Hermes: Latency Optimal Task Assignment for Resource-constrained Mobile Computing

Yi-Hsuan Kao, *Student Member, IEEE,* Bhaskar Krishnamachari, *Member, IEEE,*
Moo-Ryong Ra, *Member, IEEE* and Fan Bai, *Fellow, IEEE*

Abstract—With mobile devices increasingly able to connect to cloud servers from anywhere, resource-constrained devices can potentially perform offloading of computational tasks to either save local resource usage or improve performance. It is of interest to find optimal assignments of tasks to local and remote devices that can take into account the application-specific profile, availability of computational resources, and link connectivity, and find a balance between energy consumption costs of mobile devices and latency for delay-sensitive applications. We formulate an NP-hard problem to minimize the application latency while meeting prescribed resource utilization constraints. Different from most of existing works that either rely on the integer programming solver, or on heuristics that offer no theoretical performance guarantees, we propose Hermes, a novel fully polynomial time approximation scheme (FPTAS). We identify for a subset of problem instances, where the application task graphs can be described as serial trees, Hermes provides a solution with latency no more than $(1+\epsilon)$ times of the minimum while incurring complexity that is polynomial in problem size and $\frac{1}{\epsilon}$. We further propose an online algorithm to learn the unknown dynamic environment and guarantee that the performance gap compared to the optimal strategy is bounded by a logarithmic function with time. Evaluation is done by using real data set collected from several benchmarks, and is shown that Hermes improves the latency by 16% compared to a previously published heuristic and increases CPU computing time by only 0.4% of overall latency.

Index Terms—Mobile Cloud Computing, Computational Offloading, Approximation Algorithms, On-line Learning

1 Introduction

As more embedded devices are connected, lots of resource on the network, in the form of cloud computing, become accessible. These devices, either suffering from stringent battery usage, like mobile devices, or limited processing power, like sensors, are not capable to run computation-intensive tasks locally. Taking advantage of the remote resource, more sophisticated applications, requiring heavy loads of data processing and computation [1], [2], can be realized in timely fashion and acceptable performance. Thus, computation offloading—sending computation-intensive tasks to more resourceful severs, is becoming a potential approach to save resources on local devices and to shorten the processing time [3], [4], [5], [6].

However, implementing offloading invokes extra communication cost due to the application and profiling data that must be exchanged with remote servers. Offloading a task aims to save battery use and expedite the execution, but the additional communication spends extra energy on wireless radio and induces extra transmission latency [7], [8]. Hence, a good offloading strategy would select a subset of tasks to be offloaded, considering the balance between how much the offloading saves and how much extra cost is induced. On the other hand, in addition to targeting a single remote server, which involves only binary decision on each task, another spectrum of offloading schemes make use of other idle and connected devices in the network [9], [10], where the decision is made over multiple devices considering their availabilities and qualities of wireless channels. In sum, a rigorous **optimization formulation** of the problem and the **scalability** of corresponding algorithm are the key issues that need to be addressed.

In general, we are concerned in this domain with a task assignment problem over multiple devices, subject to constraints. Furthermore, task dependency must be taken into account in formulations involving latency as a metric. In this paper, we formulate an optimization problem that aims to minimize the latency subject to cost constraint. We show that the problem is NP-hard and propose Hermes[1], which is a fully polynomial time approximation scheme (FPTAS). For all instances, Hermes always outputs a solution that gives no more than $(1+\epsilon)$ times of the minimum objective, where ϵ is a positive number, and the complexity is bounded by a polynomial in $\frac{1}{\epsilon}$ and the problem size [13]. Table 1 summarizes the comparison of our formulation and algorithm to the existing works. To the best of our knowledge, for this class of task assignment problems, Hermes applies to more sophisticated formulations than prior works and runs in polynomial time with problem size but still provides near-optimal solutions with performance guarantee. We list our main contributions as follows.

1) **A new NP-hard formulation of task assignment considering both latency and resource cost:** Our formulation is practically useful for applications

- *Y. Kao and B. Krishnamachari are with the Department of Electrical Engineering, University of Southern California, Los Angeles, CA. E-mail: {yihsuank, bkrishna}@usc.edu*
- *M. Ra is with AT&T Research Lab, Bedminster, NJ. E-mail: mra@research.att.com*
- *F. Bai is with General Motors Global R&D, Warren, MI. E-mail: fan.bai@gm.com*

1. Because of its focus on minimizing latency, Hermes is named for the Greek messenger of the gods with winged sandals, known for his speed.

TABLE 1: Comparison between existing works and Hermes

Existing Works	**MAUI** [11]	**CloneCloud** [4]	**min k-cut** [12]	**Odessa** [2]	**Hermes**
Task Graph	serial	tree	DAG	general	subset of DAG
Objectives	energy consumption	cost and latency	communication cost	latency & throughput	latency
Constraints	latency	none	none	none	cost
Partition	2 devices	2 devices	multiple devices	2 devices	multiple devices
Complexity	exponential	exponential	exponential	no guarantee	polynomial
Performance	optimal	optimal	optimal	no guarantee	$(1+\epsilon)$-approximate

with a general task dependency described by a directed acyclic graph and allows for the minimization of total latency (makespan) subject to a resource cost constraint.

2) **Hermes, an FPTAS algorithm:** We identify for a subset of problem instances, where the application task graphs can be described as serial trees, Hermes admits a $(1+\epsilon)$ approximation and runs in $O(d_{in}NM^2\frac{l}{\epsilon}\log_2 T)$ time, where N is the number of tasks, M is the number of devices, d_{in} is the maximum indegree over all tasks, l is the length of the longest paths and T is the dynamic range.
3) **An online learning scheme to unknown dynamic environments:** We adapt a sampling method proposed in [14] to continuously probe the channels and devices, and exploit the best assignment based on the probing result. Furthermore, we prove that the performance gap is bounded by a logarithmic function of time compared to the optimal strategy assuming the statistics is known beforehand.
4) **Comparative performance evaluation:** We evaluate the performance of Hermes by using real data sets measured in several benchmarks to emulate the executions of these applications, and compare it to the previously-published Odessa scheme [2]. The result shows that Hermes improves the latency by 16% (36% for larger scale application) and increases CPU computation time by only 0.4% of overall latency, which implies the latency gain of Hermes is significant enough to compensate its CPU overhead.

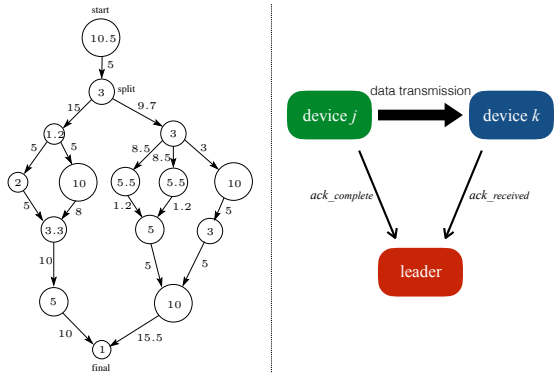

Fig. 1: An application task graph. A node specifies a computing task labeled with its workload and an edge implies data dependency labeled with the amount of data transmission. At application run time, acknowledgement is sent upon task completion and data reception. The leader takes care of node failure when acknowledge timeouts.

2 Models and Notations

2.1 System Model

We consider a mesh network, where mobile devices can communicate with each other through direct links. Before an application begins, there is a leader node collecting the available resources on each device, like released CPU cycles per second, upload bandwidth and download bandwidth. Considering the task complexity and the available CPU cycles, the leader estimates the task execution latency on each device, and the communication overhead. Finally, the leader runs Hermes for the optimal assignment strategy and notifies the helper nodes for task offloading.

At run time, the communication messages between active devices are shown in Fig. 1. When device j finishes the preceding task, it sends an acknowledgement to the leader, and transmits the necessary data to device k that is to execute the succeeding task. Upon receiving the data, device k sends another acknowledgement to the leader and starts running the task. The process repeats for each pair of tasks. Two different node failures are being tracked by the leader based on the acknowledgement timeout rule. First, if device k fails to complete the task, the leader will ask its preceding device (j) to run the task. Second, if device k fails to receive the necessary data so that it cannot run the task, the leader will also ask device j to run the task. In both cases, the leader always traces back to the preceding device that holds the previous result and data so that there will be no extra data transmission due to node failures.

2.2 Task Graph

An application profile can be described by a directed graph $G(\mathcal{V},\mathcal{E})$ as shown in Fig. 1, where nodes stand for tasks and directed edges stand for data dependencies. A task precedence constraint is described by a directed edge (m,n), which implies that task n relies on the result of task m. That is, task n cannot start until it gets the result of task m. The weight on each node specifies the workload of the task, while the weight on each edge shows the amount of data communication between two tasks. In addition to the application profile, there are some parameters related to the graph measure in our complexity analysis. We use N to denote the number of tasks and M to denote the number of available devices in the network (potential offloading candidates). For each task graph, there is an initial task (task 1) that starts the application and a final task (task N) that terminates it. A path from initial task to final task can be described by a sequence of nodes, where every pair of

consecutive nodes are connected by a directed edge. We use l to denote the maximum number of nodes in a path, i.e., the length of the longest path. Finally, d_{in} denotes the maximum indegree in the task graph. Using Fig. 1 as an example, we have $l = 7$ and $d_{in} = 2$.

2.3 Cost and Latency

Let $C_i^{(j)}$ be the execution cost of task i on device j and $C_{mn}^{(jk)}$ be the transmission cost of data between task m and n though the channel from device j to k. Similarly, the latency consists of execution latency $T_i^{(j)}$ and the transmission latency $T_{mn}^{(jk)}$. Given a task assignment strategy $\mathbf{x} \in \{1 \cdots M\}^N$, where the i^{th} component, x_i, specifies the device that task i is assigned to, the total cost can be described as follows.

$$Cost = \sum_{i \in [N]} C_i^{(x_i)} + \sum_{(m,n) \in \mathcal{E}} C_{mn}^{(x_m x_n)} \tag{1}$$

As described in the equation, the total cost is additive over nodes (tasks) and edges of the graph.

For a tree-structure task graph, the accumulated latency up to task i depends on its preceding tasks. Let $D(i, \mathbf{x})$ be the accumulated latency when task i finishes given the assignment strategy $\mathbf{x}$, which can be recursively defined as

$$D(i, \mathbf{x}) = \max_{m \in \mathcal{C}(i)} \left\{ D(m, \mathbf{x}) + T_{mi}^{(x_m x_i)} \right\} + T_i^{(x_i)}. \tag{2}$$

We use $\mathcal{C}(i)$ to denote the set of children of node i. For example, in Fig. 2, the children of task 6 are task 4 and task 5. For each branch leading by node m, the latency is accumulating as the latency up to task m plus the latency caused by data transmission between m and i. $D(i, \mathbf{x})$ is determined by the slowest branch.

2.4 Optimization Problem

Consider an application, described by a task graph, and a resource network, described by $\{C_i^{(j)}, C_{mn}^{(jk)}, T_i^{(j)}, T_{mn}^{(jk)}\}$, our goal is to find a task assignment strategy $\mathbf{x}$ that minimizes the total latency and satisfies the cost constraint, that is,

$$\begin{aligned} \mathbf{P}: \min_{\mathbf{x} \in [M]^N} \ & D(N, \mathbf{x}) \\ \textbf{s.t. } & Cost \leq B, \\ & x_N = 1. \end{aligned}$$

The $Cost$ and $D(N, \mathbf{x})$ are defined in (1) and (2), respectively. The constant B specifies the cost constraint, for example, energy consumption of mobile devices. Without the loss of generality, the final task is in charge of collecting the execution result from other devices. Hence, it is always assigned to the local device ($x_N = 1$).

Theorem 1. *Problem* $\mathbf{P}$ *is NP-hard.*

Proof. We reduce the 0-1 knapsack problem to a special case of $\mathbf{P}$, where a binary partition is made on a serial task graph without considering data transmission. Since the 0-1 knapsack problem is NP-hard [15], Problem $\mathbf{P}$ is at least as hard as the 0-1 knapsack problem.

TABLE 2: Notations

Notation	Description
α_i	workload of task i
d_{mn}	the amount of data exchange between task m and n
$G(\mathcal{V}, \mathcal{E})$	task graph with set of nodes $\mathcal{V}$ and set of edges $\mathcal{E}$)
$\mathcal{C}(i)$	set of children of node i
l	the depth of task graph (the longest path)
d_{in}	the maximum indegree of task graph
δ	quantization step size
$[N]$	set $\{1, 2, \cdots, N\}$
$\mathbf{x} \in [M]^N$	assignment strategy of tasks $1 \cdots N$
$T_i^{(j)}$	latency of executing task i on device j
$T_{mn}^{(jk)}$	latency of transmitting data between task m and n from device j to k
$C_i^{(j)}$	cost of executing task i on device j
$C_{mn}^{(jk)}$	cost of transmitting data between task m and n from device j to k
$D(i, \mathbf{x})$	accumulated latency when task i finishes, given strategy $\mathbf{x}$
w	length of exploration phase in dynamic environment

TABLE 3: Computation Offloading Solutions

Solutions	Task Offloading	Thread Offloading
Source Modification	Yes	No
Kernel Modification	No	Yes
Partition Granularity	coarse-grained	fine-grained
Data Transmission	application data	thread stack, heap and VM state

Assume that $C_i^{(0)} = 0$ for all i, the special case of Problem $\mathbf{P}$ can be written as

$$\begin{aligned} \mathbf{P}': \min_{x_i \in \{0,1\}} \ & \sum_{i=1}^{N} \left((1 - x_i) T_i^{(0)} + x_i T_i^{(1)} \right) \\ \textbf{s.t. } & \sum_{i=1}^{N} x_i C_i^{(1)} \leq B. \end{aligned}$$

Given N items with their values $\{v_1, \cdots, v_N\}$ and weights $\{w_1, \cdots, w_N\}$, one wants to decide which items to be packed to maximize the overall value and satisfies the total weight constraint, that is,

$$\begin{aligned} \mathbf{Q}: \max_{x_i \in \{0,1\}} \ & \sum_{i=1}^{N} x_i v_i \\ \textbf{s.t. } & \sum_{i=1}^{N} x_i w_i \leq B. \end{aligned}$$

Now $\mathbf{Q}$ can be reduced to $\mathbf{P}'$ by the following encoding

$$\begin{aligned} T_i^{(0)} &= 0, \ \forall i \\ T_i^{(1)} &= -v_i, \\ C_i^{(1)} &= w_i. \end{aligned}$$

By giving these inputs to $\mathbf{P}'$, we can solve $\mathbf{Q}$ exactly, hence,

$$\mathbf{Q} \leq_p \mathbf{P}' \leq_p \mathbf{P}.$$

□

In Section 4, we propose an approximation algorithm based on dynamic programming to solve this problem and show that its running time is bounded by a polynomial of $\frac{1}{\epsilon}$ with approximation ratio $(1 + \epsilon)$.

3 Related Works

3.1 Formulations and Algorithms

Table 1 summarizes the comparison of our formulation and algorithm to the existing works. Of all optimization formulations, integer linear programming (ILP) is the most common formulation due to its flexibility and intuitive interpretation of the optimization problem. In the well-known MAUI work, Cuervo *et al.* [11] propose an ILP formulation with latency constraint of serial task graphs. However, the ILP problems are generally NP-hard, that is, there is no polynomial-time algorithm to solve all instances of ILP unless P = NP [16]. Moreover, it does not address the problems of general task dependency, which is often described by a directed acyclic graph (DAG). Our previous work [17] generalizes MAUI's formulation, where we propose a polynomial time algorithm that could be applied to tree-structured task graphs. However, there is no provable performance guarantee. In addition to ILP, graph partitioning is another approach [12]. The minimum cut on weighted edges specifies the minimum communication cost and cuts the nodes into two disjoint sets, one is the set of tasks that are to be executed at the remote server and the other are ones that remain at the local device. However, it is not applicable to latency metrics. Furthermore, for offloading across multiple devices, solving the generalized version, minimum k-cut, is NP-hard [18].

3.2 Computational Offloading

There have been systems that augment computing on a resource-constrained device using computational offloading. We classify them by the types of remote computational resources that a local device has access to. One extreme is the traditional cloud-computing where a local device sends a request to a cloud that has remote servers set up by a service provider. MAUI [11] and CloneCloud [4] are systems that leverage the resources in the cloud. Odessa [2] identifies the bottleneck stage and suggests offloading strategy and leverages data parallelism to mitigate the load. On the other extreme, Mobile Cloud connects and leverages the mobile devices in close proximity to form a distributed computing platform [19]. Shi *et al.* [9] investigate the mobile helpers reached by intermittent connections. FemtoClouds [20] configures multiple mobile devices into a coordinated cloud service. Between these two extremes, MapCloud [21] is a hybrid system that makes run-time decision on using "local" cloud with less computational resources but faster connections or using "public" cloud that is distant away with more powerful servers but longer communication delay. Cardellini *et al.* [22] propose a game theoretic approach to model the interaction between selfish mobile users who want to leverage remote computational resources. COSMOS [23] finds out the customized and economic cluster in its size and its setup time considering the task complexity.

Table 3 summarizes the two approaches to implement flexible execution at run-time. Application-layer task migration involves modifying the application source code with wrapper functions and run-time decision logic, like MAUI [11] and Thinkair [5]. Thread-level or process-level migration involves modifying the kernel so that for any function execution, it gets trapped in kernel and is then assigned to local device or remote server, like CloneCloud [4]. Hermes applies to the former approach that embraces kernel independence and transmits only application data without overheads like VM state and address space. On the other hand, CloneCloud makes offloading flexible without modification on the application source code, and enables fine-grained partition on thread level, however, it requires non-trivial modification on the kernel code.

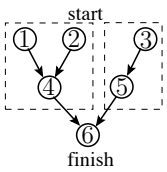

Fig. 2: A tree-structured task graph, in which the two sub-problems can be solved independently.

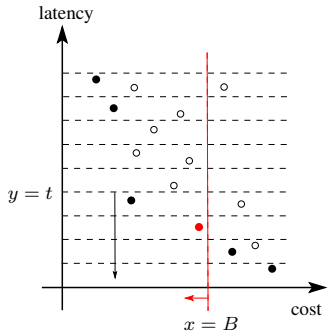

Fig. 3: Hermes solves each sub-problem for the minimum cost with latency less than t. The filled circles are the optimums of each sub-problem. Finally, it looks for minimum latency over all filled circles with cost less than B.

One crucial component that is closely related to system performance is how to partition an application and offload tasks with the awareness of resource availability at run time. To solve the optimal strategy, both MAUI and CloneCloud rely on a standard ILP solver that might cause significant computational overhead. COSMOS breaks the formulation into three sub-problems, however, the combination of the three strategies does not guarantee the global optimum. Odessa and MapCloud propose heuristics that do not have performance guarantee. As we will show in Section 6, a sub-optimal task assignment strategy may lead to significant performance loss in some scenarios. Hence, we develop Hermes that efficiently solves for near-optimal strategy. Furthermore, we are positive that Hermes can be incorporated into real systems to optimize the performance of computational offloading.

4 Hermes: FPTAS Algorithms

In this section, we first propose the approximation scheme to solve Problem **P** for a tree-structure task graph and prove that this simplest version of Hermes is an FPTAS. Then we solve more general task graphs by calling the proposed algorithm within polynomial number of times.

Algorithm 1 Find maximum latency

procedure $FIND_T(N)$
  $q \leftarrow \text{BFS}(G, N)$ ▷ run Breadth First Search on G
  for $i \leftarrow q.\text{end}, q.\text{start}$ **do** ▷ start from the leaves
    if i is a leaf **then**
      $L[i,j] \leftarrow T_i^{(j)}\ \forall j \in [M]$
    else
      for $j \leftarrow 1, M$ **do**
        $L[i,j] \leftarrow$ ▷ max latency finishing i on j

$$T_i^{(j)} + \max_{m \in \mathcal{C}(i)} \max_{k \in [M]} \{L[m,k] + T_{mi}^{(kj)}\}$$

  $T \leftarrow \max_{j \in [M]} L[N,j]$
end procedure

Algorithm 2 Hermes FPTAS for tree

procedure $FPTAS_{tree}(N, \epsilon)$
  $T \leftarrow FIND_T(N)$
  $q \leftarrow \text{BFS}(G, N)$
  for $r \leftarrow 1, \log_2 T$ **do**
    $T_r \leftarrow \frac{T}{2^{r-1}},\ \delta_r \leftarrow \frac{\epsilon T}{l 2^r}$
    $\tilde{\mathbf{x}} \leftarrow DP(q, T_r, \delta_r)$
    if $L(\tilde{\mathbf{x}}) \geq (1+\epsilon)\frac{T}{2^r}$ **then**
      return
end procedure

procedure $DP(q, T_{up}, \delta)$
  $K \leftarrow \lceil \frac{T_{up}}{\delta} \rceil$
  for $i \leftarrow q.\text{end}, q.\text{start}$ **do**
    if i is a leaf **then**

$$C[i,j,k] \leftarrow \begin{cases} C_i^{(j)} & \forall k \geq q_\delta(T_i^{(j)}) \\ \infty & \text{otherwise} \end{cases}$$

    else
      for $j \leftarrow 1, M$, $k \leftarrow 1, K$ **do**
        Calculate $C[i,j,k]$ from (6)
  $k_{min} \leftarrow \min_{j \in [M]} k$ **s.t.** $C[N,j,k] \leq B$
end procedure

4.1 Tree-structured Task Graph

We propose a dynamic programming method to solve the problem on tree-structured task graphs. For example, in Fig. 2, the minimum latency when task 6 finishes depends on when and where task 4 and 5 finish. Hence, prior to solving the minimum latency of task 6, we want to solve both task 4 and 5 first. We exploit the fact that the sub-trees rooted by task 4 and task 5 are independent. That is, the assignment strategy on task 1, 2 and 4 does not affect the strategy on task 3 and 5. Hence, we can solve the sub-problems independently and combine them when considering task 6.

We define the sub-problem as follows. Let $C[i,j,t]$ denote the minimum cost when finishing task i on device j within latency t. We will show that by solving all of the sub-problems for $i \in [N]$, $j \in [M]$ and $t \in [0,T]$ with sufficiently large T, the optimal strategy can be obtained by combining the solutions of these sub-problems. Fig. 3 shows our methodology. Each circle marks the performance given by an assignment strategy, with x-component as cost and y-component as latency. Our goal is to find out the red circle, that is, the strategy that results in minimum latency and satisfies the cost constraint. Under each horizontal line $y = t$, we first identify the circle with minimum x-component, which specifies the least-cost strategy among all of strategies that result in latency at most t. These solutions are denoted by the filled circles. In the end, we look at the one in the left plane ($x \leq B$) whose latency is the minimum.

Instead of solving infinite number of sub-problems for all $t \in [0,T]$, we discretize the time domain by using the quantization function

$$q_\delta(x) = k, \text{ if } (k-1)\delta < x \leq k\delta. \tag{3}$$

It suffices to solve all the sub-problems for $k \in \{1, \cdots, K\}$, where $K = \lceil \frac{T}{\delta} \rceil$. We will analyze how the performance is affected due to the loss of precision by doing quantization and the trade-off with algorithm complexity after we present our algorithm. Suppose we are solving the sub-problem $C[i,j,k]$, given that all of sub-problems of the preceding tasks have been solved, the recursive relation can be described as follows.

$$C[i,j,k] = C_i^{(j)} + \min_{x_m : m \in \mathcal{C}(i)} \{ \sum_{m \in \mathcal{C}(i)} C[m, x_m, k - k_m] + C_{mi}^{(x_m j)} \}, \tag{4}$$

$$k_m = q_\delta\big(T_i^{(j)} + T_{mi}^{(x_m j)}\big). \tag{5}$$

That is, to find out the minimum cost within latency k at task i, we trace back to its child tasks and find out the minimum cost over all possible strategies, with the latency that excludes the execution delay of task i and data transmission delay. As the cost function is additive over tasks and the decisions on each child task is independent with each other, we can further lower down the solution space from M^z to zM, where z is the number of child tasks of task i. By making decisions on each child task independently, we have

$$C[i,j,k] = C_i^{(j)} + \sum_{m \in \mathcal{C}(i)} \min_{x_m \in [M]} \{C[m, x_m, k - k_m] + C_{mi}^{(x_m j)}\}. \tag{6}$$

After solving all the sub-problems $C[i,j,k]$, given the final task is always assigned to the local device, the optimal strategy is solved by the following combining step.

$$\min k \ \text{ s.t. } C[N,1,k] \leq B. \tag{7}$$

Let $|I|$ be the number of bits that are required to represent an instance of our problem. As an FPTAS runs in the time bounded by a polynomial of problem size, $|I|$ and $\frac{1}{\epsilon}$ [13], we have to bound K by choosing T that is larger enough to cover the dynamic range, and choosing the quantization step size δ to achieve the required approximation ratio. To find T, we solve an unconstrained problem for maximum latency given the input instance. We also propose a polynomial-time dynamic programming to solve this problem exactly, which is summarized in Algorithm 1. To realize how the solution provided by Hermes approximates the minimum latency, we take iterative approach and reduce the dynamic range and step size for each iteration until the solution is close enough to the minimum.

We summarize Hermes for tree-structure task graph in Algorithm 2. For r^{th} iteration, we solve for half of the dynamic range with half of the step size compared to last iteration. The procedure DP solves for the minimum quantized latency based on the dynamic programming described in (6). Let $\tilde{\mathbf{x}}$ be the output strategy suggested by the procedure and $L(\tilde{\mathbf{x}})$ be the total latency. Algorithm 2 stops when $L(\tilde{\mathbf{x}}) \geq (1+\epsilon)\frac{T}{2^r}$, or after running $\log_2 T$ iterations, which implies the smallest precision has been reached.

Theorem 2. *Algorithm 2 runs in $O(d_{in}NM^2\frac{l}{\epsilon}\log_2 T)$ time and admits a $(1+\epsilon)$ approximation ratio.*

Proof. From Algorithm 2, each DP procedure solves NMK sub-problems, where $K = \lceil \frac{T_r}{\delta_r} \rceil = O(\frac{l}{\epsilon})$. Let d_{in} denote the maximum indegree of the task graph. For solving each sub-problem in (6), there are at most d_{in} minimization problems over M devices. Hence, the overall complexity of a DP procedure can be bounded by

$$O(NMK \times d_{in}M) = O(d_{in}NM^2\frac{l}{\epsilon}).$$

Algorithm 2 involves at most $\log_2 T$ iterations, hence, it runs in $O(d_{in}NM^2\frac{l}{\epsilon}\log_2 T)$ time. Since both l and d_{in} of a tree can be bounded by N, and $\log_2 T$ is bounded by the number of bits to represent the instance, Algorithm 2 runs in polynomial time of problem size, $|I|$ and $\frac{1}{\epsilon}$.

Now we prove the performance guarantee provided by Algorithm 2. For a given strategy $\mathbf{x}$, let $\hat{L}(\mathbf{x})$ denote the quantized latency and $L(\mathbf{x})$ denote the original one. That is, $L(\mathbf{x}) = D(N, \mathbf{x})$. Assume that Algorithm 2 stops at the r^{th} iteration and outputs the assignment strategy $\tilde{\mathbf{x}}$. As $\tilde{\mathbf{x}}$ is the strategy with minimum quantized latency solved by Algorithm 2, we have $\hat{L}(\tilde{\mathbf{x}}) \leq \hat{L}(\mathbf{x}^\star)$, where $\mathbf{x}^\star$ denotes the optimal strategy. For a task graph with depth l, only at most l quantization procedures have been taken. By the quantization defined in (3), it always over-estimates by at most δ_r. Hence, we have

$$L(\tilde{\mathbf{x}}) \leq \delta_r \hat{L}(\tilde{\mathbf{x}}) \leq \delta_r \hat{L}(\mathbf{x}^\star) \leq L(\mathbf{x}^\star) + l\delta_r \tag{8}$$

Since Algorithm 2 stops at the r^{th} iteration, we have

$$(1+\epsilon)\frac{T}{2^r} \leq L(\tilde{\mathbf{x}}) \leq L(\mathbf{x}^\star) + l\delta_r = L(\mathbf{x}^\star) + \epsilon\frac{T}{2^r}.$$

That is,

$$\frac{T}{2^r} \leq L(\mathbf{x}^\star).$$

From (8), we achieve the approximation ratio as required.

$$L(\tilde{\mathbf{x}}) \leq L(\mathbf{x}^\star) + l\delta_r = L(\mathbf{x}^\star) + \epsilon\frac{T}{2^r} \leq (1+\epsilon)L(\mathbf{x}^*). \tag{9}$$

□

As chain is a special case of a tree, Algorithm 2 also applies to the task assignment problem of serial tasks. Instead of using the ILP solver to solve the formulation for serial tasks proposed previously in [11], we have therefore provided an FPTAS to solve it. Furthermore, Algorithm 2 generalizes the FPTAS we have proposed in [24] in the way that we no longer assume that the input instance is bounded.

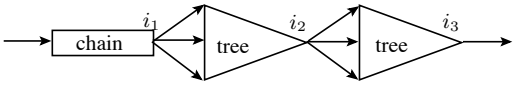

Fig. 4: A task graph of serial trees

4.2 Serial Trees

In [2], several of applications are modeled as task graphs that start from a unique initial task, then split to multiple parallel tasks and finally, all the tasks are merged into one final task. Hence, the task graph is neither a chain nor a tree. In this section, we show that by calling Algorithm 2 in polynomial number of times, Hermes can solve the task graph that consists of serial trees.

The task graph in Fig. 4 can be decomposed into 3 trees connecting serially, where the first tree (chain) terminates in task i_1, the second tree terminates in task i_2. In order to find $C[i_3, j_3, k_3]$, we independently solve for every tree, with the condition on where the root task of the former tree ends. For example, we can solve $C[i_2, j_2, k_2|j_1]$, which is the strategy that minimizes the cost in which task i_2 ends at j_2 within delay k_2 and given task i_1 ends at j_1. Algorithm 2 can solve this sub-problem with the following modification for the leaves.

$$C[i,j,k|j_1] = \begin{cases} C_i^{(j)} + C_{i_1 i}^{(j_1 j)} & \forall k \geq q_\delta(T_i^{(j)} + T_{i_1 i}^{(j_1 j)}), \\ \infty & \text{otherwise} \end{cases} \tag{10}$$

To solve $C[i_2, j_2, k_2]$, the minimum cost up to task i_2, we perform the combining step as

$$C[i_2, j_2, k_2] = \min_{j \in [M]} \min_{k_x + k_y = k_2} C[i_1, j, k_x] + C[i_2, j_2, k_y|j]. \tag{11}$$

Similarly, combining $C[i_2, j_2, k_x]$ and $C[i_3, j_3, k_y|j_2]$ gives $C[i_3, j_3, k_3]$. Algorithm 3 summarizes the steps in solving the assignment strategy for serial trees. To solve each tree involves M calls on different conditions. Further, the number of trees n can be bounded by N. The latency of each tree is within $(1+\epsilon)$ optimal, which leads to the $(1+\epsilon)$ approximation of total latency. Hence, Algorithm 3 is also an FPTAS.

4.3 Parallel Chains of Trees

We take a step further to extend Hermes for more complicated task graphs that can be viewed as parallel chains of trees, as shown in Fig. 1. Our approach is to solve each chains by calling $FPTAS_{path}$ with the condition on the task where they split. For example, in Fig. 1 there are two chains that can be solved independently by conditioning on the split node. The combining procedure consists of two steps. First, solve $C[N, j, k|j_{split}]$ by (6) conditioned on the split node. Then $C[N, j, k]$ can be solved similarly by combining two serial blocks in (11). By calling $FPTAS_{path}$ at most d_{in} times, this proposed algorithm is also an FPTAS.

4.4 Resource Contention on Parallel Tasks

In Fig. 1, the task graph consists of parallel tasks that might be running at the same device at the same time, which

Algorithm 3 Hermes FPTAS for serial trees

procedure $FPTAS_{path}(N)$ ▷ min. cost when task N finishes at devices $1, \cdots, M$ within latencies $1, \cdots, K$
  for root $i_l, l \in \{1, \cdots, n\}$ **do** ▷ solve the conditional sub-problem for every tree
    for $j \leftarrow 1, M$ **do**
      Call $FPTAS_{tree}(i_l)$ conditioning on j with modification described in (10)
  for $l \leftarrow 2, n$ **do**
    Perform combining step in (11) to solve $C[i_l, j_l, k_l]$
end procedure

causes resource contention over CPU cycles, memory usage and network access. For example, when we assign multiple parallel tasks to the same device, the resources are shared over concurrent threads (or processes [2]).

In this section, we consider the resource sharing over sibling tasks. Using Fig. 2 as an example, if task 1 and 2 are running at different devices (x_1 and x_2), they can fully utilize the available resources on the two devices, respectively. The task execution latencies remain the same as $T_1^{(x_1)}$ and $T_2^{(x_2)}$. However, if we assign them to the same device x, then the sharing over CPU cycles leads to longer latencies as $T_1^{(x)} + t$ and $T_2(x) + t$. In general, the task execution latency $T_m^{(x_m)}$ depends on the assignments on its sibling tasks $m \in \mathcal{C}(i)$. Hence, we use t_m to denote the extra latency on executing task m and consider this term when solving the sub-problem in (5).

$$C[i,j,k] = C_i^{(j)} + \min_{x_m : m \in \mathcal{C}(i)} \{ \sum_{m \in \mathcal{C}(i)} C[m, x_m, k - k_m] + C_{mi}^{(x_m j)} \}, \tag{12}$$

$$k_m = q_\delta \big(T_i^{(j)} + T_{mi}^{(x_m j)} + t_m + t_{mi} \big). \tag{13}$$

Note that t_m depends on the assignments $\{x_m : m \in \mathcal{C}(i)\}$, so we have to jointly consider the assignments on these sibling tasks in the minimization problem. On the other hand, the network resource sharing, including sharing the download bandwidth on device j that executes task i, and upload bandwidth on a potential device x_m that executes more than one sibling tasks, induces extra latency as well. Hence, we denote it as t_{mi}, which depends on $\{x_m : m \in \mathcal{C}(i)\}$ and x_j, and consider it in (13).

For resource sharing over globally parallel tasks, we no longer can make optimal decision on each sub-problem independently. This problem is highly related to makespan minimization problems in machine scheduling literature [25], [26], which have been shown to be strongly NP-hard [27]. Garey *et al.* [28] show that if P $\neq$ NP, a strongly NP-hard problem does not have an FPTAS. Hence, we cannot approximate the solution arbitrarily close within polynomial time. Considering the observation that the task graphs are in general more chain-structured with narrow width, like the face recognition and pose recognition benchmarks in [2], we propose Hermes that solves the optimal assignment strategy with low complexity, and addresses the resource contention between local parallel tasks.

2. Depending on the partition granularity, different approaches have been proposed for the system prototypes [4], [11].

5 Applying Hermes to Dynamic Environment

At the application run time, the task execution latency on a device might be affected by its CPU load, memory and other time-varying resource availability. Moreover, data transmission latency over a wireless channel varies with time due to mobility and other dynamic features. In this section, we model the execution latency on a device and the data transmission latency over a channel as stochastic processes. We adapt Hermes to two different scenarios. First, if a system keeps track of the running averages on the single-stage latencies, then given these average numbers, Hermes suggests a strategy to minimize the average latency so that the average cost is within the budget. Second, in case when these averages are unknown, we propose an online version of Hermes to learn the environment and derive its performance guarantee. This online version of Hermes guarantees the convergence to the optimal strategy with an upper bound on the performance loss due to not knowing the devices' and channels' performance at run time.

5.1 Stochastic Optimization

We aim to apply our deterministic to stochastic environment. If both latency and cost metrics are additive over tasks, we can directly apply Hermes to the stochastic environment by assuming that the profiling data is the 1st order expectation. However, it is not clear if we could apply our analysis for parallel computing as the latency metric is nonlinear. For example, for two random variables X and Y, $\mathbb{E}\{\max(X,Y)\} \neq \max(\mathbb{E}\{X\}, \mathbb{E}\{Y\})$ in general. In the following, we exploit the fact that the latency of a single branch is still additive over tasks and show that our deterministic analysis can be directly applied to the stochastic optimization problem, minimizing the expected latency such that the expected cost is less than the budget.

Let $\bar{C}[i,j,k]$ be the minimum expected cost when task i finishes on device j within expected delay k. It suffices to show that the recursive relation in (6) still holds for expected values. As the cost is additive over tasks, we have

$$\bar{C}[i,j,k] = \mathbb{E}\{C_i^{(j)}\} + \sum_{m \in \mathcal{C}(i)} \min_{x_m \in [M]} \{\bar{C}[m, x_m, k - \bar{k}_m] + \mathbb{E}\{C_{mi}^{(x_m j)}\}\}.$$

The $\bar{k}_m$ specifies the sum of expected data transmission delay and expected task execution delay. That is,

$$\bar{k}_m = q_\delta \left(\mathbb{E}\{T_i^{(j)} + T_{mi}^{(x_m j)}\} \right).$$

Based on the fact that Hermes is tractable with respect to both the application size (N) and the network size (M),

we propose an update scheme that is adaptive to dynamic resource network. The strategy is updated every period of time, which aims to minimize the expected latency in the following coherence time period. We will show how the proposed scheme adapts to the changes of network condition in Section 6.

5.2 Learning the Unknown Environment

We adapt the sampling method, deterministic sequencing of exploration and exploitation (DSEE) [14], to learn the unknown environment and derive the performance bound. The DSEE algorithm consists of two phases, exploration and exploitation. During the exploration phase, DSEE follows a fixed order to probe (sample) the unknown distributions thoroughly. Then, in the exploitation phase, DSEE exploits the best strategy based on the probing result.

In [14], learning the unknown environment is modeled as a multi-arm banded (MAB) problem, where at each time an agent chooses over a set of "arms", gets the payoff from the selected arm and tries to learn the statistical information from sensing it, which will be considered in future decision. The goal is to figure out the best arm from exploration and exploit it later on. However, the exploration costs some price due to the mismatch between the payoffs given by the explored arm and the best one [29]. Hence, we have to efficiently explore the environment and compare the performance with the optimal strategy (always choose the best arm).

The authors in [14] prove that the performance gap compared to the optimal strategy is bounded by a logarithmic function of number of trials as long as each arm is sampled logarithmically often. That is, if we get enough samples from each arm ($O(\ln V)$) compared to total trials V, we can make good enough decision such that the accumulated performance loss flats out with time, which implies we can learn and exploit the best arm without losing noticeable payoff in the end.

In the following, we adapt DSEE and combine Hermes to learn the unknown and dynamic environment, and derive the bound on performance loss compared to the optimal strategy. We model the execution latency as

$$T_i^{(j)} = \alpha_i T^{(j)}, \tag{14}$$

where α_i is the task complexity and $T^{(j)}$ is the latency of executing an unit task on device j, which is highly related to its CPU clock rate. We use linear model to simplify our analysis and presentation. In general, the task execution latency is a nonlinear function of task complexity, CPU clock rate and other factors [30]. We further assume that $T^{(j)}$ is an i.i.d. process with unknown mean $\theta^{(j)}$. Similarly, the data transmission latency $T_{mn}^{(jk)}$ can be expressed as

$$T_{mn}^{(jk)} = d_{mn} T^{(jk)}, \tag{15}$$

where d_{mn} is the amount of data exchange and $T^{(jk)}$ is the transmission latency of unit data, which is also modeled as an i.i.d. process with mean $\theta^{(jk)}$.

For some real applications, like video processing applications considered in [2], a stream of video frames comes as input to be processed frame by frame. For example, a video-processing application takes a continuous stream of image frames as input, where each image comes and goes though all processing tasks as shown in Fig. 1. Hence, for each data frame, our proposed algorithm aims to make decision on the assignment strategy of current frame, considering the performance of different assignment strategies learned from previous frames.

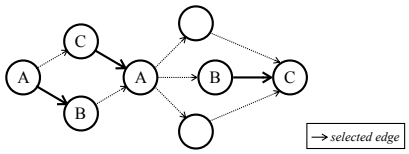

Fig. 5: The task graph has matching number equal to 3. Hence, we can sample at least 3 channels (AB, CA, BC) in one execution. We can further assign tasks that are left blank to other devices to get more samples.

We combine Hermes with DSEE to sample all devices and channels thoroughly at the exploration phase, calculate the sample means, and apply Hermes to solve and exploit the optimal assignment based on sample means. During the exploration phase, we design a fixed assignment strategy to get samples from devices and channels. For example, if task n follows after the execution of task m, by assigning task m to device j and assigning task n to device k, we could get one sample of $T^{(j)}$, $T^{(k)}$ and $T^{(jk)}$. Since sampling all the M^2 channels implies that all devices have been sampled M times, we focus on sampling all channels using as less executions of the application as possible. That is, we would like to know, for each frame (an execution of the application), what is the maximum number of different channels we can get a sample from. This number depends on the structure of the task graph, which, in fact, is lower-bounded by the matching number of the graph. A matching on a graph is a set of edges, where no two of which share a node [31]. The matching number of a graph is then the maximum number of edges that does not share a node. Taking an edge from the set, which connects two tasks in the task graph, we can assign these two tasks arbitrarily to get a sample of data transmission over our desired channel.

Fig. 5 illustrates how we design the task assignment to sample as many channels in one execution. First, we treat every directed edges as non-directed ones and find out the graph has matching number equal to 3. That is, we can sample at least 3 channels (AB, CA, BC) in one execution. There are some tasks that are left blank. We can assign them to other devices to get more samples.

In every exploration epoch, we want to get at least one sample from every channel. Hence, we want to know how many frames (executions) are needed in one epoch. We derive a bound for general case. For a DAG, its matching number is shown to be lower-bounded by $\frac{|\mathcal{E}|}{d_{max}}$, where d_{max} is the maximum degree of a node [32]. For example, the matching number of the graph in Fig. 5 is lower bounded by $\frac{10}{5} = 2$. Hence, to sample each channel at least once, we require at most $r = \lceil \frac{d_{max} M^2}{|\mathcal{E}|} \rceil$ frames.

Algorithm 4 summarizes how we adapt Hermes to dynamic environment. We separate the time (frame) horizon into epoches, where each of them contains r frames. Let $\mathcal{A}(v-1) \subseteq \{1, \cdots, v-1\}$ be the set of exploration epoches

Algorithm 4 Hermes with DSEE

procedure $Hermes_{DSEE}(w)$
  $r \leftarrow \lceil \frac{d_{max} M^2}{|\mathcal{E}|} \rceil$
  $\mathcal{A}(0) \leftarrow \emptyset$ ▷ $\mathcal{A}(v)$ defines the set of exploration epoches up to v
  for $v \leftarrow 1, \cdots, V$ **do**
    if $|\mathcal{A}(v-1)| < \lceil w \ln v \rceil$ **then** ▷ exploration phase
      for $t \leftarrow 1, \cdots, r$ **do** ▷ each epoch contains r frames
        Sample the channels with strategy $\hat{\mathbf{x}}$
      Calculate the sample means, $\bar{\theta}^{(j)}(v)$ and $\bar{\theta}^{(jk)}(v)$, for all $j, k \in [M]$
      $\mathcal{A}(v) \leftarrow \mathcal{A}(v-1) + \{v\}$
    else ▷ exploitation phase
      Solve the best strategy $\tilde{\mathbf{x}}(v)$ with input $T_i^{(j)} = \alpha_i \bar{\theta}^{(j)}(v)$ and $T_{mn}^{(jk)} = d_{mn} \bar{\theta}^{(jk)}(v)$
      for $t \leftarrow 1, \cdots, r$ **do**
        Exploit the assignment strategy $\tilde{\mathbf{x}}(v)$
end procedure

prior to v. At epoch v, if the number of exploration epoches is below the threshold ($|\mathcal{A}(v-1)| < \lceil w \ln v \rceil$), then epoch v is an exploration epoch. Algorithm 4 uses a fixed assignment strategy $\hat{\mathbf{x}}$ to get samples. After r frames have been processed, Algorithm 4 gets at least one new sample from each channel and device, and updates the sample means. At an exploitation epoch, Algorithm 4 calls Hermes to solve for the best assignment strategy $\tilde{\mathbf{x}}(v)$ based on current sample means, and uses this assignment strategy for the frames in this epoch.

In the following, we derive the performance guarantee of Algorithm 4. First, we present a lemma from [14], which specifies the probability bound on the deviation of sample mean.

Lemma 1. *Let $\{X(t)\}_{t=1}^{\infty}$ be i.i.d. random variables drawn from a light-tailed distribution, that is, there exists $u_0 > 0$ such that $\mathbb{E}[\exp(uX)] < \infty$ for all $u \in [-u_0, u_0]$. Let $\bar{X}_s = \frac{\sum_{t=1}^{s} X(t)}{s}$ and $\theta = \mathbb{E}[X(1)]$. We have, given $\zeta > 0$, for all $\eta \in [0, \zeta u_0]$, $a \in (0, \frac{1}{2\zeta}]$,*

$$\mathbb{P}\{|\bar{X}_s - \theta| \geq \eta\} \leq 2\exp(-a\eta^2 s). \tag{16}$$

Lemma 1 implies the more samples we get, the much less chance the sample mean deviates from the actual mean. From (2), the overall latency is the sum of single-stage latencies ($T_i^{(j)}$ and $T_{mn}^{(jk)}$) across the slowest branch. Hence, we would like to use Lemma 1 to get a bound on the deviation of total latency. Let β be the maximum latency solved by Algorithm 1 with the following input instance

$$T_i^{(j)} = \alpha_i, \ \forall i \in [N], j \in [M],$$
$$T_{mn}^{(jk)} = d_{mn}, \ \forall (m,n) \in \mathcal{E}, j, k \in [M].$$

Hence, if all the single-stage sample means deviate no more than η from their actual means, then the overall latency deviates no more than $\beta\eta$. In order to prove the performance guarantee of Algorithm 4, we identify an event and verify the bound on its probability in the following lemma.

Lemma 2. *Assume that $T^{(j)}$, $T^{(jk)}$ are independent random variables drawn from unknown light-tailed distributions with means $\theta^{(j)}$ and $\theta^{(jk)}$, for all $j, k \in [M]$. Let a, η be the numbers that satisfy Lemma 1. For each assignment strategy $\mathbf{x}$, let $\bar{\theta}(\mathbf{x}, v)$ be the total latency accumulated over the sample means that are calculated at epoch v, and $\theta(\mathbf{x})$ be the actual expected total latency. We have, for each v,*

$$\mathbb{P}\{\exists \mathbf{x} \in [M]^N \mid |\bar{\theta}(\mathbf{x}, v) - \theta(\mathbf{x})| > \beta\eta\}$$
$$\leq \sum_{n \in [M^2+M]} \binom{M^2+M}{n} (-1)(-2)^n e^{-na\eta^2 |\mathcal{A}(v-1)|}.$$

Proof. We want to bound the probability that there exists a strategy whose total deviation (accumulated over sample means) is greater than $\beta\eta$. We work on its complement event that the total deviation of each strategy is less than $\beta\eta$. That is,

$$\mathbb{P}\{\exists \mathbf{x} \in [M]^N \mid |\bar{\theta}(\mathbf{x}, v) - \theta(\mathbf{x})| > \beta\eta\}$$
$$= 1 - \mathbb{P}\{|\bar{\theta}(\mathbf{x}, v) - \theta(\mathbf{x})| \leq \beta\eta \ \forall \mathbf{x} \in [M]^N\}$$

We further identify the fact that if every single-stage deviation is less than η, then the total deviation is less than $\beta\eta$ for all strategy $\mathbf{x} \in [M^N]$. Hence,

$$1 - \mathbb{P}\{|\bar{\theta}(\mathbf{x}, v) - \theta(\mathbf{x})| \leq \beta\eta \ \forall \mathbf{x} \in [M]^N\}$$
$$\leq 1 - \mathbb{P}\{(\bigcap_{j \in [M]} |\bar{\theta}^{(j)} - \theta^{(j)}| \leq \eta) \cap (\bigcap_{j,k \in [M]} |\bar{\theta}^{(jk)} - \theta^{(jk)}| \leq \eta)\}$$
$$= 1 - \prod_{j \in [M]} \mathbb{P}\{|\bar{\theta}^{(j)} - \theta^{(j)}| \leq \eta\} \cdot \prod_{j,k \in [M]} \mathbb{P}\{|\bar{\theta}^{(jk)} - \theta^{(jk)}| \leq \eta\}$$
$$\leq 1 - \left[1 - 2e^{-a\eta^2 |\mathcal{A}(v-1)|}\right]^{M^2+M} \tag{17}$$
$$\leq \sum_{n \in [M^2+M]} \binom{M^2+M}{n} (-1)(-2)^n e^{-na\eta^2 |\mathcal{A}(v-1)|} \tag{18}$$

Leveraging the fact that all of random variables are independent and Lemma 1, where at epoch v, we get at least $|\mathcal{A}(v-1)|$ samples for each unknown distribution, we arrive at (17). Finally, we use the binomial expansion to achieve the bound in (18). □

In the following, we compare the performance of Algorithm 4 with the optimal strategy (assuming the actual averages, $\theta^{(j)}$ and $\theta^{(jk)}$, are known), which is obtained by solving Problem $\mathbf{P}$ with the input instance

$$T_i^{(j)} = \alpha_i \theta^{(j)}, \ \forall i \in [N], j \in [M],$$
$$T_{mn}^{(jk)} = d_{mn} \theta^{(jk)}, \ \forall (m,n) \in \mathcal{E}, j, k \in [M].$$

Theorem 3. *Let $\eta = \frac{c}{2\beta}$, where c is the smallest precision so that for any two assignment strategies $\mathbf{x}$ and $\mathbf{y}$, we have $|\theta(\mathbf{x}) - \theta(\mathbf{y})| > c$ whenever $\theta(\mathbf{x}) \neq \theta(\mathbf{y})$. Let R_V be the expected performance gap accumulated up to epoch V, which can be bounded by*

$$R_V \leq rT(w \ln V + 1) + rT \sum_{n \in [M^2+M]} \binom{M^2+M}{n} (-1)(-2)^n \left(1 + \frac{1}{na\eta^2 w - 1}\right)$$

Proof. The expected performance gap consists of two parts, the expected loss due to the use of fixed strategy during exploration (R_V^{fix}) and the expected loss due to the mismatch of strategies during exploitation (R_V^{mis}). During the exploration phase, the expected loss of each frame can be bounded by T, which can be obtained by Algorithm 1 with $\alpha_i \theta^{(j)}$ and $d_{mn}\theta^{(jk)}$ as input instance. Since the number of exploration epoches $|\mathcal{A}(v)|$ will never exceed $(w \ln V + 1)$, we have

$$R_V^{fix} \leq rT(w \ln V + 1).$$

On the other hand, R_V^{mis} is accumulated during the exploitation phase whenever the best strategy given by sample means is not the same as the optimal strategy, where the loss can also be bounded by T. That is,

$$R_V^{mis} \leq \mathbb{E}\{\sum_{v \notin \mathcal{A}(v)} rT\, \mathbb{I}(\tilde{\mathbf{x}}(v) \neq \mathbf{x}^\star)\} = rT \sum_{v \notin \mathcal{A}(v)} \mathbb{P}\{\tilde{\mathbf{x}}(v) \neq \mathbf{x}^\star\}$$

$$\leq rT \sum_{v \notin \mathcal{A}(v)} \mathbb{P}\{\exists \mathbf{x} \in [M]^N \mid |\bar{\theta}(\mathbf{x}, v) - \theta(\mathbf{x})| > \beta\eta\} \quad (19)$$

$$\leq rT \sum_{v \notin \mathcal{A}(v)} \sum_{n \in [M^2+M]} \binom{M^2+M}{n} (-1)(-2)^n e^{-na\eta^2 |\mathcal{A}(v-1)|} \quad (20)$$

$$\leq rT \sum_{n \in [M^2+M]} \binom{M^2+M}{n} (-1)(-2)^n \sum_{v=1}^{\infty} v^{-na\eta^2 w} \quad (21)$$

$$\leq rT \sum_{n \in [M^2+M]} \binom{M^2+M}{n} (-1)(-2)^n \left(1 + \frac{1}{na\eta^2 w - 1}\right) \quad (22)$$

In (19), we want to bound the probability when the best strategy based on sample means is not the optimal strategy. We identify an event, where there exists a strategy $\mathbf{x}$ whose deviation is greater than $\beta\eta$. If this event doesn't happen, in worst case, the difference between any two strategies deviates at most $2\beta\eta = c$. Hence, $\bar{\theta}(\mathbf{x}^\star, v)$ is still the minimum, which implies Algorithm 4 still outputs the optimal strategy. We further use Lemma 2 in (20) and acquire (21) by the fact that epoch v is in exploration phase implies $|\mathcal{A}(v-1)| >= w \ln v$. Finally, selecting w to be larger enough such that $a\eta^2 w > 1$ guarantees the result in (22). □

Theorem 3 shows that the performance gap consists of two parts, one of which grows logarithmically with V and another one remains the same as V is increasing. Hence, the increase of performance gap will be negligible when V (time) grows, which implies Algorithm 4 will find the strategy that matches to the optimal performance as time goes on. Furthermore, Theorem 3 provides the upper bound on the performance loss based on the worst-case analysis, in which w is a parameter left for users in Algorithm 4. A smaller w leads to less amount of probing (exploration) and hence reduces the accumulated loss during exploration, however, may increase the chance of missing the optimal strategy during exploitation. In next section, we will compare Algorithm 4 with other algorithms by simulation.

6 Evaluation of Hermes

We first verify that Hermes provides near-optimal solution with tractable complexity. Then, we apply Hermes to the dynamic environment, using the sampling method proposed in Algorithm 4. We also use the real data set of several benchmark profiles to evaluate the performance of Hermes and compare it with the heuristic Odessa approach proposed in [2]. Finally, couple of run-time scenarios like resource contention and node failure are evaluated.

6.1 Algorithm Performance

From our analysis result in Section 4, the Hermes algorithm runs in $O(d_{in} N M^2 \frac{l}{\epsilon} \log_2 T)$ time with approximation ratio $(1+\epsilon)$. In the following, we provide the numerical results to show the trade-off between the complexity and the accuracy. Given the task graph shown in Fig. 1 and $M = 3$, the performance of Hermes versus different values of ϵ is shown in Fig. 6. When $\epsilon = 0.4$, the performance converges to the minimum latency. Fig. 6 also shows the bound of worst case performance in dashed line. The actual performance is much better than the $(1+\epsilon)$ bound. We generalize our previous result in [24] that Hermes admits $(1+\epsilon)$ approximation for all problem instances, including the unbounded ones. Our previous result admits a $(1+c\epsilon)$ performance bound, where c depends on the input instance.

We examine the performance of Hermes on different problem instances. Fig. 7 shows the performance of Hermes on 200 different application profiles. Each profile is selected independently and uniformly from the application pool with different task workloads and data communications. The result shows that for every instance we have considered, the performance is much better than the $(1+\epsilon)$ bound and converges to the optimum as ϵ decreases.

6.2 CPU Time Evaluation

Fig. 8 shows the CPU time for Hermes to solve for the optimal strategy as the problem size scales. We use a less powerful laptop with very limited resources to simulate a mobile computing environment and use java management package for CPU time measurement. The laptop is equipped with 1.2GHz dual-core Intel Pentium processor and 1MB cache. For each problem size, we measure Hermes' CPU time over 100 different problem instances and show the average with vertical bar as standard deviation. As the number of tasks (N) increases in a serial task graph, the CPU time needed for the Brute-Force algorithm grows exponentially, while Hermes scales well and still provides the near-optimal solution ($\epsilon = 0.01$). From our complexity analysis, for serial task graph $l = N$, $d_{in} = 1$ and we fix $M = 3$, the CPU time of Hermes can be bounded by $O(N^2)$.

6.3 Performance on Dynamic Environment

We simulate an application that processes a stream of data frames under dynamic environment. The resource network consists of 3 devices with unit process time $T^{(j)}$ on device j.

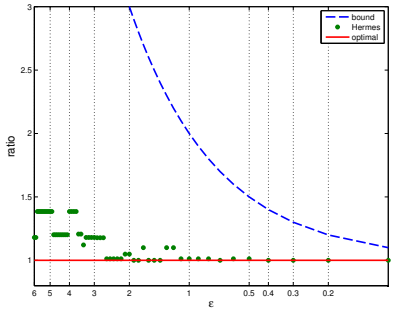

Fig. 6: Hermes performs much better than the worst case bound. When $\epsilon = 0.4$, the objective value has converged to the minimum.

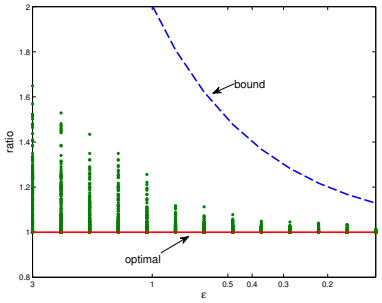

Fig. 7: The performance of Hermes over 200 different application profiles. Each dot represents an application profile that is solved with a given ϵ value.

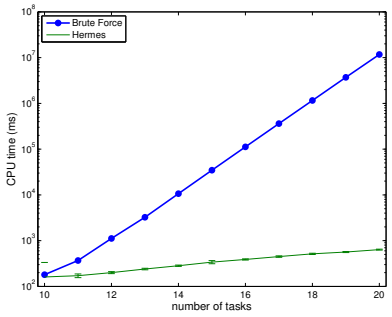

Fig. 8: The CPU time overhead for Hermes as the problem size scales ($\epsilon = 0.01$).

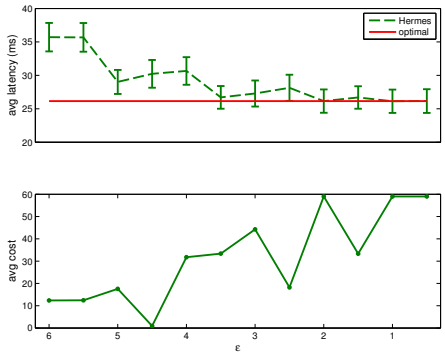

Fig. 9: The expected latency and cost over 10000 samples of resource network.

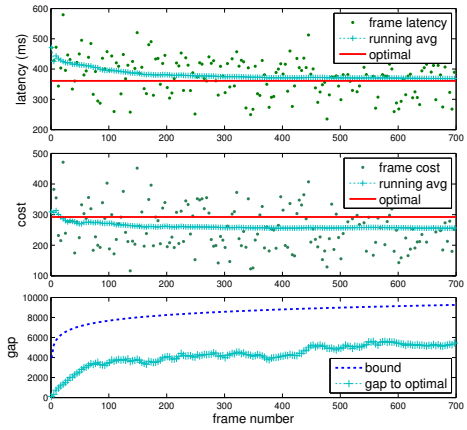

Fig. 10: The performance of Hermes using DSEE sampling method in dynamic environment. The average of frame latency approaches to the optimum and the accumulated performance gap compared to the optimal strategy flats out as the number of frames increases.

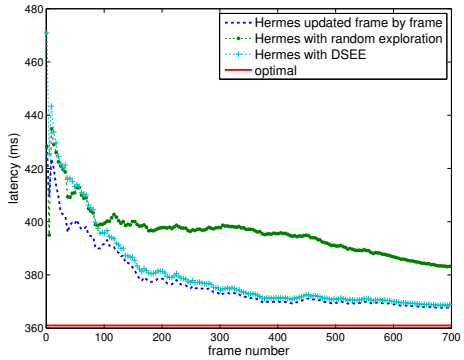

Fig. 11: Hermes using DSEE only resolves the strategy at the beginning of each exploitation phase but offers competitive performance compared to the algorithm that resolves the strategy every frame.

The devices form a mesh network with unit data transmission time $T^{(jk)}$ over the channel between device j and k. We model $T^{(j)}$ and $T^{(jk)}$ as stochastic processes that are uniformly-distributed with given means and evolve i.i.d. over time. Hence, for each frame, we draw the samples from corresponding uniform distributions, and get the single-stage latencies by (14) and (15).

6.3.1 Stochastic Optimization

If the means of these stochastic processes are known, Hermes can solve for the best strategy based on these means. Fig. 9 shows that how the strategies suggested by Hermes perform under the dynamic environment. The average performance is taken over 10000 samples. From Fig. 9, the solution converges to the optimal one as epsilon decreases, which minimizes the expected latency and satisfies the expected cost constraint.

6.3.2 Online Learning to Unknown Environment

If the means are unknown, we adapt Algorithm 4 to probe the devices and channels and exploit the strategy that is the best based on the sample means. Fig. 10 shows the performance of Hermes using DSEE as the sampling method. We see that the average latency per frame converges to the minimum, which implies Algorithm 4 learns the optimal strategy and exploits it most of the time. On the other hand, Algorithm 4 uses the strategy that costs less but performs worse than the optimal one during the exploration phase. Hence, the average cost per frame is slightly lower than the cost induced by the optimal strategy. Finally, we measure the performance gap, which is the extra latency caused by sub-optimal strategy accumulated over frames. The gap flats

out in the end, which implies the increase on extra latency becomes negligible.

We compare Algorithm 4 with two other algorithms in Fig. 11. First, we propose a randomized sampling method as a baseline. During exploration phase, Algorithm 4 designs a fixed strategy to sample the devices and channels thoroughly. However, the baseline randomly selects an assignment strategy and gather the samples. The biased sample means result in significant performance loss during exploitation phase. We propose another algorithm that resolves the best strategy every frame. That is, at the end of each frame, it updates the sample means and runs Hermes to solve for the best strategy for the next frame. We can see that by updating the strategy every frame, the performance is slightly better than Algorithm 4. However, Algorithm 4 only runs Hermes at the beginning of each exploitation phase, which only increases tolerable amount of CPU load but provides competitive performance. We will examine the extra CPU load on running Hermes in the next section.

6.4 Benchmark Evaluation

In [2], Ra *et al.* present several benchmarks of perception applications for mobile devices and propose Odessa, to improve both makespan and throughput with the help of a cloud connected server. To improve the performance, for each data frame, Odessa first identifies the bottleneck, evaluates each strategy with simple metrics and finally select the potentially best one to mitigate the load on the bottleneck. However, Odessa as a greedy heuristic does not offer any theoretical performance guarantee, as shown in Fig. 12 Hermes can improve the performance by 36% for task graph in Fig. 1.

To evaluate Hermes and Odessa on real applications, we further choose two of benchmarks proposed in [2] for comparison. Taking the timestamps of every stage and the corresponding statistics measured in real executions provided in [2], we emulate the executions of these benchmarks and evaluate the performance.

In dynamic resource scenarios, as Hermes' complexity is not as light as the greedy heuristic (86.87 ms in average) and its near-optimal strategy needs not be updated from frame to frame under similar resource conditions, we propose the following on-line update policy: similar to Odessa, we record the timestamps for on-line profiling. Whenever the latency difference of current frame and last frame goes beyond the threshold, we run Hermes based on current profiling to update the strategy. By doing so, Hermes always gives the near-optimal strategy for current resource scenario and enhances the performance at the cost of reasonable CPU time overhead due to resolving the strategy.

As Hermes provides better performance in latency but induces more CPU time overhead, we define two metrics for comparison. Let $Latency(t)$ be the normalized latency advantage of Hermes over Odessa up to frame number t. Let $CPU(t)$ be the normalized CPU advantage of Odessa over Hermes up to frame number t. That is,

$$Latency(t) = \frac{1}{t}\sum_{i=1}^{t}\big(L_O(i) - L_H(i)\big), \tag{23}$$

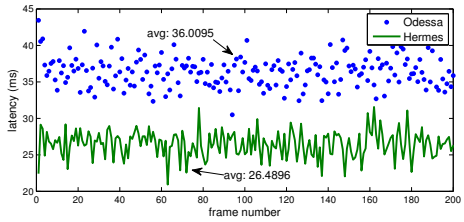

Fig. 12: Hermes can improve the performance by 36% compared to Odessa for task graph shown in Fig. 1.

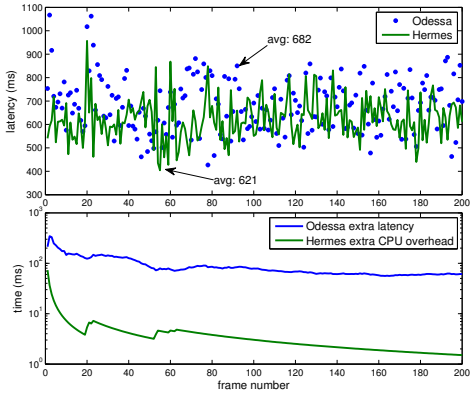

Fig. 13: Top: Hermes improves the average latency of each data frame by 10%. Bottom: the latency advantage of Hermes over Odessa ($Latency(t)$) is significant enough to compensate its CPU time overhead ($CPU(t)$).

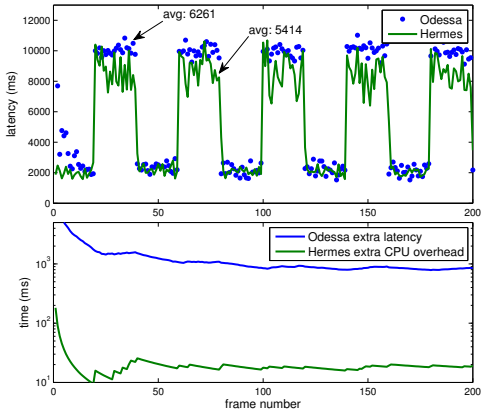

Fig. 14: Hermes improves the average latency of each data frame by 16% and well-compensates its CPU time overhead.

$$CPU(t) = \frac{1}{t}\Big(\sum_{i=1}^{C(t)} CPU_H(i) - \sum_{i=1}^{t} CPU_O(i)\Big), \tag{24}$$

where $L_O(i)$ and $CPU_O(i)$ are latency and update time of frame i given by Odessa, and the notations for Hermes are similar except that we use $C(t)$ to denote the number of times that Hermes updates the strategy up to frame t.

To model the dynamic resource network, the latency of each stage is selected independently and uniformly from a distribution with its mean and standard deviation provided by the statistics of the data set measured in real applications. In addition to small scale variation, the link coherence time is 20 data frames. That is, for some period, the link quality

TABLE 4: Mobile Energy Evaluation

Budget ($mW \cdot sec$)	Latency (sec)	Energy ($mW \cdot sec$)
50	2.058 ± 0.290	41.194 ± 13.548
40	2.212 ± 0.313	27.664 ± 11.756
30	2.205 ± 0.305	27.371 ± 11.130
20	4.364 ± 0.838	12.958 ± 7.220
local	16.710 ± 3.483	8.137 ± 3.341

degrades significantly due to possible fading situations. Fig. 13 shows the performance of Hermes and Odessa for the face recognition application. Hermes improves the average latency of each data frame by 10% compared to Odessa and increases CPU computing time by only 0.3% of overall latency. That is, the latency advantage provided by Hermes well-compensates its CPU time overhead. Fig. 14 shows that Hermes improves the average latency of each data frame by 16% for pose recognition application and increases CPU computing time by 0.4% of overall latency. When the link quality is degrading, Hermes updates the strategy to reduce the data communication, while Odessa's sub-optimal strategy results in significant extra latency. Considering CPU processing speed is increasing under Moore's law but network condition does not change that fast, Hermes provides a promising approach to trade-in more CPU for less network consumption cost.

6.4.1 Energy Consumption on Mobile Devices

We use the trace data from the pose recognition benchmark [2] and the power characteristics model proposed in [33] to evaluate the energy consumption on a mobile device for different assignment strategies. For each strategy, we evaluate the performance on latency and energy consumption over 200 frames, with mean and standard deviation as shown in Table 4. Under various budget constraints, Hermes adapts to different assignment strategies that minimize the latency and fit the budget. Compared to pure local execution, computational offloading consumes more energy due to cellular data transmission. However, Hermes identifies the offloading strategy that induces limited data transmission while offloads intensive tasks, to significantly improve latency performance under stringent budget.

6.4.2 Resource Contention and Node Failure

In Section 4.4, we adapt Hermes to consider resource contention on "local" parallel tasks and still provide the optimal strategy if the task graph can be decomposed into serial trees, like the face recognition and pose recognition benchmarks in [2]. For the task graphs that contain global parallel tasks (Fig. 1), Hermes' solution may be sub-optimal for some problem instances. In this section, we use such a task graph shown in Fig. 1 to examine Hermes' performance degradation in the worst case. That is, whenever two parallel tasks are assigned to the same device, we add up the latencies, assuming the application executes in a single thread on a single-processor device. Fig 15 shows Hermes' performance over 50 randomly-chosen problem instances, compared to the ideal parallel execution (no resource contention) and the optimal strategy. We observe that for 50% of instances, Hermes still matches the optimal performance. While for

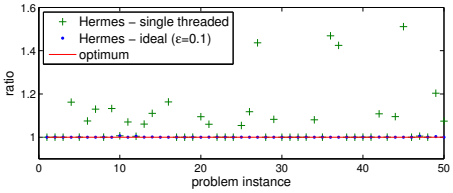

Fig. 15: Hermes' performance on a single-processor, single-threaded device

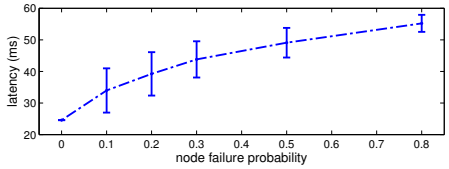

Fig. 16: Latency overhead due to node failure

the instances when Hermes assigns global parallel tasks to a single devices, it suffers from performance degradation up to 1.5 times in the worst case.

We propose a node failure recovery scheme in Section 2.1 that does not require extra data transmission but only some control signals. The system re-executes the task in the preceding device when node failure or data transmission failure happens, in order to minimize the latency overhead. We use the independent node failure model to examine the system performance, where each node fails with probability p for each task execution. Fig. 16 shows the latency overhead under different node failure probabilities. We observe that the latency overhead increases with p, up to 100% when node failure happens 80% of the time.

7 Conclusions

We have formulated a task assignment problem and provided an FPTAS algorithm, Hermes, to solve for the optimal strategy that balances between latency improvement and energy consumption of mobile devices. Compared with previous formulations and algorithms, to the best of our knowledge, Hermes is the first polynomial time algorithm to address the latency-resource tradeoff problem with provable performance guarantee. Moreover, Hermes is applicable to more sophisticated formulations on the latency metrics considering more general task dependency constraints as well as multi-device scenarios. The CPU time measurement shows that Hermes scales well with problem size. We have further emulated the application execution by using the real data set measured in several mobile benchmarks, and shown that our proposed on-line update policy, integrating with Hermes, is adaptive to dynamic network change. Furthermore, the strategy suggested by Hermes performs much better than greedy heuristic so that the CPU overhead of Hermes is well compensated. Extending Hermes to consider resource contention on a general directed acyclic task graph, known as a strongly NP-hard problem, and optimally scheduling tasks when using pipelining strategies, are worthy of detailed investigation in the future.

REFERENCES

[1] E. Miluzzo, T. Wang, and A. T. Campbell, "Eyephone: activating mobile phones with your eyes," in *ACM SIGCOMM*. ACM, 2010, pp. 15–20.
[2] 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.
[3] 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.
[4] 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.
[5] 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 *IEEE INFOCOM*. IEEE, 2012, pp. 945–953.
[6] W. Li, Y. Zhao, S. Lu, and D. Chen, "Mechanisms and challenges on mobility-augmented service provisioning for mobile cloud computing," *IEEE Communications Magazine*, vol. 53, no. 3, pp. 89–97, 2015.
[7] 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.
[8] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, S. N. Srirama, and R. Buyya, "A context sensitive offloading scheme for mobile cloud computing service," in *IEEE CLOUD*. IEEE, 2015, pp. 869–876.
[9] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: enabling remote computing among intermittently connected mobile devices," in *ACM MobiHoc*. ACM, 2012, pp. 145–154.
[10] M. Y. Arslan, I. Singh, S. Singh, H. V. Madhyastha, K. Sundaresan, and S. V. Krishnamurthy, "Cwc: A distributed computing infrastructure using smartphones," *Mobile Computing, IEEE Transactions on*, 2014.
[11] 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.
[12] C. Wang and Z. Li, "Parametric analysis for adaptive computation offloading," *ACM SIGPLAN*, vol. 39, no. 6, pp. 119–130, 2004.
[13] G. Ausiello, *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer, 1999.
[14] S. Vakili, K. Liu, and Q. Zhao, "Deterministic sequencing of exploration and exploitation for multi-armed bandit problems," *Selected Topics in Signal Processing, IEEE Journal of*, vol. 7, no. 5, pp. 759–767, 2013.
[15] R. M. Karp, *Reducibility among combinatorial problems*. Springer, 1972.
[16] G. L. Nemhauser and L. A. Wolsey, *Integer and combinatorial optimization*. Wiley New York, 1988, vol. 18.
[17] Y.-H. Kao and B. Krishnamachari, "Optimizing mobile computational offloading with delay constraints," in *IEEE GLOBECOM*. IEEE, 2014.
[18] O. Goldschmidt and D. S. Hochbaum, "A polynomial algorithm for the k-cut problem for fixed k," *Mathematics of operations research*, vol. 19, no. 1, pp. 24–37, 1994.
[19] H. Bagheri, P. Karunakaran, K. Ghaboosi, T. Bräysy, and M. Katz, "Mobile clouds: Comparative study of architectures and formation mechanisms," in *IEEE WiMob*. IEEE, 2012, pp. 792–798.
[20] K. Habak, M. Ammar, K. A. Harras, and E. Zegura, "Femto clouds: Leveraging mobile devices to provide cloud service at the edge," in *IEEE CLOUD*. IEEE, 2015, pp. 9–16.
[21] M. R. Rahimi, N. Venkatasubramanian, S. Mehrotra, and A. V. Vasilakos, "Mapcloud: mobile applications on an elastic and scalable 2-tier cloud architecture," in *IEEE/ACM UCC*. IEEE, 2012, pp. 83–90.
[22] V. Cardellini, V. D. N. Personé, V. Di Valerio, F. Facchinei, V. Grassi, F. L. Presti, and V. Piccialli, "A game-theoretic approach to computation offloading in mobile cloud computing," *Mathematical Programming*, vol. 157, no. 2, pp. 421–449, 2016.
[23] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, and E. Zegura, "Cosmos: computation offloading as a service for mobile devices," in *ACM MobiHoc*. ACM, 2014, pp. 287–296.
[24] Y.-H. Kao, B. Krishnamachari, M.-R. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," in *IEEE INFOCOM*. IEEE, 2015, pp. 1894–1902.
[25] P. Schuurman and G. J. Woeginger, "Polynomial time approximation algorithms for machine scheduling: Ten open problems," *Journal of Scheduling*, vol. 2, no. 5, pp. 203–213, 1999.
[26] K. Jansen and R. Solis-Oba, "Approximation algorithms for scheduling jobs with chain precedence constraints," in *Parallel Processing and Applied Mathematics*. Springer, 2004, pp. 105–112.
[27] J. Du, J. Y. Leung, and G. H. Young, "Scheduling chain-structured tasks to minimize makespan and mean flow time," *Information and Computation*, vol. 92, no. 2, pp. 219–236, 1991.
[28] M. R. Garey and D. S. Johnson, ""strong"np-completeness results: Motivation, examples, and implications," *Journal of the ACM (JACM)*, vol. 25, no. 3, pp. 499–508, 1978.
[29] S. Bubeck and N. Cesa-Bianchi, "Regret analysis of stochastic and nonstochastic multi-armed bandit problems," *arXiv preprint arXiv:1204.5721*, 2012.
[30] L. Luo and B. E. John, "Predicting task execution time on handheld devices using the keystroke-level model," in *ACM CHI*. ACM, 2005, pp. 1605–1608.
[31] H. N. Gabow, "An efficient implementation of edmonds' algorithm for maximum matching on graphs," *Journal of the ACM (JACM)*, vol. 23, no. 2, pp. 221–234, 1976.
[32] Y. Han, "Tight bound for matching," *Journal of combinatorial optimization*, vol. 23, no. 3, pp. 322–330, 2012.
[33] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "A close examination of performance and power characteristics of 4g lte networks," in *ACM MobiSys*. ACM, 2012, pp. 225–238.

Yi-Hsuan Kao received his B.S. in Electrical Engineering at National Taiwan University, Taipei, Taiwan, in 1998, and his M.S. and Ph.D. degrees from University of Southern California in 2012 and 2016 respectively. He is a data scientist at Supplyframe, Pasadena. His research interest is in approximation algorithms and online learning algorithms.

Bhaskar Krishnamachari received his B.E. in Electrical Engineering at The Cooper Union, New York, in 1998, and his M.S. and Ph.D. degrees from Cornell University in 1999 and 2002 respectively. He is a Professor in the Department of Electrical Engineering at the University of Southern California's Viterbi School of Engineering. His primary research interest is in the design, analysis and evaluation of algorithms and protocols for next-generation wireless networks.

Moo-Ryong Ra is a systems researcher in Cloud Platform Software Research department at AT&T Labs Research. Currently his primary research interest lies in the area of software-defined storage in a virtualized datacenter. He earned a Ph.D. degree from the Computer Science Department at University of Southern California (USC) in 2013. Prior to that, he received an M.S. degree from the same school(USC) in 2008 and a B.S. degree from Seoul National University in 2005, both from the Electrical Engineering Department.

Fan Bai Dr. Fan Bai (M05, SM15, Fellow16) is a Staff Researcher in the Electrical & Control Systems Lab., Research & Development and Planning, General Motors Corporation, since Sep., 2005. Before joining General Motors research lab, he received the B.S. degree in automation engineering from Tsinghua University, Beijing, China, in 1999, and the M.S.E.E. and Ph.D. degrees in electrical engineering, from University of Southern California, Los Angeles, in 2005. His current research is focused on the discovery of fundamental principles and the analysis and design of protocols/systems for next-generation vehicular networks, for safety, telematics and infotainment applications.