MABSTA: Collaborative Computing over Heterogeneous Devices in Dynamic Environments

Yi-Hsuan Kao*, Kwame Wright*, Po-Han Huang*, Bhaskar Krishnamachari* and Fan Bai†
*Ming-Hsieh Department of Electrical Engineering
University of Southern California, Los Angeles, CA, USA
Email: {yihsuank, kwamelaw, pohanh, bkrishna}@usc.edu
†General Motors Global R&D
Warren, MI, USA
Email: fan.bai@gm.com

***Abstract*—Collaborative computing, leveraging resource on multiple wireless-connected devices, enables complex applications that a single device cannot support individually. However, the problem of assigning tasks over devices becomes challenging in the dynamic environments encountered in real-world settings, considering that the resource availability and channel conditions change over time in unpredictable ways due to mobility and other factors. In this paper, we formulate the task assignment problem as an online learning problem using an adversarial multi-armed bandit framework. We propose MABSTA, a novel algorithm that learns the performance of unknown devices and channel qualities continually through exploratory probing and makes task assignment decisions by exploiting the gained knowledge. The implementation of MABSTA, based on Gibbs Sampling approach, is computational-light and offers competitive performance in different scenarios on the trace-data obtained from a wireless IoT testbed. Furthermore, we prove that MABSTA is 1-competitive compared to the best offline assignment for any dynamic environment without stationarity assumptions, and demonstrate the polynomial-time algorithm for the exact implementation of the sampling process. To the best of our knowledge, MABSTA is the first online learning algorithm tailored to this class of problems.**

I. Introduction

We are at the cusp of revolution as the number of connected devices is projected to grow significantly in the near future. These devices, either suffering from stringent battery constraints, or limited processing power, are not capable of running computation-intensive tasks independently. Nevertheless, the connected devices in the network, sharing resources with each other, provide a platform with abundant computational resources that enables the execution of complex applications [1]–[3]. Compared to traditional cloud services, the access to mobile devices, road-side units (RSUs) and other local devices are less reliable [4], [5]. Besides the communication over varying wireless links, the current workload on a device also affects the amount of remaining releasable resource. Hence, a system has to identify the available resources in the network and decide how to leverage them among a number of possibilities, considering the dynamic environment at run time [6], [7].

Figure 1 illustrates the concept of Wireless Collaborative Computing. Given an application that consists of multiple tasks, we want to assign them on multiple devices, considering the resource availability so that the system performance, in

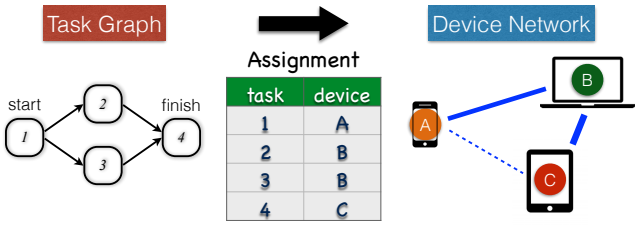

Fig. 1: To leverage the resource on heterogeneous devices in the network, a system has to find out a good task assignment strategy, considering devices' feature, workload and channel qualities between them.

metrics like energy consumption and application latency, can be improved. These resources that are accessible by wireless connections form a resource network, which is subject to frequent topology changes and has the following features:

Dynamic device behavior: The quantity of the released resource varies with devices, and may also depend on the local active processes. Moreover, some of devices may carry microprocessors that are specialized in performing a subset of tasks. Hence, the performance of each device varies highly over time and different tasks and is hard to model as a known and stationary stochastic process.

Heterogeneous network with intermittent connections: Devices' mobility makes the connections intermittent, which change drastically in quality within a short time period. Furthermore, different devices may use different protocols to communicate with each other. Hence, the link performance between devices is also highly dynamic and variable and hard to be modeled as a stationary process.

Since the resource network is subject to drastic changes over time and is hard to be modeled by stationary stochastic processes, we need an algorithm that applies to *all* possible scenarios, learns the environment at run time, and adapts to changes. Existing works focus on solving optimization problems given known deterministic profile or known stochastic distributions [8]–[11]. These problems are hard to solve. More importantly, algorithms without learning ability could be harmed by statistical changes or mismatch between the profile

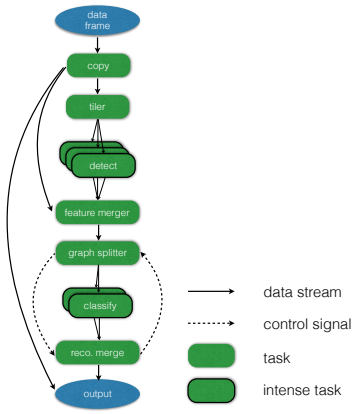

Fig. 2: A face recognition application [12] is partitioned into multiple stages. The task graph without light control signal exchange, is described by a directed acyclic graph.

(offline training) and the run-time environment. Heuristics that adapt different strategies to dynamic environments may suffer from significant performance loss in some scenarios [12]. Hence, we propose an online learning algorithm that is adaptive to the environment at run-time, and provide the performance guarantee to all practical scenarios.

We formulate the task assignment problem as an adversarial multi-armed bandit (MAB) setting that does not make any stationarity assumptions on the resource network. Unlike the Exp3 algorithm proposed in [13], which treats each strategy independently and hence becomes intractable for our problem, we propose MABSTA (Multi-Armed Bandit Systematic Task Assignment), which explores the environment and considers the dependency between the strategies. We illustrate a light implementation based on Gibbs Sampling method to approximate the sampling process, and demonstrate a polynomial-time algorithm for the exact implementation. Moreover, we provide worst-case analysis that MABSTA achieves 1-competitiveness compared with the best offline assignment for *all* dynamic environment. To the best of our knowledge, MABSTA is the first online learning algorithm in this domain of task assignment problems with provable performance guarantee.

II. Problem Formulation

Figure 2 shows the face recognition application proposed in [12], in which the system processes a video stream of multiple data frames in order. The stages like feature recognition and feature classification are intensive for cell phones and other IoT devices, considering limited computation power and battery to run the real-time application individually.

Suppose an application consists of N tasks, where their dependencies are described by a directed acyclic graph (DAG) $G = (\mathcal{V}, \mathcal{E})$. In the task graph, a node m specifies a task (stage), and a directed edge (m, n) implies that data exchange is necessary between task m and task n. Hence task n cannot start until task m finishes. There is an incoming data stream to be processed in order labeled by $1, \cdots, t, \cdots, T$. For each frame t, it is required to go through all the data-processing stages. There are M available devices. The assignment strategy of frame t is denoted by a vector $\mathbf{x}^t = x_1^t, \cdots, x_N^t$, where x_n^t denotes the device that executes task n. Given an assignment strategy, stage-wise costs apply to each node for computation and each edge for communication.

We model the stage-wise costs as sequences that vary over time. When processing frame t, let $C_n^{(i)}(t)$ be the cost of executing task n on device i. Let $C_{mn}^{(ij)}(t)$ be the cost of transmitting the data of between tasks m and n from device i to j. The cost sequences are unknown but are bounded between 0 and 1. That is, we don't make any stochastic or stationarity assumptions on the dynamic environment and hence our formulation fits a broad range of run-time scenarios. Not knowing these sequences in advance, our proposed online algorithm aims to learn the best strategy and remains competitive in overall performance.

In the energy-aware environment, we aim to minimize the total cost of processing the whole data stream. That is,

$$C_{total} = \sum_{t=1}^{T} \left(\sum_{n=1}^{N} C_n^{(x_n^t)}(t) + \sum_{(m,n)\in\mathcal{E}} C_{mn}^{(x_m^t x_n^t)}(t) \right). \quad (1)$$

If the application is delay-sensitive, we aim to minimize the total delay of processing the whole data stream,

$$D_{total} = \sum_{t=1}^{T} D_N(t). \quad (2)$$

The delay at the task n is the accumulated delay from its preceding task,

$$D_n(t) = C_n^{(x_n^t)}(t) + \max_{(m,n)\in\mathcal{E}} D_m(t) + C_{mn}^{(x_m^t x_n^t)}(t), \quad (3)$$

where C_n denotes the computation latency and C_{mn} denotes the communication latency. When the task graph is a serial graph, Equation (2) becomes the additive cost in (1).

III. MABSTA Algorithm

From Section II, our formulation implies exploring exponentially many task assignment strategies (M^N arms). Unlike the Exp3 algorithm in [14] that assumes independent arms, we propose MABSTA (Multi-Armed Bandit Systematic Task Assignment), which probes the devices' performance and channel qualities by flexibly assigning tasks to available devices, learns the best strategy and adjusts it over time. MABSTA leverages the dependencies between arms and learns the environment faster. Although there are M^N feasible assignment strategies, our performance analysis shows that MABSTA still provides the performance guarantee that is bounded by a polynomial of M and N.

In the conventional Multi-armed Bandit (MAB) setting, our assignment strategy corresponds to the arm chosen at time t. The costs can be one-to-one mapped to rewards by setting $reward = maxCost - cost$. When processing data frame t, let $R_n^{(i)}(t)$ be the reward of executing task n on device i. Let $R_{mn}^{(ij)}(t)$ be the reward of transmitting the data between tasks

Algorithm 1 MABSTA

procedure MABSTA(γ, α)
  $w_{\mathbf{x}}(1) \leftarrow 1 \ \forall \mathbf{x} \in \mathcal{F}$
  for $t \leftarrow 1, 2, \cdots, T$ **do**
    $W_t \leftarrow \sum_{\mathbf{x} \in \mathcal{F}} w_{\mathbf{x}}(t)$
    Draw $\mathbf{x}^t$ from distribution

$$p_{\mathbf{x}}(t) = (1-\gamma)\frac{w_{\mathbf{x}}(t)}{W_t} + \frac{\gamma}{|\mathcal{F}|} \tag{4}$$

    Observe $\{R_n^{(x_n^t)}(t)\}_{n=1}^N$, $\{R_{mn}^{(x_m^t x_n^t)}(t)\}_{(m,n)\in\mathcal{E}}$.
    $\mathcal{C}_{ex}^n \leftarrow \{\mathbf{y} \in \mathcal{F} | y_n = x_n^t\}, \ \forall i$
    $\mathcal{C}_{tx}^{mn} \leftarrow \{\mathbf{y} \in \mathcal{F} | y_m = x_m^t, y_n = x_n^t\}, \ \forall (m,n)$
    for $\forall i \in [M], \forall n \in [N]$ **do**

$$\hat{R}_n^{(i)}(t) = \begin{cases} \frac{R_n^{(i)}(t)}{\sum_{\mathbf{y}\in\mathcal{C}_{ex}^n} p_{\mathbf{y}}(t)} & \text{if } x_n^t = i, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

    for $\forall i, j \in [M] \prod [M], \forall (m,n) \in \mathcal{E}$ **do**

$$\hat{R}_{mn}^{(ij)}(t) = \begin{cases} \frac{R_{mn}^{(ij)}(t)}{\sum_{\mathbf{y}\in\mathcal{C}_{tx}^{mn}} p_{\mathbf{y}}(t)} & \text{if } x_m^t = i, x_n^t = j, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

    Update for all $\mathbf{x}$

$$\hat{R}_{\mathbf{x}}(t) = \sum_{n=1}^{N} \hat{R}_n^{(x_n)}(t) + \sum_{(m,n)\in\mathcal{E}} \hat{R}_{mn}^{(x_m x_n)}(t), \tag{7}$$

$$w_{\mathbf{x}}(t+1) = w_{\mathbf{x}}(t) \exp\left(\alpha \hat{R}_{\mathbf{x}}(t)\right). \tag{8}$$

end procedure

(m,n) from device i to j. We assume that only the probed rewards are observable and the rest remain unknown.

MABSTA (Algorithm 1) is a randomized algorithm. For each frame t, MABSTA selects an assignment strategy (arm $\mathbf{x}^t \in \mathcal{F}$) from a probability distribution that depends on the weights of arms ($w_{\mathbf{x}}(t)$). Then it updates the weights based on the observed performance. From (4), MABSTA randomly switches between two phases: exploitation phase (with probability $1-\gamma$) and exploration phase (with probability γ). At exploitation phase, MABSTA selects an arm based on its weight. Hence, the one with higher reward observed will be chosen more likely. At exploration phase, MABSTA uniformly selects an arm without considering its performance. The fact that MABSTA keeps probing every arm makes it adaptive to the changes of the environment, compared to the case where static strategy plays the previously best arm all the time without knowing that other arms might be performing better currently.

After selecting an assignment strategy, MABSTA collects the stage-wise rewards and estimates the performance on these stages based on the observations as shown in (5) and (6). For example, if MABSTA assigns task n to device i, then it estimates how good device i performs on running task n for future reference. In order to get an unbiased estimator,

Algorithm 2 Gibbs Sampling for MABSTA

procedure GIBBS SAMPLING(R)
  $\mathbf{x}^0 \sim Uniform\{1, \cdots, |\mathcal{F}|\}$
  for $r \leftarrow 1, 2, \cdots, R$ **do**
    for $n \leftarrow 1, 2, \cdots, N$ **do**
      Sampling x_n^r from the distribution

$$p(x | x_1^r, \cdots, x_{n-1}^r, x_{n+1}^{r-1}, \cdots, x_N^{r-1}) \tag{9}$$

  Return $\mathbf{x}^R$
end procedure

MABSTA updates the estimate $\hat{R}_n^{(i)}(t)$ by (5) such that

$$\mathbb{E}\{\hat{R}_n^{(i)}(t)\} = R_n^{(i)}(t).$$

In the end, with the estimated stage-wise performance, MABSTA calculates the estimated performance of each strategy, $\hat{R}_{\mathbf{x}}(t)$ in (7) and updates the scores in (8). The parameter α is a positive number, that is to say, higher estimated performance implies higher score and higher chance to be selected for the next frame.

With the aim to be light and competitive in dynamic environment, we design a light implementation for MABSTA, which is an approximation to the exact implementation as Algorithm 1. In Section IV, we evaluate MABSTA's competitiveness by trace-data simulation. Finally, in Section V, we present a theoretical analysis to MABSTA's performance guarantee and a polynomial-time algorithm for the exact implementation for a subset of task graphs and cost function.

A. Implementation

We identify several bottlenecks when implementing Algorithm 1. First, in (4), sampling an assignment strategy takes exponential complexity. Second, both (5) and (6) involve summing exponentially many probability densities. Furthermore, in (8), it requires updating the scores for exponentially many strategies. In the following, we propose efficient implementation on each bottleneck and present empirical evaluation.

1) Sampling an Assignment Strategy: We observe in (4) that for $(1-\gamma)$ proportion of time, we face the sampling process considering all the scores $w_{\mathbf{x}}(t)$ (M^N strategies in the worst case). Algorithm 2 describes the Gibbs sampling [15] method that takes R iterations for each data frame t. We use $\mathbf{x}^r$ to denote the sample vector at iteration r, with its element x_n^r specifying the assignment for task n. Gibbs Sampling chooses the assignment of task n from the conditional probability. Instead of looking at all $w_{\mathbf{x}}(t)$, getting the conditional probability involves only checking M scores, that is,

$$\begin{aligned} &p(x | x_1^r, \cdots, x_{n-1}^r, x_{n+1}^{r-1}, \cdots, x_N^{r-1}) \\ &\quad \propto p(x_1^r, \cdots, x_{n-1}^r, x, x_{n+1}^{r-1}, \cdots, x_N^{r-1}). \end{aligned} \tag{10}$$

Hence, choosing x_n^r from the conditional distributions is equivalent to checking $w_{\mathbf{x}}(t)$ for $x_n^r = 1, \cdots, M$ and the rest of components are fixed, with the new values sampled at the

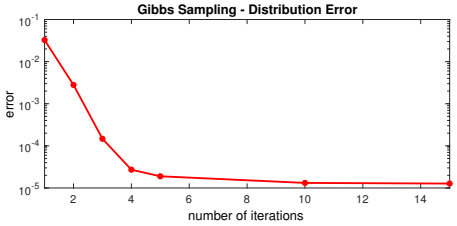

Fig. 3: The difference between the distribution given by Algorithm 2 and the real distribution decreases drastically as the number of iteration grows.

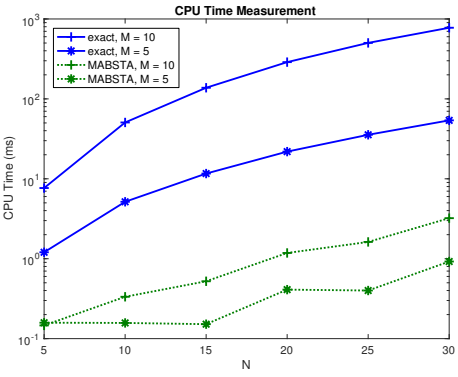

Fig. 4: CPU Time measurement (log-scale) shows that MABSTA-light achieves two orders of magnitude reduction.

current iteration up to task $n-1$ and values from the last iteration for tasks $n+1, \cdots, N$.

Figure 3 shows the difference between $\hat{p}_{\mathbf{x}}(t)$, given by Algorithm 2, and the original distribution $p_{\mathbf{x}}(t)$. The Euclidean distance $\|\hat{p}_{\mathbf{x}}(t) - p_x(t)\|$ decreases drastically as the number of iteration grows and becomes negligible when $R = 10$.

2) Summing Probability Densities: Equation (5) involves calculating the marginal probability $\mathbb{P}\{x_n^t = i\}$. To avoid this computational overhead, we use the conditional probability as an approximate to the marginal probability. That is,

$$\mathbb{P}\{x_n^t = i\} \approx \mathbb{P}\{x_n^t = i | x_1^t, \cdots, x_{n-1}^t, x_{n+1}^t, \cdots, x_N^t\},$$

where $x_1^t, \cdots, x_N^t$ are the devices chosen in the Gibbs sampling process. We will use the condensed notation $p(i|x_m^t : m \neq n)$ for the condition probability in the following discussion. By (10), this conditional probability is easy to compute. However, the approximation assumes that the assignment on task n is independent of other tasks. That is,

$$\mathbb{P}\{x_n^t = i\} = p(i|x_m^t : m \neq n) \text{ if } p(x_1^t, \cdots, x_N^t) = \mathbb{P}\{x_n^t = i\} p(x_1^t, \cdots, x_{n-1}^t, x_{n+1}^t, \cdots, x_N^t).$$

Although this is not true in general, in Section IV, we show that this approximation works well in real environments. Similarly, MABSTA approximates $\mathbb{P}\{x_m^t = i, x_n^t = j\}$ by the conditional distribution, and updates the estimates by,

$$\hat{R}_n^{(i)}(t) = \frac{R_n^{(i)}(t)}{p(i|x_m^t : m \neq n)}, \text{ if } x_n^t = i,$$

$$\hat{R}_{mn}^{(ij)}(t) = \frac{R_{mn}^{(ij)}(t)}{p(i,j|x_u^t : u \neq m,n)}, \text{ if } x_m^t = i, x_n^t = j.$$

3) Updating the Scores: In Algorithm 1, we present the update process in (8) for interpretability. In practice, it is not necessary to update $w_{\mathbf{x}}(t)$ actively. We observe that in (7), $\hat{R}_{\mathbf{x}}(t)$ relies on the estimates of each node and each edge. Hence, we rewrite (8) as

$$w_{\mathbf{x}}(t+1) = \exp\left(\alpha \sum_{n=1}^{N} \tilde{R}_n^{(x_n)}(t) + \alpha \sum_{(m,n)\in\mathcal{E}} \tilde{R}_{mn}^{(x_m x_n)}(t)\right), \tag{11}$$

where

$$\tilde{R}_n^{(x_n)}(t) = \sum_{\tau=1}^{t} \hat{R}_x^{(x_n)}(\tau), \quad \tilde{R}_{mn}^{(x_m x_n)}(t) = \sum_{\tau=1}^{t} \hat{R}_{mn}^{(x_m x_n)}(\tau). \tag{12}$$

To calculate $w_{\mathbf{x}}(t)$, it suffices to update $\tilde{R}_n^{(i)}(t)$ and $\tilde{R}_{mn}^{(i,j)}(t)$ for all $n \in [N]$, $(m,n) \in \mathcal{E}$ and $i,j \in [M]$, which cost $(NM + |\mathcal{E}|\, M^2)$. Hence, if we cache these values, it is not necessary to update $w_{\mathbf{x}}(t)$ for each $\mathbf{x}$. Instead, we can calculate $w_{\mathbf{x}}(t)$ efficiently when needed in the Gibbs Sampling process.

B. Computation Overhead Benchmark

The MABSTA system receives probing data and updates the assignment at run time. Figure 4 shows the MABSTA's average CPU overhead per data frame. We use Apple Macbook Pro equipped with 2.4GHz dual-core Intel Core i5 processor and 3MB cache as our testbed and use java management package for CPU time measurement. For each problem size, we measure the CPU time over 100 data frames and calculate the average. In this experiment, MABSTA runs 10 iterations in the Gibbs sampling process. We compare MABSTA with the exact polynomial-time implementation presented in Section V-B. The measurement result shows that the complexity of MABSTA is light. It only induces 3 ms CPU overhead for the largest problem size we have considered, indicating two orders of magnitude reduction in computational overhead.

IV. Numerical Evaluation

We examine how MABSTA adapts to dynamic environment, and perform trace-data simulation to verify MABSTA's competitive performance to the offline optimal strategy. Finally, we show that MABSTA with the Gibbs Sampling method matches the performance of the exact implementation very well.

A. MABSTA's Adaptivity

We examine MABSTA's adaptivity to dynamic environment and compare it to the optimal strategy that relies on the existing profile. We use a two-device setup, where the task execution costs of the two devices are characterized by two different Markov processes. We neglect the channel communication cost so that the optimal strategy under stationary

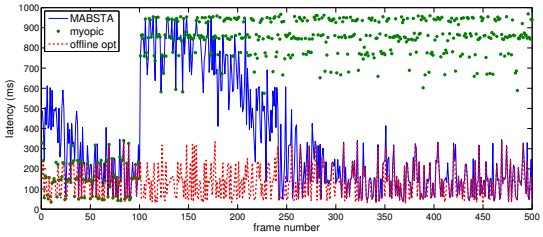

Fig. 5: MABSTA adapts to the changes at the 100^{th} frame, while the myopic policy fails to adjust its assignment strategy.

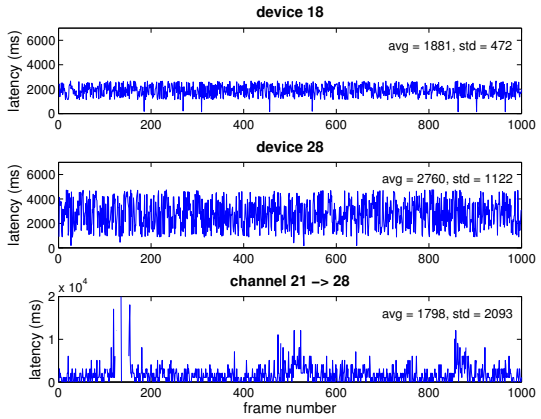

Fig. 6: Snapshots of measurement result: (a) device 18's computation latency (b) device 28's computation latency (c) transmission latency between them.

TABLE I: Parameters Used in Trace-data measurement

Device ID	# of iterations	Device ID	# of iterations
18	$\mathcal{U}(14031, 32989)$	28	$\mathcal{U}(10839, 58526)$
21	$\mathcal{U}(37259, 54186)$	31	$\mathcal{U}(10868, 28770)$
22	$\mathcal{U}(23669, 65500)$	36	$\mathcal{U}(41467, 64191)$
24	$\mathcal{U}(61773, 65500)$	38	$\mathcal{U}(12386, 27992)$
26	$\mathcal{U}(19475, 44902)$	41	$\mathcal{U}(15447, 32423)$

environment is the myopic strategy. That is, assigning the tasks to the device with the highest belief that it is in "good" state [16]. We run our experiment with an application that consists of 10 tasks and processes 500 frames in serial. The environment changes at the 100^{th} frame, where the transition matrices of two Markov processes swap with each other. From Figure 5, there exists an optimal assignment so that the performance remains as good as it was at the first 100 frames. The myopic strategy fails to adapt to the changes. From (4), MABSTA not only considers the previous observation but also keeps exploring uniformly (with probability $\frac{\gamma}{M^N}$ for each arm). Hence, when the performance of one device degrades at 100^{th} frame, this feature enables MABSTA to explore another device and adapt to the best strategy.

B. Trace-data Emulation

To obtain trace data representative of a realistic environment, we run simulations on a large-scale wireless sensor network / IoT testbed [17]. We create a network using 10 IEEE 802.15.4-based wireless embedded devices, and conduct a set of experiments to measure two performance characteristics utilized by MABSTA, namely channel conditions and computational resource availability. To assess the channel conditions, the time it takes to transfer 500 bytes of data between every pair of motes is measured. To assess the resource availability of each device, we measure the amount of time it takes to run a simulated task for a uniformly distributed number of iterations. The parameters of the distribution are shown in Table I. We use these samples as the cost sequences in the following emulation.

We present our evaluation as the regret compared to the offline optimal solution in (14). For real applications, the regret can be extra energy consumption over all nodes, or extra processing latency over all data frames. Figure 7 shows that, as time increases, the performance loss will grow slower and will finally flat out, which implies MABSTA matches the optimal performance. The ratio to the optimal performance, in the aspect of accumulated reward, also approaches to 1. The randomized baseline selects an arm with equal probability for each data frame, hence, the regret grows linear with T.

In addition to original MABSTA, we propose a more aggressive scheme by tuning the parameter γ provided in Algorithm 1. That is, for each frame t, setting

$$\gamma_t = \min\left\{1, \sqrt{\frac{M(N+|\mathcal{E}|\,M)\ln M^N}{(e-1)(N+|\mathcal{E}|)t}}\right\}. \qquad (13)$$

From (4), the larger the γ, the more likely that MABSTA will do exploration. Hence, by exploring more aggressively at the beginning and exploiting the best arm as γ decreases with t, MABSTA with varying γ learns the environment even faster and remains competitive with the offline optimal solution, where the ratio reaches 0.9 at early stage.

We compare MABSTA with the exact implementation for larger network size in Figure 8. We also present the performance for running different numbers of iterations in the Gibbs sampling process. The more iterations it takes, the more accurately it approximates the original distribution. From our observation, when $R = 10$, MABSTA performs as well as the exact implementation.

V. Performance Analysis

In this section, we analysis MABSTA's performance guarantee and design a polynomial time algorithm to the exact implementation. We follow the tradition in MAB literature and focus on maximizing a positive reward instead of minimizing the total cost. When processing data frame t, let $R_n^{(i)}(t)$ be the reward of executing task n on device i. Let $R_{mn}^{(ij)}(t)$ be the reward of transmitting the data of edge (m,n) from device i to j. The reward sequences are unknown but are bounded between 0 and 1. Our goal is to compare MABSTA's performance to a genie who always uses *the best but fixed* assignment strategy for all data frames as follows,

$$R_{total}^{max} = \max_{\mathbf{x}\in\mathcal{F}} \sum_{t=1}^{T}\left(\sum_{n=1}^{N} R_n^{(x_n)}(t) + \sum_{(m,n)\in\mathcal{E}} R_{mn}^{(x_m x_n)}(t)\right), \qquad (14)$$

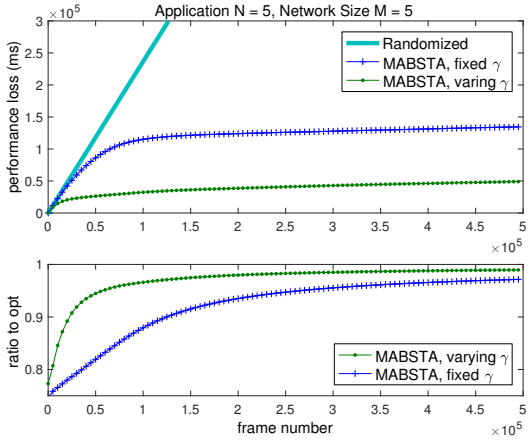

Fig. 7: MABSTA compared with other algorithms for 5-device network.

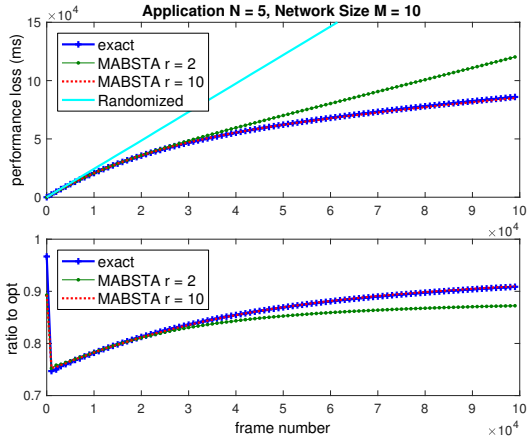

Fig. 8: MABSTA compared with other algorithms for 10-device network.

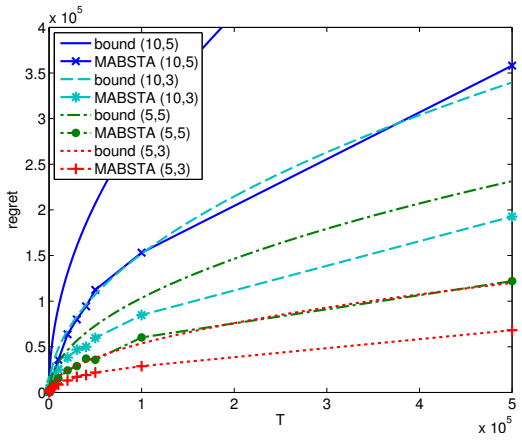

Fig. 9: MABSTA's performance with upper bounds provided by Corollary 1

where $\mathcal{F}$ represents the set of feasible solutions. The regret compared to always playing the best arm is called weak regret. The best offline policy relaxes the constraint so that the genie can switch between arms. Here, we focus on the weak regret and additive rewards. We leave the strong regret and more general metrics like non-linear latency in (2) for future works.

Theorem 1. *Assume all the reward sequences are bounded between* 0 *and* 1. *Let* $\hat{R}_{total}$ *be the total reward achieved by Algorithm 1. For any* $\gamma \in (0,1)$, *let* $\alpha = \frac{\gamma}{M(N+|\mathcal{E}|M)}$, *we have*

$$R^{max}_{total} - \mathbb{E}\{\hat{R}_{total}\} \leq (e-1)\gamma R^{max}_{total} + \frac{M(N+|\mathcal{E}|\,M)\ln M^N}{\gamma}.$$

In Theorem 1, N is the number of nodes (tasks) and $|\mathcal{E}|$ is the number of edges in the task graph. By applying the appropriate value of γ and using the upper bound $R^{max}_{total} \leq (N+|\mathcal{E}|)T$, we have the following Corollary.

Corollary 1. *Let* $\gamma = \min\{1, \sqrt{\frac{M(N+|\mathcal{E}|M)\ln M^N}{(e-1)(N+|\mathcal{E}|)T}}\}$, *then*

$$R^{max}_{total} - \mathbb{E}\{\hat{R}_{total}\} \leq 2.63\sqrt{(N+|\mathcal{E}|)(N+|\mathcal{E}|\,M)MNT\ln M}.$$

If we look at the worst case, where $|\mathcal{E}| = O(N^2)$. The regret can be bounded by $O(N^{2.5}MT^{0.5})$. From our trace-data simulation, Figure 9 validates MABSTA's performance guarantee for different problem sizes. For the cases we have considered, MABSTA's regret scales with $O(N^{1.5}MT^{0.5})$.

A. *Proof of Theorem 1*

We generalize the proof from Exp3 [14], where Lemma 1 and 2 are the direct result. However, Lemma 3 and 4 are non-trivial. We will use more condensed notations like $\hat{R}_n^{(x_n)}$ for $\hat{R}_n^{(x_n)}(t)$, $\hat{R}_{mn}^{(x_m x_n)}$ for $\hat{R}_{mn}^{(x_m x_n)}(t)$ and $p_{\mathbf{x}}$ for $p_{\mathbf{x}}(t)$ in the prove where the result holds for all t.

Lemma 1.

$$\sum_{\mathbf{x}\in\mathcal{F}} p_{\mathbf{x}}(t)\hat{R}_{\mathbf{x}}(t) = \sum_{n=1}^{N} R_n^{(x_n^t)}(t) + \sum_{(m,n)\in\mathcal{E}} R_{mn}^{(x_m^t x_n^t)}(t).$$

Lemma 2. *For all* $\mathbf{x}\in\mathcal{F}$, *we have*

$$\mathbb{E}\{\hat{R}_{\mathbf{x}}(t)\} = \sum_{n=1}^{N} R_n^{(x_n)}(t) + \sum_{(m,n)\in\mathcal{E}} R_{mn}^{(x_m x_n)}(t).$$

Lemma 3. *For* $M \geq 3$ *and* $|\mathcal{E}| \geq 3$,

$$\sum_{\mathbf{x}\in\mathcal{F}} p_{\mathbf{x}}(t)\hat{R}_{\mathbf{x}}(t)^2 \leq \frac{|\mathcal{E}|}{M^{N-2}} \sum_{\mathbf{x}\in\mathcal{F}} \hat{R}_{\mathbf{x}}(t).$$

Proof. We first expand the left-hand-side of the inequality as shown in (15). Then we derive the upper bound for each term in (15) for all $n \in [N]$, $(m,n) \in \mathcal{E}$.

$$\sum_{\mathbf{x}} p_{\mathbf{x}} \hat{R}_m^{(x_m)} \hat{R}_n^{(x_n)} = \sum_{\mathbf{x}\in\mathcal{C}_{ex}^m \cap \mathcal{C}_{ex}^n} p_{\mathbf{x}} \frac{R_m^{(x_m^t)} R_n^{(x_n^t)}}{\sum_{\mathbf{y}\in\mathcal{C}_{ex}^m} p_{\mathbf{y}} \cdot \sum_{\mathbf{y}\in\mathcal{C}_{ex}^n} p_{\mathbf{y}}}$$

$$\leq R_n^{(x_n^t)} \frac{R_m^{(x_m^t)}}{\sum_{\mathbf{y}\in\mathcal{C}_{ex}^m} p_{\mathbf{y}}} = R_n^{(x_n^t)} \hat{R}_m^{(x_m^t)} \leq \frac{1}{M^{N-1}} \sum_{\mathbf{x}} \hat{R}_m^{(x_m)} \tag{16}$$

$$\sum_{\mathbf{x}\in\mathcal{F}} p_{\mathbf{x}}(t)\hat{R}_{\mathbf{x}}(t)^2 = \sum_{\mathbf{x}\in\mathcal{F}} p_{\mathbf{x}} \left(\sum_{m,n} \hat{R}_m^{(x_m)} \hat{R}_n^{(x_n)} + \sum_{(m,n),(u,v)} \hat{R}_{mn}^{(x_m x_n)} \hat{R}_{uv}^{(x_u x_v)} + 2 \sum_m \sum_{(u,v)} \hat{R}_m^{(x_m)} \hat{R}_{uv}^{(x_u x_v)} \right) \tag{15}$$

The first inequality in (16) follows by $\mathcal{C}_{ex}^m \cap \mathcal{C}_{ex}^n$ is a subset of $\mathcal{C}_{ex}^n$ and the last inequality follows by $\hat{R}_m^{(x_m)} = \hat{R}_m^{(x_m^t)}$ for all $\mathbf{x}$ in $\mathcal{C}_{ex}^m$. Hence,

$$\sum_{m,n}\sum_{\mathbf{x}} p_{\mathbf{x}} \hat{R}_m^{(x_m)} \hat{R}_n^{(x_n)} \leq \frac{1}{M^{N-2}} \sum_{\mathbf{x}} \sum_m \hat{R}_m^{(x_m)}. \tag{17}$$

Similarly,

$$\sum_{(m,n),(u,v)} \sum_{\mathbf{x}} p_{\mathbf{x}} \hat{R}_{mn}^{(x_m x_n)} \hat{R}_{uv}^{(x_u x_v)} \leq \frac{|\mathcal{E}|}{M^{N-2}} \sum_{\mathbf{x}} \sum_{(m,n)} \hat{R}_{mn}^{(x_m x_n)}. \tag{18}$$

For the last term in (15), following the similar argument gives

$$\begin{aligned}\sum_{\mathbf{x}} p_{\mathbf{x}} \hat{R}_m^{(x_m)} \hat{R}_{uv}^{(x_u x_v)} &= \sum_{\mathbf{x}\in\mathcal{C}_{ex}^m\cap\mathcal{C}_{tx}^{uv}} p_{\mathbf{x}} \frac{R_m^{(x_m^t)} R_{uv}^{(x_u^t x_v^t)}}{\sum_{\mathbf{y}\in\mathcal{C}_{ex}^m} p_{\mathbf{y}} \cdot \sum_{\mathbf{y}\in\mathcal{C}_{tx}^{uv}} p_{\mathbf{y}}} \\ &\leq R_{uv}^{(x_u^t x_v^t)} \frac{R_m^{(x_m^t)}}{\sum_{\mathbf{y}\in\mathcal{C}_{ex}^m} p_{\mathbf{y}}} = R_{uv}^{(x_u^t x_v^t)} \hat{R}_m^{(x_m^t)} \leq \frac{1}{M^{N-1}} \sum_{\mathbf{x}} \hat{R}_m^{(x_m)}.\end{aligned}$$

Hence,

$$\sum_m \sum_{(u,v)} \sum_{\mathbf{x}} p_{\mathbf{x}} \hat{R}_m^{(x_m)} \hat{R}_{uv}^{(x_u x_v)} \leq \frac{|\mathcal{E}|}{M^{N-1}} \sum_{\mathbf{x}} \sum_m \hat{R}_m^{(x_m)}. \tag{19}$$

Applying (17), (18) and (19) to (15) gives

$$\begin{aligned}&\sum_{\mathbf{x}\in\mathcal{F}} p_{\mathbf{x}}(t) \hat{R}_{\mathbf{x}}(t)^2 \\ &\leq \sum_{\mathbf{x}\in\mathcal{F}} [\sum_m (\frac{1}{M^{N-2}} + \frac{2\,|\mathcal{E}|}{M^{N-1}}) \hat{R}_m^{(x_m)} + \sum_{(m,n)} \frac{|\mathcal{E}|}{M^{N-2}} \hat{R}_{mn}^{(x_m x_n)}] \\ &\leq \frac{|\mathcal{E}|}{M^{N-2}} \sum_{\mathbf{x}\in\mathcal{F}} \hat{R}_{\mathbf{x}}(t).\end{aligned} \tag{20}$$

The last inequality follows by the fact that $\frac{1}{M^{N-2}} + \frac{2|\mathcal{E}|}{M^{N-1}} \leq \frac{|\mathcal{E}|}{M^{N-2}}$ for $M \geq 3$ and $|\mathcal{E}| \geq 3$. Since this is the interesting regime, we will use this result in our proof of Theorem 1. □

Lemma 4. *Let $\alpha = \frac{\gamma}{M(N+|\mathcal{E}|M)}$, then for all $\mathbf{x} \in \mathcal{F}$, all $t = 1, \cdots, T$, we have $\alpha \hat{R}_{\mathbf{x}}(t) \leq 1$.*

Proof. Since $|\mathcal{C}_{ex}^m| \geq M^{N-1}$ and $|\mathcal{C}_{tx}^{mn}| \geq M^{N-2}$,

$$\hat{R}_n^{(x_n)} \leq \frac{R_n^{(x_n)}}{\sum_{\mathbf{y}\in\mathcal{C}_{ex}^n} p_{\mathbf{y}}} \leq \frac{1}{M^{N-1}\frac{\gamma}{M^N}} = \frac{M}{\gamma}, \tag{21}$$

$$\hat{R}_{mn}^{(x_m x_n)} \leq \frac{R_{mn}^{(x_m x_n)}}{\sum_{\mathbf{y}\in\mathcal{C}_{tx}^{mn}} p_{\mathbf{y}}} \leq \frac{1}{M^{N-2}\frac{\gamma}{M^N}} = \frac{M^2}{\gamma}. \tag{22}$$

Hence, we have

$$\hat{R}_{\mathbf{x}}(t) \leq N\frac{M}{\gamma} + |\mathcal{E}|\,\frac{M^2}{\gamma} = \frac{M}{\gamma}(N + |\mathcal{E}|\,M). \tag{23}$$

Let $\alpha = \frac{\gamma}{M(N+|\mathcal{E}|M)}$, we achieve the result. □

By using the above lemmas, we prove Theorem 1 as follows.

Proof. Let $W_t = \sum_{\mathbf{x}\in\mathcal{F}} w_{\mathbf{x}}(t)$. We denote the sequence of arms drawn at each frame as $\mathbf{x} = [\mathbf{x}^1, \cdots, \mathbf{x}^T]$. Then for all data frame t,

$$\begin{aligned}\frac{W_{t+1}}{W_t} &= \sum_{\mathbf{x}\in\mathcal{F}} \frac{p_{\mathbf{x}}(t) - \frac{\gamma}{|\mathcal{F}|}}{1-\gamma} \exp\left(\alpha \hat{R}_{(\mathbf{x})}(t)\right) \\ &\leq \sum_{\mathbf{x}\in\mathcal{F}} \frac{p_{\mathbf{x}}(t) - \frac{\gamma}{|\mathcal{F}|}}{1-\gamma} \left(1 + \alpha \hat{R}_{(\mathbf{x})}(t) + (e-2)\alpha^2 \hat{R}_{(\mathbf{x})}(t)^2\right) \qquad (24) \\ &\leq 1 + \frac{\alpha}{1-\gamma} \left(\sum_{n=1}^N R_n^{(x_n^t)}(t) + \sum_{(m,n)\in\mathcal{E}} R_{mn}^{(x_m^t x_n^t)}(t) \right) \\ &\quad + \frac{(e-2)\alpha^2}{1-\gamma} \frac{|\mathcal{E}|}{M^{N-2}} \sum_{\mathbf{x}\in\mathcal{F}} \hat{R}_{\mathbf{x}}(t). \qquad (25)\end{aligned}$$

Equation (24) follows by the fact that $e^x \leq 1 + x + (e-2)x^2$ for $x \leq 1$. Applying Lemma 1 and Lemma 3 we arrive at (25). Using $1 + x \leq e^x$ and taking logarithms at both sides, then summing from $t = 1$ to T gives

$$\ln \frac{W_{T+1}}{W_1} \leq \frac{\alpha}{1-\gamma} \hat{R}_{total} + \frac{(e-2)\alpha^2}{1-\gamma} \frac{|\mathcal{E}|}{M^{N-2}} \sum_{t=1}^T \sum_{\mathbf{x}\in\mathcal{F}} \hat{R}_{\mathbf{x}}(t). \tag{26}$$

On the other hand,

$$\ln \frac{W_{T+1}}{W_1} \geq \ln \frac{w_{\mathbf{x}}(T+1)}{W_1} = \alpha \sum_{t=1}^T \hat{R}_{\mathbf{x}}(t) - \ln M^N,\ \forall \mathbf{x} \in \mathcal{F}. \tag{27}$$

Combining (26) and (27) gives

$$\hat{R}_{total} \geq (1-\gamma) \sum_{t=1}^T \hat{R}_{\mathbf{x}}(t) - (e-2)\alpha \frac{|\mathcal{E}|}{M^{N-2}} \sum_{t=1}^T \sum_{\mathbf{x}\in\mathcal{F}} \hat{R}_{\mathbf{x}}(t) - \frac{\ln M^N}{\alpha}. \tag{28}$$

Equation (28) holds for all $\mathbf{x} \in \mathcal{F}$. Choose $\mathbf{x}^\star$ to be the assignment strategy that maximizes the objective in (14). Now we take expectations on both sides based on $\mathbf{x}^1, \cdots, \mathbf{x}^T$ and use Lemma 2. That is,

$$\sum_{t=1}^T \mathbb{E}\{\hat{R}_{\mathbf{x}^\star}(t)\} = \sum_{t=1}^T [\sum_{n=1}^N R_n^{(x_n^\star)}(t) + \sum_{(m,n)\in\mathcal{E}} R_{mn}^{(x_m^\star x_n^\star)}(t)] = R_{total}^{max}.$$

Hence,

$$\sum_{t=1}^T \sum_{\mathbf{x}\in\mathcal{F}} \mathbb{E}\{\hat{R}_{\mathbf{x}}(t)\} \leq M^N R_{total}^{max}.$$

Applying the result to (28) and let $\alpha = \frac{\gamma}{M(N+|\mathcal{E}|M)}$ gives

$$R_{total}^{max} - \mathbb{E}\{\hat{R}_{total}\} \leq (e-1)\gamma R_{total}^{max} + \frac{M(N + |\mathcal{E}|\,M)\ln M^N}{\gamma}.$$

□

Algorithm 3 Calculate $w_N^{(j)}$ for tree-structured task graph

procedure $\Omega(N, M, G)$
  $q \leftarrow \text{BFS}(G, N)$ ▷ run BFS and store visited nodes
  for $n \leftarrow q$.end, q.start **do** ▷ start from the last element

$$\omega_n^{(i)} \leftarrow \begin{cases} e_n^{(i)} & \text{if } n \text{ is leaf,} \\ e_n^{(i)} \prod_{m \in \mathcal{D}_n} \sum_{y_m \in [M]} e_{mn}^{(y_m i)} \omega_m^{(y_m)} & \text{otherwise.} \end{cases}$$

end procedure

B. Polynomial Time Exact Implementation

In Section III, MABSTA approximates the marginal probability in (5), (6) and the sampling process in (4) to mitigate the computation complexity. In the following, we propose polynomial-time algorithms for the exact implementation when the structure of the task graph is a subset of DAGs.

1) Marginal Probability: Equation (5) and (6) require the knowledge of marginal probabilities $\mathbb{P}\{x_n^t = i\}$ and $\mathbb{P}\{x_m^t = i, x_n^t = j\}$. From (4), without calculating W_t, we have

$$\mathbb{P}\{x_n^t = i\} - \frac{\gamma}{M} : \mathbb{P}\{x_n^t = j\} - \frac{\gamma}{M} = \sum_{\mathbf{y}: y_n = i} w_{\mathbf{y}}(t) : \sum_{\mathbf{y}: y_n = j} w_{\mathbf{y}}(t). \tag{29}$$

To calculate the sum of weights efficiently depends on the structure of the task graph.

We start from tree-structure task graphs and solve the more general graphs by calling the proposed algorithm for trees a polynomial number of times.

We drop time index t in our derivation whenever the result holds for all $t \in \{1, \cdots, T\}$. For example, $\tilde{R}_n^{(i)})$ and $\tilde{R}_{mn}^{(jk)})$ denote the sum of estimated rewards up to t, as described in 12. Furthermore, let $e_n^{(i)} = \exp(\alpha \tilde{R}_n^{(i)})$ and $e_{mn}^{(ij)} = \exp(\alpha \tilde{R}_{mn}^{(ij)})$. Then, from (11), the sum of weights can be written as

$$\sum_{\mathbf{y}} w_{\mathbf{y}}(t) = \sum_{\mathbf{y}} \prod_{n=1}^{N} e_n^{(y_n)} \prod_{(m,n) \in \mathcal{E}} e_{mn}^{(y_m y_n)}.$$

We assume that the task graph is a tree with nodes $1, \cdots, N$, following the topological ordering, where the N^{th} node is the root (final task). We define the sub-problem, $\omega_n^{(i)}$, as the sum of weights of all possible assignment on task n's preceding tasks, given task n is assigned to device i. That is,

$$\omega_n^{(i)} = e_n^{(i)} \sum_{\mathbf{y} \in [M]^{n-1}} \prod_{m=1}^{n-1} e_m^{(y_m)} \prod_{(u,v) \in \mathcal{E}} e_{uv}^{(y_u y_v)}. \tag{30}$$

Hence, the relation of weights between task n and its children $\mathcal{C}_n = m : (m, n) \in \mathcal{E}$ is given by the following equation.

$$\begin{aligned} \omega_n^{(i)} &= e_n^{(i)} \sum_{\{y_m\}_{m \in \mathcal{C}_n}} \prod_{m \in \mathcal{C}_n} e_{mn}^{(y_m i)} \omega_m^{(y_m)} \\ &= e_n^{(i)} \prod_{m \in \mathcal{C}_n} \sum_{y_m \in [M]} e_{mn}^{(y_m i)} \omega_m^{(y_m)}. \end{aligned} \tag{31}$$

Algorithm 4 Efficient Sampling Algorithm

procedure SAMPLING(γ)
  for $n \leftarrow 1, \cdots, N$ **do**
    $[\omega_n^{(i)}]_{x_1^t, \cdots, x_{n-1}^t} \leftarrow \Omega(N, M, G)_{x_1^t, \cdots, x_{n-1}^t}$
    $\mathbb{P}\{x_n^t = i | x_1^t, \cdots, x_{n-1}^t\} \propto [\omega_n^{(i)}]_{x_1^t, \cdots, x_{n-1}^t}$
end procedure

As illustrated in Algorithm 3, we first run breadth first search (BFS) from the root node. Then we start solving the sub-problems following the topological ordering. Let d_{in} denote the maximum in-degree of G. For each sub-problem, there are at most d_{in} products of summations over M terms. In total, Algorithm 3 solves NM sub-problems. Hence, it runs in $O(d_{in} N M^2)$ time.

To generalize, for a task graph that consists of serial trees rooted by $n_1, \cdots, n_R$ in order, let the sub-problem, $\omega_{n_r | n_{r-1}}^{(i_r | i_{r-1})}$, denote the sum of weights given that n_r is assigned to i_r and n_{r-1} is assigned to i_{r-1}. We can solve $\omega_{n_r}^{(i_r)}$, given previously solved $\omega_{n_r | n_{r-1}}^{(i_r | i_{r-1})}$.

$$\omega_{n_r}^{(i_r)} = \sum_{j \in [M]} \omega_{n_r | n_{r-1}}^{(i_r | j)} \omega_{n_{r-1}}^{(j)}$$

To solve $\omega_{n_r}^{(i_r)}$, we have to solve $w_{n_r | n_{r-1}}^{(i_r | i_{r-1})}$ for $i_{r-1} \in \{1, \cdots, M\}$. Hence, it takes $O(d_{in} N_{r-1} M^2) + O(M d_{in} N_r M^2)$ time, where N_r is the number of nodes in tree n_r. In sum, to solve a serial-tree task graph, it takes $O(d_{in} N M^3)$ time. This approach can be generalized to more complicated DAGs, like the one that contains parallel chains of trees, in which we solve each chain independently and combine them from their common root N.

Now we use Algorithm 3 to calculate the marginal probability. From (29), if task n is the root (node N), then Algorithm 3 solves $\omega_N^{(i)} = \sum_{\mathbf{y}: y_N = i} w_{\mathbf{y}}(t)$ exactly. For $n \neq N$, we still run Algorithm 3 to solve $[\omega_p^{(i')}]_{y_n = i}$, by fixing the assignment of task n to device i when solving from n's parent p. That is,

$$[\omega_p^{(i')}]_{y_n = i} = e_p^{(i')} e_{np}^{(i i')} \omega_n^{(i)} \prod_{m \in \mathcal{C}_p \setminus \{n\}} \sum_{y_m} e_{mp}^{(y_m i')} \omega_m^{(y_m)}.$$

In the end, we can solve $[\omega_N^{(i')}]_{y_n = i}$ from the root and

$$\sum_{\mathbf{y}: y_n = i} w_{\mathbf{y}}(t) = \sum_{i' \in [M]} [\omega_N^{(i')}]_{y_n = i}.$$

Similarly, $\mathbb{P}\{x_m^t = i, x_n^t = j\}$ can be achieved by solving the conditional sub-problems on both tasks m and n.

2) Sampling: We propose an efficient sampling policy summarized in Algorithm 4. During the exploitation phase $(1 - \gamma)$, MABSTA selects an arm based on the probability distribution $p_{\mathbf{y}}(t)$, which can be written as

$$\begin{aligned} p_{\mathbf{y}}(t) = \mathbb{P}\{x_1^t = y_1\} \cdot \mathbb{P}\{x_2^t = y_2 | x_1^t = y_1\} \\ \cdots \mathbb{P}\{x_N^t = y_N | x_1^t = y_1, \cdots, x_{N-1}^t = y_{N-1}\}. \end{aligned}$$

Hence, MABSTA assigns each task in order based on the conditional probability given the assignment on previous tasks. For each task n, the conditional probability can be calculate efficiently by running Algorithm 3 with fixed assignment on task $1, \cdots, n-1$.

Assuming the task graph belongs to the family of parallel chains of trees, calculating each marginal probability takes $O(d_{in}NM^3)$ time. Hence, the sampling process takes $O(d_{in}N^2M^3)$ time in total, which dominates the overall complexity of the exact implementation in each time frame.

VI. Related Works

A. Multi-armed Bandit Problems

The multi-armed bandit (MAB) problem is a sequential decision problem where at each time an agent chooses over a set of "arms", gets the payoff from the selected arms and tries to learn the statistical information from sensing them. Given an online algorithm to a MAB problem, its performance is measured by a regret function, which specifies how much the agent loses due to the unknown information at the beginning [18]. For example, we can compare the performance to a genie who knows the statistics of payoff functions in advance and leverages the bast policy.

The MAB formulations have been considered in the context of opportunistic spectrum access for cognitive radio wireless networks [19], [20], which focus on the channel selection. Our formulation, considering both computation cost and channel cost, makes the total payoff become a non-linear function of the action vector. Adding the interaction terms to the existing framework [21] for linear payoff function significantly degrades the performance guarantee on the regret bound. Hence, we propose MABSTA to address both computation allocation and channel selection, and present a provable and competitive performance guarantee.

Adversarial MAB problems do not make any assumptions on the stationarity of payoffs. Instead, an agent learns from the sequence given by an adversary who has complete control over the payoffs [13]. Hence, algorithms of the adversarial MAB problems apply to all bounded payoffs in dynamic environment. Auer *et al.* [14] propose Exp3 that yields a sub-linear regret with time ($O(\sqrt{TK\ln K})$), where T is the length of time and K is the number of independent arms. That is, compared to the optimal offline algorithm (always choose the best but fixed arm), Exp3 achieves asymptotically 1-competitive. However, our task assignment strategies are not independent. Direct applying Exp3 to our problem results in exploring exponentially many arms in the space, hence, the performance is not competitive. Our contribution is to consider the dependencies of the assignment strategies and provide the regret that is bounded by $O(\sqrt{T})$ and a polynomial function of network size and graph size.

B. System Prototypes

We classify the existing systems or prototypes by the types of remote computational resources that a local device has access to. One extreme is the traditional cloud-computing services where a local device sends a request to a cloud that has remote servers set up by a service provider. MAUI [22] and CloneCloud [23] are systems that leverage the resources in the distant cloud. Odessa [12] is a heuristic that identifies the bottleneck stage and suggests offloading strategy and leverages data parallelism to mitigate its load. Another extreme is to exploit the computational resources on nearby devices. Instead of targeting one service provider in the cloud, Shi *et al.* [24] investigate the nearby mobile helpers reached by intermittent connections. Between these two extremes, MapCloud [25] is a hybrid system that makes run-time decision on using "local" cloud with less computational resources but faster connections or using remote "public" cloud with more powerful servers but longer communication delay. There are hybrid systems that make best decision on offloading tasks to nearby helpers or remote cloud servers, like Cloudlets [26] and OSCC [27].

Existing systems either rely on solving optimization problems for optimal strategy based on known profiling data [9], [28], or propose heuristics that adapts different strategies to dynamic environments based on simple metrics, like Odessa [12]. First, solving the optimization problems induces considerable computation overhead. Second, lack of learning ability results in significant performance degradation due to mismatch between profiling data and the variant run-time environment. Hence, to address these issues, we propose MABSTA, an online algorithm that learns the unknown environments and globally adjusts the strategy to the changes.

VII. Conclusion

As increasing number of devices capable of computing and communicating, the concept of Wireless Collaborative Computing enables complex applications which a single device cannot support individually. However, the intermittent and heterogeneous connections and diverse device behavior make the performance highly-variant with time. In this paper, we have proposed a new Multi-armed Bandit formulation for the task assignment problem over heterogeneous network, without any stationarity assumptions on channels and devices' performance. Unlike the existing algorithm, we have presented MABSTA, which considers the dependency between different assignment strategies and makes the decision at run time. To the best of our knowledge, MABSTA is the first online learning algorithm tailored to this class of problems. We have implemented a light approximation of MABSTA using Gibbs Sampling method, and have demonstrated that it is competitive and adaptive to changes of the environment via trace-data simulation. Moreover, for a subset of task graphs and cost metric, we have demonstrated a polynomial-time algorithm for the exact implementation and have proven that MABSTA's performance matches asymptotically to the performance of the best offline assignment strategy. Interesting future work includes deriving the stronger performance guarantee for more general cost metric and more flexible offline strategy.

References

[1] D. Datla, X. Chen, T. Tsou, S. Raghunandan, S. Shajedul Hasan, J. H. Reed, C. B. Dietrich, T. Bose, B. Fette, and J. Kim, "Wireless distributed

computing: a survey of research challenges," *Communications Magazine, IEEE*, vol. 50, no. 1, pp. 144–152, 2012.

[2] M. Y. Arslan, I. Singh, S. Singh, H. V. Madhyastha, K. Sundaresan, and S. V. Krishnamurthy, "Cwc: A distributed computing infrastructure using smartphones," *IEEE Transactions on Mobile Computing*, 2014.

[3] X. Zhang, A. Kunjithapatham, S. Jeong, and S. Gibbs, "Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing," *Mobile Networks and Applications*, vol. 16, no. 3, pp. 270–284, 2011.

[4] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[5] K. Habak, M. Ammar, K. A. Harras, and E. Zegura, "Femto clouds: Leveraging mobile devices to provide cloud service at the edge," in *2015 IEEE 8th international conference on cloud computing*. IEEE, 2015, pp. 9–16.

[6] 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.

[7] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloudcmp: comparing public cloud providers," in *ACM SIGCOMM*. ACM, 2010, pp. 1–14.

[8] X. Chen, S. Hasan, T. Bose, and J. H. Reed, "Cross-layer resource allocation for wireless distributed computing networks," in *RWS, IEEE*. IEEE, 2010, pp. 605–608.

[9] Y.-H. Kao, B. Krishnamachari, M.-R. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," *IEEE Transactions on Mobile Computing*, 2017.

[10] Y.-H. Kao and B. Krishnamachari, "Optimizing mobile computational offloading with delay constraints," in *IEEE GLOBECOM*. IEEE, 2014, pp. 8–12.

[11] Y. Tao, C. You, P. Zhang, and K. Huang, "Stochastic control of computation offloading to a helper with a dynamically loaded cpu," *IEEE Transactions on Wireless Communications*, vol. 18, no. 2, pp. 1247–1262, 2019.

[12] 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.

[13] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "Gambling in a rigged casino: The adversarial multi-armed bandit problem," in *Foundations of Computer Science*. IEEE, 1995, pp. 322–331.

[14] ——, "The nonstochastic multiarmed bandit problem," *SIAM Journal on Computing*, vol. 32, no. 1, pp. 48–77, 2002.

[15] E. I. George and R. E. McCulloch, "Variable selection via gibbs sampling," *Journal of the American Statistical Association*, vol. 88, no. 423, pp. 881–889, 1993.

[16] Y. M. Dirickx and L. P. Jennergren, "On the optimality of myopic policies in sequential decision problems," *Management Science*, vol. 21, no. 5, pp. 550–556, 1975.

[17] anrg, "Tutornet: A low power wireless iot testbed," February 2014, online; accessed 11-January-2020.

[18] S. Bubeck and N. Cesa-Bianchi, "Regret analysis of stochastic and nonstochastic multi-armed bandit problems," *arXiv preprint arXiv:1204.5721*, 2012.

[19] W. Dai, Y. Gai, and B. Krishnamachari, "Online learning for multi-channel opportunistic access over unknown markovian channels," in *IEEE SECON*. IEEE, 2014, pp. 64–71.

[20] K. Liu and Q. Zhao, "Indexability of restless bandit problems and optimality of whittle index for dynamic multichannel access," *IEEE Transactions on Information Theory*, vol. 56, no. 11, pp. 5547–5567, 2010.

[21] J.-Y. Audibert, S. Bubeck, and G. Lugosi, "Regret in online combinatorial optimization," *Mathematics of Operations Research*, vol. 39, no. 1, pp. 31–45, 2013.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] M. Satyanarayanan, "Cloudlets: at the leading edge of cloud-mobile convergence," in *ACM SIGSOFT*. ACM, 2013, pp. 1–2.

[27] M. Chen, Y. Hao, C.-F. Lai, D. Wu, Y. Li, and K. Hwang, "Opportunistic task scheduling over co-located clouds in mobile environment," *IEEE Transactions on Services Computing*, 2017.

[28] Y. Tao, C. You, P. Zhang, and K. Huang, "Stochastic control of computation offloading to a dynamic helper," in *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 2018, pp. 1–6.

ACKNOWLEDGMENT

This material is based upon work supported by Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0053. Any views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.