Energy and Performance-Aware Task Scheduling in a Mobile Cloud Computing Environment

Xue Lin, Yanzhi Wang, Qing Xie, Massoud Pedram
Department of Electrical Engineering
University of Southern California
Los Angeles, U.S.
{xuelin, yanzhiwa, xqing, pedram}@usc.edu

***Abstract*—Mobile cloud computing (MCC) offers significant opportunities in performance enhancement and energy saving in mobile, battery-powered devices. An application running on a mobile device can be represented by a task graph. This work investigates the problem of scheduling tasks (which belong to the same or possibly different applications) in an MCC environment. More precisely, the scheduling problem involves the following steps: (i) determining the tasks to be offloaded on to the cloud, (ii) mapping the remaining tasks onto (potentially heterogeneous) cores in the mobile device, and (iii) scheduling all tasks on the cores (for in-house tasks) or the wireless communication channels (for offloaded tasks) such that the task-precedence requirements and the application completion time constraint are satisfied while the total energy dissipation in the mobile device is minimized. A novel algorithm is presented, which starts from a minimal-delay scheduling solution and subsequently performs energy reduction by migrating tasks among the local cores or between the local cores and the cloud. A linear-time rescheduling algorithm is proposed for the task migration. Simulation results show that the proposed algorithm can achieve a maximum energy reduction by a factor of 3.1 compared with the baseline algorithm.**

Keywords-mobile cloud computing (MCC); energy minimization; hard deadline constraint; task scheduling

I. INTRODUCTION

Mobile devices e.g., smart-phones and tablet-PCs, have become one of the major computing platforms nowadays. Unfortunately, the increase in the volumetric/gravimetric energy density of rechargeable batteries has been much slower than the increase in the power demand of these devices (which are equipped with increasing levels of advanced functionality), thus, resulting in a short battery life in mobile devices and a "power crisis" for the smart-phone technology development. At the same time, mobile devices have relatively weak computing resources compared to their "wall-powered" counterparts due to the constraints of weight, size and power.

Cloud computing has been envisioned as the next-generation computing paradigm because of the benefits that it offers, including on-demand service, ubiquitous network access, location independent resource pooling, and transference of risk [1]. In the cloud computing paradigm, a service provider owns and manages the computing and storage resources, and users have access to these resources over the Internet. With the help of wireless communication elements such as 3G, Wi-Fi, and 4G, a newly emerging mobile cloud computing (MCC) paradigm can shift the processing, memory, and storage requirements from the resource-limited mobile devices to the resource-unlimited cloud computing system [2][3][4].

MCC has the potential of improving the performance of mobile devices by (i) selectively offloading tasks of an application (e.g., object/gesture recognition, image/video editing, and natural language processing) on to the cloud and (ii) carefully scheduling task executions on both the mobile device and the cloud taking into account the task-precedence requirements. This is mainly because servers in the cloud have much larger computation capability and higher speed than the mobile processor. Moreover, MCC helps save energy in mobile devices and prolong the battery operation time by offloading executions of computation-intensive tasks onto the cloud. Experiments conducted in [5] demonstrate that (i) a large application can be partitioned into various tasks with task-precedence requirements, and (ii) the fine granularity of task-level offloading can potentially achieve both energy saving and performance improvement.

Task scheduling on limited computing resources and task offloading on to the cloud have been extensively studied and various heuristic algorithms proposed in [6]~[12]. These works are classified into two categories: (i) minimizing the overall application completion time (achieving higher performance) [6][7][8] and (ii) minimizing the total energy consumption (achieving longer battery life in battery-powered mobile devices) [9][10][11][12]. The HEFT algorithm in [6] was proposed for scheduling tasks of an application with task-precedence requirements on heterogeneous processors with the objective of achieving high performance. This algorithm computes priorities of all tasks, selects a task with the highest priority value at each step, and assigns the selected task to the processor that minimizes the task's finish time. Ra et al. [7] adopted an incremental greedy strategy and developed a runtime system which is able to adaptively make offloading and parallel execution decisions for mobile interactive perceptual applications in order to minimize the completion time of applications. A genetic algorithm was proposed in [8] to optimize the partitioning of tasks of a data stream application between a mobile device and the cloud for the maximum throughput.

Reference [9] addressed the problem of minimizing energy consumption of a computer system performing periodic tasks, assuming that the periods of tasks are large enough such that

978-1-4799-5063-8/14 $31.00 © 2014 IEEE
DOI 10.1109/CLOUD.2014.35

the positive slack time between tasks can be used for energy consumption reduction. Reference [10] formulated the task mapping problem as a maximum flow/minimum-cut problem to optimize the partition of a task graph between a mobile device and the cloud for the minimum energy consumption. However, the authors did not consider the overall application completion time and lacked a scheduling policy. Reference [11] extended the work of [6] on heterogeneous processers accounting for both the energy consumption and application completion time. However, the algorithm in [11] cannot guarantee that the scheduling result meets a hard constraint of application completion time. Kumar and Lu proposed a straightforward offloading decision strategy to minimize energy consumption according to the *computation-to-communication ratio* and the networking environment [12], whereas a practical scheduling algorithm was omitted.

Although MCC brings great benefits in performance and energy optimization for the mobile devices, it also gives rise to significant challenge in terms of designing an optimal policy to (i) determine the tasks of an application to be offloaded, (ii) map the remaining tasks onto potentially heterogeneous cores in a mobile device, and (iii) schedule tasks on the heterogeneous cores (for in-house processing) and wireless communication channels (for remote processing) such that the task-precedence requirements and application completion time constraint are satisfied with the minimum energy consumption on the mobile device side. Notice that although the mobile device cannot directly schedule tasks inside the cloud (this is the job of the cloud computing controller), it can anticipate and estimate the execution time of every task that has been offloaded to the cloud based on its prior knowledge.

To differentiate the aforementioned problem from those addressed in the previous work, we refer to it as the *MCC task scheduling* problem. In particular, there are three key issues that must be addressed.

- The application completion time constraint is a hard constraint and therefore, it should be addressed in the first place. Offloading computation-intensive tasks on to the cloud may result in a decrease in the application completion time. However, the offloading decision should be made judiciously considering the delay due to uploading/downloading data to/from the cloud.
- The total energy consumption in mobile devices, including both the energy consumed by the processing units (the potentially heterogeneous cores in the mobile device) and by the RF components for offloading tasks is the objective function to be minimized. From the perspective of energy consumption, offloading tasks to the cloud saves the computation energy but it induces energy consumption in the communication units.
- The task-precedence requirements should be enforced during the task scheduling. Unlike the conventional local task scheduling problem in [6], there exist additional task-precedence requirements between the cloud and the local cores through wireless communication channels.

In this work, we present a novel algorithm to address the MCC task scheduling problem to minimize the total energy consumption of an application in a mobile device with access to the computing resources on the cloud under a hard application completion time constraint. In particular, we generate a minimal-delay task scheduling in the first step, and after that we perform energy reduction in the second step by migrating tasks towards the cloud or other local cores that can bring great energy reduction without violation of the application completion time constraint. To avoid high time complexity, we propose a linear-time rescheduling algorithm for the task migrations. The simulation results show that the proposed algorithm can achieve a maximum energy reduction by a factor of 3.1 compared with the baseline algorithm.

To our best knowledge, this is the first task scheduling work that minimizes energy consumption under a hard completion time constraint for the task graph in the MCC environment, taking into account the joint task scheduling on the local cores and the wireless communication channels of the mobile devices as well as on the cloud.

II. MCC Task Scheduling System Model

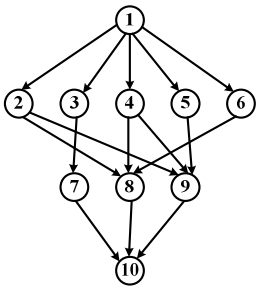
A. Applications

An application is represented by a directed acyclic *task graph* $G = (V, E)$. Each node $v_i \in V$ represents a task and a directed edge $e(v_i, v_j) \in E$ represents the precedence constraint such that task (node) v_i should complete its execution before task (node) v_j starts execution. There are a total number of N tasks (nodes) in the task graph. Given a task graph, the task without any parent is called the *entry task*, and the task without any child is called the *exit task*. As shown in Fig. 1, task v_1 is the entry task and task v_{10} is the exit task. For each task v_i, we define $data_i$ and $data'_i$ as the amount of task specification and input data required to upload to the cloud and the amount of data required to download from the cloud, if the execution of task v_i is offloaded onto the cloud.

B. MCC Environment

We consider a mobile device in the MCC environment that has access to the computing resources on the cloud. There are a number of K heterogeneous cores in the processor of the mobile device. An example is the state-of-the-art big.LITTLE architecture [13] that is adopted by Broadcom, Samsung, etc. The operating frequency of the k-th core is f_k and the (average) power consumption P_k is a super-linear function of f_k, represented by $P_k = \alpha_k \cdot (f_k)^{\gamma_k}$, where $2 \le \gamma_k \le 3$. The α_k and γ_k values may be different for different cores.

A task can be executed either locally on a core of the mobile device or remotely on the cloud. If task v_i is offloaded



Task	Core1	Core2	Core3
1	9	7	5
2	8	6	5
3	6	5	4
4	7	5	3
5	5	4	2
6	7	6	4
7	8	5	3
8	6	4	2
9	5	3	2
10	7	4	2

$$1 \le i \le N, \begin{cases} T_i^s = 3 \\ T_i^c = 1 \\ T_i^r = 1 \end{cases}$$

Figure 1. An example task graph.

on to the cloud, there are three phases in sequence associated with the execution of the task v_i: (i) the *RF sending* phase, (ii) the *cloud computing* phase, and (iii) the *RF receiving* phase. In the RF sending phase, the specification and input data of task v_i are sent to the cloud by the mobile device through the *wireless sending channel*. In the cloud computing phase, task v_i is executed in the cloud. In the RF receiving phase, the mobile device receives the output data of the task v_i from the cloud through the *wireless receiving channel*. The cloud transmits the output data of the task v_i back to the mobile device as long as it finishes processing the task v_i. We use R^s to denote the data sending rate of the wireless sending channel, and R^r to denote the data receiving rate of the wireless receiving channel. Accordingly, let P^s denote the power consumption levels of the RF component in the mobile device for sending data to the cloud.

The local core in the mobile device or the wireless sending channel can only process or send one task at a time, and preemption is not allowed in this framework. On the other hand, the cloud can execute a large number of tasks in parallel as long as there is no dependency among the tasks.

C. Task-Precedence Requirements in the MCC Environment

We use $T_{i,k}^l$ to denote the execution time of task v_i on the k-th core of the mobile device, where superscript l means "local execution". $T_{i,k}^l$ is inversely proportional to the operating frequency f_k. We use T_i^c to denote the computation time of task v_i on the cloud, where superscript c means "execution on the cloud". The time of sending task v_i onto the cloud, denoted by T_i^s, is calculated by:

$$T_i^s = data_i / R^s. \tag{1}$$

The time of receiving task v_i from the cloud, denoted by T_i^r, is calculated by:

$$T_i^r = data'_i / R^r. \tag{2}$$

For a task v_j that is already scheduled (on a local core or the cloud), we use FT_j^l, FT_j^{ws}, FT_j^c, and FT_j^{wr} to denote the finish time of task v_j on a local core, the wireless sending channel (i.e., the task has been completely offloaded to cloud), the cloud, and the wireless receiving channel (i.e., the mobile device has completely received the output data of the task from the cloud), respectively. If the task v_j is scheduled locally, $FT_j^{ws} = FT_j^c = FT_j^{wr} = 0$; otherwise (i.e., the task v_j is offloaded on to the cloud), we have $FT_j^l = 0$. Please note that the mobile device can only schedule tasks in the local cores and the wireless channels, whereas the cloud computing controller schedules tasks that have already been uploaded inside the cloud and transmits the output data back to the mobile device. However, the mobile device can anticipate the execution of tasks in the cloud and estimate the corresponding FT_j^c and FT_j^{wr} values from the parameters T_j^c, T_j^r, etc.

1) Local scheduling

Before we schedule a task v_i, all its immediate predecessors must have already been scheduled. Suppose that task v_i is to be scheduled on a local core. Then the *ready time* of task v_i, denoted by RT_i^l, is calculated as:

$$RT_i^l = \max_{v_j \in \boldsymbol{pred}(v_i)} \max\{FT_j^l, FT_j^{wr}\}, \tag{3}$$

where $\boldsymbol{pred}(v_i)$ is the set of immediate predecessors of the task v_i. The ready time RT_i^l is the earliest time when all immediate predecessors of task v_i have completed execution and their results are available for task v_i:

- If task v_j (an immediate predecessor of task v_i) has been scheduled locally, $\max\{FT_j^l, FT_j^{wr}\} = FT_j^l$. In this case we have $RT_i^l \geq FT_j^l$, which means that task v_i can start execution on a local core only after the local execution of task v_j has finished.
- If task v_j (an immediate predecessor of task v_i) has been offloaded on to the cloud, $\max\{FT_j^l, FT_j^{wr}\} = FT_j^{wr}$. In this case we have $RT_i^l \geq FT_j^{wr}$, which means that task v_i can start execution on a local core only after the mobile device has completely received the output data (results) of task v_j through the wireless receiving channel.

We can only schedule task v_i to start execution at or after its ready time RT_i^l, if the task has been scheduled on a local core. In this way the task-precedence requirements can be preserved. However, we might not be able to start executing task v_i at time RT_i^l exactly, because the cores may be executing other tasks at that time.

2) Cloud scheduling

On the other hand, suppose that task v_i is to be offloaded on to the cloud. The ready time of task v_i on the wireless sending channel, denoted by RT_i^{ws}, is calculated as:

$$RT_i^{ws} = \max_{v_j \in \boldsymbol{pred}(v_i)} \max\{FT_j^l, FT_j^{ws}\}. \tag{4}$$

RT_i^{ws} denotes the earliest start time when the task v_i can be scheduled on the wireless sending channel in order to preserve the task-precedence requirements:

- If task v_j (an immediate predecessor of task v_i) has been scheduled locally, $\max\{FT_j^l, FT_j^{ws}\} = FT_j^l$. In this case we have $RT_i^{ws} \geq FT_j^l$, which means that the mobile device can start to send task v_i through the wireless channel only after the local execution of task v_j has finished.
- If task v_j (an immediate predecessor of task v_i) has been offloaded on to the cloud, $\max\{FT_j^l, FT_j^{ws}\} = FT_j^{ws}$. In this case we have $RT_i^{ws} \geq FT_j^{ws}$, which means that the mobile device can start to send task v_i through the wireless channel only after the mobile device has completed offloading task v_j to the cloud.

The ready time of task v_i on the cloud, denoted by RT_i^c, is calculated as:

$$RT_i^c = \max\{FT_i^{ws}, \max_{v_j \in \boldsymbol{pred}(v_i)} FT_j^c\}. \tag{5}$$

RT_i^c denotes the earliest time when task v_i can start execution on the cloud. If task v_j (an immediate predecessor of task v_i) is scheduled locally, $FT_j^c = 0$. Therefore, $\max_{v_j \in \boldsymbol{pred}(v_i)} FT_j^c$ in (5) is the time when all the immediate predecessors of task v_i that are offloaded to the cloud have finished execution on the cloud. On the other hand, FT_i^{ws} is the time when task v_i has been completely offloaded to the cloud through the wireless sending channel, and therefore we have $RT_i^c \geq FT_i^{ws}$. The cloud computing controller can schedule task v_i to start execution at time RT_i^c exactly (because of the high parallelism

in the cloud), such that the task-precedence requirements can be preserved.

Finally, let RT_i^{wr} denote the ready time for the cloud to transmit back the results of task v_i, and we have:

$$RT_i^{wr} = FT_i^c. \quad (6)$$

In other words, the cloud can transmit the output data (results) of task v_i back to the mobile device immediately after it has finished processing this task.

D. Energy Consumption and Application Completion Time

If task v_i is executed locally on the k-th core of the mobile device, the energy consumption of the task is given by:

$$E_{i,k}^l = P_k \cdot T_{i,k}^l. \quad (7)$$

If task v_i is offloaded to the cloud, the energy consumption of the mobile device for offloading the task is given by:

$$E_i^c = P^s \cdot T_i^s. \quad (8)$$

The execution of task v_i on the cloud does not consume energy of the mobile device. The total energy consumption of the mobile device for running the application, denoted by E^{total}, is given by

$$E^{total} = \sum_{i=1}^{N} E_i. \quad (9)$$

where E_i equals to $E_{i,k}^l$ if task v_i is executed locally on the k-th core of the mobile device, and equals to E_i^c if the task is offloaded to the cloud.

The application completion time T^{total} is calculated by:

$$T^{total} = \max_{v_i \in exit\ tasks} \max\{FT_i^l, FT_i^{wr}\}. \quad (10)$$

The inner max block gives the finish time of an exit task v_i. It equals to FT_i^l if v_i is executed on a local core, and equals to FT_i^{wr} if v_i is offloaded to the cloud.

The MCC task scheduling problem is to (i) determine the tasks of an application to be offloaded, (ii) map the remaining tasks onto the heterogeneous cores in a mobile device, and (iii) schedule the tasks on the heterogeneous cores and wireless communication channels. The objective is to minimize E^{total} under the following constraints: (i) task-precedence requirements and (ii) the application completion time constraint $T^{total} \le T^{max}$, where T^{max} is the maximum application completion time.

III. MCC Task Scheduling Algorithm

The MCC task scheduling algorithm has two steps: initial scheduling for minimizing the application completion time T^{total}, and task migration for minimizing the energy consumption E^{total} under the application completion time constraint $T^{total} \le T^{max}$. The flow chart of the whole MCC task scheduling algorithm is shown in Fig. 2.

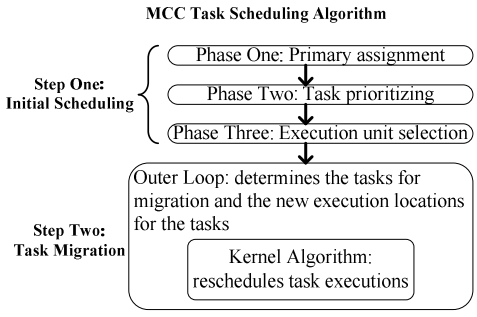


Figure 2. Flow chart of the MCC task scheduling algorithm.

In order to strictly satisfy the application completion time constraint, we minimize T^{total} in the first step and then reduce energy consumption by moving tasks from a local core to another or to the cloud in the second step. Otherwise, if we minimize energy consumption at first, the application completion time constraint can hardly be guaranteed when we move a task from one core to another or to the cloud in the subsequent step. This is because of the task-precedence requirements and the parallelism constraints on the local cores and the wireless communication channels.

A. Step One: Initial Scheduling Algorithm

In the initial scheduling algorithm, we generate the minimal-delay scheduling without considering the energy consumption of the mobile device. Reference [6] proposed the HEFT algorithm, which generates the minimal-delay scheduling for tasks running on a number of heterogeneous cores. We modify the HEFT algorithm to take into account the joint scheduling of tasks on the local cores, the wireless communication channels, and the cloud. The initial scheduling algorithm has three phases: primary assignment, task prioritizing, and execution unit selection, as shown in Fig. 2. In the following, we discuss the three phases in detail:

1) Primary assignment

In this phase, we determine the subset of tasks that are initially assigned for the cloud execution. Offloading such tasks to the cloud will result in savings of the application completion time. Please note that this primary assignment is not the final decision, since we can assign more tasks for remote execution in the "execution unit selection" phase of initial scheduling. For each task v_i, we calculate the minimum local execution time $T_i^{l,min}$ (on the fastest core) as:

$$T_i^{l,min} = \min_{1 \le k \le K} T_{i,k}^l. \quad (11)$$

We also calculate the estimated remote execution time T_i^{re} as:

$$T_i^{re} = T_i^s + T_i^c + T_i^r. \quad (12)$$

If $T_i^{re} < T_i^{l,min}$, task v_i is assigned for remote execution on the cloud. We call such a task a "cloud task".

2) Task prioritizing

In this phase, we calculate the *priority* of each task similar to the HEFT algorithm. First, we calculate the *computation cost* w_i for each task. If task v_i is a cloud task, its computation cost is given by

$$w_i = T_i^{re}. \quad (13)$$

If task v_i is not a cloud task, w_i is calculated as the average computation time of task v_i in the local cores, i.e.,

$$w_i = \operatorname*{avg}_{1 \le k \le K} T_{i,k}^l. \quad (14)$$

Then the priority level of each task v_i is recursively defined by

$$priority(v_i) = w_i + \max_{v_j \in \boldsymbol{succ}(v_i)} priority(v_j), \quad (15)$$

where $\boldsymbol{succ}(v_i)$ is the set of immediate successors of task v_i. The priority levels are recursively computed by traversing the task graph starting from the exit tasks. For the exit tasks, the priority level is equal to

$$priority(v_i) = w_i \text{ for } v_i \in exit\ tasks. \quad (16)$$

Basically, $priority(v_i)$ is the length of the critical path from task v_i to the exit tasks.

3) Execution unit selection

In this phase, tasks are selected and scheduled in the descending order of their priorities. If task v_j is the immediate predecessor of task v_i, we have $priority(v_j) > priority(v_i)$ from (15). Therefore, when task v_i is selected for scheduling in this phase, all its immediate predecessors have already been scheduled.

- If the selected task v_i is a cloud task, we calculate its ready time RT_i^{ws} on the wireless channel, and allocate the earliest available time slot on the wireless sending channel for offloading the task. Please note that the mobile device might not be able to start offloading task v_i at time RT_i^{ws} if it is offloading other tasks at that time. We calculate FT_i^{ws} from the schedule, and then the cloud will begin executing task v_i at the ready time RT_i^c (because of the high parallelism in the cloud.) Finally we calculate $FT_i^c = RT_i^c + T_i^c$ and $FT_i^{wr} = FT_i^c + T_i^r$. In this way, we have scheduled task v_i and estimated the associated finish times.
- If the selected task v_i is not a cloud task, it may be scheduled on a local core or the cloud. We need to estimate the finish time of this task if it is scheduled on each core and the finish time of this task if it is offloaded to the cloud, using the similar procedure as described above. Then we schedule task v_i on the core or offload it to the cloud such that the finish time is minimized. When we schedule the task, we need to make sure that the task-precedence requirements are satisfied according to Section II.C.

Similar to the HEFT algorithm, the computation complexity of the initial scheduling algorithm is $O(E \times K)$, where E is the number of edges in the task graph G, and K is the number of cores. We consider sparse task graphs (i.e., $E = O(N)$ where N is the number of tasks), and therefore the complexity of initial scheduling becomes $O(N \times K)$.

As an example, we perform initial task scheduling on the task graph shown in Fig. 1, assuming that there are three heterogeneous cores in the mobile device. The $T_{i,k}^l$ values are shown in the table in Fig. 1, and we use $T_i^s = 3$, $T_i^c = 1$, and $T_i^r = 1$ for all the tasks. Fig. 3 presents the task scheduling results, where the horizontal axes denote the time. For example,

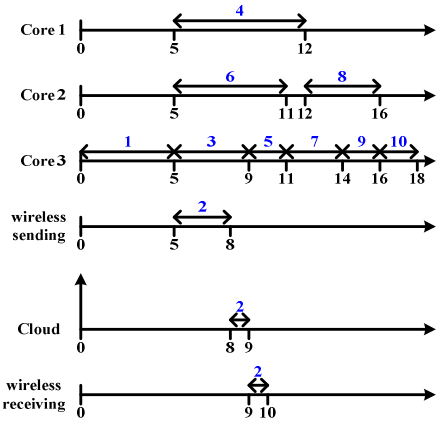


Figure 3. Task scheduling result by the initial scheduling algorithm.

task v_4 is executed on core 1 from time 5 to 12. Task v_2 is offloaded on to the cloud. The mobile device sends the specification and input data of task v_2 using the wireless sending channel from time 5 to 8. And then, task v_2 is computed on the cloud from time 8 to 9. The cloud transmits the output data (results) of task v_2 back to the mobile device from time 9 to 10. The application completion time of this example is 18, which is the finish time of the exit task v_{10}.

B. Step Two: Task Migration Algorithm

The task migration algorithm aims at minimizing the energy consumption E^{total} under the application completion time constraint $T^{total} \leq T^{max}$. The energy consumption is reduced through migrating tasks from a local core to another local core or to the cloud. The task migration algorithm is an iterative algorithm comprised of a *kernel algorithm* and an *outer loop*. In each iteration, the outer loop determines the target task for migration and the new execution location (i.e., a different local core or the cloud) in order to minimize the energy consumption E^{total}. It should also maintain the application time constraint $T^{total} \leq T^{max}$ without violation. Given the target task for migration and the new execution location, the kernel algorithm generates a new scheduling result that has the minimum application completion time T^{total} with linear time complexity.

1) Outer loop

The outer loop of the task migration algorithm determines the target tasks to migrate from one local core to another local core or to the cloud, in order to reduce the mobile device's energy consumption. It should also maintain the application time constraint $T^{total} \leq T^{max}$ without violation. Please note that the task migration algorithm does not account for the migration of a task from offloading to the cloud back to local processing, because the energy consumption of the mobile device will generally increase in this case.

In each iteration of the outer loop, let N' denote the number of tasks that are currently scheduled on the local cores. Each of them can be moved to execute on one of the other $K - 1$ cores or the cloud. Therefore, there are a total of $N' \times K$ migration choices.

- For each choice, we run the kernel algorithm to find a new schedule, and calculate the corresponding energy consumption E^{total} and application completion time T^{total}.
- We select the choice that results in the largest energy reduction compared with the current schedule and no increase in the application completion time T^{total} than the current schedule.
- If we cannot find such a choice, we select the one that results in the largest ratio of energy reduction to the increase of the application completion time. We should make sure that the new application completion time does not exceed the limit value T^{max}.

We repeat the previous steps until the energy consumption of the mobile device cannot be further minimized.

2) Kernel algorithm (i.e., rescheduling algorithm)

In a task scheduling, let k_i denote the execution location of task v_i. $k_i \neq 0$ means that task v_i is executed on the k_i-th core, whereas $k_i = 0$ means that task v_i is offloaded on to the cloud. In the kernel algorithm, we have an original scheduling of the task graph. We are given by the outer loop a task v_{tar} for

migration and its new execution location k_{tar}. The kernel algorithm should generate a new scheduling of the task graph G, where task v_{tar} is executed on the new location k_{tar} and the remaining tasks are executed on the same locations as the original scheduling. The kernel algorithm aims at minimizing the application completion time T^{total}. On the other hand, the energy consumption E^{total} is fixed and can be directly calculated using (7)~(9) once the execution locations of tasks are known. Because the kernel algorithm will be called many times from the outer loop, we propose an efficient linear-time rescheduling algorithm of the task graph as the kernel algorithm, which is more efficient than the modified HEFT algorithm when the number of cores is relatively large.

For the original scheduling, we use a *sequence set* $S_k = \{v_{(k,1)}, v_{(k,2)}, \dots\}$ to denote the sequence of tasks that are executed on the k-th local core and we use the sequence set $S_0 = \{v_{(0,1)}, v_{(0,2)}, \dots\}$ to denote the sequence of tasks that are offloaded to the cloud through the wireless sending channel. For example, if we use the scheduling result in Fig. 3 as the original scheduling, we have $S_1 = \{v_4\}$, $S_2 = \{v_6, v_8\}$, $S_3 = \{v_1, v_3, v_5, v_7, v_9, v_{10}\}$, and $S_0 = \{v_2\}$. Suppose that task v_{tar} is executed on the k_{ori}-th core in the original scheduling. We know from the outer loop that v_{tar} will be moved on to the k_{tar}-th core in the new scheduling. We should derive the new sequence sets S_k^{new} for $0 \le k \le K$, which corresponds to the sequence of tasks executed (or transmitted) on each core and the wireless sending channel in the new scheduling. In the linear-time rescheduling algorithm, we will not change the ordering of tasks in the other cores except for the k_{tar}-th core (because we are going to execute task v_{tar} in this core), i.e.,

$$S_k^{new} = S_k \backslash v_{tar} \text{ for } k = k_{ori}, \tag{17}$$

and

$$S_k^{new} = S_k \text{ for } k \neq k_{tar} \wedge k \neq k_{ori}. \tag{18}$$

In the following, we derive $S_{k_{tar}}^{new}$ by inserting v_{tar} at a "proper" location of the original schedule sequence $S_{k_{tar}}$. We need to satisfy the following task-precedence requirements on the k_{tar}-th core ($k_{tar} = 0$ means the wireless sending channel):

- For any two tasks v_i and v_j that are executed (or transmitted) on the same core or wireless communication channel, task v_i must be executed (or transmitted) before v_j if v_i is a transitive predecessor of v_j in the task graph G.

Hence, we should insert v_{tar} into $S_{k_{tar}}$ such that v_{tar} is executed (or transmitted) after all its transitive predecessors and before all its transitive successors. In order to achieve this goal, we calculate the ready time RT_{tar} of task v_{tar} in the original scheduling. RT_{tar} equals to RT_{tar}^{l} (calculated from (3)) when $k_{tar} > 0$ and equals to RT_{tar}^{ws} (calculated from (4)) when $k_{tar} = 0$. In addition, we know the start time ST_i of each task v_i in the original scheduling. Therefore, we derive $S_{k_{tar}}^{new}$ as:

$$S_{k_{tar}}^{new} = \{v_{(k_{tar},1)}, \dots, v_{(k_{tar},m)}, \boldsymbol{v_{tar}}, v_{(k_{tar},m+1)}, \dots\}, \tag{19}$$

where the start times of tasks $v_{(k_{tar},1)}, \dots, v_{(k_{tar},m)}$ are earlier than RT_{tar} and the start times of tasks $v_{(k_{tar},m+1)}, \dots$ are later than RT_{tar}. In this way, it can be proved that the task-precedence requirements on the k_{tar}-th core are preserved.

Now with the new sequence sets S_k^{new} for $0 \le k \le K$, we are going to find a new schedule of the task graph in linear time complexity $O(N)$. We maintain two vectors $ready1$ and $ready2$. $ready1_i$ is the number of immediate predecessors of task v_i that have not been scheduled. $ready2_i = 0$ if all the tasks before task v_i in the same sequence S_k^{new} have already been scheduled. In addition, we maintain a LIFO stack for storing the tasks that are ready for scheduling. The stack is initialized by pushing the task v_i's with both $ready1_i = 0$ and $ready2_i = 0$ into the empty stack. We repeat the following steps until the stack becomes empty again. Then we have scheduled all the tasks.

- Pop a task v_i from the stack.
- Suppose that task $v_i \in S_k^{new}$. If $k = 0$, we schedule the task on the wireless sending, and calculate the time when the mobile device completely receives the output data (results) of task v_i from the cloud. Otherwise, schedule the task on the k-th core.
- Update vectors $ready1$ (reducing $ready1_j$ by one for all $v_j \in \boldsymbol{succ}(v_i)$) and $ready2$, and push all the new tasks v_j with both $ready1_j = 0$ and $ready2_j = 0$ into the stack.

C. Computation Complexity of MCC Task Scheduling Algorithm and an Example of Scheduling Result

The overall computation complexity of the MCC task scheduling algorithm is $O(N^3 \times K)$, which is comparable to the reference work on task scheduling. Fig. 4 presents the task scheduling result by the MCC task scheduling algorithm for the task graph in Fig. 1. The application completion time constraint is set as $T^{total} \le 27$. Please note that Fig. 3 only presents the result of the first step of the MCC task scheduling algorithm (i.e., the initial scheduling algorithm), whereas Fig. 4 presents the result of the entire MCC task scheduling algorithm. Comparing Fig. 3 with Fig. 4, we observe that more tasks are offloaded on to the cloud in Fig. 4 for reducing the energy consumption. The application completion time in Fig. 4 is 26, which is larger than that in Fig. 3. This is mainly due to the limit on the transmission rate of the wireless sending channel. The power consumption of core 1~3 are set as $P_1 = 1$, $P_2 = 2$, and $P_3 = 4$. And the power consumption of the RF components is set as $P^s = 0.5$. In summary, we have $E^{total} = 100.5$ and $T^{total} = 18$ in Fig. 3, and we have $E^{total} = 27$ and $T^{total} = 26$ in Fig. 4. This result demonstrates that the task migration algorithm (i.e., the second step of the MCC task

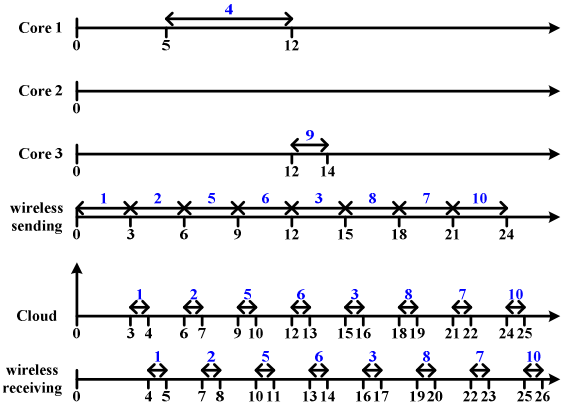


Figure 4. Task scheduling result by the MCC task scheduling algorithm.

TABLE I. COMPARISON BETWEEN THE PROPOSED ALGORITHM AND THE BASELINE ALGORITHMS ($K = 3$)

	N	T^{max}	T^{total}			E^{total}			Exe. Time (s)	
			Proposed	Baseline1	Baseline2	Proposed	Baseline1	Baseline2	Proposed	Baseline1
1	11	100	100	99	100	112	111	349	0.017123	2.810853
2	21	150	148	147	144	286	306	688	0.051820	5.155169
3	31	170	170	168	169	500	569	1018	0.117577	7.639482
4	41	210	208	209	215	747	814	1446	0.152341	10.347463
5	51	250	248	248	254	902	985	1732	0.278198	13.198991
6	61	330	328	328	322	1011	1183	1993	0.520647	15.982825
7	71	400	400	364	411	1304	1755	2818	0.633178	19.267765
8	81	450	448	434	469	1557	1877	3205	0.820833	22.775889
9	91	500	499	500	532	1705	2308	3642	1.208208	25.858333
10	101	550	548	488	582	1897	2520	4008	1.554154	29.702970

scheduling algorithm) can significantly reduce the energy consumption while satisfying the application completion time constraint.

IV. EXPERIMENTAL RESULTS

In this section, we demonstrate the effectiveness of the proposed MCC task scheduling algorithm on a set of randomly generated task graphs. We compare the scheduling results of the proposed algorithm to those of the baseline algorithms. All the algorithms are implemented in MATLAB programs executed in a 2.6 GHz Intel Core i7 processor.

We consider two baseline algorithms for comparison. The baseline1 algorithm is described as follows:

1. Generate a random vector L, where $L_i \in \{0,1,\dots,k,\dots,K\}$ denotes the computing location of task v_i. If $L_i = 0$, task v_i will be offloaded on to the cloud. If $L_i = k$, task v_i will be executed on the k-th core in the mobile device.
2. Order and schedule task executions on each local core, the cloud, and the wireless communication channels using the *modified initial scheduling algorithm*. Different from the initial scheduling algorithm described in Section III. A, in the modified initial scheduling algorithm, the execution location of each task is pre-defined by L. Calculated E^{total} and T^{total}.
3. Repeat Step1~2 for 10,000 times to find the scheduling with the minimum E^{total} under the constraint that $T^{total} \leq T^{max}$.

By comparing the proposed algorithm with the baseline1 algorithm, we will demonstrate the effectiveness and efficiency of our proposed algorithm.

The baseline2 algorithm is the same as our proposed algorithm except that it runs in the *local* mobile device environment only (i.e., the mobile device does not have access to the cloud and only the local resources can be used for task executions.) By comparing the proposed algorithm with the baseline2 algorithm, we will demonstrate that the MCC framework shows great benefits in energy saving and performance enhancement for the mobile devices.

A random task graph generator is implemented to generate task graphs with various characteristics. Input parameters of the task graph generator are given below.

- Number of tasks in the graph N.
- The density of edges in the graph α.
- Number of cores in the mobile device K.
- The average task computation time on a local core T_l^{avg}.
- The average task sending time T_s^{avg}.
- The average task receiving time T_r^{avg}.
- The average task computation time on the cloud T_c^{avg}.

A task graph can be generated with N and α. The $T_{i,k}^l$ values are generated in the following way: (i) $T_{i,1}^l$ for $1 \leq i \leq N$ are generated with the average value of T_l^{avg}, (ii) $T_{i,k+1}^l$ on the (k+1)-st core is set around $T_{i,k}^l/\beta$ for $1 \leq k \leq K-1$, where β is a factor. In addition, T_i^s/ T_i^c/T_i^r for $1 \leq i \leq N$ are generated with the average value of T_s^{avg}/T_c^{avg}/T_r^{avg}.

Now we assume there are $K = 3$ heterogeneous cores in the mobile device. The core 1 is a low-power core and the core 3 is a high-performance core. The power consumption P_k values of the three cores are set as $P_1 = 1$, $P_2 = 2$, and $P_3 = 4$. The power consumption of the RF components is set as $P^s = 0.5$. Ten task graphs with different task numbers N and different characteristics are generated for comparing the proposed algorithm with baseline algorithms. Table I shows the application completion time T^{total} and the energy consumption E^{total} of the scheduling results from all the algorithms. Table I also compares the program execution time of the proposed algorithm and the baseline1 algorithm. We do not compare the program execution time of the proposed algorithm and the baseline2 algorithm, because they are similar algorithms except that the baseline2 algorithm is designed for the mobile devices without cloud access.

In Table I, we can see that both the proposed algorithm and the baseline1 algorithm can guarantee the application completion time constraint. The proposed algorithm achieves less energy consumption than the baseline1 algorithm for task graphs 2~9. However, for task graph 1, the proposed algorithm generates a scheduling with a little bit more energy consumption. This is because the task execution locations are exhaustively searched in the baseline1 algorithm with a small N value ($N = 11$), whereas such exhaustive search is not possible for a larger N value. Please note that the execution time of the baseline1 algorithm is much larger than the proposed algorithm. On the other hand, in Table I, the scheduling results from baseline2 algorithm cannot satisfy the application completion time constraint in some cases, which demonstrates that the MCC framework can improve the performance of the mobile devices. We can observe that the proposed algorithm can achieve a maximum energy reduction by a factor of 3.1 compared with the baseline2 algorithm in Table I, demonstrating the MCC framework can greatly reduce the energy consumption in mobile devices.

TABLE II. COMPARISON BETWEEN THE PROPOSED ALGORITHM AND THE BASELINE ALGORITHMS ($K = 6$)

	N	T^{max}	T^{total}			E^{total}			Exe. Time (s)	
			Proposed	Baseline1	Baseline2	Proposed	Baseline1	Baseline2	Proposed	Baseline1
1	11	60	59	58	59	849	1060	1528	0.205388	2.760699
2	21	80	79	75	80	1380.5	2350	2604	0.269298	5.114751
3	31	110	105	98	108	2216	2951	3857	0.262893	7.606270
4	41	140	140	130	137	2694	4074	5069	0.532875	10.126989
5	51	180	178	180	180	3024	4732	5964	1.003569	13.007214
6	61	240	240	236	239	3475	6378	7184	1.963312	16.029213
7	71	300	299	576	299	3272	355	8237	3.135316	19.158112
8	81	350	347	288	349	3543	9606	8711	4.435423	22.643440
9	91	380	379	736	377	4183	455	10177	5.977805	26.011400
10	101	420	417	411	420	4475	9156	11202	7.798183	29.839775

Furthermore, we assume there are $K = 6$ heterogeneous cores in the mobile device. The core 1 is a low-power core and the core 6 is a high-performance core. The power consumption P_k values of the six cores are set as $P_1 = 1$, $P_2 = 2$, and $P_3 = 4$, $P_4 = 8$, $P_5 = 16$, and $P_6 = 32$. The power consumption of the RF components is set as $P^s = 0.5$. Ten task graphs with different task numbers N and different characteristics are generated to compare the proposed algorithm with the baseline algorithms. Table II shows the application completion time T^{total} and the energy consumption E^{total} of the scheduling results from all the algorithms. Table II also compares the program execution time of our proposed algorithm and the baseline1 algorithm. Please note that some scheduling results (for task graph 7 and 9 in Table II) from baseline1 cannot satisfy the application completion time constraint. This is because randomly generating task execution locations cannot yield good results when the number of cores and the number of tasks are relatively large. Some scheduling results of the baseline1 algorithm will offload all tasks to the cloud, and of course violate the application completion time constraint. That is why the energy consumption results of the baseline1 algorithm on task graph 7 and 9 are much lower than the proposed algorithm.

V. CONCLUSION

This work studies the MCC task scheduling problem. To our best knowledge, this is the first task scheduling work that minimizes energy consumption under a hard completion time constraint for the task graph in the MCC environment, taking into account the joint task scheduling on the local cores in the mobile device, the wireless communication channels, and the cloud. A novel algorithm is proposed that starts from a minimal-delay scheduling and subsequently performs energy reduction by migrating tasks among the local cores and the cloud. A linear-time rescheduling algorithm is proposed for the task migration such that the overall computation complexity is effectively reduced. Simulation results demonstrate significant energy reduction with the overall completion time constraint satisfied.

ACKNOWLEDGEMENT

This work is supported in part by the Software and Hardware Foundations program of the NSF's Directorate for Computer & Information Science & Engineering.

REFERENCES

[1] B. Hayes, "Cloud Computing," *Communications of the ACM*, 2008.

[2] H. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," in *Wireless Commun. and Mobile Comput.*, 2011.

[3] A. Khan and K. Ahirwar, "Mobile cloud computing as a future of mobile multimedia database," in *International Journal of Computer Science and Communication*, 2011.

[4] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing," in *Proc. of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, 2010.

[5] U. Kremer, J. Hicks, and J. M. Rehg, "A compilation framework for power and energy management on mobile computers," in *Languages and Compilers for Parallel Computing*, Springer Berlin Heidelberg, 2003.

[6] H. Topcuoglu, S. Hariri, and M. Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. on Parallel and Distributed Systems*, vol. 13, no. 3, 2002.

[7] M. R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: enabling interactive perception applications on mobile devices," in *Proc. of MobiSys*, 2011.

[8] L. Yang, J. Cao, S. Tang, T. Li, and A. T. S. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," in *Proc. of IEEE 5th International Conference on Cloud Computing*, 2012.

[9] P. Rong and M. Pedram, "Power-aware scheduling and dynamic voltage setting for tasks running on a hard real-time system," in *Proc. of Asia and South Pacific Design Automation Conference*, 2006.

[10] Z. Li, C. Wang, and R. Xu, "Task allocation for distributed multimedia processing on wirelessly networked handheld devices," in *Proc. of the International Parallel and Distributed Processing Symposium (IPDPS)*, 2002.

[11] Y. C. Lee and A. Y. Zomaya, "Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling," in *Proc. of IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2009.

[12] K. Kumar and Y. H. Lu, "Cloud computing for mobile users: can offloading computation save energy," *Computer*, 2010.

[13] P. Greenhalgh, "Big.LITTLE processing with ARM Cortex-A15 & Cortex-A7," *ARM White Paper*, 2011.