Adaptive User-managed Service Placement for Mobile Edge Computing: An Online Learning Approach

Tao Ouyang, Rui Li, Xu Chen, Zhi Zhou, Xin Tang
School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China
{ouyt9, lirui223, tangx6}@mail2.sysu.edu.cn, {chenxu35, zhouzhi9}@mail.sysu.edu.cn

***Abstract*—Mobile Edge Computing (MEC), envisioned as a cloud extension, pushes cloud resource from the network core to the network edge, thereby meeting the stringent service requirements of many emerging computation-intensive mobile applications. Many existing works have focused on studying the system-wide MEC service placement issues, personalized service performance optimization yet receives much less attention. Thus, in this paper we propose a novel adaptive user-managed service placement mechanism, which jointly optimizes a user's perceived-latency and service migration cost, weighted by user preferences. To overcome the unavailability of future information and unknown system dynamics, we formulate the dynamic service placement problem as a contextual Multi-armed Bandit (MAB) problem, and then propose a Thompson-sampling based online learning algorithm to explore the dynamic MEC environment, which further assists the user to make adaptive service placement decisions. Rigorous theoretical analysis and extensive evaluations demonstrate the superior performance of the proposed adaptive user-managed service placement mechanism.**

I. Introduction

With the widespread use of smart devices, more and more applications, as exemplified by face recognition, data stream processing, and interactive gaming, catch more and more attention [1] [2]. These kinds of applications typically require intensive computation resource and high energy consumption for processing, but a mobile device only has limited computation and energy capacity due to the physical size constraint. To address this challenge, mobile edge computing (MEC), envisioned as a new computing paradigm, has been proposed [3]–[5], in order to provide efficient computation-augmented service in close proximity to mobile devices and users.

With the sinking of the computing capabilities at the edge, new challenges would arise due to the limitation coverage of edge servers and erratic user mobility, i.e., a user may roam throughout wireless regions served by different edge servers during a continuous service. To maintain the satisfactory service performance, dynamic service migration needs to be considered. When a user moves from one served region to another, we can either continue to run the service on the original edge server with data relay through edge-edge connection, or migrate the service to another edge server to follow the user mobility. Clearly, the former would lead to long communication latency due to the expanded network distance, while the later would reduce the service latency but incur additional migration cost, such as scarce bandwidth usage, potential service interruption and even handover failure.

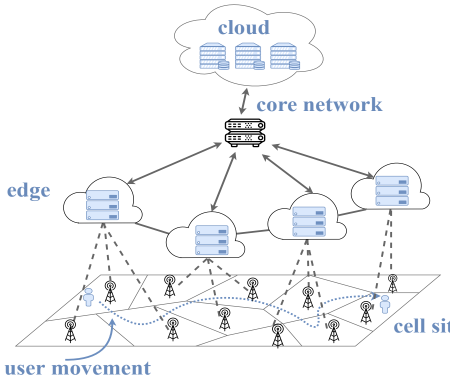

Fig. 1: An example of dynamic service placement when a user roams throughout the network in MEC.

In this paper, we advocate a novel framework of user-managed service placement, where a mobile user decides computation nodes (e.g., local device, edge server and remote cloud) to run its service, based on its observable information. The decision-making depends on the diversified application quality requirements, system dynamics and user preference. This framework is highly relevant to the scenarios when system-level service placement management is difficult. In practice, a MEC system would consist of multiple networks and edge/cloud services that are managed by different operators and it is generally difficult to achieve efficient coordinated system-level service management across multiple operators. Moreover, user-managed service placement can achieve better personalized service support tailored to user-specific preference when users' demands are highly diverse.

As illustrated in Fig. 1, during the roaming process of a user, when a mission-critical application needs to be processed, to strive for low perceived latency, the user prefers placing its service on the nearest edge server via 4G/5G/WiFi access networks. Thereafter, a latency-tolerant but computation-intensive application needs to be processed, the user prefers running its task on the remote cloud to avoid frequent service migration and shorten computing delay. Due to the unknown future information of user behavior and time-varying system

This work was supported by the National Science Foundation of China under Grants U1711265 and 61802449, the Program for Guangdong Introducing Innovative and Enterpreneurial Teams under Grant 2017ZT07X355, and the Guangdong Natural Science Funds under Grant 2018A030313032. (Corresponding author: Xu Chen.)

environment, there is no once-for-all solution for the long-term problem. Furthermore, missing accurate system information complicates the service placement problem. From the user-level perspective, collecting the system-level information, such as network condition and resource availability, needs extremely expensive communication cost.

To overcome the challenges of lacking both future system information and system-side information, we regard the user-managed service placement problem as a contextual multi-armed bandit (MAB) problem. To decouple the time dependency of the consecutive placement decisions, we construct a contextual feature vector to indicate the known user-side state information. Then we develop a Thompson sampling-based online learning algorithm to cope with the system dynamics and assist user to make adaptive service placement decision in an online manner. Furthermore, we provide a rigorous theoretical analysis for our online learning algorithm and quantify the estimated accuracy for the dynamic environment. Extensive evaluations demonstrate the superior performance of our proposed algorithm over existing schemes.

The rest of this paper is organized as follows. We first review related works in Section II. Then we present the system model and problem formulation in Section III. Next we propose offline and online service placement algorithm respectively in Section IV and Section V. Section VI presents the theoretical analysis of the proposed framework. Performance evaluation is carried out in Section VII. Section VIII concludes this paper.

II. Related Work

Mobile edge computing has received an increasing amount of attentions in recent years [3]. In particular, a central theme of many prior studies is service placement problem [6] [7]. A prime challenge of service placement in MEC is the user mobility management, exiting works for mobility management can be roughly classified into three categories.

One is based on the assumption of knowing accurate future information of the user mobility. For example, in [8] Hafid et al. make the trade-off between the execution overhead and transmission latency by using a mobility-base prediction scheme. Besides, in [9], the authors place the service by predicting the future cost incurred by data transmission, processing and service migration. The authors in [10] observe the correlation between bandwidth and geographical location based on historical data, and optimize wireless transmission times with the predicted user's location.

In general, capturing accurate future information of the user mobility is extremely challenging to the realistic environments. A few works are based on a milder assumption of user mobility that is following a Markovian process [11]–[13]. Specifically, [11] utilizes Markov chain to analyze the performance with the presence of user mobility. Both [12] and [13] try to design an optimal threshold decision policy, via modeling the service migration procedure as Markov Decision Process. Without requiring any future user mobility as a prior knowledge, some works such as [14] applies Lyapunov optimization to solve the service placement problem as a online queue stability control problem.

However, these works mainly focus on the system-wide service placement management optimization, where the scheduler knows complete system information. To deal with the challenge of lacking both future information and server-side uncertainty, [15] develops a learning-based service placement framework, using the multi-armed bandit (MAB) theory, which enables vehicles to learn the optimal neighboring vehicles to server its service. [16] utilizes MAB theory and Lyapunov optimization to minimize the joint delay with long-term energy consumption constraint in an online fashion. [17] solves the task assignment over multiple wireless devices by applying adversarial MAB to learn the unpredictable resource availability and channel condition. However, [17] only considers a static task and does not account for time-varying context constraint.

In this paper, we are standing at the user-side perspective to make adaptive service placement decision. Unlike the works [8]–[10] with the predicted scheme, nor the works [11]–[13] having a milder Markovian process assumption, we have no assumption of the user mobility. To address the uncertainty about the dynamic system environment, we propose an adaptive user-managed service placement (AUSP) framework which assists the user to select the optimal server to follow its mobility in an online learning manner. Aforementioned works such as [15]–[17] also solve the service placement problem in an online learning manner using MAB theory, but they did not consider the time dependency of sequential decisions when migrating the task from one server to another server. Along a different line, we formulate the service placement problem as a contextual MAB problem. By utilizing the contextual feature vector to describe user behavior information, we can decouple the time dependency. Furthermore, we incorporate user preference into the placement decision-making to achieve better personalized service performance.

III. System Model and Problem Formulation

As shown in Fig.1, we consider a MEC network containing a set $\mathcal{M} = \{1, 2, ..., M\}$ of base stations, each of which is endowed with an edge server. Thus, it can provide computing resource for user service in close proximity. We assume the edge servers can be interconnected with each other by kinds of network connections (e.g., via local-area network) [18]. In this way, a mobile user can continue to receive its service from the original edge server when it is no longer directly connected to the early base station. In this paper, we consider a user-managed service model for MEC. More specifically, the user can only observe its local information (e.g, the real-time location, the computation demand), but the system-wide information is not observable. To better characterize the user movement, we assume that the consecutive service placement decisions are made in a slotted structure and its timeline is discretized into time frames $t \in \mathcal{T} = \{0, 1, 2, ..., T\}$. At the beginning of each time slot t, the mobile user determines a proper computation node to run its task. During a (short) time slot, the user always stays in the original service coverage at

TABLE I: Parameter table of system model

Notation	Definition
$\mathbf{x}^t$	service placement strategy at time slot t
$\overline{\mathcal{M}}$	all computation nodes
c_i^t	the computation capacity of computation node i
λ^t	the amount of computation capacity required by user
d_{cp}^t	the computing delay for user
g_{i,l_t}^t	the communication delay when the service is placed on computation node i
d_{cm}^t	the communication delay for user
$f_{j,i}^t$	the switching cost of migrating service from source computation node j to destination computation node i
s^t	the switching cost for user

the beginning and network environment remains unchanged. Table I summarizes the key parameter notations in our paper. As discussed below, our model can allow the user's task can be time varying.

A. Service Placement Decision-making

To maintain satisfactory quality of experience, the service should be dynamically migrated across multiple servers to accommodate the user behavior, such as uncertain user mobility and service request. Here we design a binary vector $\mathbf{x}^t = [x_1^t, ..., x_M^t, x_r^t, x_d^t]$ to denote the dynamic decision making for service placement at time slot t. If the variable $x_i^t (i = 1, \ldots, M)$ is 1, it represents the task is executed on edge server node i, and $x_r = 1$ represents remote cloud server, $x_d = 1$ represents the user's local device. For ease of notation, we use $\overline{\mathcal{M}}$ to denote all computation nodes where $\overline{\mathcal{M}} = \mathcal{M} \cup \{r, d\}$). Note that at a given time slot, the user is served by one and only one computation node, we have the following decision-making constraints:

$$\sum_{i \in \overline{\mathcal{M}}} x_i^t = 1, \quad \forall t, \tag{1}$$

$$x_i^t \in \{0, 1\}, \quad \forall i, t. \tag{2}$$

Based on the above defined service placement decision, we are now readily to formulate the user-perceived latency and switching cost for service migration, which are associated with previous and current the service placement decisions.

B. User-perceived Latency

In the MEC framework, similar to many existing works such as [14], [19], the user-perceived latency is jointly determined by the computing delay and communication delay.

Computing delay: at each time slot t, a mobile task can be either locally executed on its device or offloaded to be executed on other external computation nodes (i.e., an edge server or remote cloud). Since the service request pattern is affected by various reasons, such as user mobility, some unexpected events (i.e., hot news video) etc., we denote the amount of computation demand as a time-varying variable λ^t (in terms of CPU cycles). Besides, we use c_i^t to denote the available computation capacity (i.e., CPU cycles per second) of computation node i for application processing at time slot t. Then, the computing delay can be expressed as

$$d_{cm}^t = \lambda^t / \sum_{i \in \overline{\mathcal{M}}} c_i^t x_i^t. \tag{3}$$

Taking video stream analytics as an instance [20], the amount of computation capacity required λ^t can be determined by to the input data size of the video and the corresponding computation intensity of the analytic task.

In general, a user may also purchase the cloud computation capacity (such as Amazon EC2) in advance. To account for this, we denote the user's remote cloud capacity as c_r^t.

Communication delay: communication delay is consisted of access latency and transferring latency. On the one hand, once the user offloads its service to any external computation node, the access latency to the local base station will be incurred. This access latency can be determined by the wireless environment and end device. For example, in commercial areas, concurrent accesses by large quantities of devices cause network congestion. On the other hand, if the service is not executed on the edge server attached to the local base station, the transferring latency between servers should be taken into account. Specifically, there exist two types of the transferring latency. One is edge-to-edge delay via cross-edge connection (LAN), which majorly depends on the hop distance along the shortest communication path. Another is cloud-to-edge delay via core network (wide area network), where relative low bandwidth is available for data transmission. Note that there is no communication delay when a user runs its application on the local device. For ease of notation, we use g_{i,l^t}^t to measure the communication delay for the user, where l^t indicates the the current connected base station, which depends on the real-time location of user. Obviously, the variable g_{i,l^t}^t contains the access latency to the base station l^t and transferring latency between the computation node i and l^t. Therefore, given the service placement decision x_i^t, the communication delay experienced by the user can be further expressed as:

$$d_{cm}^t = \sum_{i \in \overline{\mathcal{M}}} g_{i,l^t}^t x_i^t. \tag{4}$$

C. Switching Cost

Due to the user mobility, the connected base station can change frequently, which may trigger the service migration to maintain satisfactory service performance. However, the cross-edge service migration would incur additional operational overhead in return. More specifically, when transferring the service profile via core network, enormous usage of scarce and expensive WAN bandwidth would be caused. Besides, the cross-edge migration would generate the energy consumption of network devices such as routers and switches. Never the least, frequent switching the server would lead to a high probability handover failure and user service interruption latency (e.g., service data roaming and service virtual machine set-up delay). To avoid frequent service migration, we incorporate switching cost into our model. For ease of exposition, we use a general model $f^t_{j,i}$ to characterize the switching cost from the previous computation node j to the current computation node i. Thus the switching cost can be expressed as

$$s^t = \sum_{i\in\overline{\mathcal{M}}}\sum_{j\in\overline{\mathcal{M}}} f^t_{j,i} x^{t-1}_j x^t_i. \tag{5}$$

The physical meaning of (5) indicates that a switch cost from computation node j to node i incurs when the user chooses node j at time $t-1$ and then move to node i at time t (i.e., $x^{t-1}_j = x^t_i = 1$).

From the above description, it is obvious that aggressively minimizing user-perceived latency inevitably generates large switching cost. Then a natural question is how to balance such a delay-cost trade-off in a cost-efficient manner.

D. Objective: Total Service Cost Minimization

To optimize the conflicting objectives (i.e., perceived latency and switching cost) in a balanced manner, we assign different weights to the defined objectives and then minimize the weighted sum of them. Given a finite time horizon T, then the problem is formulated as follows:

$$\begin{aligned} \min \quad & \sum_{t=1}^{T} \omega^t_1 d^t_{cp} + \omega^t_2 d^t_{cm} + \omega^t_3 s^t \\ \text{s.t.} \quad & (1)-(2), \end{aligned} \tag{6}$$

where ω^t_1, ω^t_2 and ω^t_3 are the dynamic weights of computing delay, communication delay, and switching cost respectively, which can be set by the user as per its preference and the running application demands.

Unfortunately, optimally solving the problem in (6) requires complete information over the entire trip, including the user mobility, request pattern, available resource of all computation nodes, etc, which is extremely difficult to gain in advance. And it is impractical particularly for the user-managed scenario. Therefore, devising efficient service placement policy that operates on the fly in highly desirable. In this way, the policy can continuously adjust to accommodate system dynamics using only user's local observable information. To this end, we propose an online algorithm that can navigate the latency-cost trade-off efficiently without any future information.

IV. Offline Optimal Service Placement

We first consider the offline service placement problem by assuming complete future information for the whole process. The motivation is that we aim to provide the offline optimal algorithm as the performance benchmark for our online algorithm design later on.

Specifically, given a finite time horizon T, we can transfer the offline optimal service placement problem into an equivalent shortest path problem by carefully constructing the underlying service placement decision graph. As shown in Fig. 2, we construct a graph $G = (V, E)$ to represent all possible consecutive placement decisions within T time. Each edge weight on the graph between computation node i and j represents the perceived service cost (including computing

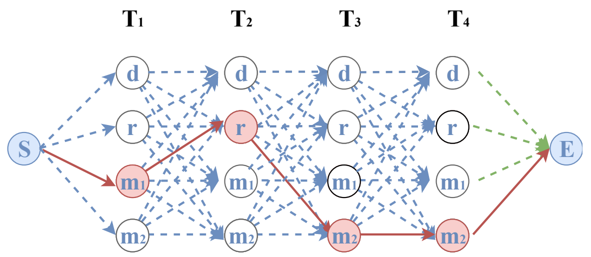

Fig. 2: Shortest-path problem transformation of offline service placement problem over T=4 time slots with two edge servers m_1, m_2, remote cloud r and local device d as the candidate computation nodes.

delay, communication delay, and switching cost if $i \neq j$) at the time slot t when service is placed on the computation node j from node i. Note that the source node S represents user's local device as the initial node and the destination node E is a dummy node to ensure that a single shortest path can be founded. And the (green) edge connecting to node E has zero cost. Clearly, due to the switching cost, there exists the time dependency of the sequential decisions over the long-run. Given all the information over T time slots are known including user's locations, the total weight of a path (e.g., the red line in Fig. 2) from source node S and destination node E can hence present user's accumulated service cost over the time horizon by choosing different computation nodes along its mobility trajectory. Naturally, the optimal placement strategy can be found by taking the shortest path from the source node S to destination node E. We can solve the above shortest-path problem based on the dynamic programing approach. The offline service placement policy algorithm is described explicitly in Algorithm 1. In the algorithm, the shortest-path (i.e., the optimal placement policy) for each time slot can be found by solving the Bellmans equation (i.e., line 13). Since our problem possess an optimal sub-structure property, the optimal placement policy at current time slot can be utilized again for finding the shortest-path in the next time slot. After iterating through the given time T, we can get the optimal offline placement policy by picking the minimum sum service cost (i.e., line 18), which is the optimal solution to the problem obviously. For searching the minimal service cost as the current optimal policy, the system needs to enumerate at most M^2 possible configuration. Thus, for the T-time period, the time-complexity of Algorithm 1 is $O(M^2T)$

V. Online Learning Based Service Placement

We now introduce the proposed online learning based AUSP framework. In the user-level perspective, the system-level network information and resource availability of servers are not accessible. To address this challenge, we first transform the information-constrained service placement problem into a contextual MAB problem. Then we propose an online learning-assisted approach by leveraging the Thompson sampling technique. Guided by this approach, the user can spontaneously learn the system dynamic from broad exploration, and determine current service placement adaptively to accommodate the system dynamics.

Algorithm 1 Offline Optimal Service Placement

```
1: Parameter Notation
2:     Variable $p$ and $q$ are the possible placement decisions in the current and previous time slot respectively.
3:     Vector $\pi_p$ and $\zeta_p$ are the optimal placement decisions from the beginning $t$ to the current and previous time slot when the decision is $p$ respectively.
4:     Variable $\phi_p$ and $\varphi_p$ are the sum service cost from the beginning $t$ to the current and previous time slot when the decision is $p$ respectively.
5:     Variable $U(p,t)$ is the received service cost after selecting computation node $p$ at time slot $t$
6: Initialization: Initialize the start time $t = 1$, the placement policy $\pi_p = \emptyset$ and the sum service cost $\phi_p = 0$
7: End initialization
8: for each time slot $\tau = t, t+1, ..., t+T-1$ do
9:     for all possible placement decision $p$ do
10:         Update the previous optimal placement decisions $\zeta_p = \pi_p$ and the sum service cost $\varphi_p = \phi_p$.
11:     end for
12:     for all possible placement decision $p$ do
13:         Determine the optimal previous decision according to current decision $p$, i.e., $q_{opt} = \arg\min_q \{\varphi_q + U(p,\tau)\}$.
14:         Update the current optimal placement policy $\pi_p(t, ..., \tau) = \zeta_{q_{opt}} \cup p$.
15:         Update the sum service cost $\phi_p = \varphi_{q_{opt}} + U(p,\tau)$.
16:     end for
17: end for
18: Pick the minimum sum service cost $p_{opt} = \arg\min_p \phi_p$ and set its corresponding policy $\pi_{p_{opt}}(t, ..., t+T-1)$ as the offline placement policy $\pi_{\text{off}}(t, ..., t+T-1)$.
19: Apply the placement policy $\pi_{\text{off}}(t, ..., t+T-1)$ in subsequent time slots $t, t+1, ..., t+T-1$
```

A. Problem Transformation via Contextual MAB

During each time frame, the mobile user is presented with the choice of taking one out of feasible computation nodes at each time slot. Before the decision making, only user-side state information is observable. After placing the service on a selected node, the user will receive a service cost of that computation node at the end of time slot, and the costs of other unselected computation nodes are unknown. To better learn the network uncertainty and resource availability, we characterize each computation node i using a feature vector $\mu_i(t) = [1/c_i^t, g_{i,1}^t, g_{i,2}^t, ..., g_{i,M}^t, f_{i,1}^t, f_{i,2}^t, ..., f_{i,M+2}^t] \in \mathbb{R}^{(2M+3)}$, which are associated with the aforementioned defined costs (i.e., computing delay, communication delay, and service switching cost). Correspondingly, to describe the current user-side state information, we use a contextual feature vector $b(t) = [\omega_1^t \lambda^t, \omega_2^t \mathbb{G}^t, \omega_3^t \mathbb{F}^t] \in \mathbb{R}^{(2M+3)}$. More explicitly, entry λ^t indicates the current service demand, the binary vector $\mathbb{G}^t$ indicates the real-time location of user, where one and only one entry is 1 at each time slot (i.e., the base station that the user is associated with at time slot t). To decouple the time dependency of the consecutive placement decisions, similar to vector $\mathbb{G}^t$, we design a binary vector $\mathbb{F}^t$ to record the computation node at previous time slot $t-1$. By doing so, we can use the context of previous time slot to assist the computation of the switching cost. Note that, since the state information of local device is easily accessed, if the user directly uses its local device to run the application, the computing delay can be accurately estimated.

After such representation transformation, when the user needs to determine which computation node is currently optimal, it can utilize the contextual feature vector $b(t)$ and historical observations $\mathcal{H}_{t-1} = \{a(\tau), r_{a(\tau)}(t), b(t), \tau = 1, ..., t-1\}$ to analyze each computation node, where $a(t)$ denotes the selected computation node, and $r_{a(t)}(t)$ denotes the corresponding received cost. In this sense, we can transform the online service placement problem into a contextual multi-armed bandit problem (MAB) by regarding the computation nodes as arms. And the received cost (i.e., the sum of computing delay in (3), communication delay in (4) and switching cost in (5)) can be obtained by the product of contextual vector $b(t)$ and the chosen feature vector $\mu_i(t)$, i.e., $r_i(t) = b(t)^\top \mu_i(t)$. Supposing the received cost $r_i(t)$ of computation node i at end of time slot is generated from a unknown distribution with the mean $b(t)^\top \mu_i$, that is,

$$\mathbb{E}[r_i(t)|b(t)] = b(t)^\top \mu_i,$$

where μ_i is the fixed but unknown expectations of the corresponding feature parameters $\mu_i(t)$ of computation node i. Let a_{op}^t be the computation node chosen by the online optimal policy, which knows the system-side information at time t, i.e., $a_{op}^t = \arg\min_i b(t)^\top \mu_i$. And the regret Δ_i^t at time slot t is defined as the difference between the mean costs of the optimal computation node and of computation node i under the identical contextual feature, i.e., $\Delta_i^t = b(t)^\top \mu_i - b(t)^\top \mu_{a_{op}^t}$. It measures the efficiency of placement decision making using the online learned network information. Thus for a long-term T period, the total regret can be expressed as following:

$$\mathcal{R}(T) = \sum_{t=1}^{T} \Delta_i^t.$$

Besides, for ease of exposition, we assume that the deviation of the received cost from its expectation $\eta_i(t) = r_i(t) - b(t)^T \mu_i$ is conditionally R-sub-Gaussian [21] for a constraint $R \geq 0$, i.e.,

$$\mathbb{E}[e^{\lambda \eta_i(t)}|\mathcal{H}_{t-1}] \leq e^{(\frac{\lambda^2 R^2}{2})}, \forall \lambda \in \mathbb{R}.$$

This assumption guarantees that the received cost satisfied the following inequality at any time slot:

$$b(t)^\top \mu_i - R \leq r_i(t) \leq b(t)^\top \mu_i + R, \forall t \in T,$$

which means the received cost $r_i(t)$ would not deviate its expectation $b(t)^\top \mu_i$ infinitely. This assumption generally holds in practice. We will also assume that $||\mu_i|| \leq c$, $||b(t)|| \leq c$, $||\Delta_i^t|| \leq c$ for all i, t (all the norms are all $\mathcal{L}_2$-norms) for some upper-bound constant c.

B. Adaptive Service Placement Learning

In this subsection, we design efficient contextual Thompson sampling learning scheme for adaptive service placement. The key idea is to properly construct a distributional likelihood function to estimate user's performance of choosing different actions over time [22] [23]. Due to its usefulness, Thompson sampling has been applied for algorithm design in other applications such as Internet advertising [24].

According to the aforementioned contextual MAB transformation of the our problem, the cost $r_i(t)$ for each computation node i can be generated as the product of the current contextual feature vector $b(t)$ and the actual but unknown feature vector μ_i. Over time, the user can gather abundant information about the underlying relation between the feature vector and service cost of chosen computation node, so that it can estimate which computation node is likely to give the minimum cost by looking at the feature vector with some certainty. Thus we let $\hat{\mu}_i(t)$ represent the estimate of the feature vector of computation node i. The variance of the empirical mean for the contextual information is related to inverse of $B_i(t)$, which can be updated based on ridge regression [25] as follows:

$$\hat{\mu}_i(t) = B_i(t)^{-1}\Big(\sum_{\tau=1}^{t-1}\mathbf{1}_{\{a(\tau)=i\}}b(\tau)r_i(\tau)\Big), \tag{7}$$

$$B_i(t) = \mathbf{I} + \sum_{\tau=1}^{t-1} b(\tau)b(\tau)^\top \mathbf{1}_{\{a(\tau)=i\}}, \tag{8}$$

where $B_i(t)$ indicates the observed previously contextual information of the computation node i. $\mathbf{I}$ is a $2M+3$ dimensional identity matrix.

We also define $s_{t,i}$ as the standard deviation of the estimated cost $b(t)^\top\hat{\mu}_i(t)$ for computation node i:

$$s_{t,i} = \sqrt{b(t)^\top B_i(t)^{-1}b(t)},$$

and $vs_{t,i}$ is the standard deviation of the sampling cost $b(t)^\top\tilde{\mu}_i(t)$, the value of v will be defined later.

According to Bayes' rule, i.e.,

$$\Pr(b(t)^\top\tilde{\mu}_i|r_i(t)) \propto \Pr(r_i(t)|b(t)^\top\tilde{\mu}_i)\Pr(b(t)^\top\tilde{\mu}_i),$$

if the prior for received cost $b(t)^\top\mu_i$ of computation node i at time slot t is assumed as $\mathcal{N}\big(b(t)^\top\hat{\mu}_i(t), v^2 b(t)^\top B_i^{-1}(t)b(t)\big)$, and the likelihood of the cost $r_i(t)$ is given by the probability density function of Gaussian distribution $\mathcal{N}\big(b(t)^\top\mu_i, v^2\big)$, where $v = R\sqrt{\frac{24}{\epsilon}(2M+3)\ln\frac{1}{\delta}}$ is a constant associated with $\epsilon \in (0,1)$, which is a control parameter in our online learning algorithm. Then the posterior distribution of $b(t+1)^\top\mu_i$ at time $t+1$ is easy to be calculated as $\mathcal{N}\big(b(t{+}1)^\top\hat{\mu}_i, v^2 b(t{+}1)^\top B_i(t{+}1)^{-1}b(t{+}1)\big)$. Hence we will use Gaussian likelihood function to abstract user's historical service cost and contextual information and estimates its performance of choosing a computation node.

Based on this, in our online learning algorithm, at each time step t, the user will first get current contextual vector, and then estimate the cost $\theta_i(t)$ of each computation node from its posterior probability $\mathcal{N}\big(b(t)^\top\hat{\mu}_i, v^2 b(t)^\top B_i^{-1}b(t)\big)$ through the history outcomes and user-side state information. Next, the user chooses the estimated optimal computation node i with the minimum cost $\theta_i(t)$ to serve its task. At the end of time slot t, the user utilizes received cost $r_i(t)$ to update the feature vector of selected computation node and variance of the empirical mean for corresponding contextual information. Hence, guided by the increasing exploration probing, the estimation accuracy for each computation node can be further improved. We summarize the online learning algorithm in Algorithm 2. Numerical results demonstrate that our contextual Thompson sampling based online service placement algorithm can out-perform over existing Linear Upper Confidence Bound (LinUCB) based MAB learning algorithms with up-to 25% performance improvement.

Algorithm 2 Adaptive User-managed Service Placement

```
1: Initialization: Initialize the cumulative context of each computation node $i: B_i = \mathbf{I}$ (i.e., $B_i$ is initiated as $2M+3$ dimensional identity matrix), the estimated feature vector of computation node $i: \hat{\mu}_i = \mathbf{0}$ (i.e., $2M+3$ dimensional zero vector), the cumulative contextual service cost of computation node $i: f_i = \mathbf{0}, i = 1, 2, ..., M$
2: End initialization
3: for each time slot $t = 1, 2, ..., T$ do
4:     Sample the aggregated cost $\theta_i(t)$ independently for each computation node $i$ from the posterior distribution $\mathcal{N}\big(b(t)^\top\hat{\mu}_i, v^2 b(t)^\top B_i^{-1}b(t)\big)$
5:     Select the computation node $a(t) = \arg\min_i \theta_i(t)$ and receive a actual aggregated cost $r_t$
6:     Update $B_{a(t)} = B_{a(t)} + b(t)b(t)^\top, f_{a(t)} = f_{a(t)} + b(t)r_t$, the estimated feature vector of computation node $a(t)$ is updated by $\hat{\mu}_{a(t)} = B_{a(t)}^{-1}f_{a(t)}$
7: end for
```

VI. Regret Analysis

In this section, we analyze the upper bound for the total regret of our online learning based AUSP framework theoretically. First, we divide all computation nodes into two groups at any time slot: saturated and unsaturated nodes, which depend on whether the standard deviation of the estimation of a computation node is smaller or larger compared to the standard deviation of the optimal computation node. Then, we provide upper bound for selecting unsaturated and saturated nodes respectively. Finally, we construct a super-martingale process to bound the regret with high probability by summing the upper bound of the saturated and unsaturated nodes.

For ease of exposition, we introduce some key definitions in the following.

During each sampling procedure in Algorithm 2, we can bound the deviation of the estimated service cost $b(t)^\top\mu_i$ and sampling cost θ_i with a certain probability, which is presented as following:

Definition 1: Define $E^{\mu}(t)$ and $E^{\theta}(t)$ as the events of estimated cost $b_i(t)^T\mu(t)$ and sampling cost $\theta_i(t)$ is concentrated to their means respectively. Thus we have

$$E^{\mu}(t): |b(t)^{\top}\hat{\mu}_i(t) - b(t)^{\top}\mu_i| \leq l(T)s_{t,i}, \forall i,$$
$$E^{\theta}(t): |\theta_i(t) - b(t)^{\top}\hat{\mu}_i(t)| \leq (g(T) - l(T))s_{t,i}, \forall i,$$

where $l(T) = R\sqrt{d\ln(T^3)\ln(\frac{1}{\delta})}+c$, $g(T) = \sqrt{4\ln(MT)}v+l(T)$.

***Lemma** 1:* For all t, $0 < \delta < 1$, the probability of event $E^{\mu}(t)$ satisfies $\mathrm{P}(E^{\mu}(t)) \geq 1 - \frac{\delta}{T^2}$. And for all possible filtrations $\mathcal{F}_{t-1}$, the probability of event $E^{\theta}(t)$ satisfies $\mathrm{P}(E^{\theta}(t)|\mathcal{F}_{t-1}) \geq 1 - \frac{1}{T^2}$.
The detailed proof is given in the technique report [26]. To provide upper regret bound of the unsaturated and saturated nodes, we give the explicit definitions of them:

***Definition** 2:* Define unsaturated nodes as those with $g(T)s_{t,i} \geq l(T)s_{t,a^t_{op}}$, saturated nodes as those with $g(T)s_{t,i} < l(T)s_{t,a^t_{op}}$,

A. Unsaturated Nodes Bound

For unsaturated computation nodes, due to the insufficient exploration about the system dynamics, the estimated cost of computation nodes is not accurate. We use $s_{t,i}$ and $s_{t,a^t_{op}}$ to derive the upper regret bound of unsaturated computation nodes at any time slot: $\Delta^t_i \leq g(T)(s_{t,i} + s_{t,a^t_{op}})$. When the computation node i is selected, the standard deviation of it would decrease according to (7) and (8).

***Lemma** 2:* If an unsaturated computation node i is chosen, the regret is at most $\frac{g(T)+l(T)}{l(T)}g(T)s_{t,i}$.
The detailed proof is given in the technique report [26].

B. Saturated Nodes Bound

For saturated computation nodes, since the estimation of the mean is quite accurate, the user can distinguish between the optimal computation node and the others. We utilize the observation of the history outcomes to show the probability of choosing saturated nodes is bounded by a function of the probability of choosing optimal computation node.

***Lemma** 3:* For all possible filtrations $\mathcal{F}_{t-1}$, if event $E^{\mu}(t)$ is true, then the

$$\mathrm{P}\big(a(t) \in C(t)|\mathcal{F}_{t-1}\big) \leq \frac{1}{p}\mathrm{P}(a(t) = a^t_{op}|\mathcal{F}_{t-1}) + \frac{1}{pT^2},$$

where $p = \frac{1}{4e^{c^2/2}\sqrt{\pi T^{\epsilon}}}$ and $C(t)$ is the set of saturated computation nodes.
The detailed proof is given in the technique report [26].

C. Construction of Super-martingale Process

In this subsection, we construct super-martingale difference process adapted to the history outcomes $\mathcal{F}_{t-1}$, which guarantees that our regret bound with high probability.

***Lemma** 4:* Let

$$X_t = \frac{\text{regret}'(t)}{g(T)} - \frac{1}{p}I(a(t) = a^t_{op})s_{t,a^t_{op}} - s_{t,a(t)} - \frac{2c}{pT^2}$$

where $\text{regret}'(t) = \text{regret}(t) \cdot I\big(E^{\mu}(t)\big)$ and let

$$Y_t = \sum_{\tau=1}^{t} X_{\tau},$$

then $(Y_t; t = 0, ..., T)$ is a super-martingale process with respect to filtration $\mathcal{F}_t$.
The detailed proof is given in the technique report []. If we can derive upper bound of the the last four items of X_t, then the total regret will not exceed a certain value with high probability.

Now, we are ready to get our regret bound using the lemmas and definitions above.

***Theorem** 1:* The total regret over the finite time T is bounded by $O\Big(M\sqrt{\frac{MT^{1+\epsilon}\ln(MT)}{\epsilon}}\big(\ln(T)\ln(\frac{1}{\delta})\big)\Big)$ under our online algorithm with probability $1 - \delta$, for any $0 < \epsilon < 1$ and $0 < \delta < 1$.
The detailed proof is given in the technique report [26]. Based on the Theorem 1, our algorithm can achieve a sub-linear deviation of the optimal solution, i.e., the regrets increases sub-linearly as time increases.

VII. Performance Evaluation

In this section, we conduct numerical studies to evaluate the proposed algorithm over a long run and to verify the derived theoretical bound for the total regret.

A. Simulation Setup

We simulate a 2km $\times$ 2km area with 25 edge servers. For simplicity, we assume edge servers are evenly deployed in this regular grid network. And each edge server can be accessed by multiple connection type, such as AP or small-cell base station in the service scope. Each edge server is equipped with multiple CPU cores and the total computation capacity for edges is uniformly distributed in $[10, 15]$ GHz, due to the different specification and type of server. Besides, we assume the computation capacity of mobile device is uniformly distributed in $[1, 1.5]$ GHz, and the purchased computation capacity of the cloud is 100 GHz. The user mobility is generated according to the widely-used ONE simulator [14]. According to [27], we roughly classify the user requests into three categories: heavy (10000 cycles/bit), medium (2500 cycles/bit) and light (250 cycles/bit) computational applications. To meet the diverse user's demand, we allow the mobile user to change the request service arbitrarily. In our simulation, for simplicity, we set a periodic interval for changing service types. The total switching cost (i.e., data migration delay, server reconfiguration cost, and energy consumption) between the servers can be affected by the network topology (e.g., the transmission path). For simplicity, we formulate it as a function of hop distant, where one hop for switching cost among the edge servers is uniformly distributed in $[0.1, 0.2]$ cost, and one hop between edges and cloud is uniformly distributed in $[0.5, 1]$ cost. Besides, the communication latency between user and cloud is normally in $[100, 300]$ ms. The access latency to connected base station or AP is distributed in $[50, 100]$ ms.

B. Performance Benchmark

To evaluate the performance of our algorithm, we compare it with the aforementioned offline strategy and following three online strategies.

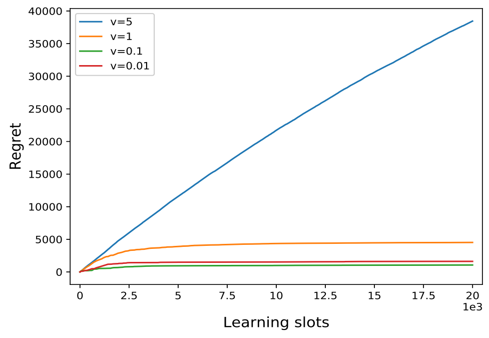

Fig. 3: Total regret with different v

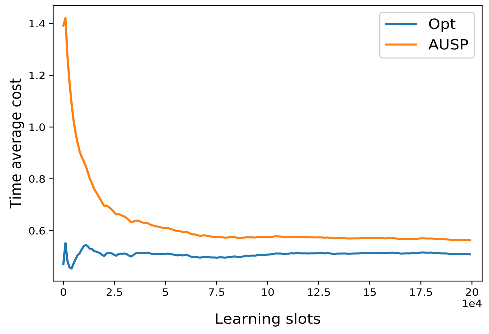

Fig. 4: Approaching to the offline optimum

1) **Serving on the cloud (SC):** without considering the kind of application, the mobile user always put the service on the remote cloud to run.
2) **Following user mobility (FUM):** without considering the switching cost, the mobile user would always select the nearest edge server to process its request at each time slot.
3) **LinUCB:** the mobile user would select the computation node based on the current minimum upper confident bound of the cost according to the classic MAB theory. The LinUCB is widely used in literature such as [25].

C. *Total Regret Analysis*

To analyze how the key elements (i.e., ϵ, δ) influence the total regret of our AUSP algorithm, for simplicity, we compare total regret with different values of v, where $v = R\sqrt{\frac{24}{\epsilon}(2M+3)\ln\frac{1}{\delta}}$ is associated with the standard deviation of the estimated computation node. As shown in Fig. 3, due to the ϵ trade-off in the theoretical bound, i.e., $O\Big(M\sqrt{\frac{MT^{1+\epsilon}\ln(MT)}{\epsilon}}\ln(T)\ln(\frac{1}{\delta})\Big)$, a smaller v may lead to large ϵ, and then generate a larger total regret, such as curves $v = 0.1$ and $v = 0.01$. However, when v continues to increase, the total regret would also increase because of the large value $\ln\frac{1}{\delta}$ and $\sqrt{\frac{1}{\epsilon}}$. Beside, as time increases, the regret curves expose a sub-linear increasing trend, which is consistent with the theoretical upper bound for total regret in Theorem 1.

D. *AUSP vs. performance benchmark*

We first trace the average costs of AUSP and offline optimal algorithm at each time slot. As shown in Fig.4, our

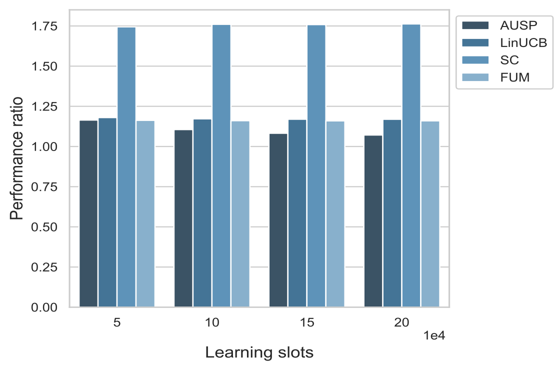

Fig. 5: The performance radio of offline optimum within a certain period

AUSP algorithm gets a decreasing average cost with the time increasing, and further it is gradually approaching to the offline optimum. Obviously, the descending trend demonstrates that our AUSP algorithm can efficiently learn the underlying relations between the cost and the computation node in the unknown system dynamics. Fig.5 depicts the performance ratio comparison between AUSP and other online service placement algorithms. It is easy to find that our algorithm achieves the best service performance among four online algorithms. At first, AUSP has almost the same performance as LinUCB and FUM, since the AUSP algorithm does not have a prior knowledge about the MEC system, so that it randomly selects computation node to explore the dynamic system environment. However, as time goes by, due to a broad exploration, AUSP acquires richer knowledge to distinguish the optimal computation node. As shown in Fig.5, although the performance ratios between AUSP and the LinUCB are gradually decreasing to their minimum, our AUSP algorithm can achieve better service performance than LinUCB. It indicates that AUSP has a powerful learning ability. However, few performance changes in SC and FUM algorithm.

E. *Different switching costs*

To analyze how the switching cost influence the service performance, we design five weights of the switching cost (i.e., ω_3) to indicate the different user preference while the others (i.e., ω_1, ω_2) are set as one. As shown in Fig.6, it is easy to find that with the weight of switching cost increasing, in other word, the user cares more about the service migration, our the service performance of our AUSP algorithm becomes worse. This is because that the service migration occurs only when the perceived latency would be dramatically reduced by choosing another computation node. For example, the user's service demand changes from the computation-intensive but latency-tolerant task to the light computational task. Due to the large switching cost, the user prefer running its service on the same computation node. Rather than large weight of switching cost, the small one provides more opportunity for AUSP to try new computation nodes. Hence, the user can learn a more complete knowledge about the dynamic system environment, and further gain a better service performance.

Besides, compared with other online algorithms, our AUSP

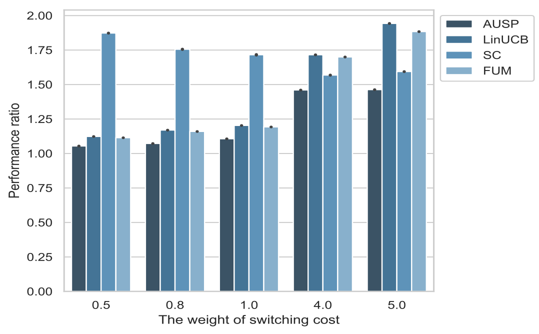

Fig. 6: Multiple algorithms under different switching cost

algorithm also achieves the best performance in any above situation. When the weight of the switching cost is small, i.e., the user cares more about the perceived latency, our AUSP have around 6% performance improvement compared with the second best performing scheme. As the weight of the switching cost increasing, a more careful trade-off between service latency and switching cost is needed, and the performance gain of our algorithm is higher with around 25% improvement over LinUCB scheme and 10% over SC scheme.

VIII. Conclusion

In this paper, we study the personalized service performance optimization problem in MEC scenario. We propose an adaptive user-managed service placement mechanism to jointly optimize the perceived-latency and service migration cost, weighted by user preferences. To deal with the unavailable future system information and unknown system dynamics, we formulate the dynamic service placement as a contextual MAB problem. To learn the underlying relation between the available computation node and its corresponding cost, we propose a Thompson-sampling based online learning algorithm to explore the dynamic system environment. Furthermore, we study the properties of our AUSP algorithm, and find that it achieves a sub-linear deviation of the online optimum with high probability. Through extensive simulations, the results demonstrate the superior performance of our algorithm.

References

[1] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. Heinzelman, "Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture," in *Proc. IEEE ISCC*, 2012, pp. 000 059–000 066.

[2] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Neal *et al.*, "Mobile-edge computing introductory technical white paper," *White Paper, Mobile-edge Computing (MEC) industry initiative*, 2014.

[3] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computinga key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[4] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[5] X. Chen, Q. Shi, L. Yang, and J. Xu, "Thriftyedge: Resource-efficient edge computing for intelligent iot applications," *IEEE Network*, vol. 32, no. 1, pp. 61–65, 2018.

[6] 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.

[7] H.-S. Lee and J.-W. Lee, "Task offloading in heterogeneous mobile cloud computing: Modeling, analysis, and cloudlet deployment," *IEEE Access*, vol. 6, pp. 14 908–14 925, 2018.

[8] A. Nadembega, A. S. Hafid, and R. Brisebois, "Mobility prediction model-based service migration procedure for follow me cloud to support qos and qoe," in *IEEE International Conference on Communications*, 2016, pp. 1–6.

[9] S. Wang, R. Urgaonkar, K. Chan, T. He, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," in *IEEE International Conference on Communications*, 2015, pp. 5504–5510.

[10] W. Zhang, R. Fan, Y. Wen, and F. Liu, "Energy-efficient mobile video streaming: A location-aware approach," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 9, no. 1, 2017.

[11] T. Taleb and A. Ksentini, "An analytical model for follow me cloud," in *IEEE Global Communications Conference (GLOBECOM)*, 2013, pp. 1291–1296.

[12] A. Ksentini, T. Taleb, and M. Chen, "A markov decision process-based service migration procedure for follow me cloud," in *Proc. IEEE ICC*, 2014, pp. 1350–1354.

[13] S. Wang, R. Urgaonkar, T. He, M. Zafer, K. Chan, and K. K. Leung, "Mobility-induced service migration in mobile micro-clouds," in *Military Communications Conference*, pp. 835–840.

[14] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2333–2345, 2018.

[15] Y. Sun, X. Guo, S. Zhou, Z. Jiang, X. Liu, and Z. Niu, "Learning-based task offloading for vehicular cloud computing systems," *arXiv preprint arXiv:1804.00785*, 2018.

[16] Y. Sun, S. Zhou, and J. Xu, "Emm: Energy-aware mobility management for mobile edge computing in ultra dense networks," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2637–2646, 2017.

[17] Y.-H. Kao, K. Wright, B. Krishnamachari, and F. Bai, "Online learning for wireless distributed computing," *arXiv preprint arXiv:1611.02830*, 2016.

[18] N. Chen, Y. Yang, T. Zhang, M.-T. Zhou, X. Luo, and J. K. Zao, "Fog as a service technology," *IEEE Communications Magazine*, vol. 56, no. 11, pp. 95–101, 2018.

[19] X. Chen, L. Pu, L. Gao, W. Wu, and D. Wu, "Exploiting massive d2d collaboration for energy-efficient mobile edge computing," *IEEE Wireless Communications*, vol. 24, no. 4, pp. 64–71, 2017.

[20] Y. Sun, S. Zhou, and J. Xu, "EMM: Energy-Aware Mobility Management for Mobile Edge Computing in Ultra Dense Networks," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2637–2646, 2017.

[21] S. Agrawal and N. Goyal, "Thompson sampling for contextual bandits with linear payoffs," in *International Conference on Machine Learning*, 2013, pp. 127–135.

[22] O. Chapelle and L. Li, "An empirical evaluation of thompson sampling," in *Advances in neural information processing systems*, 2011, pp. 2249–2257.

[23] D. J. Russo, B. Van Roy, A. Kazerouni, I. Osband, Z. Wen *et al.*, "A tutorial on thompson sampling," *Foundations and Trends in Machine Learning*, vol. 11, no. 1, pp. 1–96, 2018.

[24] D. Agarwal, B. Long, J. Traupman, D. Xin, and L. Zhang, "Laser: A scalable response prediction platform for online advertising," in *Proceedings of the 7th ACM international conference on Web search and data mining*. ACM, 2014, pp. 173–182.

[25] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 661–670.

[26] T. Ouyang, R. Li, X. Chen, Z. Zhou, and X. Tang, "Adaptive User-managed Service Placement for Mobile Edge Computing: An Online Learning Approach," Tech. Rep., 2019. [Online]. Available: https://bit.ly/2siB1MX

[27] J. Kwak, Y. Kim, J. Lee, and S. Chong, "Dream: Dynamic resource and task allocation for energy minimization in mobile cloud systems," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 12, pp. 2510–2523, 2015.