Program Placement Optimization for Storage-constrained Mobile Edge Computing Systems: A Multi-armed Bandit Approach

Mingjie Feng and Marwan Krunz
Dept. Electrical & Computer Engineering
The University of Arizona, Tucson, AZ 85719, USA.

***Abstract*—Mobile edge computing (MEC) is a promising technology to support computationally intensive mobile applications with stringent delay requirements. As MEC applications become much more diverse and complex, it becomes more challenging for an edge node (EN) with limited storage to keep the program codes of all tasks. In this paper, we investigate the problem of program placement and user association in storage-limited MEC systems. Formulating the problem as a sequential decision-making problem, we first derive the solution for a single EN by transforming the formulation into a multi-armed bandit (MBA) problem and solving it via a Thompson sampling (TS) algorithm. We then propose a solution framework for the multi-EN scenario, where we decompose the original problem into three subproblems and solve them with low-complexity approaches. The first subproblem is to learn the task popularity, which we also formulate as a MAB problem and solve it via a TS algorithm. The second subproblem is optimizing program placement under a given user association and we propose a greedy algorithm to solve it. The last subproblem relates to user association, which is solved by a dual decomposition-based approach. Simulation results show that the average latency achieved by our proposed schemes is 30% to 100% lower than two benchmark schemes and is on average less than 10% higher than a lower bound.**

***Index Terms*—Mobile edge computing; low-latency applications; storage-limited systems; program placement optimization; multi-armed bandit; Thompson sampling.**

I. Introduction

Mobile Internet of Things (IoT) applications (e.g., connected and autonomous vehicles, augmented/virtual reality, etc.) often require executing delay-sensitive yet computationally intensive tasks [1], [2]. With limited computational capability, it is quite challenging for mobile devices to execute these tasks in a timely manner. Mobile edge computing (MEC) provides an effective solution to this challenge. By deploying MEC servers within the radio access network, e.g., close to base stations (BSs) or access points (APs), the computational tasks generated by users can be offloaded to and executed by nearby *edge nodes* (ENs)[1] [3]. Due to their proximity to end users, ENs can deliver low-latency services.

To execute a task at an EN, the input data (e.g., video clips taken by the mobile device) and the program that executes the task (e.g., object recognition software) are required. Many research papers assume that the programs of all tasks are stored at each EN. This way, users can always offload any task to a nearby EN. It is also often assumed that the programs are always loaded in the random-access memory (RAM) of each EN, so that a task can be immediately executed once the input data has been uploaded by the user, without having to wait for program loading. However, these assumptions are not practical for future MEC systems. Specifically, mobile IoT applications will become more diverse, and their complexity is expected to increase (e.g., going from Level 1 autonomous driving to Level 5 autonomous driving), with a corresponding increase in the sizes of programs associated with these applications. On the other hand, with the projected massive deployment of ENs, cheap hardware with relatively small storage will likely be used to avoid high capital expenditure (CAPEX). Thus, it is quite unlikely that any EN will need to store all programs generated by different users. When the program of a requested task is not stored in a given EN, the task has to be executed at the user's own device or handed over to another EN that has the program. Both options increase the overall latency. To this end, storage utilization at an EN needs to be optimized in a way that *minimizes the average latency*.

Task requests received by different ENs vary in time and space, depending on the specific application. This means that ENs must learn to optimize their own *program placement* strategy, which includes program storage and preloading. Intuitively, frequently requested programs should have a higher priority to be stored and preloaded. However, other factors need to be taken into account, such as program file size, computational complexity, and loading time of each program. In an MEC system with multiple users and ENs, user association is a design problem that is coupled with program placement. On the one hand, users in overlapping coverage areas of neighboring ENs have multiple choices of which ENs to offload their tasks. As the ENs vary in program availability, communication link quality, and computational capability, the overall latency of a user is dictated by its EN selection. On the other hand, user association determines the traffic load at each EN and the associated communication and computational latencies. Thus, program placement strategies of neighboring ENs are coupled via user association.

In this paper, we investigate the optimization of program placement and user association in storage-constrained MEC systems, aiming to minimize the average latency of all users over a finite time interval. We develop efficient solutions based

[1]An EN refers to a combination of an MEC server and a BS/AP.

on an online learning framework called multi-armed bandit (MAB). The main contributions of this paper are as follows:

- We formulate the problem of program placement (including program storage and preloading) and user association as a sequential decision-making problem with coupled variables.
- We derive an optimal solution for the single-EN scenario by transforming the formulated problem into an MAB problem, in which a subset of program placement strategies are regarded as the arms to be played. The MAB problem is solved using a Thompson sampling (TS) algorithm.
- We develop a solution framework for the multi-EN scenario by decomposing the original problem into three subproblems that are solved with low-complexity schemes at each time slot. The first subproblem targets learning task popularity (i.e., the probability that each task will be requested by a user). This subproblem is formulated as an MAB problem, and solved by a TS algorithm that only requires ENs to perform a simple parameter update at each round of learning. The second subproblem is optimizing program placement under a given user association. We show that this subproblem is a Knapsack problem, and accordingly, we solve it via a greedy algorithm. The last subproblem is user association, for which we propose a dual decomposition-based approach to derive a near-optimal solution. The proposed user association solution is implemented in a distributed pattern and only requires limited information exchange between users and ENs. Moreover, we show that the number of iterations is only a function of the convergence threshold, which enables low-complexity implementation.
- We evaluate the performance of the proposed schemes using simulations based on a cellular network setup, in which the latency performance under different task profiles (popularity, complexity, and program file size of tasks) and network scenarios (varying numbers of ENs and users) are compared. The results show that the proposed schemes can reduce the average latency by $30\%\sim100\%$ compared to two benchmark schemes. To demonstrate that near-optimal performance can be achieved, we derive a lower bound on the latency. The results show that, on average, the latency achieved by our solution is less than 10% higher than the lower bound.

In the remainder of this paper, we overview related work in Section II. We introduce the system model and problem formulation in Sections III and IV, respectively. The solution algorithms for the single-EN and multi-EN scenarios are presented in Sections V and VI, respectively. We present our simulation results in Section VII and conclude the paper in Section VIII.

II. Related Work

As an enabling technology for IoT applications, MEC has attracted significant attention from both industry and academia. Early standardization efforts were initiated by the Industry Specification Group (ISG) of the European Telecommunications Standards Institute (ETSI) [1]. A literature overview of MEC can be found in [3]. Recently, an analytical framework for the fundamental aspects of MEC, including communication, computation, caching, and control was introduced in [4].

The problem of program placement at ENs has some similarities with content caching/prefetching in content delivery networks (CDNs), where popular content is cached at stations that are close to end users (e.g., small BSs), allowing such content to be quickly delivered to these users [8]–[10]. Because the content popularity profile is often unknown, machine learning (ML) algorithms have been developed to predict the content request pattern and optimize caching strategies (e.g., [11]–[13]). In particular, an MAB-based algorithm was employed in [13] to learn the file request pattern at a small BS, and a greedy file placement scheme was proposed based on the predicted popularity profile. MEC-supported content caching was considered in some recent works [14]–[17], where the computing capability of MEC servers was utilized to improve CDN performance. For example, the MEC server can preprocess certain files (e.g., perform video transcoding) to reduce the processing time of users. It can also help compress files to save storage space. In contrast to these works that optimize content placement for fast content delivery in CDNs, we consider optimizing the placement of *user programs* at ENs, which determines the task offloading availability, aiming to fully harvest the benefit of MEC systems in providing low-latency computing services. In addition, we consider optimizing the preloading strategy with the goal of minimizing the processing latency.

User association in MEC systems has been investigated in recent works. In [19], [20], user association was jointly optimized with computational resource allocation and power control, aiming to minimize the total energy consumption. In [6], joint optimization of EN selection and computing resources was considered to improve the quality of experience (QoE) of users. In our problem, user association is coupled with the program placement strategy of ENs, where we jointly optimize the two to minimize the average latency of all users.

III. System Model

A. Problem Setup

We consider an MEC system with J ENs, indexed by $j \in \{1,\ldots,J\} \triangleq \mathcal{J}$. These ENs provide task offloading services to K mobile user equipments (UEs), indexed by $k \in \{1,\ldots,K\} \triangleq \mathcal{K}$. Each EN has a storage space (e.g., disk) with capacity E_{H} (in bytes) that stores the program codes of different tasks. Each EN also has a RAM of capacity E_{R} (in bytes) that loads the programs from disk, allowing the CPU to read and execute these programs.

We consider a finite time interval that is divided into T slots $t = 1,\ldots,T$. For analytical tractability, we assume that the pattern of task request received by each EN remains

the same during each time interval of T slots[2]. In each time slot t, each UE randomly requests any one of the N possible computational tasks. Each task is executed by a unique program. The tasks and their associated programs are indexed by $i \in \{1, \ldots, N\} \triangleq \mathcal{N}$. We assume that the task generation processes of various UEs are mutually independent and follow the same probability distribution. This assumption is justified by the fact that different UEs act independently of each other. Let θ_i be the probability that task i is generated by any arbitrary UE; $\sum_{i=1}^{N} \theta_i = 1$. The task popularity file over N tasks is denoted by $\boldsymbol{\theta} = [\theta_1, \ldots, \theta_N]$. Before the learning process starts, $\boldsymbol{\theta}$ is unknown to ENs.

At each time slot t, the programs that are stored on the disk of EN j and loaded to the RAM are specified by two sets of binary variables $a_{i,j}^{[t]}$ and $b_{i,j}^{[t]}$, defined by:

$$a_{i,j}^{[t]} = \begin{cases} 1, & \text{if program } i \text{ is stored on disk of EN } j \\ 0, & \text{otherwise} \end{cases}$$
$$i = 1, \ldots, N,\ j \in \mathcal{J},\ t = 1, \ldots, T. \quad (1)$$

$$b_{i,j}^{[t]} = \begin{cases} 1, & \text{if program } i \text{ is preloaded to the RAM of EN } j \\ 0, & \text{otherwise} \end{cases}$$
$$i = 1, \ldots, N,\ j \in \mathcal{J},\ t = 1, \ldots, T. \quad (2)$$

Let s_i be the size of program i (in bytes) and q_i be the storage space occupied by this program (in bytes) when loaded to RAM, $q_i > s_i$. Then, for all $i \in \mathcal{N}$, $a_{i,j}^{[t]}$ and $b_{i,j}^{[t]}$ should satisfy:

$$\begin{cases} \sum_{i=1}^{N} a_{i,j}^{[t]} s_i \leq E_{\mathrm{H}},\ \forall j \in \mathcal{J},\ t = 1, \ldots, T \\ \sum_{i=1}^{N} b_{i,j}^{[t]} q_i \leq E_{\mathrm{R}},\ \forall j \in \mathcal{J},\ t = 1, \ldots, T. \end{cases} \quad (3)$$

To update the program storage, each EN downloads the programs to be added from the core network via backhaul and deletes the programs to be removed. Each EN must also decide the set of UEs associated with it. This is indicated by the following binary variables:

$$x_{k,j}^{[t]} = \begin{cases} 1, & \text{if UE } k \text{ is associated with EN } j \\ 0, & \text{otherwise,} \end{cases}$$
$$k \in \mathcal{K},\ j \in \mathcal{J},\ t = 1, \ldots, T. \quad (4)$$

When UE k is associated with EN j (i.e., $x_{k,j}^{[t]} = 1$) and it generates task i at time t, the task will be executed by EN j only if the program codes of task i are stored at EN j. Otherwise, the task can only be executed by UE k. Fig. 1 illustrates an example of task offloading with different task availability scenarios.

The sequence of program placement and user association strategies of the J ENs for $t = 1, \ldots, T$ is called a policy, which is denoted by $\boldsymbol{\pi} = \{\pi_1, \ldots, \pi_J\}$.

[2]To capture the time-varying task request pattern, the MEC system will update its configuration by performing the proposed optimizations in the next time interval of T time slots.

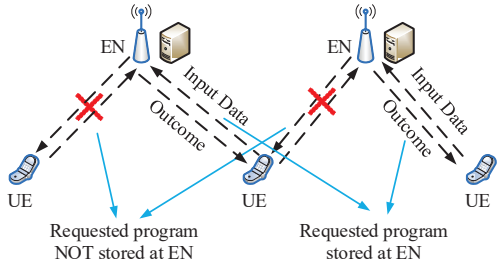

Fig. 1. Example of task offloading in a multi-EN system with limited storage. A UE can offload a computational task to its associated EN only when the task program is stored at the EN.

B. Communication Model

We consider a generic cellular network (e.g., LTE or 5G) in which UEs are served by BS's acting as ENs. UEs served by the same EN are assumed to have the same priority, hence the communication resource is equally allocated to them [18]. At time t, the data rate of UE k when associated with EN j is given by:

$$R_{k,j}^{[t]} = \frac{W \log\left(1 + \gamma_{k,j}^{[t]}\right)}{\sum_{k=1}^{K} x_{k,j}^{[t]}} \quad (5)$$

where W is the available bandwidth for each EN, and $\gamma_{k,j}^{[t]}$ is the uplink signal-to-interference-plus-noise ratio (SINR) for the transmission from UE k to EN j, which can be measured by EN j. The successful of reception UE data requires that $\gamma_{k,j}^{[t]} \geq \gamma_{\mathrm{th}}$, where γ_{th} is the SINR threshold. For simplicity, we do not consider multi-rate communications in this paper. Let $\Omega_k^{[t]}$ denote the set of candidate ENs that can be used by UE k for task offloading at time t, $\Omega_k^{[t]} = \left\{ j \in \mathcal{J} \,\middle|\, \gamma_{k,j}^{[t]} \geq \gamma_{\mathrm{th}} \right\}$. Then, $x_{k,j}^{[t]}$ must satisfy:

$$x_{k,j}^{[t]} = 0, \forall j \notin \Omega_k^{[t]}. \quad (6)$$

C. Computational Models

The computational resource required for executing a task is determined by the size of input data (in bits) and the task's computational complexity, represented in the number of CPU cycles needed to execute one bit of the task. Let $\tilde{s}_{k,i}$ be the input data size for task i of UE k and let z_i be the computational complexity of task i. Then, the number of CPU cycles required to complete task i is $\tilde{s}_{k,i} z_i$.

1) UE Computing Time: Let $c_k^{(\mathrm{L})}$ be the computational capability of UE k, measured in CPU cycles per second. Then, the execution time for task i at UE k is $\frac{\tilde{s}_{k,i} z_i}{c_k^{(\mathrm{L})}}$. Because UE k may generate any one of the N tasks, the expected task execution time (in seconds) at UE k is given by:

$$D_k^{(\mathrm{L})} = \sum_{i=1}^{N} \theta_i \frac{\tilde{s}_{k,i} z_i}{c_k^{(\mathrm{L})}}. \quad (7)$$

2) MEC Server Computing: Let $c_j^{(\mathrm{E})}$ be the computational capability of EN j. We assume that $c_j^{(\mathrm{E})}$ is equally split between all UEs associated with EN j during each time slot t.

Then, the computational capability allocated to UE k by EN j at time t is $c_{k,j}^{(\mathrm{E})^{[t]}} = \frac{c_j^{(\mathrm{E})}}{\sum_{k=1}^{K} x_{k,j}^{[t]}}$. Similar to (7), the expected computing time for executing the task of UE k at EN j is given by:

$$D_{k,j}^{(\mathrm{E})^{[t]}} = \sum_{i=1}^{N} \theta_i \frac{\tilde{s}_{k,i} z_i}{c_{k,j}^{(\mathrm{E})^{[t]}}} = \sum_{i=1}^{N} \theta_i \frac{\tilde{s}_{k,i} z_i \sum_{k=1}^{K} x_{k,j}^{[t]}}{c_j^{(\mathrm{E})}}. \quad (8)$$

D. Latency Analysis

Suppose UE k is associated with EN j at time t. Considering that UE k generates task i with probability θ_i and the generated task may be executed by EN j or by UE k depending on the program availability at EN j, the expected latency of UE k at time t is given by:

$$D_{k,j}^{[t]} = \sum_{i=1}^{N} \theta_i \left[\left(\frac{\tilde{s}_{k,i} z_i}{c_{k,j}^{(\mathrm{E})^{[t]}}} + \frac{\tilde{s}_{k,i}}{R_{k,j}^{[t]}} \right) a_{i,j}^{[t]} + \left(1 - a_{i,j}^{[t]}\right) \frac{\tilde{s}_{k,i} z_i}{c_k^{(\mathrm{L})}} + \left(1 - b_{i,j}^{[t]}\right) l_{i,j} \right] \quad (9)$$

where $l_{i,j}$ is the loading time of program i at EN j. For simplicity, the latency for downloading the result of an executed task is ignored due to its small size [6], [7]. In case the downloading time is non-negligible, it can be calculated in the same way as the uploading time, and the expression of $D_{k,j}^{[t]}$ can be modified accordingly.

The latency expression in (9) can only be applied to the case when UE k is associated with one of the J ENs. If UE k is not associated with any EN, i.e., $\sum_{j=1}^{J} x_{k,j} = 0$, the UE has to execute the task by itself, with expected latency of $D_k^{(\mathrm{L})}$. Combing the two cases, the latency of UE k is given by:

$$D_k^{[t]} = \sum_{j=1}^{J} x_{k,j}^{[t]} D_{k,j}^{[t]} + \left(1 - \sum_{j=1}^{J} x_{k,j}^{[t]}\right) D_k^{(\mathrm{L})}. \quad (10)$$

IV. Problem Formulation

In this paper, we aim to find the optimal policy $\boldsymbol{\pi}$ (i.e., the sequence of program placement and user association strategies) that minimizes the accumulated average latency of all UEs over time slots $t = 1, \ldots, T$. With $D_k^{[t]}$ given in (10), the sum latency of all UEs at time t is given by:

$$\sum_{k=1}^{K} D_k^{[t]} = \sum_{k=1}^{K} D_k^{(\mathrm{L})} + \sum_{k=1}^{K} \sum_{j=1}^{J} x_{k,j}^{[t]} \left(D_{k,j}^{[t]} - D_k^{(\mathrm{L})} \right). \quad (11)$$

Let $\Delta_{k,j}^{[t]} = D_k^{(\mathrm{L})} - D_{k,j}^{[t]}$, $\Delta_{k,j}^{[t]}$ can be interpreted as the latency reduction of UE k when associated with EN j. It can be seen from (11) that minimizing $\sum_{k=1}^{K} D_k^{[t]}$ is equivalent to maximizing $\sum_{k=1}^{K} \sum_{j=1}^{J} \Delta_{k,j}^{[t]}$. Thus, we define the system reward at time t as follows:

$$\mathcal{R}_{\boldsymbol{\pi}}^{[t]} = \sum_{k=1}^{K} \sum_{j=1}^{J} x_{k,j}^{[t]} \Delta_{k,j}^{[t]}. \quad (12)$$

Then, the problem of finding the optimal policy that maximizes the accumulated reward is formulated as:

$$\mathbf{P1}: \max_{\boldsymbol{\pi}} \sum_{t=1}^{T} \mathcal{R}_{\boldsymbol{\pi}}^{[t]} \quad (13)$$

$$\text{s.t.} \quad \sum_{i=1}^{N} a_{i,j}^{[t]} s_i \le E_{\mathrm{H}},\ j \in \mathcal{J},\ t = 1, \ldots, T, \quad (14)$$

$$\sum_{i=1}^{N} b_{i,j}^{[t]} q_i \le E_{\mathrm{R}},\ j \in \mathcal{J},\ t = 1, \ldots, T, \quad (15)$$

$$b_{i,j}^{[t]} \le a_{i,j}^{[t]},\ i \in \mathcal{N},\ j \in \mathcal{J},\ t = 1, \ldots, T, \quad (16)$$

$$\sum_{j=1}^{J} x_{k,j}^{[t]} \le 1,\ k \in \mathcal{K},\ t = 1, \ldots, T, \quad (17)$$

$$\sum_{k=1}^{K} x_{k,j}^{[t]} \le U_j,\ j \in \mathcal{J},\ t = 1, \ldots, T, \quad (18)$$

$$a_{i,j}^{[t]},\ b_{i,j}^{[t]} \in \{0, 1\},\ i \in \mathcal{N},\ j \in \mathcal{J},\ t = 1, \ldots, T, \quad (19)$$

$$x_{k,j}^{[t]} \in \{0, 1\},\ k \in \mathcal{K},\ j \in \mathcal{J},\ t = 1, \ldots, T. \quad (20)$$

In $\mathbf{P1}$, Constraints (14) and (15) reflect the storage capacities of disk and RAM, respectively. Constraint (16) is due to the fact that a program must be stored on the disk in order to be loaded to the RAM. Constraint (17) indicates that each UE can be associated with at most one EN. Finally, Constraint (18) specifies an upper bound on the number of UEs that can be served by EN j.

V. Solution for The Single-EN Scenario

We first consider the scenario in which ENs are not densely deployed, hence the coverage areas of neighbor ENs do not overlap. In this case, the user association strategies of different ENs are mutually independent. As a result, the program placement strategy can be optimized from the perspective of a single EN. Without loss of generality, we consider the program placement optimization at EN j:

$$\mathbf{P2}: \max_{\pi_j} \sum_{t=1}^{T} \sum_{k=1}^{K} x_{k,j}^{[t]} \Delta_{k,j}^{[t]} \quad (21)$$

$$\text{s.t.: } (14) - (16) \text{ and } (19).$$

Problem $\mathbf{P2}$ is a time-series decision-making problem with two sets of coupled decision variables. To solve it, we first transform the Problem $\mathbf{P2}$ into an MAB problem. Then, we apply the Thompson sampling (TS) algorithm to obtain a policy that converges to the optimal program placement strategy.

The program placement strategy at time t can be expressed by a $2N \times 1$ vector $[a_{1,j}^{[t]}, \ldots, a_{N,j}^{[t]}, b_{1,j}^{[t]}, \ldots, b_{N,j}^{[t]}]$. As the elements of this vector are binary variables, the total number of possible values (strategies) is 2^{2N}. We denote the various possible strategies by $\mathbf{z}_m$, $m = 1, \ldots, 2^{2N}$. If we model all 2^{2N} program placement strategies as arms in a MAB problem, the complexity of the problem would be prohibitively high when N is large. To this end, we reduce the dimensionality of the problem by treating the feasible program placement strategies that are possibly optimal as arms. These feasible strategies are obtained with the following steps:

(a) Among all vectors $\mathbf{z}_m$, $m = 1, \ldots, 2^{2N}$, we select the ones that satisfy all the constraints (14)-(16).

(b) Among all the vectors selected by step (a), we keep the ones that satisfy the following conditions: if the value of

any $a_{i,j}^{[t]}$ or $b_{i,j}^{[t]}$ is changed from 0 to 1, one of the storage constraints in (14) and (15) would be violated. The other vectors are removed since they cannot be the optimal solution.[3]

(c) Let M be the number of the remaining program placement strategies after steps (a) and (b). We denote the set of these strategies as $\mathcal{F} = \{\mathbf{z}_1, \ldots, \mathbf{z}_M\}$. Then, the M strategies in $\mathcal{F}$ are the arms to be played and learned. The objective of the MAB problem is to find the arm with the largest mean reward.

After the set of arms have been determined, we apply the TS algorithm in [21] to solve the MAB problem. In the formulated MAB problem, the reward for playing each arm m is a random variable that takes values in $[0, 1]$, and can be generated from any arbitrary distribution with mean ρ_m. Here, ρ_m is defined as the *normalized* expected latency reduction that results from applying the mth program placement strategy, given by:

$$\rho_m = \frac{\overline{\Delta}_j\left(\mathbf{a}_j(m), \mathbf{b}_j(m)\right)}{{\Delta_j}'} \tag{22}$$

where $\overline{\Delta}_j\left(\mathbf{a}_j(m), \mathbf{b}_j(m)\right)$ is the expected latency reduction for UEs associated with EN j under the mth program placement strategy, denoted by $\mathbf{a}_j(m) = [a_{1,j}(m), \ldots, a_{N,j}(m)]$ and $\mathbf{b}_j(m) = [b_{1,j}(m), \ldots, b_{N,j}(m)]$; ${\Delta_j}'$ is the reference latency reduction, which is calculated by setting all elements in $\mathbf{a}_j(m)$ and $\mathbf{b}_j(m)$ to 1. Note that $\overline{\Delta}_j\left(\mathbf{a}_j(m), \mathbf{b}_j(m)\right)$ and ${\Delta_j}'$ are calculated based on the long-term average value of the per-bit offloading time, i.e., the expected value of $\frac{1}{R_{k,j}}$ for UEs served by EN j.

Since the mean rewards of playing different arms (ρ_m, $m = 1, \ldots, M$) are not known a prior, EN j has a "belief" for the distribution of each ρ_m, which is updated when arm m is played. In the TS algorithm [21], the prior distribution of ρ_m ("belief") is updated based on the outcome of a Bernoulli trial. Thus, the beta distribution is a conjugate prior of the distribution of ρ_m, as the posterior distribution is still a beta distribution after each update. Let $\eta_m^{[t]}$ be the prior distribution for ρ_m at time t, it follows beta distribution with parameters $\alpha_m^{[t]}$ and $\beta_m^{[t]}$, i.e., $\eta_m^{[t]} \sim \text{Beta}(\alpha_m^{[t]}, \beta_m^{[t]})$, $m = 1, \ldots, M$, $t = 1, \ldots, T$. For a beta distribution $\text{Beta}(\alpha, \beta)$ that is conjugated with a Bernoulli trial, α and β are the numbers of observed successes and fails of the Bernoulli trial, respectively. The mean of $\text{Beta}(\alpha, \beta)$ is $\alpha/\alpha + \beta$, and the higher α and β, the tighter the distribution is concentrated around its mean. In our problem, the initial prior distributions of $\eta_m^{[t]}$ are set to be $\eta_m^{[1]} \sim \text{Beta}(1, 1)$, $m = 1, \ldots, M$, which correspond to a uniform distribution.

At each time step t, the EN samples $\eta_m^{[t]}$ from $\text{Beta}(\alpha_m^{[t]}, \beta_m^{[t]})$ and records the sampled values $\hat{\eta}_m^{[t]}$, $m = 1, \ldots, M$. Then, the EN plays the arm $m^* = \arg\max_m \hat{\eta}_m^{[t]}$ and observes the reward $r^{[t]}$. The observed reward is the normalized latency reduction achieved by all UEs associated with EN j, given by:

$$r^{[t]} = \frac{\overline{\Delta}_j\left(\mathbf{a}_j(m^*), \mathbf{b}_j(m^*)\right)^{[t]}}{{\Delta_j}'}. \tag{23}$$

Based on $r^{[t]}$, EN j performs a Bernoulli trial with success probability $r^{[t]}$ and observes the outcome $\tilde{r}^{[t]}$. Then, it updates the parameters of the distribution of $\eta_{m^*}^{[t]}$ by:

$$(\alpha_{m^*}^{[t+1]}, \beta_{m^*}^{[t+1]}) = \begin{cases} (\alpha_{m^*}^{[t]} + 1, \beta_{m^*}^{[t]}), & \text{if } \tilde{r}^{[t]} = 1 \\ (\alpha_{m^*}^{[t]}, \beta_{m^*}^{[t]} + 1), & \text{otherwise.} \end{cases} \tag{24}$$

The distributions of other arms ($m \neq m^*$) remain the same. The sampling and update processes are repeated from time $t = 1$ to $t = T$. Given a sufficiently large T, the optimal program placement strategy can be obtained. The TS algorithm for the single-EN scenario is summarized in Algorithm 1.

Algorithm 1: TS Algorithm for Optimal Program Placement Strategy in Single-EN Scenario

```
Initialize: η_m^[1] ~ Beta(1,1), m = 1,...,M ;
for t = 1 : T do
    for m = 1 : M do
        Sample η_m^[t] from Beta(α_m^[t], β_m^[t]) ;
    end
    Play arm m* = arg max_m η̂_m^[t] and observe reward r^[t] ;
    Perform a Bernoulli trial with success probability r^[t]
      and record output r̃^[t] ;
    if r̃^[t] = 1 then
        α_{m*}^[t+1] = α_{m*}^[t] + 1 and β_{m*}^[t+1] = β_{m*}^[t] ;
    else
        α_{m*}^[t+1] = α_{m*}^[t] and β_{m*}^[t+1] = β_{m*}^[t] + 1 ;
    end
end
```

VI. Solution for The Multi-EN Scenario

With the trend of *network densification* in cellular networks, the ENs are expected to be densely deployed with overlapping coverage areas. As a result, a UE may have multiple choices of ENs for task offloading, and it can optimize its EN selection depending on the program availability, communication link, and computational capability of nearby ENs. On the other hand, user association determines the load distribution among ENs and the latency performance of users associated with each EN. Thus, joint optimization of program placement and user association is necessary to achieve the full potential of latency reduction.

A. Solution Overview

As the decision variables for program placement strategy are binary integers, the number of feasible strategies of all ENs grows exponentially with the number of ENs. As a result, the TS-based solution algorithm in Section V cannot be applied in the multi-EN scenario due to the prohibitively large number of arms to be played. To derive an effective solution, we decompose the original problem into three subproblems

[3]When one more program is added to a strategy and both storage constraints are still satisfied, the resulting new strategy must be a feasible strategy that outperforms the original strategy. Thus, it is certain that the original strategy is not optimal.

to be solved at each time t and iteratively solve them to obtain a near-optimal solution. Specifically, we first apply a TS algorithm to learn the task popularity profile at time t, denoted by $\boldsymbol{\theta}^{[t]}$. With $\boldsymbol{\theta}^{[t]}$, the optimization of program placement and user association is formulated as follows:

$$\mathbf{P3}: \max_{\left\{\mathbf{a}^{[t]},\mathbf{b}^{[t]},\mathbf{x}^{[t]}\right\}} \sum_{k=1}^{K}\sum_{j=1}^{J} x_{k,j}^{[t]}\Delta_{k,j}^{[t]} \tag{25}$$

$$\text{s.t.} \quad \sum_{i=1}^{N} a_{i,j}^{[t]} s_i \le E_{\mathrm{H}},\ j \in \mathcal{J}, \tag{26}$$

$$\sum_{i=1}^{N} b_{i,j}^{[t]} q_i \le E_{\mathrm{R}},\ j \in \mathcal{J}, \tag{27}$$

$$b_{i,j}^{[t]} \le a_{i,j}^{[t]},\ i \in \mathcal{N},\ j \in \mathcal{J}, \tag{28}$$

$$\sum_{j=1}^{J} x_{k,j}^{[t]} \le 1,\ k \in \mathcal{K}, \tag{29}$$

$$\sum_{k=1}^{K} x_{k,j}^{[t]} \le U_j,\ j \in \mathcal{J}, \tag{30}$$

$$x_{k,j}^{[t]} = 0,\ k \in \mathcal{K},\ \forall j \notin \pi_k, \tag{31}$$

$$x_{k,j}^{[t]} \in \{0,1\},\ k \in \mathcal{K},\ j \in \mathcal{J}, \tag{32}$$

$$a_{i,j}^{[t]},\ b_{i,j}^{[t]} \in \{0,1\},\ i \in \mathcal{N},\ j \in \mathcal{J}. \tag{33}$$

where $\mathbf{a}^{[t]}$, $\mathbf{b}^{[t]}$, and $\mathbf{x}^{[t]}$ are the matrices $[a_{i,j}^{[t]}]_{i\in\mathcal{N},j\in\mathcal{J}}$, $[b_{i,j}^{[t]}]_{i\in\mathcal{N},j\in\mathcal{J}}$, and $[x_{k,j}^{[t]}]_{k\in\mathcal{K},j\in\mathcal{J}}$. We decompose $\mathbf{P3}$ into two levels of subproblems. The lower-level subproblem is program placement under a given user association, and we propose a greedy algorithm to solve it. The higher-level subproblem is user association given that the program placement strategy is applied, we propose a dual decomposition-based approach solve it.

B. Task Popularity Profile Acquisition

To obtain $\boldsymbol{\theta}^{[t]}$, we consider a Bernoulli bandit problem by regarding the N programs as the arms to be played, and the mean reward for playing arm i is θ_i. The values of θ_i are learned via the same TS algorithm presented in Section V. Let $\phi_i^{[t]}$ be the prior distribution of θ_i at time t, it follows beta distribution with parameters $\alpha_i^{[t]}$ and $\beta_i^{[t]}$, i.e., $\phi_i^{[t]} \sim \mathrm{Beta}(\alpha_i^{[t]}, \beta_i^{[t]})$, $i \in \mathcal{N}$. The initial distribution of $\phi_i^{[t]}$ is set to be $\phi_i^{[t]} \sim \mathrm{Beta}(1,1)$, $i \in \mathcal{N}$. We assume all ENs send their task request records to an agent. At time t, the agent samples $\hat{\phi}_i^{[t]}$ from $\mathrm{Beta}(\alpha_i^{[t]}, \beta_i^{[t]})$ and records the sampled values $\hat{\phi}_i^{[t]}$, $i \in \mathcal{N}$. Then, it plays the arm $i^* = \max_i \hat{\phi}_i^{[t]}$ and observes the reward. The reward is the proportion of UEs that have requested task i^* at time t, given by:

$$r^{[t]} = \frac{\Theta_{i^*}^{[t]}}{\sum_{i=1}^{N}\Theta_i^{[t]}} \tag{34}$$

where Θ_i is the number of requests for task i at time t. Then, the agent performs a Bernoulli trial with probability $r^{[t]}$ and observes the outcome $\tilde{r}^{[t]}$. Finally, the distribution of ϕ_i is updated by:

$$(\alpha_i^{[t+1]}, \beta_i^{[t+1]}) = \begin{cases} (\alpha_i^{[t]}+1, \beta_i^{[t]}), & \text{if } i = i^* \,\&\, \tilde{r}^{[t]} = 1 \\ (\alpha_i^{[t]}, \beta_i^{[t]}+1), & \text{if } i = i^* \,\&\, \tilde{r}^{[t]} = 0 \\ (\alpha_i^{[t]}, \beta_i^{[t]}), & \text{if } i \ne i^* \end{cases} \tag{35}$$

Algorithm 2: TS Algorithm for Obtaining Task Popularity Profile

```
Initialize: φ_i^[1] ~ Beta(1,1), i ∈ N ;
for t = 1 : T do
    for i = 1 : N do
        Sample φ_i^[t] from Beta(α_i^[t], β_i^[t]) ;
    end
    Play arm i* = arg max_i φ̂_i^[t] and observe reward r^[t] ;
    Perform a Bernoulli trial with success probability r^[t]
      and record outcome r̃^[t] ;
    if r̃^[t] = 1 then
        α_{i*}^[t+1] = α_{i*}^[t] + 1 ;
    else
        β_{i*}^[t+1] = β_{i*}^[t] + 1 ;
    end
end
```

The process for learning $\boldsymbol{\theta}^{[t]}$ is summarized in Algorithm 2.

C. Program Placement with Given User Association

With given $\phi_i^{[t]}$, the program placement optimization at time t is a Knapsack problem, which is generally NP-hard. Since such a problem needs to be solved at each time step, only a low-complexity solution can be implemented. To this end, we propose a heuristic program placement strategy that greedily allocates storage space to the programs with the highest request probability per occupied space. Specifically, we first sort the programs by the descending order of $\phi_i^{[t]}/(s_i+q_i)$ and put them into the RAM according to such order until the storage space of RAM is full, i.e., $\sum_{i=1}^{N} b_i^{[t]} q_i > E_{\mathrm{R}}$. Then, we sort the remaining programs by the descending order of $\phi_i^{[t]}/s_i$ and put them into the disk according to such order until the storage space of disk is full, i.e., $\sum_{i=1}^{N} a_i^{[t]} s_i > E_{\mathrm{H}}$.

D. Dual Decomposition-Based User Association

Let $\tilde{\Delta}_{k,j}(\mathbf{x}^{[t]})$ be the value of $\Delta_{k,j}^{[t]}$ if the greedy program placement strategy in Section VI-C is applied under user association $\mathbf{x}^{[t]}$. The user association problem is formulated as:

$$\mathbf{P4}: \max_{\left\{\mathbf{x}^{[t]}\right\}} \sum_{k=1}^{K}\sum_{j=1}^{J} x_{k,j}^{[t]}\tilde{\Delta}_{k,j}(\mathbf{x}^{[t]}) \tag{36}$$

$$\text{s.t.: } (29)-(32)$$

Problem **P4** is an integer programming problem, which is generally NP-hard. To develop an effective solution algorithm, we first relax the integer constraint by allowing all $\mathbf{x}^{[t]}$ to take values in $[0,1]$. Let **P4-Relexted** be the relaxed problem, it can be verified that the objective function of **P4-Relexted** is concave and the feasible region defined by all constraints is convex. Thus, **P4-Relexted** is a convex optimization problem and we apply a dual decomposition approach to obtain its optimal solution.

The decision variables of **P4-Relexted** are coupled with each other via the quadratic terms in the objective function. To this end, we introduce a set of auxiliary variables $V_j^{[t]} =$

$\sum_{k=1}^{K} x_{k,j}^{[t]}$, $j \in \mathcal{J}$, which are the traffic loads of ENs. At each iteration of the dual decomposition algorithm, we first fix the traffic loads and find the optimal solution of user association. The solution is then used to update the traffic loads, which will be used in the next iteration. With given $\{V_1^{[t]}, \ldots, V_J^{[t]}\}$, we have the following problem:

$$\mathbf{P5}: \max_{\{\mathbf{x}^{[t]}\}} \sum_{k=1}^{K} \sum_{j=1}^{J} x_{k,j}^{[t]} \tilde{\Delta}_{k,j}(\mathbf{x}^{[t]}) \tag{37}$$

$$\text{s.t.: } (29) - (32)$$

$$\sum_{k=1}^{K} x_{k,j}^{[t]} = V_j^{[t]}, \ j \in \mathcal{J}, \tag{38}$$

Let $\boldsymbol{\lambda}^{[t]} = [\lambda_1^{[t]}, \ldots, \lambda_J^{[t]}]$ be the Lagrangian multipliers for the constraints (38). Applying a partial relaxation on these constraints, we have the following Lagrangian function:

$$\mathcal{L}\left(\mathbf{x}^{[t]}, \boldsymbol{\lambda}^{[t]}\right) = \sum_{k=1}^{K} \sum_{j=1}^{J} x_{k,j}^{[t]} \tilde{\Delta}_{k,j}(\mathbf{x}^{[t]}) + \sum_{j=1}^{J} \lambda_j^{[t]} \left(\sum_{k=1}^{K} x_{k,j}^{[t]} - V_j^{[t]} \right). \tag{39}$$

The corresponding dual problem of **P5** is given by:

$$\textbf{P5-Dual:} \quad \min_{\{\boldsymbol{\lambda}^{[t]}\}} \ g(\boldsymbol{\lambda}^{[t]}) \tag{40}$$

where $g(\boldsymbol{\lambda}^{[t]})$ is given by:

$$g(\boldsymbol{\lambda}^{[t]}) = \max_{\{\mathbf{x}^{[t]}\}} \mathcal{L}\left(\mathbf{x}^{[t]}, \boldsymbol{\lambda}^{[t]}\right). \tag{41}$$

The problems given in (40) and (41) are solved iteratively. At each iteration, $\mathbf{x}^{[t]}$ and $\boldsymbol{\lambda}^{[t]}$ are calculated and updated by UEs and ENs, respectively.

Based on the expression of $\mathcal{L}(\mathbf{x}^{[t]}, \boldsymbol{\lambda}^{[t]})$, the problem of maximizing $g(\boldsymbol{\lambda}^{[t]})$ can be decomposed into K independent subproblems that are solved by each UE. Let $\tau = 1, 2, \ldots$ be the index of iteration for the dual decomposition algorithm.[4] At the τ th iteration, each UE solves its subproblem by selecting the EN $j^{*[t]}(\tau)$ that satisfies:

$$j^{*[t]}(\tau) = \arg\max_{j \in \pi_k} \left\{ \tilde{\Delta}_{k,j}(\mathbf{x}^{[t]}(\tau)) - \lambda_j^{[t]}(\tau) \right\}. \tag{42}$$

where $j^{*[t]}(\tau)$, $\mathbf{x}^{[t]}(\tau)$, and $\lambda_j^{[t]}(\tau)$ are the optimal selection of $j^{[t]}$, the matrix $\mathbf{x}^{[t]}$, and value of $\lambda_j^{[t]}$ at iteration τ, respectively.

After completing the selection, each UE notifies the selected EN via a message. Receiving the messages from UEs, each EN updates $\mathbf{x}_j^{[t]} = [x_{1,j}^{[t]}, \ldots, x_{K,j}^{[t]}]$ by:

$$x_{k,j}^{[t]}(\tau) = \begin{cases} 1, & j = j^{*[t]}(\tau) \\ 0, & \text{otherwise,} \end{cases} \tag{43}$$

On the other hand, Problem **P5-Dual** can be decomposed into J subproblems, each to be solved by the corresponding EN. At iteration τ, each EN updates $\lambda_j^{[t]}(\tau)$ by:

$$\lambda_j^{[t]}(\tau + 1) = \lambda_j^{[t]}(\tau) - \varepsilon_j^{[t]}(\tau) \varphi_j^{[t]}(\tau) \tag{44}$$

[4]The updates indexed by $\tau = 1, 2, \ldots$ are performed within each time period t, i.e., the updates indexed by $t = 1, \ldots, T$ are the outer loop that is performed with a larger time scale.

Algorithm 3: Dual Decomposition-Based User Association Algorithm

```
Initialize V^[t] and λ^[t] ;
do
    for k = 1 : K do
        UE k selects EN according to (42) and notifies the
          selected EN ;
    end
    for j = 1 : J do
        EN j updates x_j^[t] with (43) ;
        Updates φ_j^[t] with (45) ;
        Updates λ_j^[t] with (44) ;
        Updates V_j^[t] with (47) ;
    end
    τ++
while (x^[t] does not converge);
```

where $\varphi_j^{[t]}(\tau)$ is the gradient of $\lambda_j^{[t]}(\tau)$, given by:

$$\varphi_j^{[t]}(\tau) = V_j^{[t]}(\tau) - \sum_{k=1}^{K} x_{k,j}^{[t]}(\tau) \tag{45}$$

where $\varepsilon_j^{[t]}(\tau)$ is the step size, given by:

$$\varepsilon_j^{[t]}(\tau) = \frac{g(\boldsymbol{\lambda}^{[t]}(\tau)) - g(\boldsymbol{\lambda}^{*[t]})}{\left\| \boldsymbol{\varphi}^{[t]}(\tau) \right\|^2}. \tag{46}$$

With the updated $\lambda_j^{[t]}(\tau)$, each EN updates $V_j^{[t]}(\tau)$ by:

$$V_j^{[t]}(\tau + 1) = \min\left\{ \sum_{k=1}^{K} x_{k,j}^{[t]}(\tau), U_j \right\}. \tag{47}$$

Finally, all ENs broadcast the updated values of $\lambda_j^{[t]}(\tau)$ and $V_j^{[t]}(\tau)$. Each UE then performs EN selection for the next iteration with (42). The updates performed by UEs and ENs continue until convergence is achieved. The dual decomposition-based user association algorithm is summarized in Algorithm 3.

Lemma 1. *Algorithm 3 converges with a rate faster than the sequence* $\{1/\sqrt{\tau}\}$.

Proof. $\boldsymbol{\lambda}^{[t]}(\tau)) - g(\boldsymbol{\lambda}^{*[t]}(\tau)$

The optimality gap of $\boldsymbol{\lambda}^{[t]}$ satisfies:

$$\|\boldsymbol{\lambda}^{[t]}(\tau + 1)) - \boldsymbol{\lambda}^{*[t]}\|^2 = \left\| \boldsymbol{\lambda}^{[t]}(\tau) - \frac{g(\boldsymbol{\lambda}^{[t]}) - g(\boldsymbol{\lambda}^{*})}{\|\boldsymbol{\varphi}^{[t]}\|^2} \boldsymbol{\varphi}^{[t]} - \boldsymbol{\lambda}^{*[t]} \right\|^2$$

$$= \left\| \boldsymbol{\lambda}^{[t]}(\tau) - \boldsymbol{\lambda}^{*[t]} \right\|^2 + \left(\frac{g(\boldsymbol{\lambda}^{[t]}(\tau)) - g(\boldsymbol{\lambda}^{*[t]})}{\|\boldsymbol{\varphi}^{[t]}(\tau)\|^2} \boldsymbol{\varphi}^{[t]}(\tau) \right)^2 \left\| \boldsymbol{\varphi}^{[t]} \right\|^2$$
$$- 2\left(\boldsymbol{\lambda}^{[t]}(\tau) - \boldsymbol{\lambda}^{*[t]} \right)^{\mathrm{T}} \frac{g\left(\boldsymbol{\lambda}^{[t]}(\tau) \right) - g\left(\boldsymbol{\lambda}^{*[t]} \right)}{\|\boldsymbol{\varphi}^{[t]}(\tau)\|^2} \boldsymbol{\varphi}^{[t]}(\tau)$$
$$\overset{(a)}{\leq} \|\boldsymbol{\lambda}^{[t]}(\tau)) - \boldsymbol{\lambda}^{*[t]}\|^2 - 2\frac{\left(g\left(\boldsymbol{\lambda}^{[t]}(\tau) \right) - g\left(\boldsymbol{\lambda}^{*[t]} \right) \right)^2}{\|\boldsymbol{\eta}^{[t]}(\tau)\|^2}$$
$$+ \left(\frac{g\left(\boldsymbol{\lambda}^{[t]}(\tau) \right) - g\left(\boldsymbol{\lambda}^{*[t]} \right)}{\|\boldsymbol{\eta}^{[t]}(\tau)\|^2} \right)^2 \|\boldsymbol{\varphi}^{[t]}(\tau)\|^2$$
$$\leq \|\boldsymbol{\lambda}^{[t]}(\tau) - \boldsymbol{\lambda}^{*[t]}\|^2 - \frac{\left(g\left(\boldsymbol{\lambda}^{[t]}(\tau) \right) - g\left(\boldsymbol{\lambda}^{*[t]} \right) \right)^2}{\hat{\boldsymbol{\varphi}}^2}.$$

Inequality (a) is due to the convexity of problem **P5-dual**, given by $g\left(\boldsymbol{\lambda}^{[t]}(\tau) \right) - g\left(\boldsymbol{\lambda}^{*[t]} \right) \leq \left(\boldsymbol{\lambda}^{[t]}(\tau) - \boldsymbol{\lambda}^{*[t]} \right)^{\mathrm{T}} \boldsymbol{\varphi}^{[t]}(\tau)$. Variable $\hat{\boldsymbol{\varphi}}$ is an upper bound for $\boldsymbol{\varphi}^{[t]}(\tau)$. Since $\lim_{\tau \to \infty} \boldsymbol{\lambda}^{[t]}(\tau + 1) = \lim_{\tau \to \infty} \boldsymbol{\lambda}^{[t]}(\tau)$, we have $\lim_{\tau \to \infty} g\left(\boldsymbol{\lambda}^{[t]}(\tau) \right) = g\left(\boldsymbol{\lambda}^{*[t]} \right)$. Summing this inequality over τ, we have:

$$\sum_{\tau=1}^{\infty} \left(g\left(\boldsymbol{\lambda}^{[t]}(\tau) \right) - g\left(\boldsymbol{\lambda}^{*[t]} \right) \right)^2 \leq \hat{\boldsymbol{\varphi}}^2 \left\| \boldsymbol{\lambda}^{[t]}(1) - \boldsymbol{\lambda}^{*[t]} \right\|^2. \tag{48}$$

Suppose for contradiction, $g\left(\boldsymbol{\lambda}^{[t]}(\tau) \right)$ converges slower than $\{1/\sqrt{\tau}\}$. Then, $\lim_{\tau \to \infty} \sqrt{\tau} \left(g\left(\boldsymbol{\lambda}^{[t]}(\tau) \right) - g\left(\boldsymbol{\lambda}^{*[t]} \right) \right) > 0$. Therefore, there exists a positive number π and a sufficiently large τ' such that:

$$\sqrt{\tau} \left(g\left(\boldsymbol{\lambda}^{[t]}(\tau) \right) - g\left(\boldsymbol{\lambda}^{*[t]} \right) \right) \geq \pi, \forall \tau > \tau'. \tag{49}$$

Taking the square sum of (49) from τ' to ∞, we have:

$$\sum_{\tau=\tau'}^{\infty} \left(g\left(\boldsymbol{\lambda}^{[t]}(\tau) \right) - g\left(\boldsymbol{\lambda}^{*[t]} \right) \right)^2 \geq \pi^2 \sum_{\tau=\tau'}^{\infty} \frac{1}{\tau} = \infty. \tag{50}$$

This contradicts with the fact given in (48). We conclude that $g\left(\boldsymbol{\lambda}^{[t]}(\tau) \right)$ converges faster than the sequence $\{1/\sqrt{\tau}\}$. □

Lemma 2. *The complexity of Algorithm 3 is upper bounded by $1/\omega^2$, where ω is the threshold of convergence for $\boldsymbol{\lambda}^{[t]}$.*

Proof. According to Lemma 2, for a sufficiently large τ and a sufficiently small ω, $g\left(\boldsymbol{\lambda}^{[t]}(\tau) \right) - g\left(\boldsymbol{\lambda}^{*[t]} \right)$ is guaranteed to be smaller than ω. Thus, it takes less than $1/\omega^2$ steps for the sequence $\boldsymbol{\lambda}^{[t]}$ to achieve a optimality gap that is smaller than ω. □

VII. Simulation Results

In this section, we validate our proposed schemes with simulations. We consider multiple ENs and users randomly located in a 400 m × 400 m rectangular area. The channel is modeled by a distance-dependent path loss $140.7 + 36.7 \log_{10} d$ in dB and Rayleigh fading [6], where d is the distance between an EN and a UE in meters. The UE transmission power is 20 dBm and the noise density is -174 dBm/Hz. The system bandwidth for uplink transmission is 10 MHz. The input data size follows a truncated normal distribution in the range of $[400, 600]$ Kb with a mean of 500 Kb. The complexity of a task is uniformly distributed in $[500, 1500]$ CPU cycles/bit. The computational capabilities of UEs and ENs are 1 GHz and 20 GHz, respectively. The storage spaces of disk and RAM of each EN are in the ranges of $[50, 200]$ GB and $[5, 10]$ GB, respectively. The size of each program follows a truncated normal distribution within the range of 100 MB around its mean. The number of programs N ranges from 100 to 500. Unless otherwise stated, the default mean program size and the number of programs are 500 MB and 200, respectively; the default storage spaces of disk and RAM are 100 GB and 8 GB, respectively. The task popularity profile follows a Zipf distribution [22]. Without loss of generality, let $\theta_1 \geq \theta_2 \geq \cdots \geq \theta_N$. Then, θ_i is calculated by:

$$\theta_i = i^{-\gamma} / \textstyle\sum_{i=1}^{N} i^{-\gamma}, i \in \mathcal{N} \tag{51}$$

where γ is the parameter indicating the "skewness" of popularity profile. When $\gamma = 0$, the popularity is uniformly distributed among all tasks. As γ grows, the popularity becomes concentrated on a few popular tasks. We set the default value of γ to be 0.2. At each round of the simulation, we first generate the parameters of each task i by sampling s_i, z_i, $\tilde{s}_{k,i}$, and θ_i according to the corresponding distributions. Then, each UE generates one of the N tasks according to the values of θ_i. We focus on evaluating the long-term average latency of all users, which is the time-averaged cumulated average latency from $t = 1$ to $t = T$ by T, i.e., $\frac{\sum_{t=1}^{T} \sum_{k=1}^{K} D_k^{[t]}}{T}$

A. Single-EN Scenario

We first evaluate the performance of different schemes in the single-EN scenario. We make comparisons with two benchmark schemes. The first scheme is *random placement*, in which the programs are randomly selected and put into the disk and RAM of each EN. The second scheme is *greedy placement*, in which we first apply Algorithm 2 to obtain the task popularity at each time t, then apply the greedy program placement strategy in Section VI-C. Note that the complexity of the greedy scheme is similar to the proposed scheme, since both of them adopt TS to learn the task popularity (one is direct and the other is indirect).

We first plot the average latency versus the number of users in Fig. 2(a). It can be seen that the average latency increases as the number of users grows, since both the communication and computing resources are shared among an increased number of users, resulting in longer offloading and computing time. The proposed scheme outperforms other schemes as the optimal program placement strategy obtained from the TS algorithm is applied. The average latency under different values of γ is plotted in Fig. 2(b). Due to similar reasons, the proposed scheme achieves the lowest average latency. As γ grows, the latency of the random placement scheme increases while the

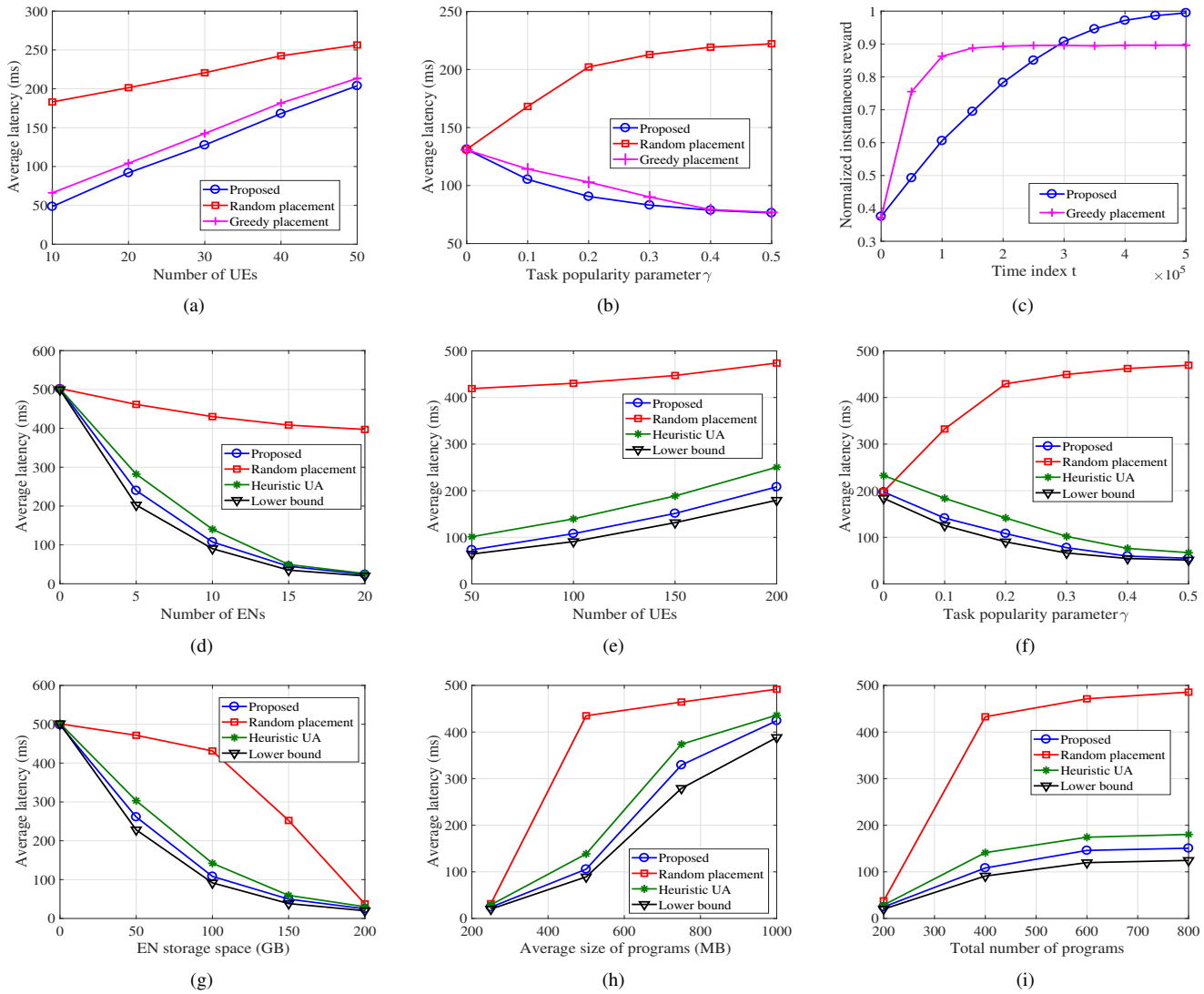

Fig. 2. Performance evaluation results. (a) Average latency vs. number of users for single-EN scenario. (b) Average latency vs. γ for single-EN scenario. Number of users is 20. (c) Convergence comparison between the proposed and greedy placement strategy. Number of programs is 50. (d) Average latency vs. number of ENs. (e) Average latency vs. number of users. (f) Average latency vs. γ for multi-EN scenario. (g) Average latency vs. EN storage space. (h) Average latency vs. average program size. (i) Average latency vs. total number of programs.

latency of the other two schemes decreases. This is because when γ is large, only a few tasks will be frequently requested, the advantage of storing popular tasks becomes significant.

Fig. 2(c) shows the convergence property of different schemes. The greedy placement scheme converges faster, since it only needs to learn the task popularity, which corresponds to solving an MAB problem with N arms. In contrast, the proposed scheme needs to learn a subset of feasible program placement strategies by solving an MAB problem with a much larger number of arms, resulting in slow convergence.

B. Multi-EN Scenario

In the multi-EN scenario, we compare with two benchmark schemes and a lower bound of latency. The first scheme is *random placement*, which is the same scheme as described in the single-EN scenario. The second scheme is *Heuristic UA*, in which each user is associated with the EN with the highest SINR. For a fair comparison, the proposed dual decomposition-based user association is applied to the random placement scheme; the proposed program placement strategy is applied to the Heuristic UA scheme. The lower bound of latency is derived by relaxing all integer constraints in Problem **P3** and solving the resulting linear programming (LP). The value of the objective function under the optimal solution of the LP is a lower bound for the sum latency of all UEs.

The average latency versus the numbers of ENs is shown in Fig. 2(d). We observe that as the number of ENs grows, the average latencies of all schemes decreases, since more UEs can employ nearby ENs for task offloading. Without optimizing program placement, the random placement scheme achieves a quite limited latency reduction compared to other schemes. The proposed scheme outperforms the heuristic UA scheme, because load balancing is achieved with our dual

decomposition-based user association, which contributes to lower average latency. Furthermore, the performance of the proposed scheme is close to the lower bound, showing that our solution is near-optimal. The average latency versus the number of UEs in the area is shown in Fig. 2(e). We observe that the latencies of all schemes are increased as more UEs join the system. This happens because the communication and computational resources are shared among all UEs. Besides, ENs receive stronger aggregated interference caused by the uplink transmission of UEs served by neighboring ENs. The proposed scheme outperforms other schemes since both user association and program placement are optimized.

The impact of γ on average latency is presented in Fig. 2(f), where similar trends exhibit in the single-EN scenario are observed. In particular, when γ is sufficiently large, the average latency of the proposed scheme becomes closer to the lower bound. This is because the task popularity is concentrated on a few tasks when γ is large, hence the greedy program placement strategy presented in Section VI-C is highly likely to be optimal. The average latency versus the storage space of EN is shown in Fig. 2(g). As the storage space increases, the latencies of our proposed scheme and heuristic UA scheme drop much faster than the random placement scheme, since the storage space is allocated to the programs that are frequently requested and occupy relatively small storage space, resulting in improved utilization. When the storage space is sufficiently large, the tension caused by limited storage is mitigated, all schemes can achieve relatively low latency.

The average latency versus the average program size is shown in Fig. 2(h). When the average program size is small, the ENs are able to store most of the programs, hence the latencies of all schemes are low. As the programs get larger, the latencies of the proposed scheme and heuristic UA scheme grow at a slower rate than the random placement scheme, due to the optimized program placement. The average latency versus the total number of programs is shown in Fig. 2(i), where similar trends are observed.

VIII. Conclusion

In this paper, we investigated the problem of optimizing program placement and user association in storage-constrained MEC systems. We formulated such a problem as a sequential decision-making problem. We first considered the single-EN scenario and proposed an MAB-based solution to derive the optimal solution. Then, we proposed a solution framework for the multi-EN scenario, in which we decomposed the original problem into three subproblems and solved them iteratively. Simulation results show that the proposed schemes reduce the average latency by $30\%\sim100\%$.

Acknowledgment

This research was supported in part by NSF (grants CNS-1910348, CNS-1563655, CNS-1731164, CNS-1813401, and IIP-1822071) and by the Broadband Wireless Access & Applications Center (BWAC). Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the author(s) and do not necessarily reflect the views of NSF.

References

[1] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing: A key technology towards 5G," *ETSI White Paper,* vol. 11, 2015.

[2] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE J. Sel. Areas Commun.,* vol. 36, no. 3, pp. 587–597, Mar. 2018.

[3] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surv. Tuts.,* vol. 19, no. 4, pp. 2322–2358, Sept–Dec. 2017.

[4] A. Ndikumana, N. H. Tran, T. M. Ho, Z. Han, W. Saad, D. Niyato, and C. S. Hong, "Joint communication, computation, caching, and control in big data multi-access edge computing," *IEEE Trans. Mobile Comput.,* to appear. DOI: 10.1109/TMC.2019.2908403.

[5] I. Blair, "Mobile app download and usage statistics (2019)," [Online]. Available: https://buildfire.com/app-statistics/

[6] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.,* vol. 68, no. 1, pp. 856–868, Jan. 2019.

[7] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Commun.,* vol. 16, no. 3, pp. 1397–1411, Mar. 2017.

[8] N. Golrezaei, A. F. Molisch, A. G. Dimakis, and G. Caire, "Femtocaching and device-to-device collaboration: A new architecture for wireless video distribution," *IEEE Commun. Mag.,* vol. 51, no. 4, pp. 142–149, Apr. 2013.

[9] W. Jiang, G. Feng and S. Qin, "Optimal cooperative content caching and delivery policy for heterogeneous cellular networks," *IEEE Trans. Mobile Comput.,* vol. 16, no. 5, pp. 1382–1393, May 2017.

[10] S. Wang. T. Wang, and X. Cao, "In-network caching: An efficient content distribution strategy for mobile networks," *IEEE Wireless Commun.,* vol. 26, no. 5, pp. 84–90, Oct. 2019.

[11] Z. Chang, L. Lei, Z. Zhou, S. Mao, and T. Ristaniemi, "Learn to cache: Machine learning for network edge caching in the big data era," *IEEE Wireless Commun.,* vol. 25, no. 3, pp. 28–35, June 2018.

[12] B. N. Bharath, K. G. Nagananda, and H. V. Poor, "A learning-based approach to caching in heterogeneous small cell networks," *IEEE Trans. Commun.,* vol. 64, no. 4, pp. 1674–1686, Apr. 2016,

[13] P. Blasco and D. Gündüz, "Learning-based optimization of cache content in a small cell base station," in *Proc. ICC'14,* Sydney, Australia, June 2014, pp. 1897–1903.

[14] K. Zhang, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Cooperative content caching in 5G networks with mobile edge computing," *IEEE Wireless Commun.,* vol. 25, no. 3, pp. 80–87, June 2018.

[15] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge AI: Intelligentizing mobile edge computing, caching and communication by federated learning," *IEEE Netw.,* vol. 33, no. 5, pp. 156–165, Sept./Oct. 2019.

[16] D. T. Hoang, D. Niyato, D. N. Nguyen, E. Dutkiewicz, P. Wang, and Z. Han, "A dynamic edge caching framework for mobile 5G networks," *IEEE Wireless Commun.,* vol. 25, no. 5, pp. 95–103, Oct. 2018.

[17] L. T. Tan and R. Q. Hu, "Mobility-aware edge caching and computing framework in vehicle networks: A deep reinforcement learning," *IEEE Trans. Veh. Tech.,* vol. 67, no. 11, pp. 10190–10203, Nov. 2018.

[18] Q. Ye, B. Rong, Y. Chen, M.A.-Shalash, C. Caramanis, and J. G. Andrews, "User association for load balancing in heterogeneous cellular networks," *IEEE Trans. Wireless Commun.,* vol. 12, no. 6, pp. 2706–2716, June 2013.

[19] S. Sardellitti, M. Merluzzi, and S. Barbarossa, "Optimal association of mobile users to multi-access edge computing resources," in *Proc. IEEE ICC'18,* Kansas City, MO, May 2018, pp. 1–6.

[20] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint computation offloading and user association in multi-task mobile edge computing," *IEEE Trans. Veh. Technol.,* vol. 67, no. 12, pp. 12313–12325, Oct. 2018.

[21] S. Agrawal and N. Goyal, "Analysis of Thompson sampling for the multi-armed bandit problem," in *Proc. of the 25th Annual Conference On Learning (COLT)*, vol. 23, pages 39.1–39.26, June 2012.

[22] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," *in Proc. IEEE INFOCOM'99,* vol. 1, pp. 126–134.