Optimizing Mobile Computational Offloading with Delay Constraints

Yi-Hsuan Kao
Department of Electrical Engineering
University of Southern California
Los Angeles, California 90089, USA
Email: yihsuank@usc.edu

Bhaskar Krishnamachari
Department of Electrical Engineering
University of Southern California
Los Angeles, California 90089, USA
Email: bkrishna@usc.edu

***Abstract*—Computation Offloading—sending computational tasks to more resourceful servers, is becoming a widely-used approach to save limited resources on mobile devices like battery life, storage, processor, etc. Given an application that is partitioned into multiple tasks, the offloading decisions can be made on each of them. However, considering the delay constraint and the extra costs on data transmission and remote computation, it is not trivial to make optimized decisions. Existing works have formulated offloading decision problems as either graph-partitioning or binary integer programming problems. The first approach can solve the problem in polynomial time but is not applicable to delay constraints. The second approach relies on an integer programming solver without a polynomial time guarantee. We provide an algorithm, DTP (Deterministic delay constrained Task Partitioning), to solve the offloading decision problem with delay constraints. DTP uses quantization and runs in polynomial time in the number of tasks. Going beyond prior work on linear delay constraints that apply only to serial tasks, we generalize the delay constraints to settings where the dependency between tasks can be described by a tree. Furthermore, we provide another algorithm, PTP (Probabilistic delay constrained Task Partitioning), which gives stronger QoS guarantees. Simulation results show that our algorithms are accurate and robust, and scale well with the number of tasks.**

I. Introduction

Mobile devices have become the primary computing platforms for millions of people. Due to the rapid growth of high-volume multimedia data like video and audio, there will be an increasing need for data processing and data management [1]. Hence, mobile devices are supposed to not only perform heavier computation but also do it more frequently. A hardware-based approach is to enhance both the clock speed and the battery capacity. However, doubling the clock speed approximately octuples the power consumption [2]. Hence, given the state of the art of battery technology, it is clear that the rate of battery improvement will not support a processor that matches the growth of data and corresponding application-specific computational requirements. On the other hand, software-based technology for reducing program power consumption offers alternative approaches [3]. One way is called computation offloading: sending resource-hungry computations to more resourceful servers. By doing so, we can extend the battery life of mobile device and shorten the response time in some cases [1], [4], [5].

However, it is not trivial to implement offloading. Offloading a task saves mobile battery and expedites the execution, while the system must pay extra prices in the form of increased data communication cost, task scheduling cost, etc [6]. Hence, to find an optimal decision relies on globally identifying tasks that are worth offloading. That is, to carefully determine the optimal offloading decision that balances the system's benefits and costs.

In this paper, our goal is to find the optimal offloading decision from a software provider's point of view. Offering an application service to users, a software provider makes offloading decisions by considering both remote computational cost and mobile energy consumption and a reasonable delay (system response time). To achieve our goal, we formulate an optimization problem subject to delay constraints, which specifies quality of service (QoS) guarantee. Then, we design a customized algorithm to solve our problem and provide its complexity analysis. Furthermore, we present another algorithm to solve the problems with stronger QoS constraints.

Existing works have been formulating optimization problems to solve for the optimal offloading decision [7], [8]. Suppose an application program is represented by a directed graph, called task graph, where the vertices represent tasks and the edges specify the data dependency between two tasks [9]. The nodes in the graph are functions of the application and the edges specify the data communication between them. Given such a task graph, an offloading decision is represented by a cut of the graph, which separates the vertices into two disjoint sets, one representing tasks that are executed at the mobile device and the other representing tasks that are offloaded to the remote server. Hence, finding the optimal offloading decision is equivalent to partitioning the graph such that the objective function is minimized [10]. However, this method does not consider delay constraints at all. Prior work [7] on offloading with delay constraints has relied on a general purpose integer linear programming (ILP) formulation without polynomial run-time guarantees. Instead of solving the optimization problem by either graph partitioning or an ILP formulation, we present customized dynamic programming algorithms that are proved to be running in polynomial time and can also handle more sophisticated delay constraints.

We summarize our contribution as follows.

TABLE I
NOTATIONS

Notation	Description
A_n	measure of complexity of task n
B	channel bandwidth
$\mathcal{C}(n)$	set of children of node n
$D^{(n)}$	delay up to when task n finishes
I_n	offloading decision of task n
K	number of quantization levels
Δ	quantization step size
T_n^l/T_n^r	mobile / remote execution delay of task n
c_n^r	remote clock rate to which task n is sent
c^l	mobile clock rate
d_n	succeeding data of task n
d_{-n}	preceding data of task n (leaf only)
$g(\cdot)$	remote computational cost function
p^c	mobile computation power
p^{RF}	mobile power consumption of RF components
t_{max}	maximum tolerable delay
$\bar{R}$	expectation of random variable R

Contribution 1: A customized and polynomial-time algorithm to solve the optimization problem with deterministic delay constraint. Cuervo *et al.* [7] formulate their optimization problem with deterministic delay constraint and solve it by a standard ILP solver without polynomial run-time guarantee. We present a customized algorithm, **DTP** (Deterministic delay constrained Task Partitioning), which is applicable to tree structure task graph. Further, we prove that **DTP** runs in polynomial time in the number of tasks.

Contribution 2: A stochastic analysis that is applicable to probabilistic delay constraint. The variation of channel states and user actions suggests that probabilistic delay guarantee is more desirable than deterministic delay guarantee. In this paper, we formulate a stochastic optimization problem and present another algorithm, **PTP** (Probabilistic delay constrained Task Partitioning), to solve it.

II. Models and Notations

Consider an application that is separated into N tasks and the corresponding task graph is a rooted tree as shown in Fig. 1. We assume that task 1 initializes the application and task N, which is the rooted task, is executed in the end. The edge (m, n) on the graph specifies that task n relies on the result of task m. That is, task n can not start before task m finishes.

Let the binary variable I_n denote the offloading decision of task n. That is, $I_n = 1$ implies that task n is offloaded to the remote server. Our goal is to find these offloading decisions $\{I_n\}_{n=1}^N$ that minimizes the cost function subject to the total delay constraints. Table I lists the notations in this paper.

From a software provider's point of view, the cost function consists of the remote computational cost and the mobile energy consumption. Given a series of offloading decisions, $\{I_n\}_{n=1}^N$, we calculate the cost as follows.

$$\text{Cost} = \sum_{n=1}^{N} I_n g(A_n, c_n^r) + \omega \left[\sum_{n=1}^{N} (\neg I_n) p^c T_n^l + \sum_{n=1}^{N} \sum_{m \in \mathcal{C}(n)} p^{RF} (I_n \oplus I_m) \frac{d_m}{B} \right]. \quad (1)$$

In this equation, $\neg$ denotes NOT operator and $\oplus$ denotes XOR operator. The remote computational cost of task n is a function of task complexity A_n and the remote clock rate c_n^r. On the other hand, the mobile energy consumption consists of the costs of computation and RF components. When task n is executed at mobile that induces the delay T_n^l, the computation energy is simply $p^c T_n^l$. While before executing task n, data transmission is necessary if task n and the preceding task m are executed at different places, which induces the delay $\frac{d_m}{B}$. Hence, the total RF energy consumption before executing task n is the summation over all transmissions of its preceding tasks $m \in \mathcal{C}(n)$. For illustrative purpose, we assume that the transmitting power and the receiving power are the same. Finally, we use a positive weight ω to represent for the desired balance. For example, when the remaining battery is low, one might want to increase ω to bias the offloading decisions.

The total delay depends on the topology of the task graph. For a rooted tree, we recursively define the accumulated delay when task n finishes as

$$D^{(n)} = \max_{m \in \mathcal{C}(n)} \left\{ D^{(m)} + (I_m \oplus I_n) \frac{d_m}{B} \right\} + (\neg I_n) T_n^l + I_n T_n^r. \quad (2)$$

It consists of the execution delay of task n and data transmission delay between task n and its preceding tasks. The delay $D^{(n)}$ is dominated by the slowest branch. If the task graph is a chain, the total delay, $D^{(N)}$, can be further simplified as

$$D_{serial}^{(N)} = \sum_{n=1}^{N} ((\neg I_n) T_n^l + I_n T_n^r) + \sum_{n=1}^{N-1} (I_{n+1} \oplus I_n) \frac{d_n}{B},$$

which is the same delay constraint considered in [7]. In the following sections, we provide efficient algorithms to solve the optimal offloading strategies for our optimization problems.

III. Deterministic Delay Constrained Task Partitioning

In this section we formulate our deterministic optimization problem and present an algorithm, **DTP** (Deterministic delay constrained Task Partitioning) that runs in $\Theta(NMK)$ time, where N is the number of tasks, M is the maximum in-degree of the task graph and K is the number of quantization levels in time domain. We will further bound K by a polynomial of N. Given a task graph with nodes representing tasks from

Algorithm 1 Deterministic delay constrained Task Partitioning (DTP)

procedure DTP(N, t_{max}) ▷ min. cost from root N with delay less than t_{max}
  $q \leftarrow$ BFS (G, N) ▷ run BFS of G from node N and store visited nodes in order in q
  for $n \leftarrow q$.end, q.start **do** ▷ start from the last element in q
    if n is a leaf **then** ▷ initialize OPT values of leaves
      $OPT^l[n, t_k] \leftarrow \begin{cases} \omega P^c T_n^l & \forall k \ni t_k \geq q_{up}(T_n^l) \\ \infty & \text{otherwise} \end{cases}$
      $OPT^r[n, t_k] \leftarrow \begin{cases} g(A_n, c_n^r) + \omega P^{RF} \frac{d_{-n}}{B} & \forall k \ni t_k \geq q_{up}\left(T_n^r + \frac{d_{-n}}{B}\right) \\ \infty & \text{otherwise} \end{cases}$
    else
      for $k \leftarrow 1, K$ **do**
        Calculate $OPT^l[n, t_k]$ and $OPT^r[n, t_k]$ from (4) and (5)
      end for
    end if
  end for
  Trace back the optimal decision from $OPT^l[N, t_{max}]$
end procedure

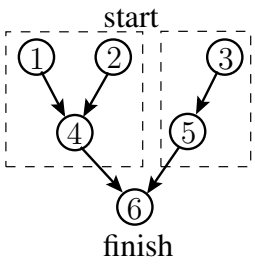

Fig. 1. A tree-structured task graph.

$1, \cdots, N$, we formulate the optimization problem as

$$\begin{aligned} &\min \ \text{Cost} \\ &\text{s.t. } I_1 = I_N = 0, \ I_n \in \{0, 1\} \ \forall n \in \{2, \cdots, N-1\}, \\ &\quad D^{(N)} < t_{max}. \end{aligned}$$

The cost function and the total delay are defined in (1) and (2), respectively. To solve this problem, we exploit the fact that the tasks at the same depth can be viewed as independent sub-problems. For example, Fig. 1 shows a task graph with 6 tasks. To find the minimum cost when finishing task 6 subject to the delay constraint, we can define the sub-problem as follows: to find the minimum cost when finishing task 4 subject to the delay that excludes the execution delay of task 6 and possible data transmission delay between task 4 and 6 (depends on the offloading decisions). Similarly, we can define the sub-problem of which task 5 is the root. Since the task graph is a tree, these two sub-problems can be solved independently.

In the following, we present our algorithm, **DTP**, based on dynamic programming. Let $OPT^l[n, t]$ denote the minimum cost when finishing task n at mobile (i.e. the local device) with the delay less than t. Similarly, $OPT^r[n, t]$ denotes the minimum cost when finishing task n remotely. To find out the optimal solution, it is suffice to solve $OPT^l[n, t]$ and $OPT^r[n, t]$ for all $n \in \{1, \cdots, N\}$ and all $t \in [0, t_{max}]$. However, since the time domain is continuous, quantization is necessary when we run dynamic programming algorithms. We uniformly partition the interval $[0, t_{max}]$ into K intervals and define the quantizer as follows.

$$q_{up}(x) = t_k, \ \forall x \in (t_{k-1}, t_k], \ \forall k \in \{1, \cdots, K\}. \quad (3)$$

We can solve $OPT^l[n, t_k]$ and $OPT^r[n, t_k]$ for all n and all $k \in \{1, \cdots, K\}$ by the following update equations.

$$\begin{aligned} OPT^l[n, t_k] = \omega P^c T_n^l \\ + \sum_{m \in \mathcal{C}(n)} \min \Big\{ & OPT^l\left[m, t_k - q_{up}\left(T_n^l\right)\right], \\ & OPT^r\left[m, t_k - q_{up}\left(T_n^l + \tfrac{d_m}{B}\right)\right] + \omega P^{RF} \tfrac{d_m}{B} \Big\} \end{aligned} \quad (4)$$

$$\begin{aligned} OPT^r[n, t_k] = g(A_n, c_n^r) \\ + \sum_{m \in \mathcal{C}(n)} \min \Big\{ & OPT^r\left[m, t_k - q_{up}\left(T_n^r\right)\right], \\ & OPT^l\left[m, t_k - q_{up}\left(T_n^r + \tfrac{d_m}{B}\right)\right] + \omega P^{RF} \tfrac{d_m}{B} \Big\} \end{aligned} \quad (5)$$

We summarize our algorithm, **DTP**, as shown in Alg. 1. We first perform breadth-first search on the task graph from the root with every edges reversed and put every visited nodes in order in a queue. By traversing back from the end of the queue, we guarantee that all the sub-problems of node n have been solved when solving node n. For the leaf nodes, we initialize the values as described in Alg. 1. Especially, we define the preceding data d_{-n} for leaf tasks such that executing them remotely will induce extra cost and delay. As we assume that task 1 and N can only be executed at the mobile device, we set d_{-1} to be infinity to rule out the decision $I_1 = 1$. In the end, we trace back from $OPT^l[N, t_{max}]$ to get the optimal offloading decisions.

It can be shown that **DTP** runs in $\Theta(NMK)$ time. First, running breadth-first search on a tree takes linear time with N. Second, from (4) and (5), each OPT value involves at most M comparisons. In total, our algorithm solves for $2NK$ OPT values. Hence, it completes in $\Theta(NMK)$ time. Similarly, it needs $\Theta(NMK)$ storage to store these OPT values.

We further investigate the value of K to guarantee that **DTP** always provides the optimal solution under quantization loss. For real applications, there exists a smallest precision of delay measurement ϵ. That is, the system is no longer sensitive to any delay less than ϵ. For example, $\epsilon = 1$ ms is precise enough for most mobile applications. Hence, the fractions below 1 ms are negligible. Let Δ be the quantization step size, i.e. $K = \frac{t_{max}}{\Delta}$. Suppose t_{max} grows with $\Theta(d)$, where d is the depth of the task graph. That is, as N grows, t_{max} grows linearly with the longest path of the task graph. Since the quantizer defined in (3) always overestimates the delay, the solution error happens when the quantization error is large enough so that **DTP** judges the optimal solution as a non-feasible one. From (4) and (5), quantization error accumulates over tasks on each path. Hence, the maximum quantization error is at most $d\Delta$. As the delay constraint in our optimization problem is a strict inequality, the delay of the optimal solution is at least ϵ away from t_{max} if we neglect the fractions below ϵ. Hence, we choose the quantization step size to be $\frac{\epsilon}{d}$ to guarantee that given any instance, the optimal solution is always considered as a feasible solution by **DTP**. In other words, K can be bounded by $O\left(\frac{d^2}{\epsilon}\right)$. For the worst case when the task graph is a chain ($d = N$), **DTP** runs in $\Theta\left(N^3 M \frac{1}{\epsilon}\right)$ time.

IV. Probabilistic Delay Constrained Task Partitioning

In this section, we perform stochastic analysis of the offloading decision problem subject to a probabilistic delay constraint. Let A_n be the random variable representing the measure of complexity of task n. The execution delay can be specified as

$$T_n^l = \delta \frac{A_n}{c^l},$$

where δ is a constant and c^l is the local clock rate. For the remote server, T_n^r can be defined in a similar way. The randomness of task complexity comes from different user interactions. Hence, the task execution delay is now a random variable with distribution that depends on task complexity. Our goal becomes to minimize the average cost subject to a probabilistic delay constraint. That is,

$$\begin{aligned} &\min \ \overline{\text{Cost}} \\ &\text{s.t. } I_1 = I_N = 0,\ I_i \in \{0,1\}\ \forall i \in \{2,\cdots,N-1\}, \\ &\quad \mathbb{P}\left\{D^{(N)} \le t_{max}\right\} > p_{obj}. \end{aligned}$$

We define $OPT^l[n, p_k]$ as the minimum cost when finishing task n at mobile under the constraint

$$q_{low}\left(\mathbb{P}\left\{D^{(n)} \le t_{max}\right\}\right) = p_k,$$

where p_k is the k^{th} quantization step between $[p_{obj}, 1]$ and the quantizer is defined as

$$q_{low}(x) = p_k,\ \forall x \in (p_k, p_{k+1}],\ \forall k \in \{1,\cdots,K\}. \quad (6)$$

Since the delay is increasing as we solve the sub-problems from leaves to root, it is sufficient to just deal with the interval $[p_{obj}, 1]$. However, instead of simply excluding the delays induced after node m as the case in deterministic analysis, the links between $OPT^*[m, p_m]$ and $OPT^*[n, p_n]$ ($*$ could be l or r) are not obvious for arbitrary p_m and p_n. For a node n that has two children, say m_1 and m_2, suppose we want to find out $OPT^l[n, p_k]$. For illustrative purpose, we only consider the case when task m_1 and m_2 are executed at the mobile device. In this example, we have to identify all possible cases $\left(OPT^l[m_1, p_{k_1}], OPT^l[m_2, p_{k_2}]\right)$ satisfying

$$\begin{aligned} p_{k_1} &= q_{low}\left(\mathbb{P}\left\{D^{(m_1)} \le t_{max}\right\}\right), \\ p_{k_2} &= q_{low}\left(\mathbb{P}\left\{D^{(m_2)} \le t_{max}\right\}\right), \\ p_k &= q_{low}\left(\mathbb{P}\left\{\max\left\{D^{(m_1)}, D^{(m_2)}\right\} + T_n^l \le t_{max}\right\}\right). \end{aligned} \quad (7)$$

The delay $D^{(m_1)}$ depends on the offloading decisions that have been made up to task m_1 and these decisions are related to the probabilistic constraint p_{k_1}. Hence, we have to find the appropriate (p_{k_1}, p_{k_2}) such that the delays $D^{(m_1)}$ and $D^{(m_2)}$ satisfy (7). Since $D^{(m_1)}$ and $D^{(m_2)}$ are independent random variables, we can equivalently write (7) as

$$q_{low}\left(\int_0^{t_{max}} F_{D^{(m_1)}}(t)\, F_{D^{(m_2)}}(t)\, f_{T_n^l}(t_{max} - t)\, dt\right) = p_k, \quad (8)$$

where F_* represents for cumulative distribution function (CDF) and f_* represents for probability density function (PDF). We calculate these integrals and create links from child nodes to node n associated with the resulting probability values. If the result of (8) is less than p_{obj}, we discard the corresponding combination. When we make decision on node n with the constraint p_k, we compare the cost of all possible offloading decisions up to node n based on these links.

For a general case, by performing channel estimation, we assume the channel bandwidth B remains constant over its coherence time. Hence, Eq. (8) will also involve some shifts of CDFs if constant data transmission delay is induced. Suppose node n has M children, we have to consider all possible offloading decisions on its children and all possible p_k's. We can write the set of all possible combinations as

$$\left(OPT^*[1, p_{k_1}], OPT^*[2, p_{k_2}], \cdots, OPT^*[M, p_{k_M}]\right).$$

Each $*$ can be independently chosen from $\{l, r\}$ and k_m can be independently chosen from $\{1,\cdots,K\}$ for all m in $\{1,\cdots,M\}$. Hence, creating links for a node n, at the worst case, involves $(2K)^M$ integrals with the form as shown in (8).

We summarize our algorithm, **PTP** in Alg. 2. **PTP** runs in $\Theta\left(NK^M\right)$ time. By the similar analysis in Sec. III, K can be further bounded by $O\left(\frac{N}{\epsilon}\right)$, as now the interval $[p_{obj}, 1]$ remains constant as N grows.

Algorithm 2 Probabilistic delay constrained Task Partitioning (PTP)

procedure PTP(N, p_{obj}, t_{max}) ▷ min. cost from root N s.t. $\mathbb{P}\left\{D^{(N)} \leq t_{max}\right\} > p_{obj}$

  $q \leftarrow \text{BFS}\,(G, N)$ ▷ run BFS of G from node N and store visited nodes in order in q

  for $n \leftarrow q$.end, q.start **do** ▷ start from the last element in q

    if n is a leaf **then** ▷ initialize OPT values of leaves

      $OPT^l\left[n, p_k\right] \leftarrow \begin{cases} \omega P^c \bar{T}_n^l & \text{if } p_k = q_{low}\left(F_{T_n^l}\left(t_{max}\right)\right) \\ \infty & \text{otherwise} \end{cases}$

      $OPT^r\left[n, p_k\right] \leftarrow \begin{cases} \bar{g}\left(A_n, c_n^r\right) + \omega P^{RF} \frac{d_{-n}}{B} & \text{if } p_k = q_{low}\left(F_{T_n^r}\left(t_{max} - \frac{d_{-n}}{B}\right)\right) \\ \infty & \text{otherwise} \end{cases}$

    else

      for all possible combinations $\left(OPT^*\left[1, p_{k_1}\right], OPT^*\left[2, p_{k_2}\right], \cdots, OPT^*\left[M, p_{k_M}\right]\right)$ **do**

        link to $OPT^l\left[n, p^*\right]$ if $p^* = q_{low}\left(\int_0^{t_{max}} \Pi_{m \in \mathcal{C}(n)} F_{D^{(m)}}\left(t - I_m \frac{d_m}{B}\right) f_{T_n^l}\left(t_{max} - t\right) dt\right)$

        link to $OPT^r\left[n, p^*\right]$ if $p^* = q_{low}\left(\int_0^{t_{max}} \Pi_{m \in \mathcal{C}(n)} F_{D^{(m)}}\left(t - \left(\neg I_m\right) \frac{d_m}{B}\right) f_{T_n^r}\left(t_{max} - t\right) dt\right)$

      end for

      for $k \leftarrow 1, K$ **do**

        Calculate $OPT^l\left[n, p_k\right]$ and $OPT^r\left[n, p_k\right]$ by choosing the minimum from their links

      end for

    end if

  end for

  Trace back the optimal decision from $\min_{k \in \{1, \cdots, K\}} OPT^l\left[N, p_k\right]$

end procedure

TABLE II
SIMULATION PROFILE

Notation	Value	Description
p^c	0.3 W	local computation power
p^{RF}	0.7 W	local RF power
c^l	1 Hz	local clock rate
B	10 MB/s	channel bandwidth

V. SIMULATION RESULTS

In this section, we first verify the accuracy of **DTP** and **PTP** and then design an iterative algorithm (**IDTP**) that iteratively runs **DTP** as a baseline for comparison with **PTP**. From the simulation results, we show that both of **DTP** and **PTP** achieve high accuracy with tractable complexity. On the other hand, **IDTP** can provide a suboptimal solution that are close to the optimal one given by **PTP** within fixed number of iterations.

Our simulation is done as follows. For every sample, we fix the task graph as a perfect binary tree while the simulation profile is varying. We first uniformly choose a profile from a pool. Then we solve the offloading strategies by our algorithms and finally compare the solution with the one given by brute force algorithm, which simply checks over all offloading decisions and chooses the optimal solution from the feasible ones. A profiling pool is defined as follows.

$$\mathcal{P}_N = \left\{(\mathbf{A}, \mathbf{c}^r, \mathbf{d}) \in [1, 10]^N \times [100, 1000]^N \times [0.1, 10]^{N-1}\right\}$$

The n^{th} element of vector $\mathbf{A}$ denotes the complexity of task n. Similarly, $\mathbf{c}^r$ denotes the remote clock rates in Hz and $\mathbf{d}$ denotes the succeeding data in MB of each task except for the final task. The rest parameters remain constants as shown in table II. For the stochastic case, we assume that the task complexities are independent exponential random variables with expectations chosen from the pool. Further, we assume that the system performs channel estimation so that the bandwidth B remaining constant within the coherence time.

A. The Performance of Proposed Algorithms

Fig. 2 shows the performance of **DTP** that solves the deterministic optimization problem. In general, we classify the solution errors into three types: first, **DTP** may provide a feasible solution which is not the optimal one; second, **DTP** may not find any feasible solution but there exists at least one; third, **DTP** may provide a non-feasible solution. By the definition of the quantizer in (3), **DTP** always overestimates the delay. That is, to some extent **DTP** may miss some feasible solutions while it will never provide a non-feasible one. Fig. 2 shows both the solution error and the missing probability (type 2) with different values of K. We can see that the error probabilities decrease as K grows. Moreover, K falls in the order of tens to provide good performance for a binary tree with depth less than 4.

Fig. 3 shows the performance of **PTP** that solves the stochastic optimization problem for a task graph with depth 3. Similar to the deterministic case, the quantizer in (6) underestimates the probability that the total delay is less than t_{max}. Hence, **PTP** will never give a non-feasible solution but may miss some feasible solutions for small K. For probabilistic delay constraints, K is related to the interval size $[p_{obj}, 1]$. In general applications, the probabilistic delay constraints provide more sophisticated QoS, in which p_{obj} is chosen to be close

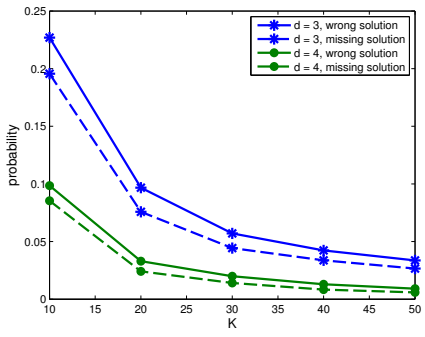

Fig. 2. Error probability of **DTP**

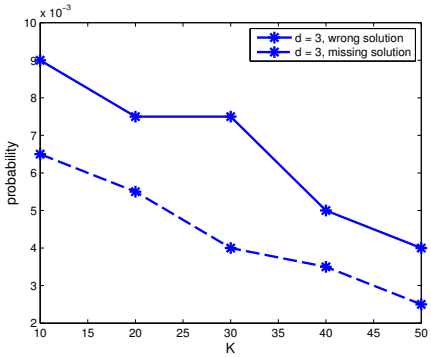

Fig. 3. Error probability of **PTP**

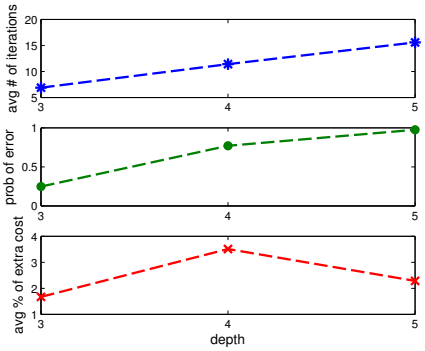

Fig. 4. The performance of **IDTP**

to 1. Hence, from our simulation result, $K = 10$ (with corresponding step size 0.01 and $p_{obj} = 0.9$) provides a good performance with error probability 0.009.

B. An Iterative Alternative to Compare with PTP

We design a simple algorithm, **IDTP**, as an iterative alternative to **PTP**. At every step we get a solution from **DTP** and simulate the corresponding probability that the delay is less than t_{max}. Based on the result, we update the deterministic delay constraint and run **DTP** for another iteration. To make the running time reasonable, we run **DTP** for at most 100 iterations. Fig. 4 shows the performance of **IDTP**. We record the average number of iterations that **IDTP** takes when it finds the optimal solution. When it runs out of iterations but fails to find the optimum, we do not count in the statistics but record as an error instead. When a suboptimal solution is provided, we define the extra cost percentage as

$$\text{ExtraCost } \% = \frac{(\text{Cost} - \text{Cost}_{min})}{\text{Cost}_{min}} \times 100,$$

where Cost_{min} denotes the cost of optimal solution. Further, we take the average over the samples at which our algorithm provides suboptimal solutions. From Fig. 4, the average number of iterations grows linearly with the depth of the task graph. Although **IDTP** fails to find the optimal solution most of the time as the depth of the task graph grows, the suboptimal solution it provides shows that the average extra cost does not exceed 4% in all cases.

VI. Conclusion

We formulated two computational offloading decision problems, Deterministic delay constrained Task Partitioning and Probabilistic delay constrained Task Partitioning, and provided respective algorithms, **DTP** and **PTP**, to solve them. Comparing with linear delay constraints that are limited to a chain of tasks, our algorithms are applicable to more generalized schemes, in which the dependency between tasks can be described by a tree. Instead of relying on an integer programming formulation without a polynomial time guarantee, we showed that our algorithms run in polynomial time with the problem size. In addition to deterministic delay constraints, we performed stochastic analysis to settle problems with probabilistic delay constraints. **DTP** runs in $\Theta\left(N^3\right)$ time, where N is the number of tasks. On the other hand, **PTP** runs in $\Theta\left(N^M\right)$ time, where M is the maximum in-degree of the task graph. Furthermore, by running **DTP** iteratively as an alternative to **PTP**, we showed that it provides robust suboptimal solutions in all cases we considered.

Our formulations and algorithms are generally applicable to mobile applications. More comprehensive costs can be taken into consideration without modifying the solution scheme. More importantly, the efficiency of our algorithms can potentially operate tasks partitioning on finer granularity. Implementation on real systems will be a stronger evidence to identify potential applications. Furthermore, instead of binary decisions, solving for multiple offloading decisions on an intermittent network will be an interesting problem that can support a wilder range of applications.

References

[1] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Networks and Applications*, vol. 18, no. 1, pp. 129–140, 2013.

[2] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, vol. 43, no. 4, pp. 51–56, 2010.

[3] S. Gurun and C. Krintz, "Addressing the energy crisis in mobile computing with developing power aware software," *Memory*, vol. 8, no. 64MB, p. 512MB, 2003.

[4] D. Shivarudrappa, M. Chen, and S. Bharadwaj, "Cofa: Automatic and dynamic code offload for android," *University of Colorado, Boulder*.

[5] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: enabling interactive perception applications on mobile devices," in *ACM MobiSys*. ACM, 2011, pp. 43–56.

[6] M. V. Barbera, S. Kosta, A. Mei, and J. Stefa, "To offload or not to offload? the bandwidth and energy costs of mobile cloud computing," in *IEEE INFOCOM*. IEEE, 2013, pp. 1285–1293.

[7] 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 *ACM MobiSys*. ACM, 2010, pp. 49–62.

[8] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *ACM Computer systems*. ACM, 2011, pp. 301–314.

[9] H. El-Rewini and T. G. Lewis, "Scheduling parallel program tasks onto arbitrary target machines," *Journal of parallel and Distributed Computing*, vol. 9, no. 2, pp. 138–153, 1990.

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