execution schedule for application i

$$W_{j,k}^i = y_{j,k}^i + \alpha_{j,k}^i d_{j,k}^i/s, \quad j,k \in \mathcal{M}^i, e_{j,k}^i \in \mathcal{E}^i, \tag{2}$$

$$T_k^i = \tau_k^i + x_k^i c_k^i + (1-x_k^i)m_k^i, \quad k \in \mathcal{M}^i, \tag{3}$$

$$\alpha_{j,k}^i = |x_j^i - x_k^i|, \quad j,k \in \mathcal{M}^i, e_{j,k}^i \in \mathcal{E}^i, \tag{4}$$

Here, relation (2) means that task k receives the data sent from task j at time instant $y_{j,k}^i$ with a delay of $\alpha_{j,k}^i d_{j,k}^i/s$. Relation (3) shows that the ending time of the execution of task k is determined by the computation either at a mobile device or at the server. Finally, relation (4) shows that the value of $\alpha_{j,k}^i$ is determined by whether two consecutive tasks j and k are executed at different locations, i.e., if tasks j and k are executed at different locations, $\alpha_{j,k}^i = 1$ and otherwise $\alpha_{j,k}^i = 0$. On condition that relations (2) - (4) are satisfied, we can formulate the task offloading problem as a mixed integer programming (MIP) problem that minimizes the average application response time as follows.

$$\min \quad T = \frac{1}{N}\sum_{i\in\mathcal{N}}\left(T_{M^i}^i - \tau_0^i\right), \tag{5}$$

subject to

$$y_{j,k}^i \geq \tau_j^i + x_j^i c_j^i + (1-x_j^i)m_j^i, \ j,k \in \mathcal{M}^i, e_{j,k}^i \in \mathcal{E}^i, i \in \mathcal{N}, \tag{6}$$

$$\tau_k^i \geq y_{j,k}^i + \alpha_{j,k}^i d_{j,k}^i/s, \ j,k \in \mathcal{M}^i, e_{j,k}^i \in \mathcal{E}^i, i \in \mathcal{N}, \tag{7}$$

$$y_{j',k'}^{i'} \geq W_{j,k}^i - I(3-\beta_{j,k,j',k'}^{i,i'} - \alpha_{j,k}^i - \alpha_{j',k'}^{i'}), \ j,k \in \mathcal{M}^i, j',k' \in \mathcal{M}^{i'},$$
$$e_{j,k}^i \in \mathcal{E}^i, e_{j',k'}^{i'} \in \mathcal{E}^{i'}, i,i' \in \mathcal{N}, (i,j,k) \neq (i',j',k'), \tag{8}$$

$$x_j^i \in \{0,1\}, \ y_{j,k}^i \geq 0, j,k \in \mathcal{M}^i, e_{j,k}^i \in \mathcal{E}^i, i \in \mathcal{N}, \tag{9}$$

where we let $x_0^i = 0$ and $x_{M^i}^i = 0$ for $i \in \mathcal{N}$. Note that in problem (5) x_j^i and $y_{j,k}^i$ are decision variables. Constraint (6) shows the data transmission from task j to k should be after the completion of task j. Constraint (7) indicates that task k can only be executed after receiving all the input data from its previous tasks j ($e_{j,k}^i \in \mathcal{E}^i$). Constraint (8) guarantees that at any time instant there can be only one data transmission on the shared channel. For two transmissions, from task j to k of application i and from task j' to k' of application i' ,i.e., $\alpha_{j,k}^i = \alpha_{j',k'}^{i'} = 1$, if the transmission beginning time $y_{j,k}^i$ is earlier than $y_{j',k'}^{i'}$,i.e., $\beta_{j,k,j',k'}^{i,i'} = 1$, $y_{j',k'}^{i'}$ must be later than the transmission ending time $W_{j,k}^i$. A positive constant, denoted by I, is set to be greater than any other variables and is used to guarantee that $y_{j',k'}^{i'}$ can be earlier than $W_{j,k}^i$ when one of $\alpha_{j,k}^i$, $\alpha_{j',k'}^{i'}$ and $\beta_{j,k,j',k'}^{i,i'}$ is 0. Finally, constraint (9) shows the integrality and the nonnegativity constraints for x_i^j and $y_{j,k}^i$, respectively. Unfortunately, it is difficult to find the optimal solution for the problem (5) and in this paper we propose a heuristic offloading approach that yields very efficient solutions in realistic system scenarios.

IV. Proposed Computation Offloading Algorithm

Our proposed offloading algorithm consists of three sub-algorithms: the first one determines the optimal offloading tasks for an application with a simple task chain, the second one considers only one application in the system and makes the optimal offloading decision for each task on the task flow graph shown in Fig.3, and the third one considers multiple applications and the applications may compete with one another for the shared communication channel.

A. Single-User Chain-Application (SUCA) Offloading Algorithm

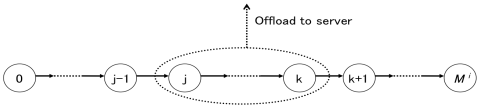

Fig. 4. A chain application.

We first consider how to offload a simple application that contains only a task chain as shown in Fig.4. For the sake of simplicity, we omit i from the superscripts of the variables. That is, x_j^i, m_j^i, c_j^i, $d_{j,k}^i$, $e_{j,k}^i$, $\mathcal{M}^i$, and $\mathcal{E}^i$ are simply denoted by x_j, m_j, c_j, $d_{j,k}$, $e_{j,k}$, $\mathcal{M}$, and $\mathcal{E}$. The relations (2)–(4) hold for this single-user chain-application offloading (SUCA) problem and the problem can be formulated as follows.

$$\min \quad T = T_M - \tau_0, \tag{10}$$

subject to

$$y_{j,j+1} \geq \tau_j + x_j c_j + (1-x_j)m_j, \ j,j+1 \in \mathcal{M}, e_{j,j+1} \in \mathcal{E}, \tag{11}$$

$$\tau_{j+1} \geq y_{j,j+1} + \alpha_{j,j+1} d_{j,j+1}/s, \ j,j+1 \in \mathcal{M}, e_{j,j+1} \in \mathcal{E}, \tag{12}$$

$$x_j \in \{0,1\}, \ y_{j,j+1} \geq 0, \ j,j+1 \in \mathcal{M}, e_{j,j+1} \in \mathcal{E}, \tag{13}$$

where we let $x_0 = 0$ and $x_M = 0$. We see from [13], [14] that a chain application can be offloaded at most only once. Furthermore, we can show the beginning and the ending offloaded tasks satisfy the following theorem.

Theorem 1: *For a mobile application with a simple task chain from 0 to M, suppose that there exists an optimal offloading decision, with the beginning and ending offloaded tasks j^* and k^*, respectively, and that the offloading decision yields the least response time. Then, the optimal beginning task j^* remains unchanged even if the ending offloaded task k^* is moved to $M-1$. Similarly, the optimal ending offloaded task k^* remains unchanged even if the beginning offloaded task j^* is moved to 1.*

The proof of Theorem 1 is omitted due to the space limitation. According to **Theorem 1**, we can find the optimal beginning and ending offloaded tasks respectively by scanning the task chain at most twice. The following SUCA algorithm is

used to determine the optimal beginning and ending offloaded tasks for a chain application. Here, the beginning and the ending offloaded tasks are chosen from the ranges of $[1, J_0]$ and $[K_0, M-1]$, respectively, where J_0 and K_0 are given parameters. The computation complexity of the SUCA algorithm is bounded by $O(M)$.

Algorithm 1 Single-user chain-application (SUCA) offloading algorithm.

Input: : m_j, c_j $(j \in \mathcal{M}), d_{j,k}$ $(e_{j,k} \in \mathcal{E}), s, J_0, K_0$
1: compute completion time, $T_{\max} = \sum_{j=1}^{M-1} m_j$, without offloading any task
2: compute the completion time with offloading tasks from j ($j \in [1, J_0]$) to $M-1$ and determine task j^* that yields the shortest completion time
3: compute the completion time with offloading tasks from 1 to k ($k \in [K_0, M-1]$) and determine task k^* that yields the shortest completion time
4: compute completion time T_{j^*,k^*} with j^* to k^*
5: **if** $T_{j^*,k^*} \geq T_{\max}$ **then**
6: **return** null
7: **else**
8: **return** beginning and ending offloaded tasks j^* and k^*
9: **end if**

B. Single-User General-Application (SUGA) Offloading Algorithm

In SUGA, we consider only one application whose tasks can be processed parallelly and/or sequentially as shown in Fig.3 but the collision of data transmission is not taken into account. We also omit i here from the symbols used for the variables. Note that the relations (2)–(4) also hold here and the offloading problem can be formulated as follows.

$$\min \quad T = T_M - \tau_0, \tag{14}$$

subject to

$$y_{j,k} \geq \tau_j + x_j c_j + (1-x_j) m_j, \quad j,k \in \mathcal{M}, e_{j,k} \in \mathcal{E}, \tag{15}$$

$$\tau_k \geq y_{j,k} + \alpha_{j,k} d_{j,k}/s, \quad j,k \in \mathcal{M}, e_{j,k} \in \mathcal{E}, \tag{16}$$

$$x_j \in \{0,1\},\ y_{j,k} \geq 0, \quad j,k \in \mathcal{M}, e_{j,k} \in \mathcal{E}, \tag{17}$$

where we let $x_0 = 0$ and $x_M = 0$.

The offloading decision for a single-user general application are based on the following two considerations. 1) In order to minimize the completion time of an application we need to minimize the path length from task 0 to M on the task flow graph. The length of a path means the sum of the task execution times and the transmission delays along the path from task 0 to M on the task flow graph. 2) Since a path from task 0 to M can be considered as a task chain, the tasks on any path can be offloaded only once and the offloaded tasks should be consecutive. The SUGA algorithm is described in Algorithm 2 in details. In each iteration, we decide at least one task that should be offloaded, and therefore the worst case computation complexity of SUGA is bounded by $O(M^3)$.

C. Multi-User General-Application (MUGA) Offloading Algorithm

In real systems, there are multiple applications in the system and transmission collisions may occur if more than one mobile device try to send data simultaneously. In this paper, we consider the problem of how to offload multiple applications via a shared channel and formulate the offloading problem as a MIP problem (5). Since the problem is NP-hard, we propose a heuristic algorithm, called *MUGA*, that determines an offloading schedule for all the applications so as to minimize the average application response time. The MUGA algorithm is adaptive in nature in the sense that an application can be executed at any time. When an application i is newly executed, the MUGA algorithm determines the offloading schedule for application i and, if necessary, updates the offloading schedules for other applications i' $(i' < i)$ that have been executed earlier than application i. The MUGA algorithm is described in Algorithm 3 and consists of the following three phases.

Algorithm 2 Single-user general-application (SUGA) offloading algorithm.

Input: : $c_j, m_j, d_{j,k}$ $(j,k \in \mathcal{M}, e_{j,k} \in \mathcal{E}), s$
1: let $L = \{\mathcal{M}\}, C = \emptyset$ and $x_j = 0$ $(j \in \mathcal{M})$
2: calculate the completion time along the longest path from task 0 to M, $T_{\max}$
3: **repeat**
4: find the current longest path p from task 0 to M
5: **if** $x_j > 0$ such that task j is on path p **then**
6: set the first and the last offloaded tasks along p to be J_0 and K_0, respectively
7: **else**
8: set the next task from 0 and the previous task to M along p to be K_0 and J_0, respectively
9: **end if**
10: run **Algorithm 1** to determine the beginning and ending offloaded tasks j^* and k^* respectively on path p
11: set $x_j = 1$ for each task on the bypass of path p from j^* to k^* and let $C = C \cup j, L = L \setminus j$
12: calculate the completion time from task 0 to M along path p, T_p
13: **if** $T_p > T_{max}$ **then**
14: Let $x_j = 0$ $(j \in \mathcal{M})$ **break** from repeat loop
15: **end if**
16: **for** each bypass p_{ij} from an offloaded task i to j $(i,j \in C)$ **do**
17: **if** task k on path p_{ij} and $k \in L$ **then**
18: Let $x_k = 1$ and $C = C \cup k, L = L \setminus k$
19: **end if**
20: **end for**
21: **until** no new offloading task can be found
22: **return** offloading decision $x = \{x_0, x_1, \cdots, x_M\}$

1) Initial schedule determination: The offloading decision for each task k of application i, i.e., $x_k^i, k \in \mathcal{M}^i$, is made by using the SUGA algorithm. The beginning and ending times of the execution of task j, i.e., τ_j^i and T_j^i, and the beginning and ending times for data transmission from task j to k, i.e., $y_{j,k}^i$ and $W_{j,k}^i$, are determined without considering the channel capacity constraints, respectively. The initial schedule, S^i, is obtained by steps $2 \sim 3$ of Algorithm 3.

2) Elimination of interruption to existing application: Since the transmission collision is not considered in Phase 1, a data transmission of application i may collide with another transmission of existing applications i' $(i' < i)$. Therefore, we need to determine whether to postpone the collided data transmissions of application i or change the offloading decisions. If a data transmission of application i from task j to k collides with another data transmission of applications i' and can be postponed, the offloading decision of task k along with its descendant offloaded tasks are kept no change. Otherwise, if it is beneficial to process task k locally at the mobile device, the offloading decision will be changed; i.e., let $x_k^i = 0$ and its descendant offloaded tasks will be examined recursively. That is, if a data transmission of application i from task j to k with the earliest data transmission time $y_{j,k}^i$ collides with another data transmission of existing application i' from task j' to k', we compare the completion time when offloading task k with that when not offloading task k, i.e., by comparing

$T + W^{i'}_{j',k'} - y^i_{j,k}$ with T'. If $T' \geq T + W^{i'}_{j',k'} - y^i_{j,k}$, the offloading decision will not be changed; otherwise, task k will be processed locally and schedule S^i will be updated.

3) Rescheduling colliding transmissions: Once the offloading decision of application i is made, the data transmissions of all the applications to and back from the edge server are examined and if necessary rescheduled based on a First-In-First-Out (FIFO) basis. Furthermore, if a task needs to transfer data to more than one following task via the shared channel, we need to determine to which one the data should be transferred first. For example, if task j' needs to send data to two tasks k' and k'', we need to determine to which one (k or k') the data should be transferred first. In the MUGA algorithm, the longer of two paths from task j' to $M^{i'}$ via tasks k' and k'' will be chosen to transfer data first.

The worst case computation complexity of MUGA is bound by $O\left(|M^i|^3\right)$, where $|M^i|$ denotes the number of tasks of application i.

V. Performance Evaluation

We examined the performance of our proposed algorithm compared with two previous algorithms [5], [6], denoted by *"Energy-based"* and *"Partial offloaded"* in the figures, respectively. Two extreme cases, denoted by *"Full-offloaded"* and *"No offloaded"*, respectively, were also simulated and in the former all the tasks except task 0 and M^i of application i are offloaded while in the latter no task will be offloaded. We simulated a network model with 1000 mobile devices each of which runs a face recognition application [6]. Each application consists of 22 tasks and is represented by a flow graph as shown in Fig. 5. We assumed that the data size transmitted between tasks and the execution times of the tasks of each application are different from others.

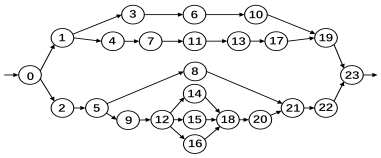

Fig. 5. Flow graph of face recognition application.

The computation times of tasks from m^i_1 to m^i_{22} were generated randomly between $[4, 12]$s but the computation times of tasks m^i_0 and m^i_{23} were 0. The inter-execution time of an application was assumed to follow the exponential distribution. The size of data sent between two tasks was generated randomly between $[10, 40]$MB. The results shown in the figures are the sample means obtained by the simulation with 95% confidence interval. The half widths of the confidence intervals are all less than 3% of the sample means and therefore are not shown in the figures. Simulation program was developed using Python 3.5 on a Microsoft Windows 10 computer with the Intel E3-3.0GHz CPU and 24GB memory.

A. Effect of communication speed

Fig.6 shows the average application response times obtained by various algorithms when changing the speed of the shared channel. The ratio of computation speed of the server to a mobile device was fixed to be 4 and the average inter-execution time of an application was fixed to be 40s. From Fig.6, we see that our proposed algorithm performs much better than other algorithms for a wide range of communication speed. On one hand, when the communication speed is extremely slow it is not beneficial to offload any task to the server due to the communication delay, and we can see that our proposed algorithm performs similarly to the case of "No offloaded". On the other hand, when the communication speed is very fast, offloading tasks to the server leads to large improvement over the response time and we can see in this case that our proposed algorithm performs similarly to the case of "Full offloaded".

We also see that our proposed algorithm outperforms the previous algorithms denoted by "Energy-based" [5] and "Partial offloaded" [6]. In order not to cause any bias to the two previous algorithms, we assumed that they know the exact transmission

Algorithm 3 The multi-user general-application (MUGA) offloading algorithm.

Input: : $c^i_j, m^i_j, d^i_{j,k}$ $(j, k \in \mathcal{M}^i, e^i_{j,k} \in \mathcal{E}^i, i \in \mathcal{N})$, s, $S(S = \bigcup_{i\in\mathcal{N}} S^i)$
Output: : updated schedule S

calculate the completion time of application i, denoted by T^i_L, without offloading any task, and the total completion times of all the other applications, $T_O = \sum_{l=1}^{i-1}(T^l_{M^l} - \tau^l_0)$
/* steps $2 \sim 3$: determine initial schedule S^i */
run **Algorithm 2** to obtain an offloading decision for application i, x^i
determine the initial schedule for application i, S^i, using x^i without considering transmission collision
/* steps $4 \sim 18$: eliminate interruption to existing applications */
while any $x^i_l > 0$ $(l \in \mathcal{M}^i)$ **do**
  calculate the completion time of application i, denoted by T, based on the current S^i
  find the earliest beginning time for transmission from task j to k, $y^i_{j,k}$, for application i
  find the last ending time for data transmission from task j' to k', $W^{i'}_{j',k'}$, for application i' $(i' < i)$ such that $T^{i'}_{j'} \leq y^i_{j,k} < W^{i'}_{j',k'}$
  if application i' exists **then**
    calculate the completion time of application i, denoted by T', with $x^i_k = 0$ $(k \in \mathcal{M}^i)$
    if $T' \geq T + W^{i'}_{j',k'} - y^i_{j,k}$ **then**
      break from **while** loop /* delaying transmission is acceptable */
    else
      $x^i_k = 0$ and update S^i
    end if
  else
    break from **while** loop
  end if
end while
/* steps $19 \sim 32$: reschedule colliding data transmissions */
while $y^{i'}_{j',k'} \leq y^{i''}_{j'',k''} < W^{i'}_{j',k'}$ $((i', i'' \leq i),\ (i', j', k') \neq (i'', j'', k''))$ **do**
  if $T^{i'}_{j'} < T^{i''}_{j''}$ **then**
    $y^{i''}_{j'',k''} = W^{i'}_{j',k'}$ and update $S^{i''}$
  else if $T^{i'}_{j'} = T^{i''}_{j''}$ $(i' = i'', j' = j'', k' \neq k'')$ **then**
    calculate completion times of applications i' and i'' from task j' to $M^{i'}$ via k' (denoted by $t^{i'}_{j',k'}$) and task j'' to $M^{i''}$ via k'' (denoted by $t^{i''}_{j'',k''}$), respectively
    if $t^{i'}_{j',k'} \geq t^{i''}_{j'',k''}$ **then**
      $y^{i''}_{j'',k''} = W^{i'}_{j',k'}$ and update $S^{i''}$
    end if
  end if
  if $\sum_{l=1}^{i}(T^l_{M^l} - \tau^l_0) \geq T^i_L + T_O$ **then**
    $x^i_k = 0$ $(k \in \mathcal{M}^i)$ and update S^i
    restore schedules of all the applications $S^{i'}$ $(i' < i)$ to their initial ones before scheduling user i
  end if
end while

delay at each time instant. Since these algorithms do not consider the transmission collisions in their offloading decisions, their performance degrades fast when the communication speed is slow since frequent incorrect decisions are made.

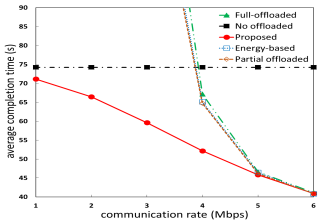

Fig. 6. Effect of communication speed.

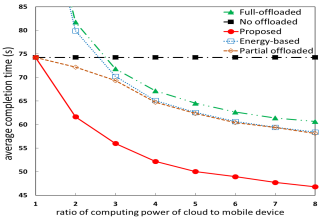

Fig. 7. Effect of ratio of computing power of server vs. device.

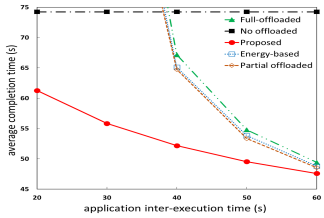

Fig. 8. Effect of application inter-execution time.

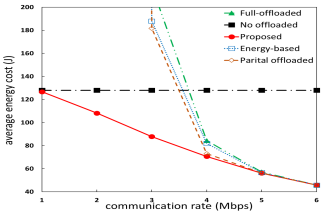

Fig. 9. Energy consumption comparison.

B. Effect of computing power of server versus a mobile device

Fig.7 shows the average application response times obtained by various algorithms for different ratios of computing power of the server to a mobile device. In this experiment, the transmission speed of the shared channel was fixed to be 4Mbps and the average inter-execution time of an application was fixed to be 40s. From Fig.7, we can see that our proposed algorithm performs much better than all the other algorithms. When the computing power of the server is the same as a mobile device, no task will be offloaded since there is no benefit to offload due to the additional communication delay. Again, since our proposed algorithm focuses on the application response time and furthermore takes the transmission collisions into account, it outperforms other algorithms.

C. Effect of application execution frequency

Fig.8 shows the average application response times of the algorithms under consideration when changing the application execution frequency. The shared channel speed was fixed to be 4Mbps and the ratio of computing power of the server to that of a mobile device was fixed to be 4. From Fig.8, we see that our proposed algorithm behaves more stable than other algorithms when the application execution rate changes. When the applications are executed frequently, all of the previous algorithms degrade much faster than our proposed algorithm.

D. Energy consumption comparison

We considered four key energy consumption parameters: *computation power* for computation, *state holding power* for keeping the active state but not in computation, *data transmission/receiving power* for transferring/receiving data, and *transmission waiting power* for collision avoidance. The former three parameters were set to 0.6, 0.3, and 1.3W, by referring to [6], while the fourth parameter was set to 1W. Fig.9 shows the average energy consumption by an application for the algorithms under consideration when changing the speed of the shared channel. The ratio of computation speed of the server to a mobile device was fixed to be 4 and the average inter-execution time of an application was fixed to be 40s. From Fig.9, we see that our proposed algorithm performs better than other algorithms, even though the objective of our proposed algorithm is for the minimization of the application completion time. The main reason of this result is that in our proposed algorithm the server determines the data transmissions of all the applications thoroughly and clearly so that there is no need to attempt a data transmission. Additionally, more accurate offloading decision is made in our algorithm and therefore the waste of power consumption can be avoided.

VI. Conclusion

In this paper, we focused on the problem of how to offload computation tasks of mobile applications in order to minimize the average application response time. We first formulated the offloading problem as a mixed integer programming (MIP) problem and then proposed an heuristic offloading algorithm. We evaluated our proposed algorithm compared with previous algorithms by simulation experiments. The results show that our proposed algorithm outperforms significantly previous algorithms for a wide range of system parameters. Furthermore, we examined the energy consumption of our proposed algorithm with realistic system parameters and the results also show that our proposed algorithm yields much less energy consumption than previous algorithms and when the network speed is slow the energy reduction becomes significantly large.

References

[1] X. Fan, J. Cao, and H. Mao, "A survey of mobile cloud computing", *ZTE Communications*, Vol. 9, No. 1, 2010; 4–8.

[2] F. Liu, *et al.* "Gearing Resource-Poor Mobile Devices with Powerful Clouds: Architecture, Challenges and Applications", *IEEE Wireless Communications*, Vol. 20, No. 3, 2013; 14–22.

[3] M. V. Barbera, *et al.* "To offload or not to offload? the bandwidth and energy costs of mobile cloud computing", *Proc. IEEE Int. Conf. Computer Communications (INFOCOM2013)*, 2013; 1285–1293. 10

[4] M. R. Ra, *et al.* "Odessa: enabling interactive perception applications on mobile devices", *Proc. ACM 9th Int. Conf. Mobile Systems, Applications, and Services (MobiSys2011)*, 2011; 43–56.

[5] Y. Tao, Y. Zhang, and Y. Ji, "Efficient Computation Offloading Strategies for Mobile Cloud Computing", *Proc. IEEE 29th Int. Conf. Advanced Information Networking and Applications (AINA2015)*, 2015; 626–633.

[6] H. Wu, W. Knottenbelt, K. Wolter, and Y. Sun, "An Optimal Offloading Partitioning Algorithm in Mobile Cloud Computing", *Int. Conf. Quantitative Evaluation of Systems (QEST2016)* in LNCS 9826, eds. G. Agha and B.V. Houdt, Springer, 2016; 311–328.

[7] A. B. Craig, "Understanding Augmented Reality: concepts and applications", 1st Ed., Morgan Kaufmann, 2013.

[8] X. Chen, "Decentralized computation offloading game for mobile cloud computing", *IEEE Trans. Parallel and Distributed Systems*, Vol. 26, No. 4, 2015; 974–983.

[9] X. Chen, *et al.* "Efficient multi-user computation offloading for mobile-edge cloud computing", *IEEE/ACM Trans. Networking*, 2015; 1–14.

[10] E. Meskar, *et al.* "Energy efficient offloading for competing users on a shared communication channel", *Proc. IEEE Int. Conf. Communications (ICC2015)*, 2015; 3192–3197.

[11] M. Jia, J. Cao, and L. Yang, "Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing", *Proc. IEEE Int. Conf. Computer Communications Workshops (INFOCOM WKSHPS)*, 2014; 352–357.

[12] L. Yang, *et al.* "Multi-user computation partitioning for latency sensitive mobile cloud applications", *IEEE Trans. Computers*, Vol. 64, No. 8, 2015; 2253–2266.

[13] S. Yang, *et al.* "Application offloading based on R-OSGi in mobile cloud computing", *Proc. IEEE Int. Conf. Mobile Cloud Computing, Services, and Engineering (MobileCloud2016)*, 2016; 7 pages.

[14] Y. Zhang, *et al.* "To offload or not to offload: An efficient code partition algorithm for mobile cloud computing", *Proc. IEEE Int. Conf. Cloud Networking (CLOUDNET2012)* 2012; 80–86.