Communication Augmented Latest Possible Scheduling for Cloud Computing with Delay Constraint and Task Dependency

Sowndarya Sundar and Ben Liang
Department of Electrical and Computer Engineering
University of Toronto, Ontario, Canada
{ssundar, liang}@ece.utoronto.ca

***Abstract*—We consider a system consisting of a remote cloud and a network of heterogeneous local processors. We aim to identify the optimal scheduling decision for a mobile application comprising of dependent tasks, such that the total cost is minimized subject to an application deadline. The total cost for application execution accounts for the execution of tasks and the communication of input/output data between the mobile device and the remote cloud. We propose the Communication Augmented Latest Possible Scheduling (CALPS) algorithm to obtain an approximate solution for this NP-hard problem in polynomial time. Through simulation, we study the effect of communication delay and application deadline on the application cost and feasibility. Furthermore, we compare the CALPS algorithm with the optimal solution, a lower bound, and an existing state-of-the-art algorithm to demonstrate the advantage of CALPS in performance and computational complexity.**

I. Introduction

Cloud computing can augment the capabilities of resource-poor mobile devices with the help of resourceful servers. It allows reduction of energy consumption at the mobile device by offloading resource-hungry applications to the cloud, which helps improve the mobile device's battery lifetime. Existing research work has resulted in several computational offloading systems such as energy-aware migration decisions at run-time [1], multiple virtual machine images [2], or trusted cloudlets [3].

There are several mobile applications that can benefit from offloading and have been recognized in literature, such as image and language processing [1], sharing GPS/Internet data [4], and crowd computing [5]. Each mobile application can be modelled as a number of tasks, and each task is executed either locally at the mobile device or remotely at the cloud. This results in greater flexibility on selectively offloading tasks in finer granularity [6], [7]. By performing task scheduling, we can identify the best possible offloading decision (e.g., one that minimizes cost/energy) for the entire application, subject to latency constraints on the execution of the application [1], [6], [8].

This work has been funded in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada under grant STPGP 447497-13.

In this paper, we aim to determine the task scheduling decision that minimizes the total cost of running a mobile application subject to an overall application completion deadline. We consider a single mobile application consisting of dependent tasks that have possible data communication and precedence constraints. We wish to schedule these tasks onto a network of a finite number of heterogeneous local processors and a remote cloud. Each local processor can process just one task at a time whereas the remote cloud is viewed as an additional processor having infinite capacity, while there exist a time and cost associated with task execution and data communication.

The aforementioned objective of minimizing cost under an application deadline helps us obtain a suitable trade-off rather than aiming to minimize just the energy/cost [9] or just the completion time [10]. Furthermore, the finite-capacity consideration of the local processors on the mobile device renders the problem more practical in comparison to the system model examined in existing research work, in which mobile devices are assumed to be capable of simultaneously processing any number of tasks [1], [6], [8]. Nevertheless, in order to meet the application deadline, the scheduling decision must take into account the waiting times of the tasks, in addition to the dependencies and possible communication delay between the tasks. The decision-making process is thus complicated by the aforementioned requirements.

We formulate the proposed problem as a constrained optimization problem over binary scheduling decision variables. As this problem is NP-hard in nature, we cannot guarantee an optimal solution in polynomial time. Therefore, we propose a novel and efficient heuristic algorithm, named Communication Augmented Latest Possible Scheduling (CALPS). Through simulation, we study the impact of the communication delay and the application deadline on the cost and feasibility performance. Furthermore, we relax the formulated optimization problem to a convex form in order to obtain a lower bound to the optimal solution. We compare the CALPS algorithm with the optimal solution, the lower bound, and an existing state-of-the-art algorithm that assumes an infinite number of processors at the mobile device [8], demonstrating that CALPS

is computationally efficient and is effective in cost reduction under a deadline constraint.

The rest of the paper is organized as follows. In Section II, we present the related work. Section III describes the system model and the problem formulation. In Section IV, we present the two proposed solution approaches. Section V presents the simulation results, and we conclude the paper in section VI.

II. Related Work

A closely related problem is to achieve minimal mobile energy consumption given a delay constraint. This has been approached from different angles in the literature. In [11], the energy-optimal execution policy for the whole application is obtained by solving two constrained optimization problems, i.e., how to optimally configure the clock frequency to complete CPU cycles for mobile execution, and how to optimally schedule the data transmission for cloud execution. In [6], this problem is investigated under a Markovian stochastic channel subject to a deadline. However, only a specific case of linear task topology is considered, as opposed to generally dependent tasks, and only an approximate solution is obtained to reduce complexity.

There are several techniques that have been employed to find the scheduling decision for a general application consisting of dependent tasks, such as integer linear programming [1] and graph partitioning [6]. However, [6] does not consider an application deadline and [1] has high computational complexity and does not provide a polynomial-time guarantee. Additionally, both [1] and [6] assume that the mobile device and the cloud cannot be active simultaneously, i.e, the mobile device must be idle while the cloud is busy and vice-versa. In [8], dynamic programming is utilized to identify this decision but under an unrealistic assumption that the mobile device can simultaneously execute any number of tasks without impacting the processing time of each task. The problem in [12] considers dependent tasks and heterogeneous processors, but it does not take into account data communication between tasks and it only addresses the problem of minimizing the makespan of the application.

In this work, we address the problem of minimizing the overall cost with an application deadline, for an application consisting of dependent tasks and a network of heterogeneous local processors and a remote cloud. To the best of our knowledge, this problem has not been addressed in the literature.

III. System Model and Problem Formulation

A. Local Processors and the Cloud

We consider a system with M local processors and one remote cloud. We denote the set of processors, including the cloud, by $\mathcal{P}$. The total number of processors is

$$M' = M + 1. \tag{1}$$

We assume that the communication time between local processors is negligible. In many practical scenarios, these processors belong to the same local device. However, our study is more generally applicable, possibly to scenarios where nearby devices pool their processors in a shared cloudlet [3]. Each local processor is capable of executing only one task at a time, while the remote cloud is capable of executing an infinite number of tasks simultaneously.

TABLE I
Notations

Notation	Description
t_{ij}	execution time for task i on processor j
p_j	processing cost per unit time on processor j
p^c	communication cost per unit delay
e_{ik}	amount of data to be communicated from task i to task k
d	delay per unit data between local processors and cloud
L	application deadline

B. Task Dependency Graph

Consider a single application that is partitioned into tasks and must be completed before a deadline L. The dependencies between the tasks is modeled as a Directed Acyclic Graph (DAG) $G = \langle \mathcal{V}, \mathcal{E} \rangle$ where $\mathcal{V}$ is the set of tasks and $\mathcal{E}$ is the set of edges. The edge (i, k) on the graph specifies that there is some required data transfer, e_{ik}, from task i to task k and hence, i cannot start before k finishes. Furthermore, if they are scheduled at different locations (one locally and the other at the cloud), the communication delay is $e_{ik}d$ and the communication cost is $p^c e_{ik} d$, where d is the delay per unit data and p^c is the communication cost per unit time. When task i is executed on processor j, the execution time is t_{ij}, and the execution cost is $p_j t_{ij}$, where p_j is the processing price per unit time on processor j.

We assume that an application is initiated at a local processor and must end at a local processor. To model this requirement, for a given DAG representing an application, we insert two dummy nodes, i.e., tasks having zero execution time and zero communication cost. One dummy task is inserted at the start to trigger the application at the local device and another task is inserted at the very end to receive all the results back at the local device. This insertion is without loss of generality since it preserves the application. Hence, the total number of tasks can be considered to be

$$N' = |\mathcal{V}| + 2. \tag{2}$$

C. Task Scheduling

As every local processor can process only a single task at a time, the task scheduling decision must also contain the order of the tasks allocated to the processor, which should satisfy the precedence constraints and the application deadline. We define the scheduling decision variables as follows:

$$x_{ijr} := \begin{cases} 1 & \text{if task } i \text{ is on processor } j \text{ in position } r, \\ 0 & \text{if otherwise,} \end{cases} \tag{3}$$

$\forall\, i = 1, \ldots N', j = 1, \ldots M'$ and $r = 1, \ldots N'$.

Each task is to be scheduled to exactly one of the existing positions on the processors. Hence,

$$\sum_{j=1}^{M'}\sum_{r=1}^{N'} x_{ijr} = 1, \qquad \forall i = 1, \ldots, N'. \tag{4}$$

Furthermore, each position on each processor can be assigned to atmost one task, which is given by

$$\sum_{i=1}^{N'} x_{ijr} \leq 1, \qquad \forall r = 1, \ldots, N', \quad j = 1, \ldots, M'. \tag{5}$$

The positions in each processor are filled by the tasks sequentially, i.e., until one position on a processor is occupied, tasks cannot be assigned to subsequent positions. This is imposed by the following constraint:

$$\sum_{i=1}^{N'} x_{ijr} - \sum_{i=1}^{N'} x_{ij(r-1)} \leq 0, \quad \forall r = 2, \ldots, N', \quad j = 1, \ldots, M'. \tag{6}$$

The two dummy tasks inserted are required to be scheduled on a local processor, so we have

$$\sum_{r=1}^{N'} x_{11r} = 1, \quad \sum_{r=1}^{N'} x_{N'1r} = 1. \tag{7}$$

Furthermore, our task scheduling decision is required to meet the application deadline, which imposes constraints on the finishing times of the tasks. If $\forall i = 1, \ldots N'$, F_i is the finish time of task i, then

$$F_{N'} \leq L \tag{8}$$

ensures that the last dummy task, and consequently the overall application, is completed by the deadline. In addition,

$$F_1 = 0 \tag{9}$$

sets the finish time of the first dummy task to zero as it has zero execution time.

The relationship between the finish times of the tasks and the decision variables is given by

$$F_i - F_k + C(2 - x_{ijr} - x_{kj(r-1)}) \geq t_{ij}, \quad \forall i, k = 1, \ldots N', \quad r = 2, \ldots, N', \quad j = 1, \ldots, M \tag{10}$$

where we assign C to be a large positive number. This ensures that the finish time of a task in sequence on a local processor is at least equal to the sum of the finish time of the preceding task and the processing time of the present task. Note that $2 - x_{ijr} - x_{kj(r-1)}$ is zero if and only if tasks k and i are placed consecutively on processor j.

The tasks of the application are dependent in nature. This implies that the finish time of a task must be greater than its predecessor by the amount of its execution time and communication time from its predecessor. Thus, we have

$$F_i - F_k \geq \sum_{r=1}^{N'}\sum_{j=1}^{M'} t_{ij}x_{ijr} + e_{ki}d\sum_{t=1}^{N'}\sum_{r=1}^{N'}\sum_{j=1}^{M}(x_{ijr}x_{kM't} + x_{iM'r}x_{kjt}), \quad \forall i = 1, \ldots N', \quad (k,i) \in \mathcal{E}. \tag{11}$$

The first term of (11) is the execution time, and the second term is the communication time, which occurs when task i is executed on one of the M local processors and k is executed on the cloud or vice-versa.

D. Minimizing the Cost of Application Execution

The total cost of execution of the application is the sum of the total execution cost and the total communication cost. Our goal is to identify the schedule that minimizes this total cost, subject to the deterministic application deadline, L. This can be formulated as an optimization problem as follows:

$$\underset{\{x_{ijr}\}}{\text{minimize}} \sum_{r=1}^{N'}\sum_{j=1}^{M'}\sum_{i=1}^{N'} p_j t_{ij} x_{ijr} + p^c d \sum_{k=1}^{N'}\sum_{i=1}^{N'} e_{ki}\left[\sum_{t=1}^{N'}\sum_{r=1}^{N'}\sum_{j=1}^{M}(x_{ijr}x_{kM't} + x_{iM'r}x_{kjt})\right], \tag{12}$$

$$\text{subject to} \quad (4) - (11), \tag{13}$$

$$x_{ijr} \in \{0,1\}, \quad i = 1, \ldots N', r = 1, \ldots, N', j = 1, \ldots, M'. \tag{14}$$

Constraint (14) forces the decision variables to take on binary values.

IV. Proposed Approaches

The proposed problem is NP-hard in nature as the Generalized Assignment Problem (GAP), which is a special case of this problem, is NP-hard. Hence, there is no guarantee of obtaining the optimal solution in polynomial time. As a reasonable alternative, we propose the CALPS algorithm, which is an efficient heuristic. Furthermore, for benchmarking, we also present a lower bound to the optimal solution by continuity relaxation.

A. CALPS Algorithm

This algorithm consists of the following major steps.

- **Step 1:** Allocate deadlines to the individual tasks derived from the overall application deadline.
- **Step 2:** Pick an unscheduled task i whose predecessors have already been scheduled. Assign it to the processor j wherein it completes its execution before its individual deadline and produces minimum additional cost.
- **Step 3:** Repeat Step 2 untill all tasks are scheduled or infeasibility is revealed.

Step 1 of this algorithm sets, for each task i, the values LST_i and LFT_i, which refer to the latest starting time and the latest finish time of task i respectively. These values are computed as follows:

$$\text{LST}_i := \begin{cases} L & \text{if } i = N' \\ \min\limits_{k:(i,k)\in\mathcal{E}}(\text{LST}_k - \min\limits_{j\in\mathcal{P}} t_{ij} - wde_{ik}) & \text{if otherwise,} \end{cases} \tag{15}$$

$$\text{LFT}_i := \begin{cases} L & \text{if } i = N' \\ \text{LST}_i + \min\limits_{j\in\mathcal{P}} t_{ij} & \text{if otherwise,} \end{cases} \tag{16}$$

where w is a parameter used to tune the importance of communication delay for individual tasks.

Once the individual deadlines are set in Step 1, Step 2 of the algorithm aims at scheduling a processor s_i to each task i. For each task, the algorithm calculates its accumulated execution delay d_{ij} and cost c_{ij}, due to the execution of i on processor j, according to the following equations:

$$d_{ij} = \max_{(k,i)\in\mathcal{E}} (\mathrm{ST}_k + t_{ks_k} + D_{ki}) \tag{17}$$

$$c_{ij} = p_j t_{ij} + p^c \sum_{(k,i)\in\mathcal{E}} D_{ki} \tag{18}$$

where

$$D_{ki} := \begin{cases} de_{ki} & \text{if task } k \text{ and } i \text{ are in different locations} \\ 0 & \text{if otherwise} \end{cases}$$

and ST_k is the actual starting time of task k.

The accumulated delay in (17) does not take into account the waiting time for a task on a processor if the processor is already running another task. We, thus, keep a tab on the total busy time or schedule length SL_j for each processor j. Thus, in order for a task to complete execution by its deadline, the following condition must be satisfied:

$$\max\{d_{ij}, \mathrm{SL}_j\} + t_{ij} \leq \mathrm{LFT}_i. \tag{19}$$

We then schedule task i to processor s_i as follows:

$$s_i = \underset{j}{\operatorname{argmin}}\{c_{ij} \text{ given (19) is satisfied}\}. \tag{20}$$

If there is no processor for which (19) is satisfied, infeasibility occurs and the algorithm fails to produce a schedule corresponding to the given application deadline. Alternatively, if all the tasks have been scheduled to some processor in accordance with the deadlines, then a feasible decision is obtained. These two possibilities result in termination of the algorithm as indicated in Step 3.

An important component of the algorithm is the scaling factor w which mainly allows us to consider possible communication delay between tasks by accounting for the weighted delay while setting task deadlines. Additionally, this ensures that the initial tasks do not take away a large chunk of the overall deadline and helps maintain nearly uniform priority for all tasks by countering the greedy aspect of the algorithm.

The details of CALPS are given in Algorithm 1.

Algorithm 1 CALPS algorithm

Input: DAG $G = \langle \mathcal{V}, \mathcal{E} \rangle$, $\mathcal{P}$, L.
Output: Decision variables $\{x_{ijr}\}$ defined in (3)

```
for all unmarked task i ∈ V whose predecessors are all marked do
    LST_i ← from (15)
    LFT_i ← from (16)
    Mark task i
end for
SL_j ← 0 for all j ∈ P
ST_i ← 0 for all i ∈ V
while there exist tasks not scheduled do
    Choose unscheduled task i with minimum LST_i
    for all j ∈ P do
        Calculate d_ij from (17)
        Calculate c_ij from (18)
    end for
    Find s_i from (20)
    if s_i = ∅ then
        No feasible decision produced.
        return
    end if
    ST_i ← max{d_ij, SL_i} {Setting actual starting time}
    if s_i ≤ M then
        SL_{s_i} ← ST_i + t_{is_i} {Updating schedule length for local processors}
    end if
    x_ijr ← 0 for all i, j and r
    Sort the tasks scheduled to each single processor in increasing order of ST_i and obtain their positions r_i.
    for all i ∈ V do
        x_{is_i r_i} = 1
    end for
end while
```

B. Lower Bound to the Optimum

Since the proposed formulation in Section III is NP-hard, we cannot obtain an optimal solution in reasonable time for large systems in order to evaluate the performance of the CALPS algorithm. Thus, it helps to obtain a lower bound to the optimal solution.

The formulated optimization problem is a mixed integer program and it is non-convex in nature due to (12), (11), and (14). However, using the following continuity relaxation technique, we can obtain a convex problem on variables $\{x_{ijr}\}$:

- Replace the integer constraints in (14) with linear constraints and simply restrict the decision variables to be positive.
- Replace the $x_{ijr}x_{iM't}$ and $x_{iM'r}x_{ijt}$ terms in (12) and (11) by max(x_{ijr}+$x_{iM't}$-1,0) and max($x_{iM'r} + x_{ijt}$-1,0) respectively, which are equivalent for binary variables.

Thus, equations (11), (12), and (14) of the problem are replaced with (22), (21), and (23) below, respectively. In particular, replacing (14) with (23) implies that we now allow a single task to be distributed and executed partially across several processors and positions.

$$\begin{aligned} \underset{\{x_{ijr}\}}{\text{minimize}} & \sum_{r=1}^{N'} \sum_{j=1}^{M'} \sum_{i=1}^{N'} p_j t_{ij} x_{ijr} \\ & + p^c d \sum_{k=1}^{N'} \sum_{i=1}^{N'} e_{ki} [\sum_{t=1}^{N'} \sum_{r=1}^{N'} \sum_{j=1}^{M} \max(x_{ijr} + x_{iM't} - 1, 0) \\ & + \max(x_{iM'r} + x_{ijt} - 1, 0)]. \end{aligned} \tag{21}$$

$$F_i - F_k \geq \sum_{r=1}^{N'} \sum_{j=1}^{M'} t_{ij} x_{ijr} + e_{ki} d \sum_{t=1}^{N'} \sum_{r=1}^{N'} \sum_{j=1}^{M} [\max(x_{ijr} + x_{iM't} - 1, 0) + \max(x_{iM'r} + x_{ijt} - 1, 0)], \ \forall i = 1, \ldots N', \ (k,i) \in \mathcal{E}. \quad (22)$$

$$x_{ijr} \geq 0, \qquad i = 1, \ldots N', \ r = 1, \ldots, N', \ j = 1, \ldots, M'. \quad (23)$$

After continuity relaxation, one may discretize the resultant fractional solution to obtain binary values for $\{x_{ijr}\}$. The following procedure is based on the discretization approach commonly adopted in the literature:

1) Pick an unscheduled task i whose predecessors are already scheduled.
2) Find $[j', r'] = \text{argmax}\, x_{ijr}$.
3) Check if this value of $x_{ij'r'}$ satisfies all the constraints.
   a) If so, mark i as scheduled and go to Step 1.
   b) If not, set $x_{ij'r'} = 0$. If $x_{ijr} = 0 \ \forall j$ and r, no feasible solution is obtained, else go to Step 2.

However, our numerical experiments based on a wide range of parameter settings indicate that this common relaxation-discretization approach generally performs poorly. In fact, due to the many restrictive constraints on $\{x_{ijr}\}$, including (5), (6), and (10), this approach often cannot give a feasible binary solution for $\{x_{ijr}\}$. Therefore, we conclude that continuity relaxation for this problem only serves to provide a lower bound to the optimal objective, for numerical performance benchmarking.

V. Simulation Results

In order to assess our proposed CALPS algorithm, we run simulation for different scenarios and investigate its cost and feasibility performance. In our simulation, the DAGs are randomly generated in terms of the graph structure, task execution times on the processors, and input/output data between tasks. For each parameter setting, we run several of these randomly generated iterations and plot the average. For each sample point, we run the CALPS algorithm for 25 different values of scaling factor w ranging from 0 to 6, and we pick the one that gives the lowest cost.

Figure 1 depicts the cost performance of the CALPS algorithm, the lower bound, and the optimal solution, for $|\mathcal{V}| = 5$ and $|\mathcal{E}| = 8$. We consider energy as the cost in our formulation, so that the values that we use in our simulation correspond to the practical values involved if the objective is to minimize the total energy consumption. We assume two local processors, with $p_1 = 0.944$ watt [13], $p_2 = 0.4$ watt, and $t_{i2} = 2t_{i1}$ for all tasks i, and a remote cloud with $p_3 = 10$ watt and $t_{i3} = 0.12t_{i1}$ for all tasks i. Here, $t_{i1} = \frac{\text{number of cycles}}{1.2\text{GHz}}$ wherein the processor speed is 1.2GHz and the number of cycles is drawn from a uniform distribution in the interval (100, 200) mega cycles. The DAG structures are randomly generated and dummy nodes are inserted. The edge weights are then drawn uniformly from the interval (1, 2) MB and p^c is taken to be 0.935 watt [13]. The optimal and lower bound solutions are obtained using the CVX programming package, by employing the Gurobi solver for the original optimization problem and the SDPT3 solver for the lower-bound convex program. We study the variation of cost with application deadline L, for multiple values of communication delay, d.

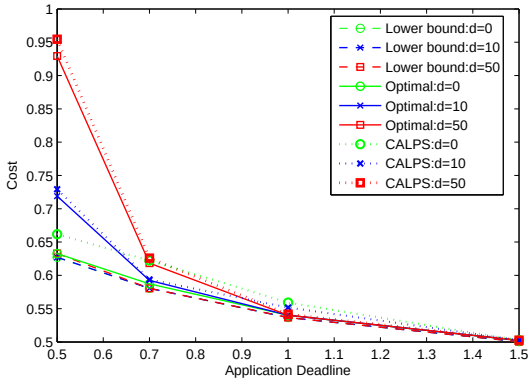

Fig. 1. Cost vs. application deadline for small applications.

We see that the cost performance of CALPS is very close to that of the optimum for all the different values of application deadline and communication delay. We see that it follows a similar trend to the optimal and the lower bound, wherein the cost decreases with increase in the application deadline. Though not shown here, we further observe that CALPS has 100% feasibility in this entire range with respect to the optimum.

We are also interested in investigating the performance of CALPS for larger systems. However, all of the existing work in literature dealing with minimizing cost/energy under an application deadline has assumed that the local device can run any number of tasks simultaneously without affecting the processing time of each task. In particular, the work in [8] uses dynamic programming to obtain an optimal solution for this problem. We study the performance of the algorithm proposed in [8] when applied to our practical system, allowing only a finite number of tasks to be run simultaneously. In other words, we run their algorithm and obtain a scheduling decision and use this decision in our system by queuing the tasks when their number exceeds M. Figures 2 and 3 depict the feasibility and cost, where the 'Dyn Prog' curves refer to the results obtained from [8] by adopting the aforementioned technique. For fair comparison, we only plot those cost points corresponding to feasibility $> 70\%$. We take $|\mathcal{V}| = 15$, $|\mathcal{E}| = 25$, $d = 20$ and vary the number of local processors M. We assume in this case that the local processors are identical with $p_j = 0.944$ and t_{ij} is the same for task i on every processor j.

We see that the feasibility performance of the CALPS algorithm is much better than the dynamic programming algorithm. As the number of local processors increases, the feasibility improves and is nearly the same (barring minor quantization error) for $M = 16$, which is equivalent to the case of infinite capacity for an application of size $|\mathcal{V}| = 15$. Thus, for large practical application sizes, we expect the dynamic

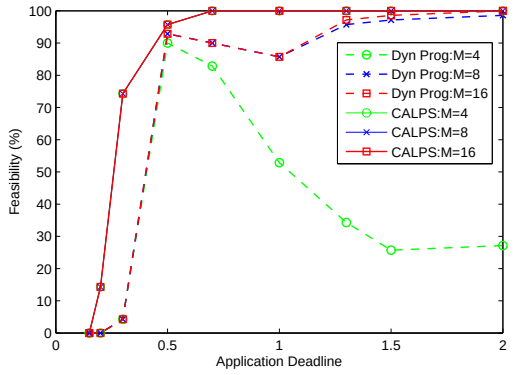

Fig. 2. Feasibility vs. application deadline for large applications.

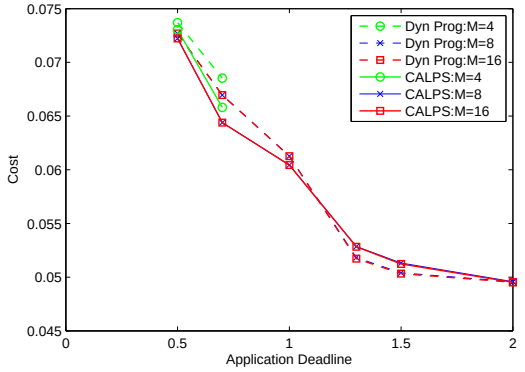

Fig. 3. Cost vs. application deadline for large applications.

TABLE II
RUN-TIME COMPARISON

M	**Dyn Prog** ($K = 100$)	**CALPS** ($\|\mathcal{W}\| = 25$)
4	0.3304	0.08
8	0.3258	0.095
16	0.3253	0.1775

programming algorithm to perform poorly when subjected to our finite-capacity local device. Further, it runs in $\Theta(|\mathcal{V}|DK)$ time where D is the maximum in-degree of the task graph and K is the number of quantisation levels. We take $K = 100$ and compare the run-time of this algorithm with CALPS, $\mathcal{O}(|\mathcal{W}|M(|\mathcal{V}|+|\mathcal{E}|))$ where $|\mathcal{W}|$ is the set of all scaling factor w values used, in Table II. We observe that CALPS has much lower complexity in typical scenarios.

The figures so far depict the performance after trying out multiple values of w and picking the best one, namely, w_{opt}, for each application. We can improve the run-time of the algorithm if we can pick a single suitable value of w. Toward this goal, in Figure 4, we plot the average w_{opt} over all applications, for the simulation scenario of Figure 1. We observe that w_{opt} is sensitive to the application deadline. It is an increasing function when the application deadline is small to moderate, and then it decreases when the application deadline is more relaxed.

VI. CONCLUSION

We have formulated an optimization problem to find the minimum cost in scheduling dependent tasks under a deterministic application deadline for a practical system model consisting of finite-capacity local processors and the cloud. We propose the CALPS algorithm to obtain a polynomial-time solution for this NP-hard problem. Through simulation, we compare its performance with the optimal solution and other alternatives. We also use simulation to identify suitable scaling factors to improve its run-time. Our proposed algorithm and formulation can also be extended to the case of computational offloading to peer devices in addition to the remote cloud.

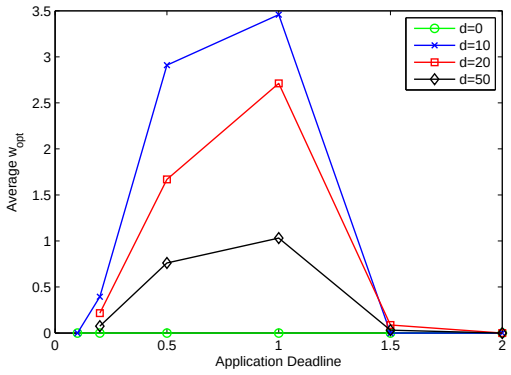

Fig. 4. Average optimal scaling factor vs. application deadline.

REFERENCES

[1] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proc. ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pp. 49–62, 2010.

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

[3] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[4] N. Vallina-Rodriguez and J. Crowcroft, "Erdos: achieving energy savings in mobile os," in *Proc. ACM workshop on MobiArch*, pp. 37–42, 2011.

[5] M. Satyanarayanan, "Mobile computing: the next decade," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 15, no. 2, pp. 2–10, 2011.

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

[7] P. Balakrishnan and C.-K. Tham, "Energy-efficient mapping and scheduling of task interaction graphs for code offloading in mobile cloud computing," in *Proc. IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pp. 34–41, 2013.

[8] B. Y.-H. Kao and B. Krishnamachari, "Optimizing mobile computational offloading with delay constraints," in *Proc. Global Communication Conference (Globecom)*, pp. 8–12, 2014.

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

[10] M. Jia, J. Cao, and L. Yang, "Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing," in *Proc. IEEE INFOCOM Workshop on Computer Communications*, pp. 352–357, 2014.

[11] Y. Wen, W. Zhang, and H. Luo, "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones," in *Proc. IEEE INFOCOM*, pp. 2716–2720, 2012.

[12] M.-A. Hassan Abdel-Jabbar, I. Kacem, and S. Martin, "Unrelated parallel machines with precedence constraints: application to cloud computing," in *Proc. IEEE International Conference on Cloud Networking (CloudNet)*, pp. 438–442, 2014.

[13] B. Flipsen, J. Geraedts, A. Reinders, C. Bakker, I. Dafnomilis, and A. Gudadhe, "Environmental sizing of smartphone batteries," in *Proc. IEEE Electronics Goes Green (EGG)*, pp. 1–9, 2012.