Energy-efficient Task Execution for Application as a General Topology in Mobile Cloud Computing

Weiwen Zhang and Yonggang Wen, *Senior Member, IEEE*

Abstract—Mobile cloud computing has been proposed as an effective solution to augment the capabilities of resource poor mobile devices. In this paper, we investigate energy-efficient collaborative task execution to reduce the energy consumption on mobile devices. We model a mobile application as a general topology, consisting of a set of fine-grained tasks. Each task within the application can be either executed on the mobile device or on the cloud. We aim to find out the execution decision for each task to minimize the energy consumption on the mobile device while meeting a delay deadline. We formulate the collaborative task execution as a delay-constrained workflow scheduling problem. We leverage the partial critical path analysis for the workflow scheduling; for each path, we schedule the tasks using two algorithms based on different cases. For the special case without execution restriction, we adopt *one-climb* policy to obtain the solution. For the general case where there are some tasks that must be executed either on the mobile device or on the cloud, we adopt Lagrange Relaxation based Aggregated Cost (LARAC) algorithm to obtain the solution. We show by simulation that the collaborative task execution is more energy-efficient than local execution and remote execution.

Index Terms—Mobile cloud computing, energy efficiency, general topology, task execution

1 Introduction

The proliferation of mobile devices has driven the emergence of a large number of mobile applications. Cisco VNI report [1] predicts that by 2019 there will be nearly 1.5 mobile devices per capita. Indeed, according to the statistics of Apple and AppBrain, there were 1.3 million applications on iPhone App Store by July 2014 [2] and more than 1.5 million Android applications on Google Play by April 2015 [3]. These numbers could continue to grow, resulting in great impact on ubiquitous computing.

However, mobile devices, due to the small physical size, are resource constrained for the applications that are computation intensive. Although there has been technical innovation on mobile devices (e.g., smartphones and tablets), computation capability and memory capacity still fall behind their desktop counterparts. Furthermore, as the battery system has not been improved on pace with the computation capability and memory capacity, the short battery lifetime has become a bottleneck of running computation intensive mobile applications [4]. The mobile applications, such as image processing and object recognition, can consume a large amount of energy on mobile devices. Therefore, energy issue of mobile devices should be taken into consideration for designing mobile applications.

Mobile cloud computing [5], [6] has been touted as an effective solution to mitigate the resource constraints of mobile devices. Two categories of cloud-assisted mobile application platforms have been proposed, including infrastructure-based cloud and ad-hoc virtual cloud. Empowered by the remote servers, the infrastructure-based cloud provides sufficient computation resources to mobile devices, such as MAUI [7], CloneCloud [8], Cloudlets [9] and Weblet [10]. Rather than relying on remote servers, the ad-hoc virtual cloud, is formed by a group of mobile devices nearby that work cooperatively to accomplish application offloading [11], [12]. By application offloading, the tussle between the computation intensive mobile applications and resource poor mobile devices can be alleviated. Nevertheless, it is not always energy efficient to offload applications to the cloud for execution [4]. The decision of application offloading, i.e., *offloading policy*, should be investigated to consider the tradeoff between the amount of computation and communication.

The design of offloading policy for optimal application execution in mobile cloud computing has received extensive concerns from researchers. In [13], we derived the offloading policy for the application modeled as an entire task, which is a case for coarse-grained task topology. In [14], we presented the offloading policy for the application modeled as a linear chain, which requires that the tasks should start to execute when its precedent task has been completed. However, general task topology of the applications is more common in practice, which requires that the execution of a task may depend on the output of a set of tasks. One typical example is the computer vision based applications on the mobile device, including object, face, pose and gesture recognition [15]. To obtain the offloading decision for the image based object recognition as a general task topology, Yang et al. in [15] presented a framework for maximizing the throughput of the application. Giurgiu et al. in [16] modeled the ap-

- *Weiwen Zhang is with Computing Science Department, Institute of High Performance Computing (IHPC), Agency for Science, Technology and Research (A*STAR), Singapore 138632. Email: zhangww@ihpc.a-star.edu.sg.*
- *Yonggang Wen is with School of Computer Engineering, Nanyang Technological University, Nanyang Avenue, Singapore 639798. Email: ygwen@ntu.edu.sg.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2511727, IEEE Transactions on Cloud Computing

plication as a consumption graph and adopted a min-cut approach to minimize the interaction time and the amount of exchanged data. Chun et al. in [8] and Cuervo et al. in [7] presented a formulation of 0-1 integer linear program to decide the execution of the tasks to minimize the energy consumption on the mobile device. Nevertheless, solving the integer linear program is computation intensive, which may not be practical for delay sensitive mobile applications. Thus, the research of offloading decision for general task topology is still limited.

In this paper, we investigate energy-efficient collaborative task execution for the application that is modeled as a general topology, where a task is referred to as a method within the application. We consider the architecture in [17], [18] that each mobile device is associated with a cloud clone for task execution. Each task within the application can be either executed on the mobile device or on the cloud clone under collaborative task execution. The objective is to reduce the energy consumption on the mobile device while completing all the tasks under a delay deadline. We take a graph model approach to represent the tasks within the application as a directed acyclic graph. With the graph representation, we formulate the collaborative task execution as a delay-constrained workflow scheduling problem to minimize the energy consumption on the mobile device.

We propose scheduling algorithms for the collaborative task execution to conserve the energy on the mobile device under the delay deadline. The proposed scheduling algorithms are based on partial critical path analysis [19], which enable us to find out the critical path formed by a set of critical parents. Specifically, the critical parent is defined as the parent node of a task that results in the maximum value of the earliest start time of the task. After the identification of the partial critical path, we find its sub-deadline and schedule the tasks on the path using two algorithms based on different application scenarios. Notice that there could be some tasks that must be executed on the mobile device or on the cloud clone; we refer it to as execution restriction. For a special case without the execution restriction, we apply *one-climb* policy [14], in which there exists at most one migration from the mobile device to the cloud if ever for the minimum energy consumption. For a general case with the execution restriction, we adopt Lagrange Relaxation based Aggregated Cost (LARAC) algorithm [20] to schedule the tasks on the partial critical path.

We conduct performance evaluation for the collaborative task execution. We consider mesh, tree and more general topology as the task organization in the application for the special case and the general case without and with the execution restriction, respectively. Simulation results demonstrate that as the data rate increases, more tasks will be offloaded to the cloud for execution and more energy consumption can be saved for both the special case and the general case. Moreover, the collaborative task execution can save energy consumption compared to the local execution and is more flexible than the remote execution. Thus, by exploring the granularity of tasks in the application, we can have a more energy-efficient collaborative task execution for mobile applications. In addition, the collaborative task execution under the special case without execution restriction consumes less energy than that under the general case with the execution restriction. This suggests that one needs to have a reasonable partition of tasks within the mobile application for energy-efficient collaborative task execution.

The rest of the paper is organized as follows. We present the related work of offloading policy for mobile cloud computing in Section 2. In Section 3, we present the mathematical models and a problem formulation of the collaborative task execution. Section 4 provides the scheduling algorithms of the collaborative task execution for the application as a general topology. Simulation results are presented to evaluate the effectiveness of the proposed scheduling algorithms in Section 5. Section 6 concludes the paper and suggests future work.

2 Related Work

Some researchers have provided analysis of offloading decision for coarse-grained applications. Kumar et al. in [4] presented an energy model to analyze whether to offload applications to the cloud. The analysis was made based on the tradeoff between computation energy for mobile execution and communication energy under a static network for cloud execution. Miettinen et al. in [21] had similar analysis and demonstrated that workload, data communication patterns and technologies are the main factors that affect the energy consumption of mobile applications for application offloading. But its analysis was roughly based on measurements and investigations. In [13], the offloading decision was made by solving two optimization problems, i.e., optimal clock frequency configuration on the mobile device for local execution and optimal data transmission over stochastic wireless channel for remote execution. The policy was then to select the execution mode that results in less energy consumption on the mobile device. However, it can only be applied for the application as a single task.

For fine-grained applications, graph representation and integer program formulation have been typical approaches for obtaining the offloading policy.

Graph representation: Given an application, Li et al. in [22] constructed a cost graph and divided the application into server tasks and client tasks for execution. Branch-and-bound algorithm was used to solve the optimization problem for minimizing the energy consumption of computation and data communication cost for task execution. In [23], Wang and Li modeled a task graph and proposed a min-cut approach to minimize the cost for task execution between the mobile device and the cloud. Giurgiu et al. in [16] constructed a graph to represent an application and presented algorithms to find a cut in the graph for the application execution between the mobile device and the cloud. The cut was determined by solving an unconstrained optimization problem, the objective function of which is a combination of the interaction time and the amount of exchanged data. Nevertheless, their cut-based approach cannot be used in our research problem, since they do not consider the energy consumption and time delay constraint.

Integer program formulation: CloneCloud [8] considered the energy cost as a function of CPU state, display state and network state, and chose which tasks in the application to be migrated for execution. To minimize the energy consumption, the authors formulated an optimization problem

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2511727, IEEE Transactions on Cloud Computing

as an integer linear program, but that work did not provide the guarantee of the execution time. Similarly, MAUI [7] presented an integer linear program formulation to decide at runtime which tasks should be remotely executed, in order to optimize the energy savings under application execution time constraint. Wishbone [24] also leveraged integer linear program to partition the program for execution between servers and embedded nodes, aiming to minimize a combination of network bandwidth and CPU consumption. Yang et al. in [15] proposed a framework to determine the execution of data stream applications on the mobile device with cloud computing. The authors considered an image-based object recognition application modeled by a data flow graph. The objective was to maximize the throughput in processing the stream data, but the energy consumption was not considered. Among those previous work, MAUI [7] is quite similar with our work, in which the objective is to minimize the energy consumption while satisfying the application delay. Although it can achieve the optimal solution for the ILP formulation, its computation complexity is high, which is not efficient when the number of tasks within the application is large. In this paper, we will present efficient algorithms for the task scheduling.

This work investigates offloading policy for application as a general topology, which is an extension of our previous work [25]. Compared to [25], we have the following contributions. First, we have a general framework for the offloading policy, considering the execution restriction for the applications. Second, apart from the *one-climb* policy as the solution for a special case, we import LARAC algorithm into the task scheduling to obtain the solution for a general case. Finally, based on the task scheduling algorithm, we provide more results for offloading policy.

3 System Models and Problem Formulation

In this section, we present the models, including task graph model, computation cost model and communication cost model, and problem formulation for the collaborative task execution between the mobile device and the cloud clone.

3.1 Task Graph Model

A mobile application can consist of a set of tasks in the granularity of either method [7] or module [16]. Within an application, a task can call other tasks for execution. The invoke of a task needs the output data from other tasks. Thus, the tasks are connected through the corresponding dependencies (shown in Figure 1 (a) and Figure 1 (b)).

To model the dependencies among the tasks, we represent the tasks by a directed acyclic graph $G = (V, A)$. We have the following assumptions for the task graph model:

- We do not consider the cyclic graph, where methods or modules can be invoked recursively; indeed, we can regard them as single tasks.
- We assume that the tasks have been identified based on the design and implementation of the mobile application; we do not consider how we partition the application into tasks, but focus on the scheduling for the tasks.

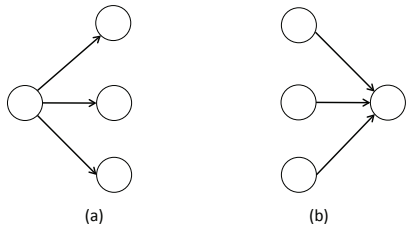


Fig. 1. Tasks within the application. A node represents a task and an arc denotes the data dependency between tasks. (a) The completion of a task can call other tasks for execution. (b) The execution of a task can require the output data of other tasks.

- We assume that we have the information in advance, for which tasks must be executed on the mobile device and on the cloud clone, respectively.

Without loss of generality, we suppose that there are n tasks in the application. We denote V as the node set for the tasks, i.e., $V = \{v_i\}$. In this paper, we use the terms node and task interchangeably. We add two dummy nodes into the graph, i.e., v_0 and v_{n+1} to denote the initialization and termination of the application, as shown in Figure 2. We denote A as the arc set for data dependencies among the tasks. An arc a_{ij} in A indicates the dependency between the adjacent task i and task j, which requires that task j cannot be started if its parent, task i, is not completed. In addition, we denote X as the execution decision for the tasks, i.e., $X = \{x_i\}$. The value of x_i is either 0 denoting that task i is executed on the mobile device, or 1 denoting that on the cloud clone. As the initialization and the termination of the mobile application should be on the mobile device, we have $x_0 = 0$ and $x_{n+1} = 0$ for the two dummy tasks.

The task graph model has been demonstrated in the interactive mobile applications of object, pose and gesture recognition [15], [26]. For example, in object and pose recognition [26], an image goes through a scale transformer before a set of SIFT features are extracted from the image (corresponding to Figure 1(a)); and these SIFT features will be delivered to the component of feature merger for finding objects of interest (corresponding to Figure 1(b)). After that, the features of each object are clustered to separate the objects and a consensus algorithm is subsequently used to recognize each object and determine its pose. As such, the process of object and pose recognition, consisting of image scale transformer, feature extractor, feature merger, cluster and recognition algorithm, can be modeled as a general task topology. Those two works, however, only aim to achieve the maximum throughput of the application, without considering the energy consumption on the mobile device. In this paper, we aim to reduce the energy consumption of completing the tasks within the application by collaborative task execution.

3.2 Computation Cost Model

Computation cost of a task depends on its execution decision, i.e., mobile device or cloud clone. We consider the computation time and computation energy on the mobile device for the task execution as follows.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2511727, IEEE Transactions on Cloud Computing

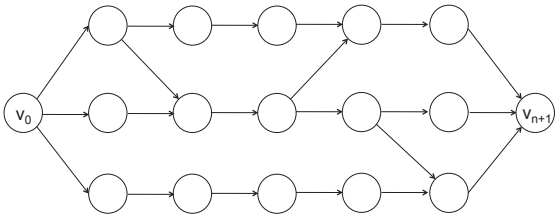


Fig. 2. A graph representation of tasks within an application as a general topology.

First, we denote $T_{v_i}^{comp}(x_i)$ as the computation time of task i, where x_i is the decision for the execution of task i. Then, the computation time of task i is given by,

$$T_{v_i}^{comp}(x_i) = \begin{cases} \omega_i f_m^{-1}, & x_i = 0, \\ \omega_i f_c^{-1}, & x_i = 1, \end{cases} \tag{1}$$

where ω_i is the workload of task i, f_m and f_c are the clock frequency of the mobile device and the cloud clone, respectively. We assume that $f_c > f_m$.

Second, we denote $E_{v_i}^{comp}(x_i)$ as the energy consumption on the mobile device for the execution of task i. Then, the energy consumption on the mobile device for the execution of task i is given by,

$$E_{v_i}^{comp}(x_i) = \begin{cases} \omega_i p_m f_m^{-1}, & x_i = 0, \\ \omega_i p_0 f_c^{-1}, & x_i = 1, \end{cases} \tag{2}$$

where p_m and p_0 are power of the mobile device for computation and being idle, respectively. We assume that $p_m > p_0$.

3.3 Communication Cost Model

Communication cost of tasks depends on their execution decisions. We assume that the communication cost is negligible if the two adjacent tasks are assigned to the same device. In other words, for the tasks v_i and v_j with $(v_i, v_j) \in A$, if $x_i = x_j$, then the data communication time between task i and task j is zero and thus, the resulted communication energy is also zero. We consider the communication time and communication energy on the mobile device for the task execution as follows.

First, we denote $T_{v_i,v_j}^{comm}(x_i, x_j)$ as the communication time on the mobile device for data communication between task i and task j, where v_j is a child node of v_i. Then, the communication time between task i and task j is given by,

$$T_{v_i,v_j}^{comm}(x_i, x_j) = \begin{cases} 0, & x_i = 0, x_j = 0, \\ \beta_{ij} R^{-1}, & x_i = 0, x_j = 1, \\ \beta_{ij} R^{-1}, & x_i = 1, x_j = 0, \\ 0, & x_i = 1, x_j = 1, \end{cases} \tag{3}$$

where β_{ij} is the output data size of task i to task j and R is the average rate of data transmission. We assume that we have the information of the average data rate over the wireless channel by profiling and then the communication time can be approximated by Eq. (3).

Second, we denote $E_{v_i,v_j}^{comm}(x_i, x_j)$ as the communication energy on the mobile device for data communication between task i and task j. Then, the communication energy between task i and task j is given by,

$$E_{v_i,v_j}^{comm}(x_i, x_j) = \begin{cases} 0, & x_i = 0, x_j = 0, \\ p_s \beta_{ij} R^{-1}, & x_i = 0, x_j = 1, \\ p_r \beta_{ij} R^{-1}, & x_i = 1, x_j = 0, \\ 0, & x_i = 1, x_j = 1, \end{cases} \tag{4}$$

where p_s and p_r are the power of the mobile device sending and receiving data, respectively.

For more complicated scenarios (e.g., ad-hoc virtual cloud with other mobile devices, and user interaction during the application execution), we can adjust the computation cost and the communication cost, i.e., T^{comp}, E^{comp}, T^{comm}, and E^{comm} accordingly and thus, our models are still valid.

3.4 Problem Formulation for Collaborative Task Execution

In this paper, we investigate how to minimize the energy consumption on the mobile device while completing all the tasks within the delay deadline, by determining the execution of each task within the application. Specifically, the energy consumption of the application execution is the sum of the weights of nodes and arcs in the graph G. The completion time of the application execution is when the last task in the graph (i.e., v_{n+1} in Figure 2) completes.

The collaborative task execution can be formulated as a constrained workflow scheduling problem. Mathematically, we have

$$\min_{X} \quad E(X) = \sum_{v_i \in V} E_{v_i}^{comp}(x_i) + \sum_{(v_i,v_j) \in A} E_{v_i,v_j}^{comm}(x_i, x_j) \tag{5}$$

$$\text{s.t.} \quad T(X) = T_{v_{n+1}}^{finish} \leq T_d, \tag{6}$$

$$x_i = 0, \forall v_i \in M, \tag{7}$$

$$x_j = 1, \forall v_j \in C, \tag{8}$$

where $T_{v_{n+1}}^{finish}$ is the finish time of task v_{n+1}, M denotes the set for the tasks that must be executed on the mobile device, and C denotes the set for the tasks that must be executed on the cloud clone. We aim to find a schedule X that minimizes the energy consumption on the mobile device (Eq. (5)) while meeting the delay deadline (Eq. (6)). Notice that some tasks may need to access the sensor on the mobile device or access the database in the cloud; thus, we include Eq. (7) and Eq. (8) as the constraints into the optimization problem, respectively. Specifically, Eq. (7) indicates that some tasks must be executed on the mobile device if $M \neq \emptyset$. Similarly, Eq. (8) indicates that some tasks must be executed on the cloud clone if $C \neq \emptyset$. We refer Eq. (7) and Eq. (8) as to *execution restriction*. The optimization problem with the execution restriction is a general case for the previous work [25] as a special case. Since the scheduling problem is NP-complete, we leverage heuristic algorithms, which is discussed in Section 4.

4 Design of Workflow Scheduling Algorithm

In this section, we adopt a two-step approach to design the workflow scheduling algorithm for the collaborative task

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2511727, IEEE Transactions on Cloud Computing

execution. First, we leverage partial critical path analysis to find the path that has the largest finish time. Second, we schedule the tasks on the partial critical path to achieve the minimum energy consumption. A summary of the design of the scheduling algorithm is given as follows:

- ***Partial Critical Path***: Partial Critical Path (PCP) in [19] was proposed as a heuristic algorithm to solve the scientific workflow scheduling problem in the cloud. A partial critical path of a task v consists of critical parent v' of the task v and the partial critical path of v'. In [19], the proposed approach using PCP did not consider the communication cost in the internal cloud and thus data transfer cost between cloud instances is zero. In contrast, for the collaborative task execution, the energy cost between the mobile device and the cloud cannot be ignored. Moreover, the corresponding energy of transmitting and receiving data can be different for the mobile device. As a result, the approach by [19] cannot be directly applied into our workflow scheduling for the collaborative task execution. In this paper, we adapt the partial critical path analysis with the consideration of the communication cost to design the workflow scheduling algorithm.
- ***Task Scheduling Strategy***: After finding a partial critical path, we schedule the tasks on the partial critical path for the collaborative task execution. We leverage *one-climb* policy [14] or LARAC algorithm [20] to obtain the execution decision for each task within the application. If there is no execution restriction (i.e., $M = \emptyset$ and $C = \emptyset$) in the optimization problem (we refer it as to special case), we adopt the *one-climb* policy for task scheduling, which indicates that there exists at most one migration from the mobile device to the cloud if ever. However, if there is execution restriction (we refer it as to general case), the *one-climb* policy may not be applicable and we adopt the LARAC algorithm to schedule the tasks for execution. More details will be elaborated in the following subsections.

4.1 Scheduling Metrics

We define two metrics for the partial critical path analysis. We denote T^{es} for each task as the earliest time for which it will start the execution, and T^{lf} as the latest time for which it will finish the execution.

We approximate the earliest start time T^{es} and the latest finish time T^{lf} as follows. Given different execution decisions on the tasks, computation time of the task execution (mobile device or cloud clone) and communication time between tasks are different. Thus, we cannot have the exact T^{es} and T^{lf}. In this case, we compute the T^{es} and T^{lf} by approximation for each unscheduled task [27]. For the approximation, suppose that all the tasks are offloaded to the cloud for execution, except the tasks that must be executed on the mobile device. Then, the earliest start time of each task is

$$\begin{aligned} T^{es}_{v_0} &= 0 \\ T^{es}_{v_i} &= \max_{v_j \in \mathcal{P}(v_i)} \{T^{es}_{v_j} + T^{comp}_{v_j} + T^{comm}_{v_j,v_i}(x_j, x_i)\}, \end{aligned} \quad (9)$$

where $\mathcal{P}(v_i)$ is the set of parent tasks of v_i. Similarly, the latest finish time of each task is

$$\begin{aligned} T^{lf}_{v_{n+1}} &= T_d \\ T^{lf}_{v_i} &= \min_{v_j \in \mathcal{C}(v_i)} \{T^{lf}_{v_j} - T^{comp}_{v_j} - T^{comm}_{v_i,v_j}(x_i, x_j)\}, \end{aligned} \quad (10)$$

where $\mathcal{C}(v_i)$ is the set of child tasks of v_i. The calculation of T^{es} and T^{lf} can help to find out the partial critical path before we can schedule the tasks and the sub-deadline for the task scheduling. In addition, T^{es} and T^{lf} will be updated after the task scheduling.

4.2 Framework of Scheduling Algorithm

We design the scheduling algorithm of the collaborative task execution in Algorithm 1. Given the task graph and the requirement of execution restriction, we aim to find the decision of task scheduling. First, we compute $T^{es}_{v_i}$ and $T^{lf}_{v_i}$ for each task by Eq. (9) and Eq. (10). Second, we set tasks v_0 and v_{n+1} as scheduled. It should be noted that when all the tasks are marked as scheduled, the scheduling algorithm terminates (as shown in Algorithm 2). Third, we call *scheduleParents* to schedule the tasks. We start from v_{n+1} to find its partial critical path and schedule the tasks on the path (as shown in Algorithm 3 for the special case and Algorithm 4 for the general case). The procedure *scheduleParents* recursively finds other PCPs for task scheduling, which will be detailed in Algorithm 2.

Algorithm 1 Scheduling algorithm of collaborative task execution

Input: $G(V, A)$, M, C
Output: Schedule X
1: Compute T^{es} and T^{lf} for each task in G by Eq. (9) and (10)
2: Set tasks v_0 and v_{n+1} as scheduled
3: Call *scheduleParents*(v_{n+1})

4.3 Sub Procedure for Finding Partial Critical Path

The partial critical path can be found recursively by Algorithm 2. In Algorithm 2, we find the partial critical path for a given a task v. In the outer while loop (line 2), Algorithm 2 terminates when all the tasks are scheduled. In the inner while loop (line 4), we find the critical parent of a particular task. Particularly, the critical parent of a task is its parent node that results in the maximum value of earliest start time T^{es},

$$v^*_j = \arg \max_{v_j \in \mathcal{P}(v_i)} \{T^{es}_{v_j} + T^{comp}_{v_j} + T^{comm}_{v_j,v_i}(x_j, x_i)\}. \quad (11)$$

By finding a set of critical parents in the inner while loop (line 4), we can constitute a partial critical path PCP. In line 3 and 10, we add two tasks that have been scheduled at the end and at the beginning of the PCP, respectively. There are two reasons for this addition. First, these two tasks have the T^{es} and T^{lf} updated, which can decide the sub-deadline for the scheduling of the PCP. Second, unlike [19], the cost of the arcs between the scheduled task and the unscheduled task (i.e., T^{comm} and E^{comm}) should also be taken into consideration.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2511727, IEEE Transactions on Cloud Computing

Figure 3 illustrates that a set of tasks are constituted to form the PCP for scheduling. The tasks in grey have been scheduled. The other tasks in between are to be scheduled, based on the procedure *schedulePath*. Specifically, Figure 3(a) represents a special case where there are no execution restriction and Figure 3(b) represents a general case where some tasks must be executed on the mobile device (the task with dots) and some tasks must be executed on the cloud clone (the task with grids). The scheduling of the unscheduled tasks on the PCP are determined by the procedure *schedulePath* (line 11). After scheduling, those tasks will be set to be scheduled. For more details, we will present Algorithm 3 and Algorithm 4 to implement the procedure *schedulePath* for the special case and general case, respectively. After that, the earliest start time T^{es} (line 12) and the latest finish time T^{lf} (line 13) of the tasks on the PCP are updated, which turn out to be actual start time and actual finish time of the task, respectively. The update of the earliest start time and the latest finish time are conducted as follows: the child of a task will have its T^{es} updated,

$$T^{es}_{v_i} = T^{es}_{v_j} + T^{comp}_{v_j}(x_j) + T^{comm}_{v_j,v_i}(x_j, x_i), \quad (12)$$

where v_j is a parent node of v_i in Eq. (12); and the parent of a task will have its T^{lf} updated,

$$T^{lf}_{v_i} = T^{lf}_{v_j} - T^{comp}_{v_j}(x_j) - T^{comm}_{v_i,v_j}(x_i, x_j), \quad (13)$$

where v_j is a child node of v_i in Eq. (13). After the update, we will find other PCPs in the graph recursively (line 15).

Algorithm 2 Procedure of *scheduleParents*

1: Procedure *scheduleParents*(v)
2: **while** v has an unscheduled parent **do**
3: Set $PCP = [v]$ and $u = v$
4: **while** there exists an unscheduled parent of u **do**
5: Find the critical parent of u that is unscheduled (denoted by w)
6: Add w into the beginning of PCP
7: Set $u = w$
8: **end while**
9: Find the critical parent of u that is scheduled (denoted by w')
10: Add w' into the beginning of PCP
11: Call *schedulePath(PCP)*
12: Update T^{es} of tasks on the PCP by Eq. (12)
13: Update T^{lf} of tasks on the PCP by Eq. (13)
14: **for** $i = 1$ to $|PCP|$ **do**
15: Call *scheduleParents*($PCP(i)$)
16: **end for**
17: **end while**
18: End Procedure

4.4 Sub Procedure for Task Scheduling on the Path

As mentioned, we consider two cases, i.e., special case and general case, for the task scheduling. We will adopt *one-climb* policy to schedule the tasks on the path for the special case and LARAC algorithm for the general case, respectively.

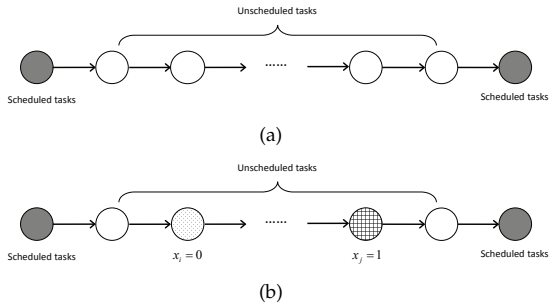


Fig. 3. An illustration of task scheduling on the path. (a) Special case: the tasks in shades have already been scheduled while the tasks without shades have no execution restriction and are to be scheduled based on the *one-climb* policy. (b) General case: the tasks with dots must be executed on the mobile device, the tasks with grids must be executed on the cloud clone, and the unscheduled tasks are to be scheduled based on the LARAC algorithm.

4.4.1 Special Case

As shown in Figure 3(a), the first task and the last task on the PCP have been scheduled, and we determine the execution decision of other tasks in between without execution restriction. We design Algorithm 3 based on *one-climb* policy to schedule the tasks on the PCP for the minimum energy consumption under the sub-deadline of the PCP. The sub-deadline of the PCP can be computed by

$$T_{sd} = T^{lf}_{u_m} - T^{es}_{u_1}, \quad (14)$$

where m is the number of tasks on the PCP, u_1 is the first task on the PCP and u_m is the last task on the PCP. We have the following scheduling strategy in Algorithm 3.

- If both the first task u_1 and the last task u_m are executed on the mobile device, then we select one from two options that result in less energy consumption under the subdeadline: either all the tasks in between are executed on the mobile device (line 8) or one migration occurs from mobile device to the cloud for application offloading (line 9).
- If the first task u_1 is executed on the mobile device and the last task u_m is executed on the cloud clone, then we need to find which task in between should be offloaded for execution in the first place (line 10-12).
- If the first task u_1 is executed on the cloud clone and the last task u_m is executed on the mobile device, then we need to find which task in between should be returned to the mobile device for execution in the first place (line 13-15).
- If both the first task u_1 and the last task u_m are executed on the cloud clone, then all the tasks in between should be executed on the cloud clone (line 17).

After the task scheduling on the PCP, we mark the tasks as scheduled and update the execution decision of the tasks.

4.4.2 General Case

As shown in Figure 3(b), there can be some tasks that must be executed either on the mobile device or on the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2511727, IEEE Transactions on Cloud Computing

Algorithm 3 Procedure of *schedulePath* under *one-climb* policy for Special Case

1: Procedure *schedulePath*(PCP)
2: Let m be the number of tasks on the PCP
3: Let y_1 be the execution decision of the first task u_1 on the PCP
4: Let y_m be the execution decision of the last task u_m on the PCP
5: Compute the sub-deadline T_{sd} by Eq. (14)
6: **if** $y_1 == 0$ and $y_m == 0$ **then**
7: Set y_2, ..., y_{m-1} by choosing one of the following options which results in less energy under the sub-deadline T_{sd}
8: 1) all the tasks are executed on the mobile device
9: 2) one migration occurs from the mobile device to the cloud clone
10: **else if** $y_1 == 0$ and $y_m == 1$ **then**
11: Determine task i that should be offloaded to the cloud clone in the first place such that the energy consumption is minimal under the sub-deadline T_{sd}
12: Set $y_2, ..., y_{i-1}$ to be 0, and $y_i, ..., y_{m-1}$ to be 1
13: **else if** $y_1 == 1$ and $y_m == 0$ **then**
14: Determine task i that should be returned back to the mobile device in the first place such that the energy consumption is minimal under the sub-deadline T_{sd}
15: Set $y_2, ..., y_{i-1}$ to be 1, and $y_i, ..., y_{m-1}$ to be 0
16: **else**
17: Set $y_i = 1$ for task i on the PCP, where $i = 2, ..., m-1$
18: **end if**
19: Set all the tasks on the PCP as scheduled
20: Update X
21: End Procedure

cloud clone. For this general case, we will leverage LARAC algorithm to determine the execution decision of other tasks in between without execution restriction. LARAC algorithm serves as an approximate algorithm to the task scheduling policy under time delay constraint, which has less complexity than the enumeration based on the *one-climb* policy [14].

We construct a graph to represent the collaborative task execution between the mobile device and the cloud clone on the PCP, as shown in Figure 4. We add two dummy nodes S and D to denote the initialization and termination of the task execution on the PCP. The node k represents that the kth task has been completed on the mobile device, while the node k' represents that the kth task has been completed by the cloud clone, where $k = 1, 2, ..., m$ (m is the number of tasks on the PCP). Notice that if task k must be executed on the mobile device, then we set the weight of arc connecting with k' to be ∞. Likewise, if task k must be executed on the cloud clone, then we set the weight of arc connecting with k to be ∞. Under this graph representation, we can find the shortest path between the source node S and the destination node D in terms of energy consumption and time delay. Notice that this graph representation can be extended for multiple cloud clones through adding more layers in the graph to denote the task execution on other cloud clones.

We define the following function for the task scheduling,

$$L(\lambda) = \min \{E_\lambda(p) | p \in P(S, D)\} - \lambda T_{sd}, \quad (15)$$

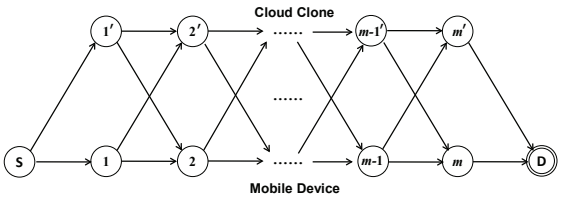


Fig. 4. The representation of the task execution flow. Node k represents that the kth task on the PCP has been completed on the mobile device, while node k' represents that the kth task on the PCP has been completed on the cloud clone. If task k must be executed on the mobile device, then we set the weight of arc connecting with k' to be ∞. If task k must be executed on the cloud clone, then we set the weight of arc connecting with k to be ∞.

where p is a path in the set of $P(S, D)$ from S to D on the PCP, λ is the Lagrangian multiplier, and $E_\lambda(p)$ is the aggregated cost as a combination of energy consumption and time delay. The aggregated cost $E_\lambda(p)$ is defined as

$$E_\lambda(p) = E(p) + \lambda T(p), \quad (16)$$

where $E(p)$ and $T(p)$ denote the energy consumption and time delay of the path p, respectively.

Then, we can design the algorithm in Algorithm 4 for the general case. Algorithm 4 first finds a path with the minimum energy consumption between the source node S and the destination node D by calling the function *minimumEnergyPath* (line 4). If this path can have a time delay less than the subdeadline (line 5), we can set the task execution of the PCP accordingly (line 6). Otherwise, we will find a path with the minimum aggregated cost by calling the function *minimumCostPath* between the source node S and the destination node D (line 12) with the Lagrangian multiplier λ (line 11) updated in an iteration. Notice that the functions *minimumEnergyPath*, *minimumDelayPath* and *minimumCostPath* in Algorithm 4 can be implemented under backward iteration [14]. The while loop (line 10) terminates when it finds the optimal Lagrangian multiplier λ [20] and the corresponding task scheduling (line 14). After that, we can set all the tasks on the PCP as scheduled (line 25).

It is likely that there is no solution that can satisfy the sub-deadline of the partial critical path by LARAC algorithm and *one-climb* policy. This could happen because we use the greedy algorithm to minimize the energy consumption on the partial critical path by scheduling the task execution, which would violate the delay constraint of other paths in the graph. In this case, we need to use backtracking and redistribute the sub-deadline for each path. As an alternative, we can simply schedule the tasks with the largest execution time of the PCP to satisfy the sub-deadline.

4.5 Complexity Analysis

The complexity analysis of the workflow scheduling algorithm is given as follows. There are $|V| = n + 2$ nodes in the graph including dummy nodes. Then, there are at most $|A| = \frac{n(n-1)}{2} + 2n$ arcs for the data dependency. The initialization of the earliest start time and the latest finish time is to process the nodes and arcs, which results in the complexity of $O(|V|+|A|) = O(n^2)$. Then, the procedure *scheduleParents*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2511727, IEEE Transactions on Cloud Computing

Algorithm 4 Procedure of *schedulePath* under LARAC Algorithm for General Case

```
Procedure schedulePath(PCP)
Let m be the number of tasks on the PCP
Compute the sub-deadline T_sd by Eq. (14)
Find the path with the minimum energy consumption from S to D, p_E = minimumEnergyPath(S, D) in Figure 4
if T(p_E) ≤ T_sd then
    Set x on the PCP based on p_E
else
    Find the path with the minimum time delay from S to D, p_T = minimumDelayPath(S, D) in Figure 4
    Set flag = 0
    while flag == 0 do
        λ = (E(p_E) − E(p_T)) / (T(p_T) − T(p_E))
        Find the path with the minimum aggregated cost from S to D, p_λ = minimumCostPath(S, D) in Figure 4
        if E_λ(p_λ) = E_λ(p_E) then
            Set x on the PCP based on p_T
            Set flag = 1
        else
            if T(p_λ) ≤ T_sd then
                p_T = p_λ
            else
                p_E = p_λ
            end if
        end if
    end while
end if
Set all the tasks on the PCP as scheduled
Update X
End Procedure
```

in Algorithm 2 is called, with the complexity of $O(n^2)$ to constitute the PCP. After that, the procedure of *schedulePath* is called to schedule the tasks on the PCP. For the special case, the complexity of Algorithm 3 is $O(n^3)$ [25]. For the general case, the complexity of Algorithm 4 is $O(n^2 log^2 n)$ [14]. Finally, the update of the earliest start time and the latest finish time is $O(|V| + |A|) = O(n^2)$. Therefore, the overall complexity of the workflow scheduling algorithm is $O(n^3)$ under the one-climb policy and $O(n^2 log^2 n)$ under the LARAC algorithm, respectively.

5 Performance Evaluation

In this section, we present simulation results of the collaborative task execution for the special case and the general case.

5.1 Application Profile

Table 1 illustrates the parameters of the mobile device and the cloud clone in system models mentioned in Section 3.

As an example, we consider the application that consists of 11 tasks. Figure 5 shows three task topologies,

- Mesh: a set of linear chains;
- Tree: a tree-based structure;
- General: a combination of the mesh and the tree.

We add two dummy nodes v_0 and v_{12} as task 0 and task 12, respectively. The workload for these two nodes are 0. In addition, the value of the arc between the nodes denotes the amount of data in kb units for transmission if the two corresponding tasks are executed on different devices.

Based on these task topologies, we investigate three cases under different execution restrictions as follows:

- Case 1: no execution restriction;
- Case 2: some tasks must be executed on the mobile device;
- Case 3: some tasks must be executed on the mobile device while some are on the cloud clone.

For case 1, we will leverage *one-climb* policy for the task execution. For case 2 and 3, we will leverage LARAC algorithm for the task execution. As an example, for case 2, suppose $M = \{v_5, v_{11}\}$, then we set $x_5 = 0$ and $x_{11} = 0$ to denote that task 5 and task 11 must be executed on the mobile device; for case 3, suppose $M = \{v_5\}$ and $C = \{v_{11}\}$, then we set $x_5 = 0$ and $x_{11} = 1$ to denote that task 5 must be executed on the mobile device and task 11 must be executed on the cloud clone. Our approach is not limited to these specific cases.

TABLE 1
Parameters of simulation

Data transmission power of the mobile device	$p_s = 0.1W$
Data receiving power of the mobile device	$p_r = 0.05W$
Computation power of the mobile device	$p_m = 0.5W$
Idle power of the mobile device	$p_0 = 0.001W$
CPU frequency of the mobile device	$f_m = 500MHz$
CPU frequency of the cloud clone	$f_c = 3000MHz$

5.2 Simulation Results

We present the solution of the proposed algorithm in comparison with of the optimal solution, analyze the effect of the execution restriction on the execution decision, and compare the collaborative task execution with the local execution and the remote execution.

5.2.1 Comparison with the Optimal Solution

We compare the solution of the proposed workflow scheduling algorithm with the optimal solution using brute-force enumeration for different cases. (1) For case 1 under the *one-climb* policy, we observe that with the low data rate (e.g., $R = 10kb/s$) and high data rate (e.g., $R = 100kb/s$), our proposed algorithm achieves the same energy consumption as the brute-force enumeration method for the topologies shown in Figure 5. We thus only plot the energy ratio between the proposed workflow algorithm and the brute-force enumeration with $R = 50kb/s$. Figure 6(a) shows that the ratio is 1 for the general topology, indicating that the proposed workflow scheduling algorithm achieves the optimal solution. However, the ratio is greater than 1 for mesh and tree topology given some delay deadlines with $R = 50kb/s$. This is because, the proposed algorithm greedily finds the optimal energy consumption under the sub-deadline on the PCP but it only achieves the local optimum, which does not

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2511727, IEEE Transactions on Cloud Computing

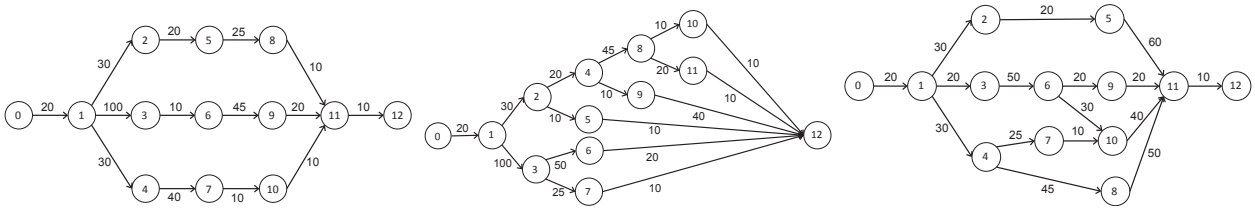


Fig. 5. Task topologies: (a) mesh; (b) tree; (c) general topology. There are 13 nodes in each task topology. v_0 and v_{12} are added into the task topologies as task 0 and task 12, respectively. The workload for each task is $\omega = [0, 10, 25, 2, 12, 15, 67, 54, 24, 50, 9, 20, 0]$ M cycles. The value of the arc between the adjacent nodes refers to the data communication.

guarantee for the global optimum of the overall workflow scheduling. (2) For case 2 under the LARAC algorithm, we also plot the energy ratio between the proposed workflow algorithm and the optimal solution with $R = 50kb/s$ in Figure 6(b). The feasible set of the optimization problem is smaller due to the execution decision, and the energy ratio is much smaller. (3) Moreover, for case 3, we also plot the energy ratio between the proposed workflow algorithm and the optimal solution in Figure 6(c). Since there is no solution for the general topology with $R = 50kb/s$, we present the results with $R = 100kb/s$. Also notice that there are no solutions for tree and general topology under some delays. These results in Figure 6 indicate that the proposed algorithm can achieve a good solution compared with the optimal solution.

5.2.2 Characterization of the Execution Decision

We present the process of finding partial critical path and its execution decision in Table 2. Specifically, we investigate the execution decision under general topology with $R = 100kb/s$ and $T_d = 1.0s$ for the three specific cases, respectively. As shown in Table 2, since each case has different execution restrictions, the resulted PCPs are different, leading to different execution decisions.

We characterize the solution of the workflow scheduling for collaborative task execution under different delay deadlines. Table 3 shows the execution decision of each task for the topologies under the delay deadline 0.6s, 0.8s, 1.0s and 1.3s for $R = 50kb/s$ without execution restriction (i.e., case 1). We observe that with the increase of the deadline T_d, there are more tasks to be offloaded to the cloud for execution. In addition, given the same delay deadline, different task topologies would have different task execution decisions. The results for the collaborative execution with execution restriction (e.g., case 2 and case 3) can also be similarly obtained.

5.2.3 Comparison with Local Execution and Remote Execution

We make a comparison with two alternative execution modes: local execution and remote execution. For the local execution, all the tasks are executed on the mobile device, assuming that there is no execution restriction for the tasks to be executed on the cloud clone. For the remote execution, all the tasks are executed on the cloud clone, assuming that there is no execution restriction for the tasks to be executed on the mobile device. We plot the energy ratio between

TABLE 2
Execution decision under general topology for three cases. The number with an underline denotes that the task must be executed on the mobile device. The number with an overline denotes that the task must be executed on the cloud clone. $R = 100kb/s$. $T_d = 1.0s$

(a) Case 1: no execution restriction ($M = \emptyset$ and $C = \emptyset$)

PCP	Decision
$\underline{0} \to 1 \to 3 \to 6 \to 9 \to 11 \to \underline{12}$	$x_1 = x_3 = x_6 = x_9 = x_{11} = 1$
$1 \to 4 \to 7 \to 10 \to 11$	$x_4 = x_7 = x_{10} = 1$
$1 \to 2 \to 5 \to 11$	$x_2 = x_5 = 1$
$4 \to 8 \to 11$	$x_8 = 1$

(b) Case 2: task 5 and task 11 must be executed on the mobile device ($M = \{v_5, v_{11}\}$ with $x_5 = 0$ and $x_{11} = 0$)

PCP	Decision
$\underline{0} \to 1 \to 4 \to 8 \to \underline{11} \to \underline{12}$	$x_1 = 1, x_4 = 1, x_8 = 1$
$1 \to 3 \to 6 \to 10 \to \underline{11}$	$x_3 = 1, x_6 = 1, x_{10} = 0$
$4 \to 7 \to 10$	$x_7 = 1$
$6 \to 9 \to \underline{11}$	$x_9 = 1$
$1 \to 2 \to \underline{5} \to \underline{11}$	$x_2 = 1$

(c) Case 3: task 5 must be executed on the mobile device and task 11 must be executed on the cloud clone ($M = \{v_5\}$ and $C = \{v_{11}\}$ with $x_5 = 0$ and $x_{11} = 1$)

PCP	Decision
$\underline{0} \to 1 \to 2 \to \underline{5} \to \overline{11} \to \underline{12}$	$x_1 = 0, x_2 = 0$
$1 \to 3 \to 6 \to 9 \to \overline{11}$	$x_3 = 1, x_6 = 1, x_9 = 1$
$1 \to 4 \to 7 \to 10 \to \overline{11}$	$x_4 = 1, x_7 = 1, x_{10} = 1$
$4 \to 8 \to \overline{11}$	$x_8 = 1$

the collaborative task execution and the local execution for the special case without execution restriction (i.e., case 1) in Figure 7. It shows that when the data rate is low (e.g., $R = 10kb/s$), the collaborative execution becomes the local execution and the energy ratio is thus 1 for all the three topologies. This is because, it takes time to transmit the data and consumes a large amount of energy if a task is offloaded to the cloud for execution. Under the delay deadline, tasks are forced to be executed on the mobile device. When the data rate is medium (e.g., $R = 50kb/s$), the collaborative task execution could consume less energy, with the ratio about 0.7, 0.5, 0.2 with respect to the local execution for the mesh ($T_d = 0.6s$), tree ($T_d = 1.3s$) and general topology ($T_d = 0.7s$), respectively. When the data rate is high (e.g., $R = 100kb/s$), the energy consumption by the collaborative task execution is about 0.1 times of that by the local execution for the mesh and the general topology, and about 0.9 to 0.25 times for the tree topology. As shown in Figure 7(a) and 7(c), the collaborative task execution becomes the remote execution for both the mesh

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2511727, IEEE Transactions on Cloud Computing

TABLE 3
Decision of collaborative task execution under different topologies and delay deadlines without execution restriction (case 1). $R = 50kb/s$.

Mesh	Task 0	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8	Task 9	Task 10	Task 11	Task 12
$T_d = 0.6s$	0	0	0	0	0	0	1	0	0	1	1	1	0
$T_d = 0.8s$	0	0	0	0	0	1	1	0	1	1	1	1	0
$T_d = 1.0s$	0	0	1	0	1	1	1	1	1	1	1	1	0
$T_d = 1.3s$	0	0	1	0	1	1	1	1	1	1	1	1	0
Tree	Task 0	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8	Task 9	Task 10	Task 11	Task 12
$T_d = 0.6s$	0	0	0	0	0	0	0	0	0	0	0	0	0
$T_d = 0.8s$	0	0	0	0	0	0	0	0	0	0	0	0	0
$T_d = 1.0s$	0	0	0	0	0	0	0	0	0	0	0	0	0
$T_d = 1.3s$	0	1	1	1	1	1	1	1	1	1	1	1	0
General	Task 0	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8	Task 9	Task 10	Task 11	Task 12
$T_d = 0.6s$	0	0	0	0	0	0	0	0	0	0	0	0	0
$T_d = 0.8s$	0	1	1	1	1	1	1	1	1	1	1	1	0
$T_d = 1.0s$	0	1	1	1	1	1	1	1	1	1	1	1	0
$T_d = 1.3s$	0	1	1	1	1	1	1	1	1	1	1	1	0

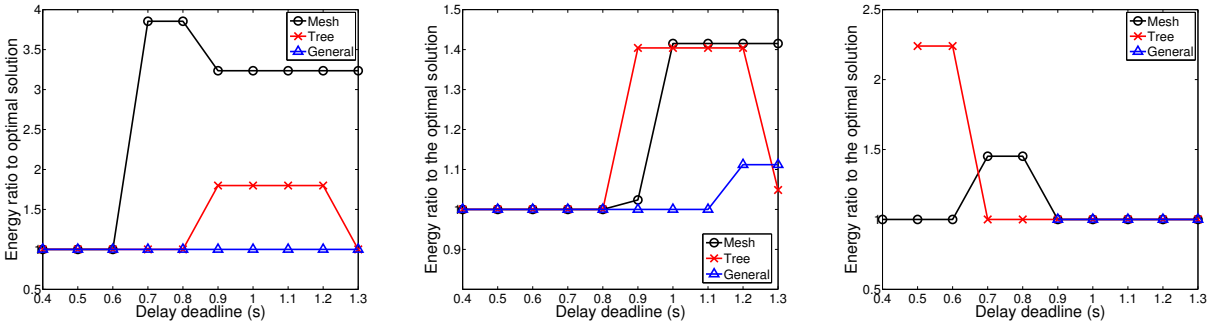


Fig. 6. Energy ratio between the proposed workflow scheduling algorithm and the brute-force enumeration. (a) Case 1 as a special case under *one-climb* policy with $R = 50kb/s$. (b) Case 2 as a general case under LARAC algorithm with $R = 50kb/s$. (c) Case 3 as a general case under LARAC algorithm with $R = 100kb/s$.

and the general topology. For the tree topology, however, as shown in Figure 7(b), under the deadline 0.6s, almost all the tasks are executed on the cloud except task 9. Since task 9 has a large amount of data for the output (i.e., the arc between node 9 and 12 in Figure 5(b)), it is executed on the mobile device to avoid the data transmission to meet the delay deadline. But as the application deadline increases, all the tasks are offloaded such that more energy consumption can be saved on the mobile device.

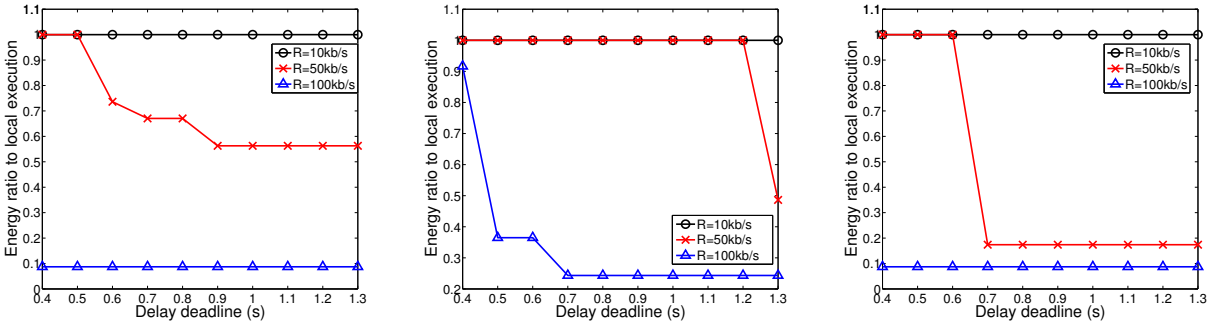
We also plot the energy ratio between the collaborative task execution and local execution for the general case with execution restriction in Figure 8. We have the same simulation settings except that the task 5 and task 11 must be executed on the mobile device (i.e., case 2 with $x_5 = 0$ and $x_{11} = 0$). Compared to Figure 7, we observe that the energy ratio with the execution restriction is higher than that without the execution restriction. This indicates that it consumes more energy consumption on the mobile device for this case. We do not plot the energy ratio for the case 3 with $x_5 = 0$ and $x_{11} = 1$, since it is not comparable with the local execution.

Finally, we observe that the collaborative task execution is more flexible than the remote execution. As an example, we investigate the three topologies under case 1 that has no execution restriction. For the tree topology, when the data rate is $10kb/s$ and $50kb/s$, the execution time of the remote execution is $6.0323s$ and $1.2323s$, respectively. Hence, there are no solutions using remote execution under the delay deadline from $0.6s$ to $1.2s$. To meet the delay deadline, some tasks should be executed on the mobile device. When the data rate is $100kb/s$, the execution time is $0.6323s$; thus, the remote execution cannot meet the delay deadline $0.6s$. Similarly, for the mesh topology and the general topology, the execution time of the remote execution is $0.6497s$ when the data rate is $50kb/s$. As a result, there is no solution for the remote execution under the delay deadline $0.6s$. The collaborative task execution, due to its flexibility of task scheduling, can address the issue of remote execution violating the delay deadline. Therefore, the collaborative task execution is more effective in reducing the energy consumption while meeting the delay deadline.

6 Conclusion

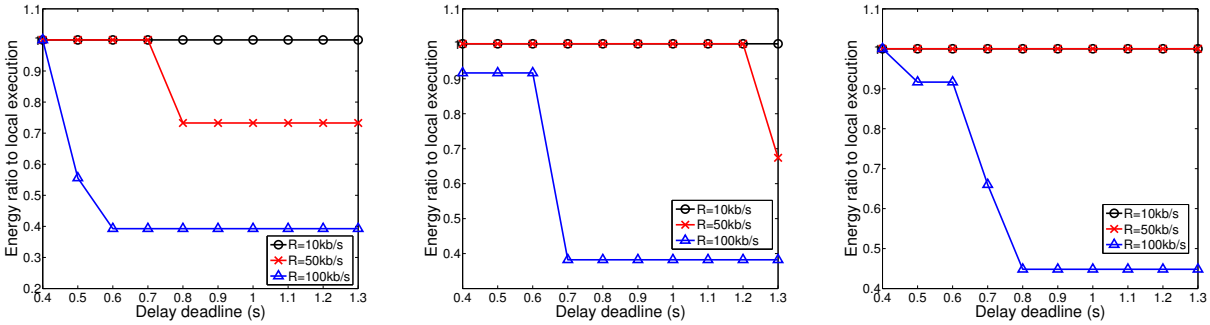
In this paper, we proposed the collaborative task execution for the application as a general topology in mobile cloud computing. Each task within the general topology is either executed on the mobile device or on the cloud clone. The objective was to minimize the energy consumption on the mobile device while meeting the delay deadline under the collaborative task execution. We formulated the collaborative task execution as a delay-constrained workflow scheduling problem. Based on the partial critical path

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2511727, IEEE Transactions on Cloud Computing



(a) Mesh. (b) Tree. (c) General topology.

Fig. 7. Case 1 as a special case: energy ratio between the collaborative execution and the local execution under different delay deadlines and data rates for mesh, tree and general topology using workflow scheduling.



(a) Mesh. (b) Tree. (c) General topology.

Fig. 8. Case 2 as a general case: energy ratio between the collaborative execution and the local execution under different delay deadlines and data rates for mesh, tree and general topology using workflow scheduling.

analysis, we scheduled the tasks on the partial critical path. Specifically, we provided a general solution using LARAC algorithm to solve the general case with the consideration of execution restriction under which some tasks must be executed either on the mobile device or on the cloud clone. We also provided a solution using *one-climb* policy to solve the special case without the consideration of the execution restriction. Performance evaluation showed that the proposed workflow scheduling algorithm can obtain a reasonable solution compared with the brute-force enumeration. In addition, the collaborative task execution is more energy-efficient and flexible than the local execution and the remote execution.

In the future, we will consider the collaborative task execution among ad-hoc mobile devices. In addition, we will build a cloud-based mobile application platform and evaluate the performance of our proposed collaborative task execution for real mobile applications.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their insightful comments. This work was supported by Start-Up Grant from NTU, MOE Tier-1 Grant (RG 17/14) from Singapore MOE and EIRP02 Grant from Singapore EMA, a research grant from Microsoft Research Asia and a research grant from Cisco Systems, Inc.

REFERENCES

[1] *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update 2014 - 2019 White Paper*, Cisco, 2015.
[2] "How many apps are in the iphone app store?" last accessed on May 4, 2015. [Online]. Available: http://ipod.about.com/od/iphonesoftwareterms/qt/apps-in-app-store.htm
[3] "Number of android applications," last accessed on May 4, 2015. [Online]. Available: http://www.appbrain.com/stats/number-of-android-apps
[4] K. Kumar and Y. H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *IEEE Computer*, vol. 43, no. 4, pp. 51–56, 2010.
[5] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587–1611, 2013.
[6] Y. Cui, X. Ma, H. Wang, I. Stojmenovic, and J. Liu, "A survey of energy efficient wireless transmission and modeling in mobile cloud computing," *Mobile Networks and Applications*, vol. 18, no. 1, pp. 148–155, 2013.
[7] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: Making smartphones last longer with code offload," in *International conference on Mobile systems, applications, and services*, 2010, pp. 49–62.
[8] B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proceedings of the 6th European Conference on Computer Systems*, 2011, pp. 301–314.

[9] M. Satyanarayanan, R. C. P. Bahl, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.
[10] X. W. Zhang, A. Kunjithapatham, S. Jeong, and S. Gibbs, "Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing," *Mobile Networks and Applications*, vol. 16, no. 3, pp. 270–284, 2011.
[11] G. Huerta-Canepa and D. Lee, "A virtual cloud computing provider for mobile devices," in *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, 2010, p. 6.
[12] N. Fernando, S. W. Loke, and W. Rahayu, "Dynamic mobile cloud computing: Ad hoc and opportunistic job sharing," in *Proceedings of the Fourth IEEE International Conference on Utility and Cloud Computing*, 2011, pp. 281–286.
[13] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. O. Wu, "Energy-optimal mobile cloud computing under stochastic wireless channel," *IEEE Transactions on Wireless Communications*, vol. 12, no. 9, pp. 4569–4581, 2013.
[14] W. Zhang, Y. Wen, and D. O. Wu, "Collaborative task execution in mobile cloud computing under stochastic wireless channel," *IEEE Transactions on Wireless Communications*, vol. 14, no. 1, pp. 81–93, 2015.
[15] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, and A. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," *ACM Sigmetrics Performance Evaluation Review*, vol. 40, no. 4, pp. 23–32, 2013.
[16] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso, "Calling the cloud: enabling mobile phones as interfaces to cloud applications," in *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, 2009, pp. 83–102.
[17] W. Zhang, Y. Wen, J. Wu, and H. Li, "Toward a unified elastic computing platform for smartphones with cloud support," *IEEE Network*, pp. 34–40, 2013.
[18] W. Zhang, Y. Wen, and X. Zhang, "Towards virus scanning as a service in mobile cloud computing: Energy-efficient dispatching policy under n-version protection," *IEEE Transactions on Emerging Topics in Computing (In Press)*, 2015.
[19] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013.
[20] A. Juttner, B. Szviatovski, I. Mécs, and Z. Rajkó, "Lagrange relaxation based method for the QoS routing problem," in *Proceedings of IEEE International Conference on Computer Communications*, vol. 2, 2001, pp. 859–868.
[21] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing," in *Proceedings of the 2nd USENIX conference on hot topics in cloud computing*, 2010, pp. 4–11.
[22] Z. Li, C. Wang, and R. Xu, "Computation offloading to save energy on handheld devices: A partition scheme," in *Proceedings of the international conference on Compilers, architecture, and synthesis for embedded systems*, 2001, pp. 238–246.
[23] C. Wang and Z. Li, "Parametric analysis for adaptive computation offloading," *ACM SIGPLAN Notices*, vol. 39, no. 6, pp. 119–130, 2004.
[24] R. Newton, S. Toledo, L. Girod, H. Balakrishnan, and S. Madden, "Wishbone: Profile-based partitioning for sensornet applications," in *NSDI*, vol. 9, 2009, pp. 395–408.
[25] W. Zhang and Y. Wen, "Cloud-assisted collaborative execution for mobile applications with general task topology," in *2015 IEEE International Conference on Communication*, 2015, pp. 6815 – 6821.
[26] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: enabling interactive perception applications on mobile devices," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*. ACM, 2011, pp. 43–56.
[27] S. Abrishami and M. Naghibzadeh, "Deadline-constrained workflow scheduling in software as a service cloud," *Scientia Iranica*, vol. 19, no. 3, pp. 680–689, 2012.

Weiwen Zhang received his Ph.D. degree in computer engineering from Nanyang Technological University (NTU), Singapore in 2015; his Master degree in computer science and his Bachelor degree in software engineering from South China University of Technology (SCUT) in 2011 and 2008, respectively. From January 2014 to June 2014, he was a visiting scholar in Purdue university. He was also a research associate in NTU from November 2014 to June 2015. Currently, he is a scientist at Computing Science Department, Institute of High Performance Computing (IHPC), Agency for Science, Technology and Research (A*STAR), Singapore. His research interests include cloud computing, mobile computing, green data center and multimedia.

Yonggang Wen (S99-M08-SM14) received his PhD degree in Electrical Engineering and Computer Science (minor in Western Literature) from Massachusetts Institute of Technology (MIT), Cambridge, USA. Currently, Dr. Wen is an assistant professor with school of computer engineering at Nanyang Technological University, Singapore. Previously he has worked in Cisco to lead product development in content delivery network, which had a revenue impact of 3 Billion US dollars globally. Dr. Wen has published over 130 papers in top journals and prestigious conferences. His work in Multi-Screen Cloud Social TV has been featured by global media (more than 1600 news articles from over 29 countries) and received ASEAN ICT Award 2013 (Gold Medal). His work on Cloud3DView, as the only academia entry, has won the Data Centre Dynamics Awards 2015 APAC. He is a co-recipient of 2015 IEEE Multimedia Best Paper Award, and a co-recipient of Best Paper Awards at EAI/ICST Chinacom 2015, IEEE WCSP 2014, IEEE Globecom 2013 and IEEE EUC 2012. He serves on editorial boards for IEEE Wireless Communications, IEEE Communications Survey & Tutorials, IEEE Transactions on Multimedia, IEEE Transactions on Signal and Information Processing over Networks, IEEE Access Journal and Elsevier Ad Hoc Networks, and was elected as the Chair for IEEE ComSoc Multimedia Communication Technical Committee (2014-2016). His research interests include cloud computing, green data center, big data analytics, multimedia network and mobile computing.