Offloading Dependent Tasks with Communication Delay and Deadline Constraint

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

***Abstract*—We study the scheduling decision for an application consisting of dependent tasks, in a generic cloud computing system comprising a network of heterogeneous local processors and a remote cloud server. We formulate an optimization problem to find the offloading decision that minimizes the overall application execution cost, subject to an application completion deadline. Since this problem is NP-hard, we propose a heuristic algorithm termed Individual Time Allocation with Greedy Scheduling (ITAGS) to obtain an efficient solution. ITAGS first uses a binary-relaxed version of the original problem to allocate a completion deadline to each individual task, and then greedily optimizes the scheduling of each task subject to its time allowance. Through trace-based simulation using real applications, as well as various randomly generated task trees, we study the performance of ITAGS, highlighting the effect of the application deadline, communication delay, number of processors, and number of tasks. We further demonstrate the substantial performance advantage of ITAGS over existing alternatives.**

I. Introduction

Computational offloading refers to the migration of computationally-intensive portions of an application, typically from a resource-limited mobile device, to powerful servers at the cloud [1], [2]. An application can be partitioned into a number of tasks, and task scheduling decisions can be made such that the overall offloading leads to reduced energy consumption [3], [4], makespan [5], [6], [7], or combination of both [1], [8], [9]. This allows flexibility in offloading tasks in finer granularity than entire applications. Nevertheless, offloading to the remote cloud can incur significant delays, particularly if a large amount of data needs to be communicated between the cloud and the mobile device. Hence, cloudlets and edge servers that are in closer proximity to the mobile device provide an attractive alternative to offload tasks that require low latency [10], [11]. Tasks can also be offloaded to peer devices such as nearby mobiles [12], [13], [14]. Therefore, the future landscape of computing is expected to be highly networked, with diverse sources of computation services.

In this paper, we consider the offloading of a single application comprising multiple tasks, over a generic cloud computing system consisting of a network of heterogeneous local processors and a remote cloud. The local processors can represent the processing cores in a single mobile device, local peer devices, and/or nearby cloudlets, depending on their computational speed and communication distance from the user device. There is a time and a cost associated with task execution, which depend on both the task and the processor where the task is scheduled.

While many prior studies in the literature consider the offloading of independent tasks [6], [8], [10], [12], [15], [16], [17], the computational tasks of an application often depend on each other for data input or timing precedence. Hence, in this work, we consider inter-dependent tasks with possible data communication between them. Each task may have predecessor tasks that must be completed before the task can start. Furthermore, if data need to be transferred between tasks on different processors, a communication delay, as well as communication cost, is incurred.

The objective of this work is to identify a task scheduling decision that minimizes the total cost of running an application, subject to an application completion deadline. Our cost model is general, which for example may include energy consumption or usage charges for task processing and communication. We observe that the precedence constraints and data transfer requirement between tasks can drastically complicate their scheduling decision [3], [4], [5], [7]. Furthermore, the need to account for both the cost and the run-time of the application adds to the challenge [1], [9], [18], [19], [20]. Prior studies have assumed simplified processor models to facilitate tractable analysis, such as non-concurrent local and remote processors [1], infinite-capacity local processors [9], [18], [19], and negligible delay between local processors [21], [20]. We use a more realistic processor model in this study. Our problem can be shown to be NP-hard, and, as such, there is no polynomial run-time guarantee for finding an optimal solution.

The contributions of this work are as follows:

- We formulate a problem of cost minimization in scheduling an application with dependent tasks and a completion deadline, over a generic cloud computing system with heterogeneous processors and communication delay. Since the problem is non-convex, we first relax its binary constraints to obtain a convex problem and a lower bound to the optimal objective of the original problem.
- We observe that a scheduling solution obtained by directly discretizing the binary-relaxed solution does not provide satisfactory performance. Therefore, we propose a new heuristic algorithm, termed Individual Time Allo-

This work has been funded in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

978-1-5386-4128-6/18/$31.00 ©2018 IEEE

cation with Greedy Scheduling (ITAGS), which utilizes the binary-relaxed solution to allocate a completion deadline to each individual task and then greedily optimizes the scheduling of each task subject to its time allowance.
- Through trace-based simulation with real-world applications from [22], as well as various randomly generated task trees, we study the impact of the application deadline and other system settings on the performance of ITAGS. We further compare ITAGS with the dynamic programming approach from [9], [18], other alternatives including the above discretization heuristic, and the cost lower bound, demonstrating its superior effectiveness.

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 motivation and details of ITAGS. Section V presents the simulation results, and concluding remarks are given in Section VI.

II. Related Work

The offloading of dependent tasks is complicated by the need to satisfy precedence constraints and data transfer requirement. Earlier works on offloading dependent tasks often focus on a single performance metric, either energy or makespan. In [3] and [4], the authors aim to identify the task scheduling decision to reduce energy consumption. Both assume that the tasks on a mobile device and the cloud do not run simultaneously, which can be justified since no gain on energy can be achieved by parallel execution [4]. However, exploiting parallelism between the mobile device and the cloud can greatly improve the makespan of the application. For example, load-balancing heuristics [5] and genetic algorithms [7] have been proposed for makespan reduction with dependent tasks. However, in [5] the tasks dependency is assumed to be sequential, while in [7] no data communication is considered between tasks.

Cost minimization and application deadline are jointly considered in [1] and [9]. In [1], the authors aim to maximize the energy savings at a mobile device with task offloading, subject to an application deadline. Their formulation is based on the simplifying assumption that tasks on the mobile device and cloud cannot run simultaneously, and task computation and data communication cannot occur simultaneously. In [9], a dynamic programming algorithm is proposed for deterministic and stochastic application deadlines. Both [1] and [9] assume that the mobile device has infinite capacity and can execute any number of tasks simultaneously without impacting the processing time of each task. This assumption is unrealistic since user devices are usually resource limited. In our work, the computing system consists of finite-capacity devices. We allow data communication and task execution to take place simultaneously, as well as parallel execution between the local device and the cloud. Additionally, [1] and [9] only consider offloading from a mobile device to a remote cloud. In contrast, our modeling of the processor network is general, potentially consisting of a wide variety of mobile processors, edge computers, cloudlets, and remote cloud.

The problem of offloading dependent tasks to multiple types of processors have been considered in [18], [19], and [20]. In [18], a fully polynomial time approximation scheme is proposed to minimize the overall latency under a resource cost constraint while offloading dependent tasks to multiple devices. However, the devices are again assumed to possess infinite capacity in terms of the number of tasks that can be processed simultaneously without reduction in the processing speed for each task. The authors of [19] consider a similar model of offloading to a cluster of mobile devices and propose generic task scheduling heuristics with the objective of maximizing the overall useful computation by the cluster. In [21], the problem of scheduling an application consisting of dependent tasks is considered, and a heuristic algorithm is suggested. However, the local processor cores are assumed to exist on a single mobile device, and an objective of only energy consumption by the mobile device is considered. Similarly, in [20], we investigate the objective of cost minimization subject to an application deadline, for heterogeneous local and remote processors. However, the delay between the local processors is assumed to be negligible. In this work, we account for the delay between all processors in order to arrive at a general model that encompasses scenarios such as offloading to peer devices and cloudlets in addition to the cloud. The local processors have finite capacity, and there are time and cost associated with both task execution and data communication between any two locations. This leads to a unique problem formulation that has not been considered in the existing literature.

III. System Model and Problem Formulation

A. Local Processors and Remote Cloud

We consider a system with a finite number of local processors. These processors may be installed in mobile edge computing hosts, cloudlet devices, or peer mobile devices. These processors may have different speeds but are assumed to be *unary*, i.e., each processor executes one task at a time, while the other tasks assigned to the processor wait in a queue. We emphasize that, with respect to the cost and delay in task processing, this assumption is without loss of generality.[1]

We further assume a remote cloud center that provides an essentially infinite number of processors, possibly through leasing of virtual machines. Consequently, the remote cloud can be viewed as an additional processor having infinite capacity in terms of the number of tasks it can process simultaneously. Let the set of all processors, including the remote cloud, be $\mathcal{P}$ and its size be M. Let d_{ij} be the delay per unit data transfer between processors i and j. For simplicity of illustration, we assume $d_{ij} = d_{ji}$ and $d_{ij} = 0$ if $i = j$. An example of such a system is depicted in Figure 1.

[1]It is easy to see that there is no benefit in processor-sharing with respect to the sum queueing-and-execution delay of the tasks.

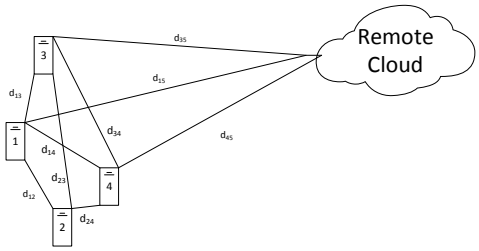


Fig. 1: Example network of local processors and cloud.

B. Task Dependency Graph

Consider a single application that must be completed before a deadline L. The application is partitioned into tasks, whose dependency 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, k cannot start before i finishes. Furthermore, if they are scheduled at different processors j and v respectively, the communication delay is $e_{ik}d_{jv}$ and the communication cost is $ce_{ik}d_{jv}$, where c is the communication cost per unit time. It is clear that the delay values are often smaller while offloading to nearby local processors in comparison with the delay to offload to the remote cloud.

If 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. In practice, the processing times and data transfer requirement may be obtained by applying a program profiler as shown in experimental studies such as MAUI [1] and Thinkair [2]. In this work, we proceed assuming that such information in already given.

We assume that an application is initiated at a particular local processor and must end at the same 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

$$N' = |\mathcal{V}| + 2. \tag{1}$$

C. Problem Formulation

The task scheduling decision contains both the mapping between tasks and processors and the order of the tasks allocated to each processor. We define the scheduling decision variables as follows:

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

for all $i = 1, \dots N', j = 1, \dots M$ and $r = 1, \dots N'$. Each task is to be scheduled to exactly one of the existing positions on the processors. Hence,

$$\sum_{j=1}^{M} \sum_{r=1}^{N'} x_{ijr} = 1, \qquad \forall i = 1, \dots, N'. \tag{2}$$

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

$$\sum_{i=1}^{N'} x_{ijr} \leq 1, \qquad \forall r = 1, \dots, N', \quad j = 1, \dots, M. \tag{3}$$

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:

$$\sum_{i=1}^{N'} x_{ijr} - \sum_{i=1}^{N'} x_{ij(r-1)} \leq 0, \qquad \forall r = 2, \dots, N', \quad j = 1, \dots, M. \tag{4}$$

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

$$\sum_{r=1}^{N'} x_{11r} = 1, \quad \sum_{r=1}^{N'} x_{N'1r} = 1. \tag{5}$$

Furthermore, our task scheduling decision is required to meet the application deadline, which imposes constraints on the finishing times of the tasks. Let F_i be the finish time of task i, for $i = 1, \dots N'$. Then

$$F_{N'} \leq L \tag{6}$$

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

$$F_1 = 0 \tag{7}$$

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

The relationship between the finish times of tasks on the same local processor j and the decision variables is given by

$$F_i - F_k + C(2 - x_{ijr} - x_{kj(r-1)}) \geq t_{ij}, \qquad \forall i, k = 1, \dots N', \quad r = 2, \dots, N', \quad j = 1, \dots, M, \tag{8}$$

where we assign a large positive number to C. This ensures that the finish time of a task on processor j 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.

Finally, since the tasks of the application are dependent, the finish time of a task must be greater than that of each of its predecessors by the amount of its predecessor's execution time and communication time from its predecessor. Thus, we have

$$F_i - F_k \geq \sum_{r=1}^{N'} \sum_{j=1}^{M} t_{ij} x_{ijr} + \sum_{j=1}^{M} \sum_{t=1}^{N'} \sum_{r=1}^{N'} \sum_{v=1}^{M} e_{ki} d_{vj} x_{ijr} x_{kvt}, \qquad \forall i = 1, \dots N', \quad (k, i) \in \mathcal{E}. \tag{9}$$

The first term on the right hand side of (9) is the total execution time, and the second term is the total data communication time,

which occurs when task i is executed on processor j and its predecessor k is executed on another processor v.

We define the total cost of application execution as 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 application deadline, L. This can be formulated as an optimization problem as follows:

$$\begin{aligned}\underset{\{x_{ijr}\}}{\text{minimize}} \quad & \sum_{r=1}^{N'}\sum_{j=1}^{M}\sum_{i=1}^{N'} p_j t_{ij} x_{ijr} \\ & + \sum_{i=1}^{N'}\sum_{k=1}^{N'}\sum_{j=1}^{M}\sum_{v=1}^{M}\sum_{r=1}^{N'}\sum_{t=1}^{N'} ce_{ik} d_{jv} x_{ijr} x_{kvt}, \quad (10)\\ \text{subject to} \quad & (2)-(9), \\ & x_{ijr} \in \{0,1\}, \quad \forall i = 1,\dots N', \\ & \qquad r = 1,\dots,N',\ j = 1,\dots,M. \quad (11)\end{aligned}$$

This problem is NP-hard since it contains the Generalized Assignment Problem (GAP) as a special case, and GAP is NP-hard. Hence, we do not expect to find an optimal solution in polynomial time. Consequently, we propose the ITAGS algorithm and study its effectiveness in solving this problem.

IV. Individual Time Allocation with Greedy Scheduling (ITAGS)

The ITAGS algorithm is built on the concept of appropriately allocating the application deadline among the individual tasks. To provide a guideline on the suitable amount of individual time allocation, we first consider a binary-relaxed version of the original problem in the next subsection. We follow that by discussing how one might design, as an inferior alternative to ITAGS, a feasible binary solution via direct discretization. We then present the details of ITAGS, concluding with a discussion of its feasibility and computational complexity.

A. Binary Relaxation and Individual Time Allowance

Optimization problem (10) is a mixed integer program, and it is non-convex due to its non-convex objective and constraints (9). However, we note that the communication delay and cost terms in (9) and (10) can be modified as follows:

$$\begin{aligned}& \sum_{j=1}^{M}\sum_{t=1}^{N'}\sum_{r=1}^{N'}\sum_{v=1}^{M} e_{ki} d_{vj} x_{ijr} x_{kvt} \quad (12)\\ =& \sum_{j=1}^{M}\sum_{v=1}^{M} e_{ki} d_{vj} \left(\sum_{r=1}^{N'} x_{ijr}\right)\left(\sum_{t=1}^{N'} x_{kvt}\right) \\ =& \sum_{j=1}^{M}\sum_{v=1}^{M} e_{ki} d_{vj} \max\left[\left(\sum_{r=1}^{N'} x_{ijr}\right)+\left(\sum_{t=1}^{N'} x_{kvt}\right)-1, 0\right], \quad (13)\end{aligned}$$

where the last equality holds because $\{x_{ijr}\}$ are binary. This converts the non-convex (12) to a convex form in (13).

Therefore, we perform the following two-step binary relaxation on the original problem:

- Replace the communication terms in (10) and (9) by (13);
- Replace the binary constraints in (11) with linear constraints by simply restrict the decision variables to be non-negative.

This leads to the following convex problem over decision variables $\{x_{ijr}\}$:

$$\begin{aligned}\underset{\{x_{ijr}\}}{\text{minimize}} \quad & \sum_{r=1}^{N'}\sum_{j=1}^{M}\sum_{i=1}^{N'} p_j t_{ij} x_{ijr} \\ & + \sum_{j=1}^{M}\sum_{v=1}^{M} e_{ki} d_{vj} \max\left[\left(\sum_{r=1}^{N'} x_{ijr}\right)+\left(\sum_{t=1}^{N'} x_{kvt}\right)-1, 0\right] \quad (14)\\ \text{subject to} \quad & (2)-(8), \\ & F_i - F_k \geq \sum_{r=1}^{N'}\sum_{j=1}^{M} t_{ij} x_{ijr} \\ & + \sum_{j=1}^{M}\sum_{v=1}^{M} e_{ki} d_{vj} \max\left[\left(\sum_{r=1}^{N'} x_{ijr}\right)+\left(\sum_{t=1}^{N'} x_{kvt}\right)-1, 0\right], \\ & \qquad \forall i = 1,\dots N',\ (k,i) \in \mathcal{E}. \quad (15)\\ & x_{ijr} \geq 0, \quad \forall i = 1,\dots N', \\ & \qquad r = 1,\dots,N',\ j = 1,\dots,M. \quad (16)\end{aligned}$$

An optimal solution to problem (14) can be efficiently computed using convex programming solvers such as CVX. Note that replacing (11) with (16) is equivalent to allowing a single task to be distributed and executed partially across several processors and positions. This is unrealistic, but solving this relaxed problem is useful for two purposes. First, since the relaxed problem has a larger feasible set, it serves as a lower bound to the optimum of the original problem, which can be used for numerical performance benchmarking. Second, the relaxed solution can be leveraged to recover a binary solution to the original problem. In particular, as a part of the ITAGS algorithm, it supplies the individual time allowance for each task as explained in Section IV-C.

B. Alternative Discretization Heuristic

Before presenting the details of ITAGS, we first consider a conventional approach to recover a binary solution from the above relaxed solution, by discretizing the fractional solution x_{ijr} to binary values. We will show later that such an approach, although non-trivial, does not provide satisfactory performance. Therefore, it will be used mainly for performance benchmarking against ITAGS.

We note that discretizing the fractional x_{ijr} solutions is challenging. Directly rounding them to binary values will violate some constraints of the original problem. In particular, the constraints on relative positions of tasks on a processor need to be taken into consideration, to ensure that the scheduled tasks are in proper order to satisfy the dependency requirement. Consequently, we consider the following algorithm, term *discretization heuristic*, which 1) disregards the task positions in the relaxed solution, 2) schedules each task to a processor

based on the fractional solution, and 3) calculates the resultant task starting times to obtain their relative position values for the final binary solution.

1) Reduction to Task-on-Processor: In this step, we assign x_{ijr} values to their corresponding task-on-processor variables y_{ij} as follows:

$$y_{ij} = \sum_{r=1}^{N'} x_{ijr}, \quad \forall i = 1, \dots N',\ j = 1, \dots, M. \tag{17}$$

Thus, the y_{ij} variables contain just the fractional solution for each task i on each processor j, which disregards the position information in x_{ijr}. It should be noted that y_{ij} obey the scheduling constraint: $\sum_{j=1}^{M} y_{ij} = 1, \forall i = 1, \dots N'$.

2) Discretization: We next discretize the fractional y_{ij} solutions to decide the processor assignment decision s_i for every task i by picking the processor that has the maximum y_{ij} value. The intuition behind this is that a y_{ij} value can be viewed as the probability of scheduling task i on processor j, and thus we take the decision with the highest probability:

$$s_i = \underset{j}{\operatorname{argmax}}\ y_{ij}, \quad \forall i = 1, \dots N'. \tag{18}$$

Thus, the decision for every task i is

$$y_{ij} := \begin{cases} 1 & \text{if } j = s_i, \\ 0 & \text{if } j \neq s_i. \end{cases}$$

3) Mapping to Positions: Although we have determined the processor on which each task needs to be scheduled, we still need to decide the positions of tasks on each processor, or the starting times for each task. Towards this end, we sort the tasks in the order of increasing F_i values from the solution to the relaxed problem (14). This sorting will ensure that the precedence constraints in (9) are obeyed between any two consecutive tasks. Thus, scheduling the tasks to their assigned processors in this order will give us their corresponding positions and starting time values.

4) Feasibility Check: For the above task schedule, we check if the total delay meets the application deadline L. If so, the corresponding cost is the resulting solution, or else the algorithm fails to produce a feasible schedule. In the latter case, we will use the same fallback procedure as ITAGS described below, to offload all tasks to the cloud.

C. ITAGS Algorithm

The task scheduling decision is required to meet the overall application completion deadline. A purely greedy algorithm might schedule each task to the processor where it achieves the minimum cost such that the overall application deadline is still met. However, such an algorithm prioritizes the tasks that are scheduled in the beginning as these tasks would be able to take away a larger chunk of the overall deadline allowance and make cost-effective decisions for themselves. On the other hand, the tasks that are scheduled later in the greedy process would have a relatively smaller portion of the overall deadline allowance available, resulting in possible infeasibility and performance degradation.

Thus, the guiding principle behind the design of ITAGS is that the overly greedy aspect of the above approach should be countered, by assigning individual deadlines to the tasks to ensure uniform priority for all tasks regardless of their scheduling order. ITAGS consists of three major steps: 1) Set individual time allowance for each task; 2) Schedule each task to a processor based on a greedy approach subject to its individual deadline; and 3) Check feasibility by testing if the last task meets the overall application deadline.

1) Individual Time Allocation: In Step 1, we identify the time allowance to be given to each task. This is achieved by performing binary relaxation on the original problem as detailed in Section IV-A, and solving the relaxed problem to obtain the finish time F_i for each task i. These finish times are treated as individual task deadlines in the next step.

2) Task Scheduling: Once the individual deadlines are set in Step 1, Step 2 of ITAGS aims at assigning a processor s_i for each task i. This task scheduling process has a principled greedy nature as the algorithm takes one task at a time and schedules it to the processor where the task 1) can complete its execution before its individual deadline and 2) incurs minimum additional cost.

ITAGS schedules the tasks starting from the top of the DAG and works its way down to the bottom. Specifically, the tasks are scheduled in the increasing order of individual deadline F_i. We note that this ensures that a task is scheduled only after its predecessors have been scheduled since (15) ensures that the F_i value of task i exceeds that of its predecessors. The topmost task is the first dummy task, and it is always scheduled to the local processor where the application is initiated. Then, as ITAGS moves down the list of unscheduled tasks, for each task i, we decide its start time ST_i and processor s_i.

First, for each potential processor j, we compute the accumulated execution delay D_{ij} and cost C_{ij}, due to the execution of i on processor j, as follows:

$$D_{ij} = \max_{(k,i)\in\mathcal{E}} (\text{ST}_k + t_{ks_k} + T_{ki}) \tag{19}$$

$$C_{ij} = p_j t_{ij} + c \sum_{(k,i)\in\mathcal{E}} T_{ki} \tag{20}$$

where $T_{ki} = d_{s_k j} e_{ki}$ is the communication delay from processor s_k to processor j with respect to the data from task k to task i. In (19), the sum inside $\max$ calculates the time when a parent task k completes execution and its data transfer to task i has arrived at processor j. Therefore, D_{ij} is the earliest start time of task i on processor j, by taking into account all parents of task i. Note that if both tasks k and i are scheduled onto the same processor, i.e., s_k=j, then the communication delay per unit data $d_{s_k j} = 0$ and consequently $T_{ki} = 0$.

However, knowing D_{ij} is not sufficient to decide whether task i should be place on processor j, since D_{ij} does not take into account the waiting time for a task on processor j if the processor is local and is already executing another task. Therefore, we keep a tab on the end of busy time on each local processor j, denoted by SL_j, and we update it every

time a task is scheduled onto the processor. In other words, every time that some task k is assigned to processor j, we set

$$\mathrm{SL}_j = \mathrm{ST}_k + t_{kj}. \tag{21}$$

This takes into account the amount of time that a task will have to wait for processor j if it is assigned to this processor. Note that for the remote cloud M, SL_M is always zero as we assume that the cloud has infinite capacity in terms of the number of tasks it can process simultaneously, resulting in zero waiting time for any task scheduled to the cloud.

As a result, the start time of a task i assigned to processor j is the maximum of the accumulated execution delay D_{ij} and current end of busy time SL_j. Thus, in order for task i to complete execution by its individual deadline F_i, the following condition must be satisfied:

$$\max\{D_{ij}, \mathrm{SL}_j\} + t_{ij} \leq F_i. \tag{22}$$

We then choose processor s_i to schedule task i as follows:

$$s_i = \begin{cases} \mathrm{argmin}_{j \in J_i} C_{ij} & \text{if } J_i \neq \emptyset \\ \mathrm{argmin}_j D_{ij} & \text{if } J_i = \emptyset \end{cases} \tag{23}$$

where J_i is the set of all processors for which (22) is satisfied for task i. From (23) we see that if the individual deadline F_i is too tight and cannot be met by any processor, ITAGS gracefully falls back to a greedy-time algorithm, i.e., one that tries to minimize makespan.

3) Feasibility Check: The process outlined in Step 2 is repeated until the last dummy task. This dummy task is to be scheduled to the local processor that initiated the application in order to obtain the results at the initiating device. If this last task does not meet the overall application deadline L, infeasibility occurs and the algorithm fails to produce a feasible schedule. Alternatively, if every task has been scheduled successfully to some processor, then a feasible decision is obtained. These two possibilities result in termination of the algorithm and constitutes Step 3.

The details of ITAGS are given in Algorithm 1.

Algorithm 1 ITAGS algorithm (after Step 1)

Input: DAG $G = \langle \mathcal{V}, \mathcal{E} \rangle$, $\mathcal{P}$, L, and solution to problem (14).
Output: Scheduling decision variables $\{x_{ijr}\}$

```
SL_j ← 0 for all j ∈ P
ST_i ← 0 for all i ∈ V
s_1 = 1 {Schedule first dummy task to initiating processor}
while there exist tasks not scheduled do
    Choose unscheduled task i with minimum F_i
    for all j ∈ P do
        Calculate D_ij from (19)
        Calculate C_ij from (20)
    end for
    if i = N' then
        s_N' = 1 {Schedule last dummy task to initiating processor}
    else
        Find s_i from (23)
    end if
    ST_i ← max{D_{i s_i}, SL_{s_i}} {Setting actual starting time}
    if s_i < M then
        SL_{s_i} ← ST_i + t_{i s_i} {Updating the end of busy time for local processors}
    end if
end while
if D_{N' s_N'} > L then
    No feasible decision produced.
    return
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_{i s_i r_i} = 1
end for
```

D. Feasibility and Complexity Analysis

For our NP-hard optimization problem (10), neither the discretization heuristic nor ITAGS provides a feasibility guarantee. Consequently, for practical cases, we may consider a fallback option that simply offloads all tasks belonging to the application to the remote cloud, if a feasible solution is not found by the algorithm. Such a fallback option is applicable under the assumption that the cloud has fast processors and high-speed access, so that offloading to the cloud can meet the overall application deadline, albeit with an added cost.

The computational complexity of the discretization heuristic is $\mathrm{O}(2|\mathcal{V}| + |\mathcal{E}|)$. On the other hand, the computational complexity for the ITAGS algorithm excluding the time to compute the lower bound solution is $\mathrm{O}(|\mathcal{P}|(|\mathcal{V}| + |\mathcal{E}|))$, which is polynomial with respect to the size of the application. The time to compute the lower bound is dependent on the algorithms used by the software to arrive at the solution. Assuming that a primal barrier algorithm and a υ-self concordant barrier function with μ as the barrier parameter is used, the number of iterations in the algorithm to arrive at the solution is $\mathrm{O}(\sqrt{\upsilon}\log(\frac{\upsilon\mu}{\epsilon}))$ for a convex program [23].

V. Trace-driven and Randomized Simulations

We investigate the performance of ITAGS with extensive simulation over multiple offloading scenarios and applications, using both real-world application and randomly generated task trees with practical parameter values.

A. Comparison Targets

We compare ITAGS with the following alternatives:

- Discretization heuristic in Section IV-B.
- Purely local: Scheduling all tasks on the local device, i.e., user's own device/processor.
- Purely remote: Scheduling all tasks on the remote cloud.
- Greedy algorithm: Picking tasks starting from the top of the DAG and scheduling each task onto the processor where it has the least accumulated cost such that the overall application deadline is still met.

- Kao's dynamic programming: The dynamic programming method proposed in [9], [18]. Since the local device in [9], [18] can execute any number of tasks simultaneously without increasing the required processing time of each task, it is essentially assumed to have an infinite number of identical unary-capacity processors. Furthermore, there is zero delay between the local device and the remote cloud. Thus, we study the performance of this dynamic programming algorithm by allowing only a finite number of identical local processors and practical delay between these processors. In other words, we run their algorithm to obtain a scheduling decision and apply this decision to our system by queuing the tasks appropriately and calculating the cost and deadline accounting for inter-processor delay.

All of the above algorithms are provided with the same fallback option as ITAGS. The lower bound solution, described in Section IV-A, is also observed for benchmarking purposes. It is calculated using the SDPT3 solver of CVX.

B. Trace-Driven Simulation Results

The delayed constrained cost minimization problem under consideration and the ITAGS algorithm have general application to different network topologies. Because of page limitation, here we present only the following two scenarios that often arise in practice:

- *Scenario 1*: identical local processors (including the initiating processor) and a remote cloud
- *Scenario 2*: three-tier architecture (peer device, cloudlet and remote cloud) given in Figure 2.

Note that we apply Kao's dynamic programming scheme only to Scenario 1, as the system model considered in [9], [18] cannot be extended to the three-tier system in Scenario 2. We always label the local processor initiating the application as processor 1.

We use the application DAG structures presented in [22] for Gaussian elimination and the FFT algorithm, as well as addition information provided in [22] with respect to the computation and communication times, to test our proposed algorithms for the aforementioned scenarios. We consider the Gaussian elimination application with a matrix size of 5. We generate random values for the processing time of a single loop uniformly from the interval $(0.5, 5)$ ms and allocate processing times t_{i1} for each task i accordingly based on the number of loops required for the execution of the task in the Gaussian elimination algorithm. Similarly, the input/output data is drawn uniformly from the interval $(10, 100)$ KB. For the FFT algorithm, we generate the processing times t_{i1} for each task i uniformly in $(0.5, 20)$ ms and input/output data amount is drawn uniformly from the interval $(10, 100)$ KB. Further, we enforce that the computation times for the tasks in each level are equal and the communication times between the tasks at two particular levels are equal as given in [22]. The rest of the parameter values are kept the same as those of the Gaussian elimination application.

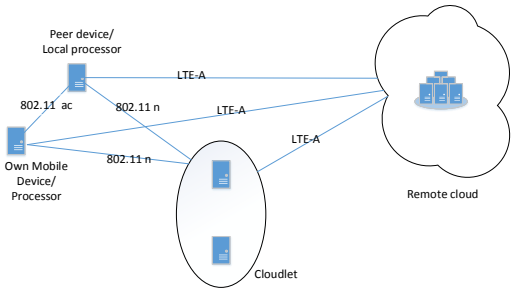


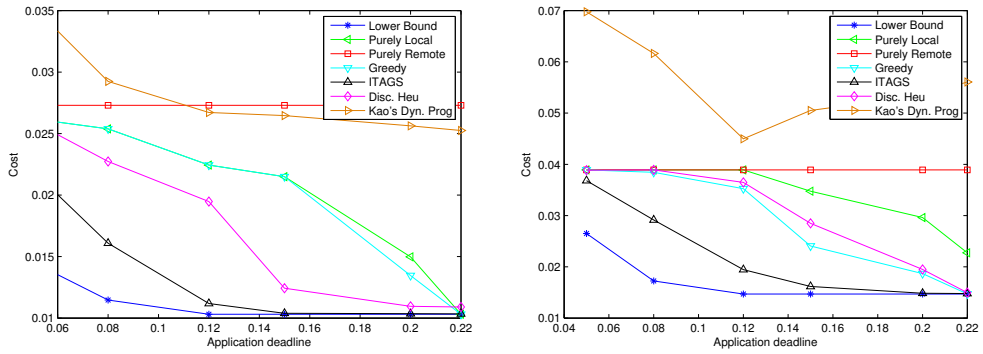
Fig. 2: Simulation topology

We use energy consumption as the measurement of cost. We set $c = 0.935$ watt [24]. The local processor initiating the application has $p_1 = 0.944$ watt [24], and a single additional local processor, representing the peer device, has $p_2 = 1.5$ watts and $t_{i2} = 0.75t_{i1}$ for task i. For Scenario 1 specifically, we consider 3 additional local processors which may all be on the same initiating device or on different devices. We assume that all these local processors have $p_j = 0.944$ watt and $t_{ij} = t_{i1}$, for each task i and processor $j = 2, 3, 4$. For Scenario 2, we consider a cloudlet consisting of two processors $p_3 = p_4 = 4$ watts and $t_{i3} = t_{i4} = 0.5t_{i1}$ for each task i. We consider a more powerful and consequently more expensive remote cloud, consisting of an infinite number of processors, with $p_3 = 10$ watts and $t_{iM} = 0.12t_{i1}$ for each task i. For communication between processors, we consider practical communication delay based on the links as given in Figure 2, with 6.15 ns/byte for 802.11ac, 17.77 ns/byte for 802.11n, and 80 ns/byte for Long Term Evolution Advanced (LTE-A).

Figures 3 and 4 depict the cost versus application deadline for the Gaussian elimination and FFT applications. We see from these figures that ITAGS performs consistently better than the other alternatives. From Figures 3a and 3b, we see that the dynamic programming approach in [9], [18] performs poorly when subjected to practical constraints such as finite capacity processors and inter-processor delay. Naive algorithms such as purely local, purely remote, and greedy do not give satisfactory cost, particularly for non-trivial values of application deadline. The discretization heuristic generally performs better than the naive alternatives but is out-performed by ITAGS. We also see that with increasing values of application deadline, the cost decreases due to the cost-time tradeoff. For large values of application deadline, the decisions tend towards being purely local as the local device is the cheapest and the slowest in our settings.

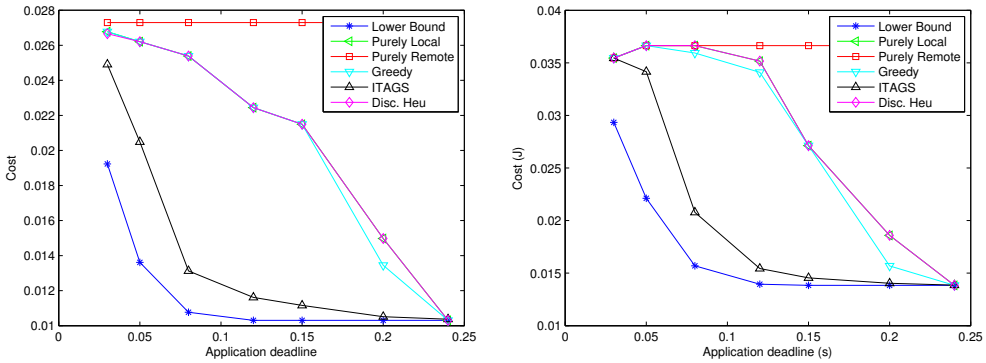
C. Simulation with Randomly Generated Task Trees

In order to further assess the behavior of ITAGS over richer parameter settings, we conduct further simulation based on randomly generated task trees in terms of the DAG structure, task execution times on the processors, and input/output data between tasks. For each parameter setting, we observe the average performance of various algorithms over multiple



(a) Gaussian elimination (b) FFT algorithm

Fig. 3: Cost vs. application deadline for Gaussian elimination and FFT in Scenario 1



(a) Gaussian elimination (b) FFT algorithm

Fig. 4: Cost vs. application deadline for Gaussian elimination and FFT in Scenario 2

realizations of the randomly generated task trees. From our observations in the previous section, the discretization heuristic mostly outperforms the other naive alternatives. Hence, we present comparison only with the discretization heuristic, but also use the lower bound solution for benchmarking.

In Figure 5, we study the effect of various parameters on the performance of ITAGS. We again use energy consumption as the measurement of cost. We consider a general topology with local processors and a remote cloud. We assume the local processor initiating the application, labeled as processor 1, has $p_1 = 0.944$ watt [24] and any additional local processors, representing faster cloudlets or peer devices, has $p_j = 1.5$ watts and $t_{ij} = 0.75t_{i1}$ for each task i and processor $j = 2, \ldots, (M-1)$. We consider a more powerful and consequently more expensive remote cloud with $p_3 = 10$ watts and $t_{iM} = 0.25t_{i1}$ for each task i. Here, $t_{i1} = \frac{\text{number of cycles}}{1.2\text{GHz}}$ where the processor speed is 1.2GHz and the number of cycles is drawn from a uniform distribution in the interval $(100, 200)$ mega cycles. We set by default $M = 3$, $N = 5$, and $c = 0.935$ watt [24] but vary each of them in different plots. The input/output data amount is drawn uniformly from the interval $(1, 3)$ MB. The communication delay is taken as 10 ns/byte between the local processors and 50 ns/byte between a local processor and the remote cloud.

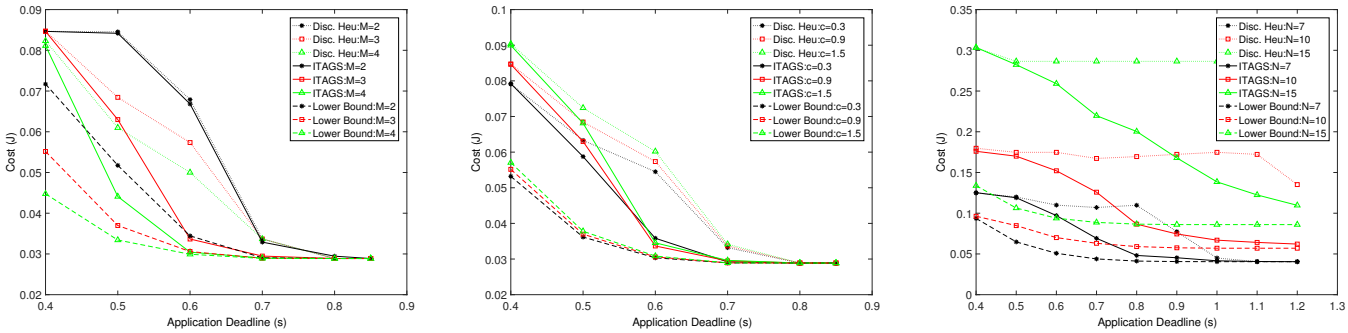
We see that ITAGS substantially outperforms the discretization heuristic, and by inference the other alternatives, over a wide range of parameter values in the number of processors, the communication price, and the application size. Furthermore, as the application deadline increases, ITAGS converges to the lower bound, and hence also converges to the optimum, much faster than the alternatives.

TABLE I: Run-time (sec)

N	Disc. Heu.	ITAGS	Disc. Heu.	ITAGS
	M=3		M=4	
5	5.0612	5.0620	8.6019	8.6029
7	10.0160	10.0167	17.0474	17.0484
10	22.9529	22.9538	41.5981	41.5991
15	79.7471	79.7484	178.5292	178.5308

D. Run-Time Comparison

In Table I, we show the run-time of ITAGS under the settings of Figure 5a, averaged over all L values. We observe that the run-time of ITAGS is nearly identical to that of the discretization heuristic. Therefore, the substantial performance benefit of ITAGS is achieved with negligible run-time penalty. Furthermore, ITAGS scales well with respect to the application size.



(a) Effect of the number of processors M (b) Effect of communication cost c (c) Effect of the number of tasks N

Fig. 5: Cost vs. application deadline for randomly generated task trees

VI. Conclusion

We study the scheduling of dependent tasks on heterogeneous processors with communication delay and an application completion deadline. The proposed cost minimization formulation is generic, allowing different cost structures and processor topologies. To overcome the obstacles of task dependency and deadline constraint, we have developed the ITAGS approach, where the scheduling of each task is assisted by an individual time allowance obtained from a binary-relaxed version of the original optimization problem. Through trace-driven and randomized simulations, we show that ITAGS substantially outperforms a wide range of known algorithms. Furthermore, as the deadline constraint is relaxed, it converges to optimality much faster than other alternatives.

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 Conference on Computer Communications (INFOCOM)*, pp. 945–953, 2012.

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

[4] Z. Li, C. Wang, and R. Xu, "Computation offloading to save energy on handheld devices: a partition scheme," in *Proc. ACM International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pp. 238–246, 2001.

[5] 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.

[6] Y. Zhang, H. Liu, L. Jiao, and X. Fu, "To offload or not to offload: an efficient code partition algorithm for mobile cloud computing," in *Proc. IEEE International Conference on Cloud Networking (CLOUDNET)*, pp. 80–86, 2012.

[7] 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.

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

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

[10] 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.

[11] B. Liang, "Mobile edge computing," in *Key Technologies for 5G Wireless Systems*, V. W. S. Wong, R. Schober, D. W. K. Ng, and L.-C. Wang, Eds., Cambridge University Press, 2017.

[12] R. K. Balan, D. Gergle, M. Satyanarayanan, and J. Herbsleb, "Simplifying cyber foraging for mobile devices," in *Proc. ACM International Conference on Mobile Systems, Applications and Services (MobiSys)*, pp. 272–285, 2007.

[13] A. Mtibaa, K. A. Harras, and A. Fahim, "Towards computational offloading in mobile device clouds," in *Proc. IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, vol. 1, pp. 331–338, 2013.

[14] A. Mtibaa, K. A. Harras, K. Habak, M. Ammar, and E. W. Zegura, "Towards mobile opportunistic computing," in *Proc. IEEE International Conference on Cloud Computing (CLOUD)*, pp. 1111–1114, 2015.

[15] X. 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.

[16] Y. Jararweh, L. Tawalbeh, F. Ababneh, and F. Dosari, "Resource efficient mobile computing using cloudlet infrastructure," in *Proc. IEEE International Conference on Mobile Ad-hoc and Sensor Networks (MSN)*, pp. 373–377, 2013.

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

[18] Y.-H. Kao, B. Krishnamachari, M.-R. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," in *Proc. IEEE Conference on Computer Communications (INFOCOM)*, pp. 1894–1902, 2015.

[19] K. Habak, M. Ammar, K. A. Harras, and E. Zegura, "Femto clouds: Leveraging mobile devices to provide cloud service at the edge," in *Proc. IEEE International Conference on Cloud Computing (CLOUD)*, pp. 9–16, 2015.

[20] S. Sundar and B. Liang, "Communication augmented latest possible scheduling for cloud computing with delay constraint and task dependency," in *Proc. IEEE INFOCOM Workshop on Green and Sustainable Networking and Computing (GSNC 2016)*, 2016.

[21] X. Lin, Y. Wang, Q. Xie, and M. Pedram, "Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment," *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 175–186, 2015.

[22] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[23] F. A. Potra and S. J. Wright, "Interior-point methods," *Journal of Computational and Applied Mathematics*, vol. 124, no. 1, pp. 281–302, 2000.

[24] 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.