Computer Networks 246 (2024) 110395

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Online optimal service caching for multi-access edge computing: A constrained Multi-Armed Bandit optimization approach

Weibo Chu [a,*], Xiaoyan Zhang [a], Xinming Jia [a], John C.S. Lui [b], Zhiyong Wang [b]

[a] Northwestern Polytechnical University, Xi'an, China
[b] The Chinese University of Hong Kong, Hong Kong

ARTICLE INFO

Keywords:
Multi-access edge computing
Service selection
Service caching
Constrained multi-armed bandit
Online algorithm

ABSTRACT

In order to fully exploit the power of Multi-Access Edge Computing, services need to be cached at the network edge in an adaptive and responsive way to accommodate the high system dynamics and uncertainty. In this paper, we study the online service caching problem in MEC, with the goal to minimize users' perceived latency while at the same time, ensure the rate of tasks processed by the edge server is no less than a preset threshold. We model the problem with a Constrained stochastic Multi-Armed Bandit formulation, and propose a simple yet effective online caching algorithm called Constrained Confidence Bound (CCB). CCB achieves $O(\sqrt{T \ln T})$ bounds on both regret and violation of the constraint, and is able to achieve a good balance between them. We further consider the scenario when there is cost (i.e., delay) due to service switches, and propose two service switch-aware caching algorithms — Explore-First (EF) and Successive Elimination-based (SE) caching, together with a novel sampling scheme. We prove that EF achieves $O(T^{\frac{2}{3}}(\ln T)^{\frac{1}{3}})$ bound on regret and violation, whereas SE achieves $O(\sqrt{T \ln T})$ and converges significantly faster. Lastly, we conduct extensive simulations to evaluate our algorithms and results demonstrate their superior performance over baselines.

1. Introduction

It has been shown repeatedly that innovative network architectures and key technologies enable and empower crucial applications. It is our consensus today that the reverse is also true as applications are shaping/changing the network architectures and technologies. Take for example emerging applications such as autonomous driving [1], AR/VR [2], and networked gaming. Being resource-hungry and delay-sensitive, these applications impose a stringent requirement on both computing and networking capacity, but which cannot be met solely by the existing cloud systems due to the long propagation delay and unstable network connections. In this context, a new network computing paradigm called multi-access edge computing (MEC [3, 4]), has been put forward. The key feature of MEC is that services are hosted at various type of edge nodes endowed with computing/storage/communication capacities, so that low-latency access to services are possible.

With MEC, users can offload their tasks to the edge nodes (a.k.a MEC servers) for high energy-efficiency, fast responses and enhanced security/privacy protection [5]. However, as compared with cloud infrastructure (e.g., data centers) which can virtually host all the services with abundant resources, MEC servers are often resource-constrained and can only accommodate a limited number of services. For example, network operators usually implement cloudlet based mobile computing using a computing server with small resources or a cluster with medium resources. This raises the *service placement problem* as when and where to host the services at the edge nodes. Apparently, the performance of MEC varies significantly depending on service placement.

The service placement problem (SPP), sometimes also referred to as *service caching* [6], has attracted a lot of research in the past few years, and various algorithms [7–10], i.e., exact/approximate, static/dynamic, centralized/decentralized, have been proposed. Yet designing an optimal policy for service caching remains a challenge due to the high heterogeneity and dynamics of both the system and workload. The problem becomes even more challenging when we consider it in an online setting where the caching decisions have to be made as system operates, but without a priori knowledge of user-generated workload and network condition. In fact, these critical information such as task offloading delays are stochastic and unobservable unless the services are cached. Moreover, from a practical point of view, it is expected that the online caching algorithms can provide us provable performance guarantee.

* Corresponding author.
E-mail addresses: wbchu@nwpu.edu.cn (W. Chu), zhangxy301@mail.nwpu.edu.cn (X. Zhang), jxm12f@mail.nwpu.edu.cn (X. Jia), cslui@cse.cuhk.edu.hk (J.C.S. Lui), zywang21@cse.cuhk.edu.hk (Z. Wang).

https://doi.org/10.1016/j.comnet.2024.110395
Received 19 September 2023; Received in revised form 29 February 2024; Accepted 2 April 2024
Available online 16 April 2024

In this paper, we study the online service caching problem for a generic multi-access edge computing system, with the goal to minimize users' perceived latency. We adopt a Multi-Armed Bandits (MAB) [11, 12] optimization framework, which is a simple but very powerful tool for online learning that allows a decision maker to estimate parameters and perform optimization overtime under uncertainty. Our model differs substantially from traditional MAB in that: (1) in addition to cumulative reward (here, cumulative latency) that measured by *regret*, we also consider a QoS constraint from MEC service provider that the rate of tasks processed by the edge server is no less than a preset threshold. This formulation captures a key feature in many real-world optimization tasks that network operators seek some other performance guarantees (or constraints, measured by *violation*) in addition to the optimal solution; (2) Besides bandit feedback, our model also involves other type of feedback, i.e., full feedback, complementary feedback, and even state-dependent feedback. This significantly extends the existing MAB framework, while at the same time, poses challenges to design efficient algorithms and characterize their performance.

More specifically, in this paper we formulate the online service caching problem as a Constrained Multi-Armed Bandits (CMAB) optimization problem. To balance between minimizing the aggregate delay and satisfying the QoS constraint, we first propose a simple yet effective algorithm, called Constrained Confidence Bound (CCB), which is a probabilistic algorithm that achieves *sub-linear* bounds on both regret and violation with high probability, i.e., at time T with a probability $1-\delta$, we have $Reg(T) = O((K-L)\sqrt{KT\ln\frac{2KT}{\delta}})$ and $Vio(T) = O(K\sqrt{KT\ln\frac{2KT}{\delta}})$, where K is the number of services in system and L is the server capacity. Our simulation results suggest that CCB serves as a good candidate to online caching if our goal is more about the long-term performance.

We then consider the scenario when there is service switching cost, i.e., delays for fetching a new service from cloud and then instantiate it at the MEC server. We show that with service switching cost, the performance of CCB degrades due to the inaccurate estimate of parameters and frequent service switches. To this end, we propose two new service switch-aware online caching algorithms — Explore-First (EF) and Successive Elimination-based (SE) caching, both of which can effectively deal with service switches during the caching process. A common ingredient of the two algorithms is a novel sampling scheme that we propose, which is designed to sample services uniformly at the same rate with small sampling cost, while at the same time, which can avoid too many service switches. We show that both algorithms are able to identify the optimal solution efficiently, with the difference that a large number of samples are required by EF whereas the sampling cost can be significantly decreased by SE through gradually eliminating non-optimal services with high confidence during the learning process. This allows SE to converge much faster and also more computationally cost-effective. Furthermore, we prove that both algorithms simultaneously achieve *sub-linear* bounds on regret and violation.

The remainder of this paper is organized as follows. In Section 2 we describe system model and the CMAB formulation for the online service caching problem. Section 3 presents CCB, its performance analysis and simulation results. In Section 4 we extend the model to problems with service switching cost, and elaborate the EF and SE algorithm and their performance evaluation. Section 5 gives related work. We discuss some future research directions and conclude the paper in Section 6.

2. System model and problem formulation

2.1. System model

We consider a Multi-Access Edge Computing system as shown in Fig. 1, which consists of multiple user equipments (UEs), an MEC server and the cloud. To make our model generic, this MEC server can be micro data center, an edge cloud or a computing server with high capacity. Resource-hungry and delay-sensitive tasks are generated from UEs and routed to the MEC server. If the required services are hosted, the corresponding tasks are performed locally and superior user experience can be achieved. Otherwise, they are directed to the cloud for remote execution at the cost of large delay. Throughout the paper, we assume that the cloud hosts all the services; however, due to resource (CPU, memory, etc.) constraints, the MEC server can only accommodate a limited number of services simultaneously, which raises the *service caching problem*, i.e., how to properly select a subset of services from a large service pool and instantiate them at the MEC server so as to optimize the system performance.

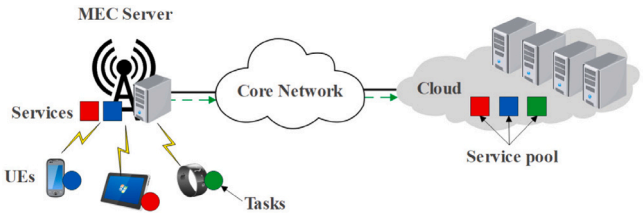


Fig. 1. A Multi-Access Edge Computing system.

Due to its combinatorial nature, service caching problem is generally hard to tackle even when all system parameters are available, i.e., the problem is often modeled as a 0–1 programming problem. It is much more challenging when we consider it in an online setting where one has to make caching decisions as system operates, but without any priori knowledge of the stochastic user-generated tasks and network condition. Simply caching the most popular services does not always guarantee low latency when, i.e., the computation workload of the tasks are high. Likewise, caching the services with the best network condition is also sub-optimal when the popularity of the service is low. In fact, system parameters such as MEC-offloading delay and service switching cost are unobservable unless the service is selected/cached.

In this paper, we study this online service caching problem and our goal is to minimize users' perceived latency. We seek efficient online caching algorithms which can optimally select a subset of services while at the same time, which can run at the MEC server without any collaboration with UEs or the cloud. This makes our mechanism easy to deploy, scalable and of low cost. To achieve this, we make the following assumptions: (1) tasks for the services arrive independently; (2) a task can only be offloaded to the MEC server if the required service is present; otherwise, it will be directed to the cloud; and (3) the network delay between UEs and the MEC server is negligible as compared to the delay between the MEC server and the cloud, which implies that the MEC-offloading delay for a task equals to the time it is performed at the MEC server, and the cloud computing delay is the time for the task being processed at cloud plus the time it takes for transmitting data and computation results. It follows that both of the two delays can be observed at the MEC server.

Let $\mathcal{K} = \{1, 2, \ldots, K\}$ be the set of services in system, and assume they are of equal sizes. The MEC server can accommodate up to $L < K$ services. Time is divided into successive slots with equal length $\mathcal{T} = \{1, 2, \ldots, T\}$. At each time slot t, we associate with each service $i \in \mathcal{K}$ the following three quantities: (1) β_i^t, which is the arrival rate of tasks for service i at time t; (2) m_i^t, which is the aggregate latency of tasks for service i at time t if they are computed at the MEC server, i.e., the MEC-offloading delay; and (3) c_i^t, which is the aggregate latency of tasks for service i at time t if they are performed at cloud, i.e., the cloud computing delay. Note that depending on the caching state of service i, at an arbitrary time slot t we can only observe m_i^t or c_i^t, but not both of them. However, we can always observe β_i^t since all tasks from UEs are routed to the MEC server for potential offloading. Moreover, without loss of generality, we assume $\beta_i^t \in [0,1]$, $m_i^t \in [0,1]$ and $c_i^t \in [0,1]$. Table 1 gives main notations used in this paper.

Table 1
Main notations.

Symbol	Definition
$\mathcal{K}$	Set of services in system, $\|\mathcal{K}\| = K$
L	Server capacity
h	A threshold denoting the rate of tasks processed by the MEC server
δ	Failure probability
β_i^t	Arrival rate of tasks for service i at time t
m_i^t	MEC offloading delay of tasks for service i at time t
c_i^t	Cloud computing delay of tasks for service i at time t
(β_i, m_i, c_i)	Expectation of $(\beta_i^t, m_i^t, c_i^t)$
$(\bar{\beta}_i^t, \bar{m}_i^t, \bar{c}_i^t)$	Empirical means of $(\{\beta_i^t\}, \{m_i^t\}, \{c_i^t\})$ up to time t
$(\check{\beta}_i^t, \check{m}_i^t, \check{c}_i^t)$	Lower Confidence Bound for (β_i, m_i, c_i) at time t
$(\hat{\beta}_i^t, \hat{m}_i^t, \hat{c}_i^t)$	Upper Confidence Bound for (β_i, m_i, c_i) at time t
$m_{i,11}^t$	MEC offloading delay of tasks for service i of type 11 at time t
$\mathrm{UCB}_t(m_{i,11})$	Upper Confidence Bound for $m_{i,11}$ at time t
$\mathrm{LCB}_t(m_{i,11})$	Lower Confidence Bound for $m_{i,11}$ at time t
$\mathbf{x}^*$	Optimal caching policy
$\mathbf{x}^t$	Caching decision at time t
$\mathbf{x}^t(\pi)$	Caching decision made by algorithm π at time t
$\mathcal{L}_t(\pi)$	Set of services selected by algorithm π at time t

Let $\mathbf{x^t} = (x_1^t, x_2^t, \dots, x_K^t)^{\mathrm{T}}$ be the caching decision at time slot t, where $x_i^t = 1$ if service i is cached, and $x_i^t = 0$ otherwise.[1] Also denote $\mathbf{m^t} = (m_1^t, m_2^t, \dots, m_K^t)^{\mathrm{T}}$, $\mathbf{c^t} = (c_1^t, c_2^t, \dots, c_K^t)^{\mathrm{T}}$ and $\boldsymbol{\beta}^t = (\beta_1^t, \beta_2^t, \dots, \beta_K^t)^{\mathrm{T}}$. Here, in addition to optimize users' perceived latency, we also enforce another constraint to the caching policy that the aggregate rate of tasks processed by the MEC server is no less than a preset threshold $h > 0$. Note that this constraint can be regarded as the QoS requirement from the service providers. Indeed, one of the most significant benefits that a service provider can expect from multi-access edge computing is that the vast majority of their tasks are processed at the network edge and the workload on cloud thus can be dramatically decreased. This at the same time alleviates network congestion for the infrastructure provider.

Let $\mathbf{1}$ be the one column vector, i.e. $\mathbf{1} = (1, 1, \dots, 1)^{\mathrm{T}}$. With the above notations, we can formulate the online service caching problem as the following optimization problem:

$$\underset{\{\mathbf{x^t}\}}{\text{Min:}} \quad \sum_{t=1}^{T} \mathbf{x^t}^{\mathrm{T}} \mathbf{m^t} + (\mathbf{1} - \mathbf{x^t})^{\mathrm{T}} \mathbf{c^t} \tag{1a}$$

$$\text{s.t.:} \quad \mathbf{1}^{\mathrm{T}} \mathbf{x^t} \le L, \forall t \le T, \tag{1b}$$

$$\boldsymbol{\beta}^{t\mathrm{T}} \mathbf{x^t} \ge h, \tag{1c}$$

$$x_i^t \in \{0, 1\}, \forall i \in \mathcal{K}, t \le T, \tag{1d}$$

where $\sum_{t=1}^{T} \mathbf{x^t}^{\mathrm{T}} \mathbf{m^t} + (\mathbf{1} - \mathbf{x^t})^{\mathrm{T}} \mathbf{c^t}$ is the aggregate users' perceived delay over the whole time horizon T. Constraint (1b) is for the resource limitation at the MEC server, and (1c) is for the QoS requirement.

Problem (1) is an integer linear programming (ILP) problem that can be well solved by existing algorithms. The key obstacle here is that in an online setting, $\{\mathbf{m^t}, \mathbf{c^t}, \boldsymbol{\beta}^t\}$ are not known in advance when it comes to time slot t, but rather they are revealed after the caching decision is made.

Remark. The assumption that the network delay between UEs and the MEC server can be ignored is for the following two considerations: (1) MEC servers are generally deployed at the network edge in close proximity to end-users, whereas cloud are located much further away. This implies that the delay between UEs and the MEC server is much smaller than that between the MEC server and cloud. Furthermore, it is typically also true that the transmission delay between UEs and the MEC server is much smaller than task processing delay; (2) To implement multi-access edge computing, network infrastructure providers usually deploy MEC servers at locations such as a base station in a cellular network, or a gateway in an enterprise local area network. In such a setup, it is common that we have identical delays between UEs and the MEC server. The overall effect is that this delay can be ignored when we want to optimize users' perceived latency, as doing this will not make too much differences.

[1] Unless otherwise specified, all vectors defined in this paper are column vectors.

2.2. Problem formulation

To address the online caching problem (1), we model it with a Constrained Multi-Armed Bandit (CMAB) formulation. More specifically, each service is regarded as an arm and caching a service is equivalent to pulling an arm. The set of arms thus can be written as $\mathcal{K} = \{1, 2, \dots, K\}$. For each arm $i \in \mathcal{K}$, we associate it with three feedbacks: $\{m_i^t\}_{t=1}^T$, $\{c_i^t\}_{t=1}^T$ and $\{\beta_i^t\}_{t=1}^T$. We assume that all these sequences are made of iid. random variables.

Let $m_i = \mathbb{E}[m_i^t]$, $c_i = \mathbb{E}[c_i^t]$ and $\beta_i = \mathbb{E}[\beta_i^t]$. Also denote $\mathbf{m} = (m_1, m_2, \dots, m_K)^{\mathrm{T}}$, $\mathbf{c} = (c_1, c_2, \dots, c_K)^{\mathrm{T}}$ and $\boldsymbol{\beta} = (\beta_1, \beta_2, \dots, \beta_K)^{\mathrm{T}}$. Let $\mathcal{X}$ be the set of all valid caching decisions, i.e., $\mathcal{X} = \{\mathbf{x} \in \{0, 1\}^K | \mathbf{1}^{\mathrm{T}}\mathbf{x} \le L\}$. When all these information are available, the best (and static) caching policy $\mathbf{x}^*$ can be put as:

$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathcal{X}, \boldsymbol{\beta}^{\mathrm{T}}\mathbf{x} \ge h}{\text{argmin:}} \ \mathbf{x}^{\mathrm{T}}\mathbf{m} + (\mathbf{1} - \mathbf{x})^{\mathrm{T}}\mathbf{c} \tag{2}$$

As we have mentioned above, the expectations and distributions of the three feedbacks are unknown beforehand, and therefore we have to learn and make decisions based on their estimates. At each time slot t, an algorithm π makes caching decision $\mathbf{x}^t(\pi) \in \mathcal{X}$, and selects services $\mathcal{L}_t(\pi) \subset \mathcal{K}$. It then observes $\{m_i^t, \beta_i^t\}$ for each $i \in \mathcal{L}_t(\pi)$, and $\{c_j^t, \beta_j^t\}$ for each $j \in \mathcal{K} \setminus \mathcal{L}_t(\pi)$. Our objective is to design an algorithm π to decide the caching set $\mathcal{L}_t(\pi)$ for $t = 1, 2, \dots, T$ such that it achieves the minimal *regret*, i.e., the accumulated difference between the latency under π and that under the optimal policy $\mathbf{x}^*$, which is defined as:

$$Reg_\pi(T) = \sum_{t=1}^{T} \Big(\sum_{i \in \mathcal{L}_t(\pi)} m_i^t + \sum_{j \in \mathcal{K} \setminus \mathcal{L}_t(\pi)} c_j^t \Big) - T(\mathbf{x}^{*\mathrm{T}}\mathbf{m} + (\mathbf{1} - \mathbf{x}^*)^{\mathrm{T}}\mathbf{c}) \tag{3}$$

It is worthy noting that an algorithm π which achieves a low regret may violate the QoS constraint, especially when it has little information about services/arms. We define *violation* of the algorithm as the gap between the accumulated rate of tasks processed by the edge server and the target rate, as follows:

$$Vio_\pi(T) = [hT - \sum_{t=1}^{T} \sum_{i \in \mathcal{L}_t(\pi)} \beta_i^t]^+ \tag{4}$$

where $[x]^+ = \max\{x, 0\}$.

Both the regret and violation are important performance indexes and both should be taken into account when we design the algorithm. A small regret means the caching policy by the algorithm is close to the optimal one, and a small violation implies that the QoS requirement is well satisfied during the service caching process. In practice, an algorithm with *sub-linear* bounds on both regret and violation is considered acceptable and applicable.

3. CCB and its performance evaluation

In this section, we present our Constrained Confidence Bound (CCB) algorithm and its performance under simulation. The idea behind CCB is straightforward: (1) at each time slot t we can derive a valid caching policy through solving the optimization problem (1); and (2) although the system parameters are unknown at the time of decision, we can replace them by estimates. This should work well as long as the algorithm can provide us with good estimates, i.e., Lower/Upper Confidence Bound.

3.1. Algorithm

Denote by $\mathcal{H}_t = \{\mathcal{L}_\tau, m_i^\tau, c_i^\tau, \beta_i^\tau : i \in \mathcal{L}_\tau, 1 \le \tau \le T\}$ be the history of caching decisions and the observed feedback up to time slot t. CCB maintains the following empirical means for each arm $i \in \mathcal{K}$ at each time t:

$$\bar{m}_i^t = \frac{\sum_{\tau<t, i\in\mathcal{L}_\tau} m_i^\tau}{N_{i,\mathrm{M}}^t + 1} \tag{5}$$

$$\bar{c}_i^t = \frac{\sum_{\tau<t, i\notin\mathcal{L}_\tau} c_i^\tau}{N_{i,\mathrm{C}}^t + 1} \tag{6}$$

$$\bar{\beta}_i^t = \frac{\sum_{\tau<t, i\in\mathcal{L}_\tau} \beta_i^\tau}{t} \tag{7}$$

where $N_{i,\mathrm{M}}^t$ and $N_{i,\mathrm{C}}^t$ are the number of times arm i is selected and not selected before time t, respectively. Obviously, $N_{i,\mathrm{M}}^t + N_{i,\mathrm{C}}^t + 1 = t$.

Let $R(\mu, n) = \sqrt{\frac{\gamma\mu}{n}} + \frac{\gamma}{n}$ as in [13] and γ is a positive constant. Define the following Lower Confidence Bound for m_i and c_i at each time t:

$$\check{m}_i^t = \max\{0, \bar{m}_i^t - 2R(\bar{m}_i^t, N_{i,\mathrm{M}}^t + 1)\} \tag{8}$$

$$\check{c}_i^t = \max\{0, \bar{c}_i^t - 2R(\bar{c}_i^t, N_{i,\mathrm{C}}^t + 1)\} \tag{9}$$

and Upper Confidence Bound for β_i:

$$\hat{\beta}_i^t = \min\{1, \bar{\beta}_i^t + 2R(\bar{c}_i^t, t)\} \tag{10}$$

Denote by $\check{\mathbf{m}}^\mathbf{t} = (\check{m}_1^t, \check{m}_2^t, \ldots, \check{m}_K^t)$, $\check{\mathbf{c}}^\mathbf{t} = (\check{c}_1^t, \check{c}_2^t, \ldots, \check{c}_K^t)$, and $\hat{\boldsymbol{\beta}}^\mathbf{t} = (\hat{\beta}_1^t, \hat{\beta}_2^t, \ldots, \hat{\beta}_K^t)$. As depicted in Alg. 1, the input of CCB includes the arm set $\mathcal{K}$, the server capacity L, the QoS requirement h, the time horizon T, and $\delta \in (0, 1)$ which is a failure probability. CCB starts with γ set to $72 \ln \frac{2KT}{\delta}$. It then solves problem (1) at each time t with the system parameters $(\mathbf{m}, \mathbf{c}, \boldsymbol{\beta})$ being replaced by $(\check{\mathbf{m}}^\mathbf{t}, \check{\mathbf{c}}^\mathbf{t}, \hat{\boldsymbol{\beta}}^\mathbf{t})$ to get the selected arms $\mathcal{L}_t$. After that, it updates the upper/low confidence bounds for each arm. The process repeats until time T.

Procedure 1 Constrained Confidence Bound algorithm for Caching Services at the MEC server.

Input: $\mathcal{K}$, L, h, T, δ;

Output: Selected arms/services at each time slot;

1: $\gamma = 72 \ln \frac{2KT}{\delta}$, $\check{\mathbf{m}}^1 = \check{\mathbf{c}}^1 = \hat{\boldsymbol{\beta}}^1 = \mathbf{0}$, $N_{i,\mathrm{M}}^1 = N_{i,\mathrm{C}}^1 = 0, \forall i \in \mathcal{K}$.

2: **for** $t = 1, 2, \ldots, T$ **do**

3: Solve the following optimization problem:

$$\mathbf{x}^t = \underset{\mathbf{x}\in\mathcal{X}, \hat{\boldsymbol{\beta}}^{\mathbf{t}\mathrm{T}}\mathbf{x}\ge h}{\operatorname{argmin}}: \ \mathbf{x}^\mathrm{T}\check{\mathbf{m}}^\mathbf{t} + (\mathbf{1} - \mathbf{x})^\mathrm{T}\check{\mathbf{c}}^\mathbf{t} \tag{11}$$

4: Select arms $\mathcal{L}_t$ according to $\mathbf{x}^t$, and do the following updates for each arm $i \in \mathcal{K}$:

$$N_{i,\mathrm{M}}^{t+1} = \begin{cases} N_{i,\mathrm{M}}^t + 1, & \forall i \in \mathcal{L}_t \\ N_{i,\mathrm{M}}^t, & \forall i \in \mathcal{K} \setminus \mathcal{L}_t \end{cases}. \tag{12}$$

$$N_{i,\mathrm{C}}^{t+1} = \begin{cases} N_{i,\mathrm{C}}^t + 1, & \forall i \in \mathcal{K} \setminus \mathcal{L}_t \\ N_{i,\mathrm{M}}^t, & \forall i \in \mathcal{L}_t \end{cases}. \tag{13}$$

5: Based on the received feedback, calculate $(\check{\mathbf{m}}^{t+1}, \check{\mathbf{c}}^{t+1}, \hat{\boldsymbol{\beta}}^{t+1})$ accordingly.

The following theorem holds for CCB:

Theorem 3.1. *By running CCB, we have a probability at least* $1 - \delta$ *such that:*

$$Reg(T) = O((K - L)\sqrt{KT \ln \frac{2KT}{\delta}}),$$

$$Vio(T) = O(K\sqrt{KT \ln \frac{2KT}{\delta}}).$$

Proof. See Appendix A. □

Remark. CCB is based on Con-UCB [14], but with the following differences: (1) Con-UCB is proposed to tackle the online decision problem where the objective function is a function of one parameter only (can be in the form of multi-level feedback), whereas CCB can be applied when there are two or more such parameters in the objective function; (2) In Con-UCB, all feedback are bandit feedback, i.e., observations can only be made if the arm is selected. On the other hand, CCB allows both bandit feedback and full feedback, i.e., $\{\beta_i^t\}$. Moreover, CCB allows feedback that are complementary, i.e., one parameter that can be observed if the arm is selected, and the other one be observed if the arm is not selected, but which cannot be simultaneously observed, i.e., $\{m_i^t\}$ and $\{c_i^t\}$. These feedback need to be properly handled in the performance analysis. As a result, CCB is a generalization of Con-UCB.

3.2. Simulation results

We use simulations to investigate the performance of CCB, with both synthetic workload and real dataset. For synthetic workload, we assume the popularity of services follows a Zipf distribution with skewdness parameter $s \in \{0.6, 0.8, 1.0\}$. Task parameters are configured as follows: we divide task sizes into a set of intervals as [0.1 MB, 0.3 MB], [0.3 MB, 0.5 MB], [0.5 MB, 0.8 MB], [0.8 MB, 1 MB], [1 MB, 3 MB], [3 MB, 5 MB], [5 MB, 8 MB], and [8 MB, 10 MB] [15,16]. Note that these intervals are not of the same lengths. The task size of each service falls in one of these intervals that are randomly picked, and once fixed, the size of a task for the service is uniformly chosen from that interval. The computing intensity of tasks (in CPU cycles per bit) are drawn randomly from [100, 200, 300, 400, 500] [17], which represents certain amount of task heterogeneity and skewed workload distribution.

Meanwhile, we assume tasks arrive independently and the aggregate request rate for services is 100 req/sec. The network bandwidth between cloud and the MEC server is 5 Mbps [18], and each service is allocated to 5.6 GHz and 2.8 GHz CPU resources from cloud and the MEC server, respectively. The time horizon is set as $T = 100\,000$, and the length of each slot is 100 s. Without otherwise specified, we set the number of services in system as $K = 100$ and that can be hosted at the MEC server as $L = 10$.

For performance evaluation, we adopt the following two algorithms as baselines: (1) *Random-Caching*: this is the algorithm that randomly picks L services to cache at the MEC server at each time slot; and (2) *Top-Rate-Caching*: this is the algorithm that always cache the top L most popular services, assuming that the knowledge of service popularity is given a priori.

Fig. 2 shows how each algorithm performs with different parameter settings in simulation. From these figures, we can see that: (1) as expected, in all cases Random-Caching performs worst as both the regret and violation grow linearly in time; (2) the Top-Rate-Caching, which is able to satisfy the QoS constraint consistently, also does not provide a satisfactory delay performance for the linearly growing regret (although it grows much more slowly than Random-Caching). This implies that in general, the top L most popular services does not coincide with the set of services that provides the most caching gain. The reason is that workload distribution can be inconsistent with service popularity distribution, as we configured in simulation; and (3) CCB gives the best performance among the three algorithms, in that both the regret and violation grow sub-linearly as time elapses. Moreover, it can be observed that CCB behaves exactly the same way as Random-Caching at the very beginning (i.e., $t < 20\,000$), for the fact that during this period not enough samples are collected for services and as a result, CCB is not able to differentiate them but have to randomly pick services. After that period, CCB gradually identifies/learns the optimal services and the regret grows sub-linearly then.

Fig. 3 shows the performance of each algorithm in trace-driven simulation. We adopt a dataset from [19], which contains packet inter-arrival times from 5 applications generated by 36 wireless devices. The

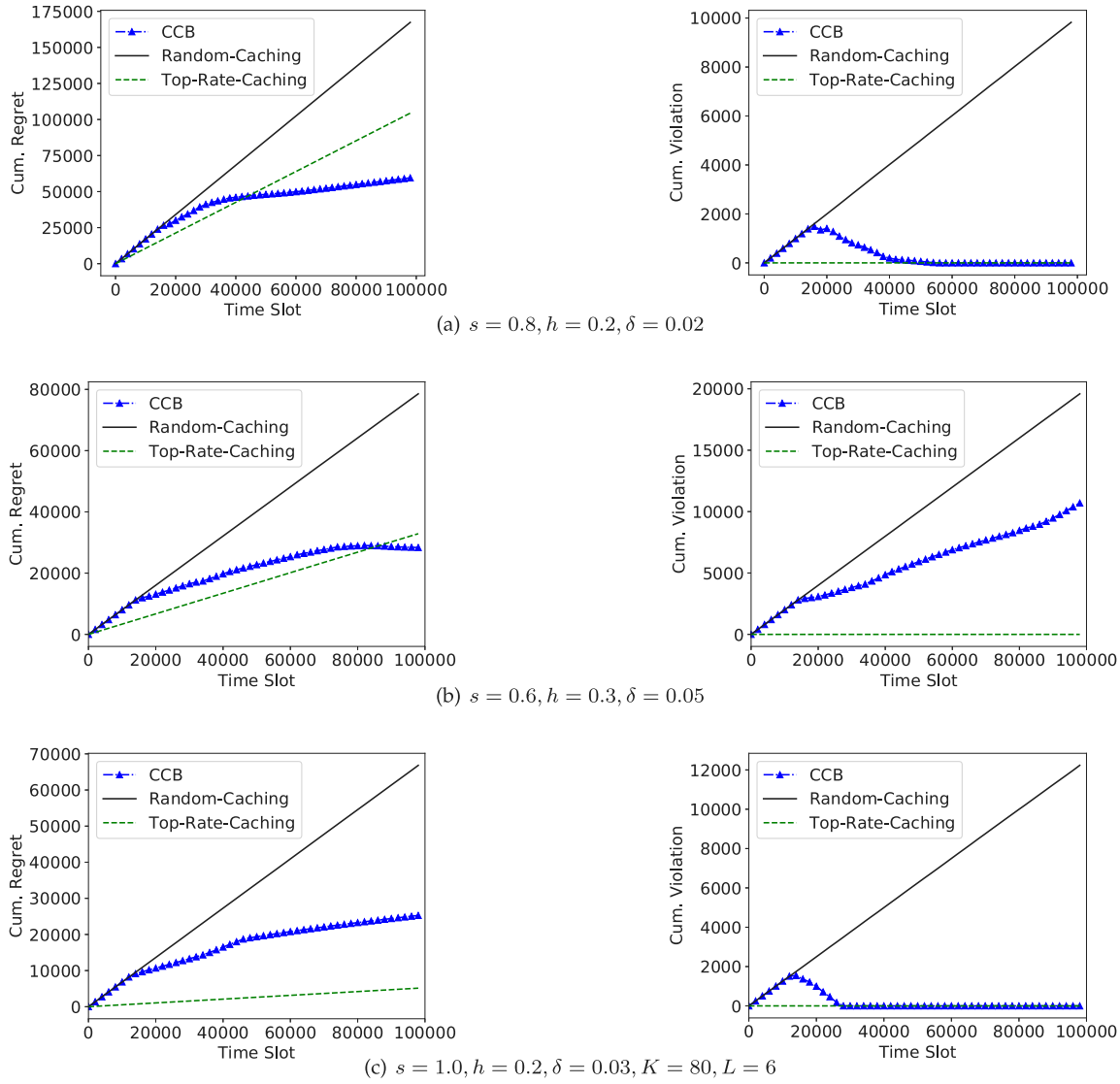


Fig. 2. Performance of CCB and the two baseline algorithms under different system settings.

wireless traces are used to generate workload, where each packet is regarded as a request and each <device, application> as a service. In this way, we get workload for 94 services in total. Meanwhile, to have enough data for simulation, we stretch time axis in each trace by 1000 (so a millisecond becomes a second). Here, since the optimal set of services is not known, we give the cumulative latency instead of regret. Again, we can see that CCB outperforms the two baselines whose latency and violation keep growing linearly all the time, whereas CCB slows down the growth, i.e., when $t > 40\,000$.

All in all, we believe that CCB serves as a good solution to online caching problem if our goal is more about the long-term system performance. In the next section, we will present an algorithm that converges much faster, which at the same time, grows much slower in both regret and violation than CCB.

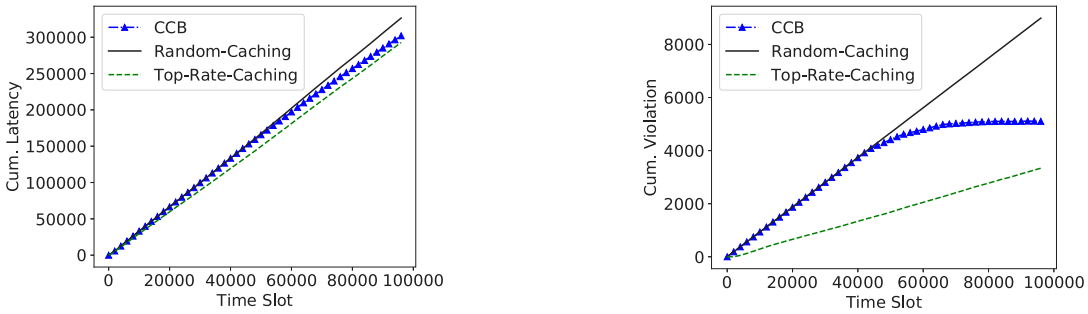
4. Service switch-aware online caching

In Section 3, we formulate the online caching problem and propose CCB, without taking into account service switching cost. According to the current technology, services are often hosted by VMs or containers, whose image (e.g., data, code) needs to be fetched and then loaded into system before the service is available. Note that during this period of time, tasks for the service will not be performed at the MEC server, but instead they have to be directed to the cloud for remote execution. The cost of service switches raises two new problems for online caching algorithms, if they were designed without properly considering it: (1) biased/inaccurate estimate of parameters, in particular, the MEC-offloading delay; and (2) system performance degradation due to frequent service switches. Take CCB for example, Fig. 4 shows its performance when there is no switching cost VS. there is cost, where in the latter scenario we set the time to load a new service as 20 s. It is evident that in call cases, the performance of CCB decreases due to service switches, i.e., the regret grows faster and it takes more time to converge. These results suggest that in practice, we need online caching algorithms that can properly handle the service switching cost.

4.1. Problem formulation

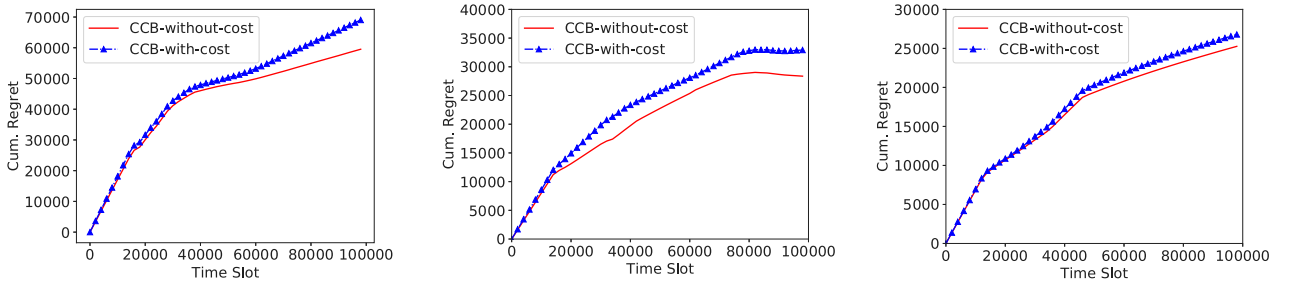
With service switching cost, the MEC-offloading delay for each service depends on whether the service is cached or not in the previous time slot, i.e., *the caching state*. Accordingly, at each time slot t we can divide services into 4 categories:

- type 00 — this is the set of services not cached at time t and will also be absent at $t + 1$;
- type 01 — this is the set of services not cached at t but will be cached at $t + 1$;



(a) Cummulative Latency (b) Cummulative Violation

Fig. 3. Performance of CCB and the two baseline algorithms in trace-driven simulation: $L = 10, h = 0.2, \delta = 0.01$.



(a) $s = 0.8, h = 0.2, \delta = 0.02$ (b) $s = 0.6, h = 0.3, \delta = 0.05$ (c) $s = 1.0, h = 0.2, \delta = 0.03, K = 80, L = 6$

Fig. 4. Performance reductions of CCB when there is service switching cost.

- type 10 — this is the set of services cached at t but will be absent at $t + 1$;
- type 11 — this is the set of services cached at t and will also be present at $t + 1$.

Note that among the four categories, service switching cost is incurred only for services of type 01, i.e., when a new service is loaded.

Likewise, for each service i, let $m^t_{i,11}$ and $m^t_{i,01}$ denote the delays of type 11 and 01, respectively, and denote by c^t_i the delays of type 00 and 10 (both for cloud computing). The online caching problem then becomes:

$$\underset{\{\mathbf{x^t}\}}{\text{Min:}} \sum_{t=1}^{T} \sum_{i \in \mathcal{K}} x_i^{t-1} x_i^t m_{i,11}^t + (1 - x_i^{t-1}) x_i^t m_{i,01}^t + (1 - x_i^t) c_i^t \tag{14a}$$

$$\text{s.t.:} \quad \mathbf{1}^{\mathrm{T}} \mathbf{x^t} \le L, \forall t \le T, \tag{14b}$$

$$\boldsymbol{\beta}^{t\mathrm{T}} \mathbf{x^t} \ge h, \tag{14c}$$

$$x_i^t \in \{0, 1\}, \forall i \in \mathcal{K}, t \le T, \tag{14d}$$

The above problem is a 0–1 quadratic programming problem that is much more complicated than problem (1), in that the caching decision at each time slot not only depends on the unknown delays but also depends on the current caching state. A simple and heuristic algorithm is to select services at each time slot t through solving the following 0–1 LP problem given the current system state $\mathbf{x^{t-1}}$:

$$\mathbf{x^t} =: \tag{15a}$$

$$\underset{\mathbf{x^t} \in \mathcal{X}, \hat{\boldsymbol{\beta}}^{\mathbf{t}\mathrm{T}} \mathbf{x^t} \ge h}{\text{argmin:}} \sum_{i \in \mathcal{K}} x_i^{t-1} x_i^t \check{m}_{i,11}^t + (1 - x_i^{t-1}) x_i^t \check{m}_{i,01}^t + (1 - x_i^t) \check{c}_i^t \tag{15b}$$

where $\check{m}^t_{i,11}$ and $\check{m}^t_{i,01}$ are LCBs of $m_{i,11}$ and $m_{i,01}$, respectively. This approach, although straightforward, is greedy in nature, and far from optimal as shown in Fig. 5, where we can see that the performance of the algorithm is identical or even worse (see Fig. 5(b)) than CCB.

4.2. Explore-first algorithm

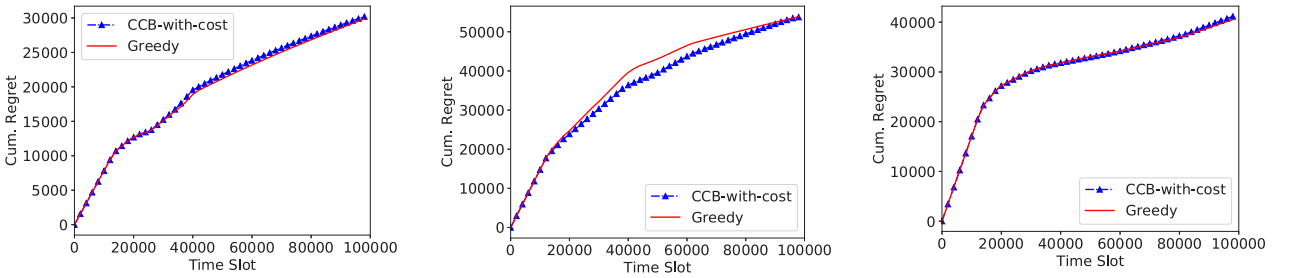
The first algorithm we propose to deal with service switches is Explore-First (EF). To start with, let us re-examine the online caching problem. We make the following observations: (1) Our goal is to minimize the user-perceived latency, that is, to cache the L services with the largest delay savings (under the given constraint), i.e., the gap between MEC-offloading delay and cloud computing delay; and (2) The optimal policy is:

$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathcal{X}, \boldsymbol{\beta}^{\mathrm{T}} \mathbf{x} \ge h}{\text{argmin:}} \mathbf{x}^{\mathrm{T}} \mathbf{m_{11}} + (\mathbf{1} - \mathbf{x})^{\mathrm{T}} \mathbf{c} \tag{16}$$

Note that both do not involve the switching cost. It follows that if we can estimate $\mathbf{m_{11}}$, $\mathbf{c}$ and $\boldsymbol{\beta}$ accurately based on feedback, then we can always find a good solution. Meanwhile, since service switching cost is incurred only when we explore new services, and multiple services needs to be selected at each time slot, we need an efficient sampling scheme with the following properties: (1) low cost ($\ll \mathrm{C}_K^L$); (2) arms/services are sampled uniformly, so as to simplify the algorithm design and its performance analysis; and (3) the frequency of service switches are well controlled so that we can avoid too much switching cost.

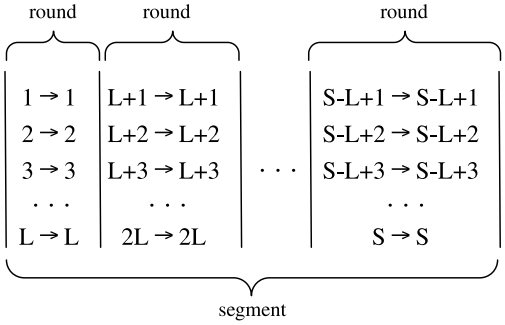
Our Explore-First algorithm is based on a novel sampling scheme with the above properties. We use *segment* as the basic unit to sample a given set of services, where each segment contains multiple rounds, and each round consists of two successive time slots. The structure of a segment is depicted in Fig. 6. We discriminate two scenarios according to whether the number of services to sample S can be divided by L:

- $S \mod L = 0$. In this case, whenever a new round begins, we select L new services (at the first time slot), and keep hosting these services at the second time slot, as shown in Fig. 6(a). To sample all the services, each segment contains S/L rounds, and $2S/L$ time slots in total.

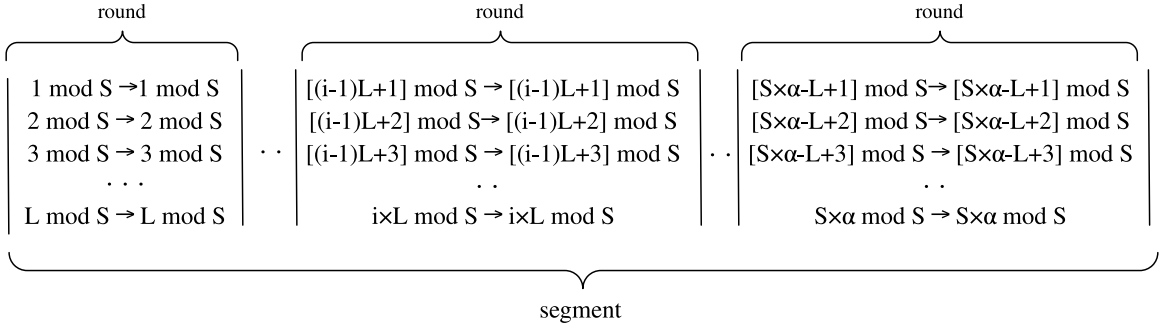


(a) $s = 0.8, h = 0.2, \delta = 0.02$ (b) $s = 0.6, h = 0.3, \delta = 0.05$ (c) $s = 1.0, h = 0.2, \delta = 0.03, K = 80, L = 6$

Fig. 5. Performance comparison between CCB and the greedy algorithm when there is service switching cost.



(a) $S \mod L = 0$



(b) $S \mod L \neq 0$

Fig. 6. A segment used to sample S services.

- $S \mod L \neq 0$. Let α be the smallest positive integer such that $S\alpha \mod L = 0$. In this case, each segment contains $S\alpha/L$ rounds, and we select at the jth round the following services: $\{[(j-1)L + 1] \mod S, [(j-1)L + 2] \mod S, \ldots, jL \mod S\}$, as depicted in Fig. 6(b). Note that it is our policy that service switch occurs whenever a new round begins, regardless of whether the service to cache has been hosted in the previous round.

It can be easily verified that with this sampling scheme, each service is sampled at the same rate, i.e., for both two scenarios we have:

$$n(i, \mathrm{M}) = \alpha \tag{17}$$

$$n(i, \mathrm{C}) = 2(\frac{S\alpha}{L} - \alpha) \tag{18}$$

$$n(i) = 2S\alpha/L \tag{19}$$

where we set $\alpha = 1$ when $S \mod L = 0$. $n(i, \mathrm{M})$ and $n(i, \mathrm{C})$ denote the number of times service i is hosted at the MEC server (without switching cost) or it is played at cloud, in each segment, respectively, and $n(i)$ denotes the number of times i is requested.

The idea behind Explore-First (EF) is simple: we explore services uniformly with the above sampling scheme, and pick an empirically best arm set for exploitation, regardless of what has been observed previously. More specifically, we divide the time horizon T into two phases, *exploration phase* and *exploitation phase*, as shown in Fig. 7(a). The exploration phase consists of N successive segments, where each segment is used to sample the ground set of services $\mathcal{K}$. The exploitation phase follows which consists of remaining time slots that always

play the empirically best arm set, derived by solving the optimization problem 20.

Procedure 2 Explore-First algorithm for Caching Services at the MEC server.

Input: $\mathcal{K}$, L, h, T;

Output: Selected arms/services at each time slot;

1: Exploration phase: Sample the set of arms $\mathcal{K}$ with N segments.

2: Select the arm set $\mathcal{L}(\mathbf{x}^*)$ by solving the following optimization problem:

$$\mathbf{x}^* = \underset{\mathbf{x}\in\mathcal{X}, \bar{\boldsymbol{\beta}}^{\mathrm{T}}\mathbf{x}\geq h}{\operatorname{argmin}}: \ \mathbf{x}^{\mathrm{T}}\bar{\mathbf{m}}_{\mathbf{11}} + (\mathbf{1}-\mathbf{x})^{\mathrm{T}}\bar{\mathbf{c}} \tag{20}$$

3: Exploitation phase: Play $\mathcal{L}(\mathbf{x}^*)$ in all remaining time slots.

As show in Alg. 2, here N is a parameter chosen to minimize the regret. It is a function of the time horizon T, the number of arms K and L. In Appendix B, we will show how to properly set it.

We define the following regret for performance evaluation:

$$Reg_{\pi}(T) =: \sum_{t=1}^{T}\Big(\sum_{i\in\mathcal{L}_t(\pi)} m_{i,11} + \sum_{j\in\mathcal{K}\setminus\mathcal{L}_t(\pi)} c_j\Big) - T(\mathbf{x}^{*\mathrm{T}}\mathbf{m}, \mathbf{11} + (\mathbf{1}-\mathbf{x}^*)^{\mathrm{T}}\mathbf{c}) \tag{21}$$

Note that this definition is different from (3) as it uses the expected latencies whereas realized latencies are adopted in (3). The definition of violation remains unchanged as (4).

Theorem 4.1. *By running Explore-First, we have the following bounds on regret and violation:*

$$Reg(T) = O\left((\frac{K^4}{L})^{\frac{1}{3}}T^{\frac{2}{3}}(\ln T)^{\frac{1}{3}}\right),$$

$$Vio(T) = O\left(K^{\frac{1}{3}}L^{\frac{2}{3}}T^{\frac{2}{3}}(\ln T)^{\frac{1}{3}}\right).$$

Proof. See Appendix B. □

4.3. Successive elimination-based algorithm

Explore-First is able to identify the optimal set of services precisely given sufficient samples, however, the performance in the exploration phase may be poor, especially when most of the arms have a large gap compared with the optimal one. Here we present another caching algorithm, called Successive Elimination-based (SE) caching, that can significantly decrease the sampling cost while at the same time, improve the bound on both regret and violation.

The main idea behind SE is as follows: (1) we divide time horizon T into two phases: *exploration/elimination phase* and *exploitation phase*, as shown in Fig. 7(b). The exploration/elimination phase consists of successive segments, where each segment is used to sample a given set of (active) arms; (2) In the end of each segment, one or more arms may be deactivated according to the elimination rule; and (3) The elimination phase completes when there is no more arms to be deleted, and the exploitation phase follows which keeps playing the remaining arms.

More specifically, SE maintains the following quantities (LCB/UCB) for each service i at time t:

$$\mathrm{UCB}_t(m_{i,11}) = \bar{m}_{i,11} + \sqrt{2\ln T/n_t(i,\mathrm{M})} \tag{22}$$

$$\mathrm{LCB}_t(m_{i,11}) = \bar{m}_{i,11} - \sqrt{2\ln T/n_t(i,\mathrm{M})} \tag{23}$$

$$\mathrm{LCB}_t(c_i) = \bar{c}_i - \sqrt{2\ln T/n_t(i,\mathrm{C})} \tag{24}$$

where $n_t(i,\mathrm{M})$ denotes the number of times service i of type 11 is played, and $n_t(i,\mathrm{C})$ denotes the number of times i is absent before t. $m_{i,11}$ and c_i denote the aggregate latency when tasks of service i are computed at the MEC server (without switching cost) and the cloud, respectively.

Let S_t be the set of arms remaining active at time t. SE deactivates arms according to the following rules:

Rule 1: At the end of each segment (assuming at time t), identify the set of arms $\mathcal{A}$ such that arm $i \in \mathcal{A}$ if we can find some other arm $j \in S_t$ with $\mathrm{UCB}_t(c_i) - \mathrm{LCB}_t(m_{i,11}) < \mathrm{LCB}_t(c_j) - \mathrm{UCB}_t(m_{j,11})$;

Rule 2: Solve the following optimization problem for S_t:

$$\mathbf{x}^t = \underset{\mathbf{x}\in\mathcal{X}, \bar{\boldsymbol{\beta}}^{\mathrm{T}}\mathbf{x}\geq h}{\operatorname{argmin}}: \ \mathbf{x}^{\mathbf{T}}\mathrm{UCB}_t(\boldsymbol{m_{11}}) + (\mathbf{1}-\mathbf{x})^{\mathbf{T}}\mathrm{UCB}_t(\boldsymbol{c}) \tag{25}$$

Denote by $\mathcal{L}(\mathbf{x}^t)$ the set of selected arms, obtain the set $\mathcal{B} = S_t \setminus \mathcal{L}(\mathbf{x}^t)$;

Rule 3: Deactivate from S_t the arms in both $\mathcal{A}$ and $\mathcal{B}$.

Note that rule 1 is used to identify arms with small contribution, i.e., these arms are likely not the optimal ones. Rule 2 is used to identify arms that are not optimal with high confidence, where the QoS constraint has been properly taken into account. Therefore, the intersection of the two sets gives us arms that can be deactivated with high confidence. One can image that during the very first few segments $\mathcal{A} = \emptyset$ and $\mathcal{B}$ is a random set, since not enough samples are collected and the algorithm is not able to differentiate arms. As time elapses, $\mathcal{A}$ becomes larger and $\mathcal{B}$ becomes more accurate. Once an arm is in $\mathcal{A}\cap\mathcal{B}$, we are highly confident that it does not belong to the optimal set and thus can be eliminated. Moreover, the active set becomes smaller as time elapses since more and more arms are deactivated, which significantly decreases the sampling cost. See Alg. 3 for more details.

Procedure 3 Successive Elimination-based Algorithm for Caching Services at the MEC server.

Input: $\mathcal{K}$, L, h, T;

Output: Selected arms/services at each time slot;

1: $t = 1$; $S = \mathcal{K}$. # S is the set of active arms;

2: **for** $t \leq T$ **do**

3: Sample S with a segment.

4: **if** $|S| > L$ **then**

5: $\mathcal{A} = \emptyset$; # $\mathcal{A}$ denotes set of arms to be potentially deactivated;

6: **for** $i \in S$ **do**

7: **if** $\exists j \in S$ such that $\mathrm{UCB}_t(c_i) - \mathrm{LCB}_t(m_{i,11}) < \mathrm{LCB}_t(c_j) - \mathrm{UCB}_t(m_{j,11})$ **then**

8: $\mathcal{A} = \mathcal{A} \cup \{i\}$;

9: Solve the following optimization problem for S, and denote the selected arms as $\mathcal{L}(\mathbf{x}^t)$:

$$\mathbf{x}^t = \underset{\mathbf{x}\in\mathcal{X}, \bar{\boldsymbol{\beta}}^{\mathrm{T}}\mathbf{x}\geq h}{\operatorname{argmin}}: \ \mathbf{x}^{\mathbf{T}}\mathrm{UCB}_t(\boldsymbol{m_{11}}) + (\mathbf{1}-\mathbf{x})^{\mathbf{T}}\mathrm{UCB}_t(\boldsymbol{c}) \tag{26}$$

10: Deactivate arms in both $\mathcal{A}$ and $S \setminus \mathcal{L}(\mathbf{x}^t)$:

$$S = S \setminus (\mathcal{A} \cap \{S \setminus \mathcal{L}(\mathbf{x}^t)\}) \tag{27}$$

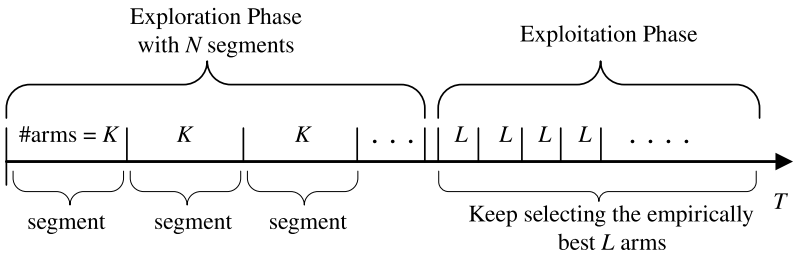
Theorem 4.2. *Let t be the time slot that the elimination phase completes, by running SE we have the following bounds on regret and violation:*

$$Reg(T) \leq O(\sqrt{KTt\ln T}),$$

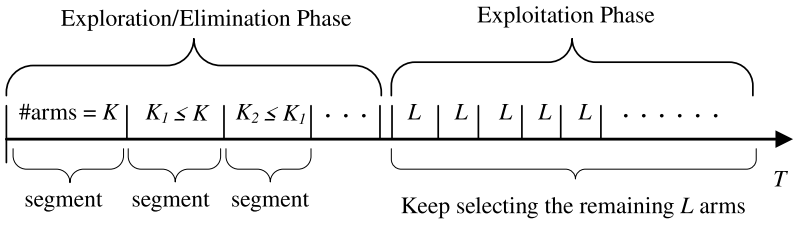
$$Vio(T) \leq O(t\sqrt{KT\ln T}).$$

Proof. See Appendix C. □

Theorem 4.2 gives an *instance-dependent* upper bound on regret by SE since t is a function of parameter distributions, i.e., $\{m_{i,11}, c_i, \beta_i\}$. On the other hand, it is well known that UCB-like algorithms achieve an *instance-independent* upper bound on the order of $O(\sqrt{KT\ln T})$ for multi-armed bandit problems. We argue that there is no conflict in that the two bounds are of the same order, as stated in the following lemma.



(a) Explore-First algorithm



(b) Successive Elimination-based algorithm

Fig. 7. Two service switch-aware algorithms for online caching.

Lemma 4.1. *SE achieves regret:*

$Reg(T) \leq O(\sqrt{KLT\ln T})$

Proof. See Appendix D. □

4.4. Simulation results

4.4.1. Performance over baselines

Fig. 8 gives performance of EF, SE and other algorithms when there is switching cost. Again, we set the time to load a new service as 20 s. From the figure, we can see that in all cases: (1) both EF and SE can accurately identify the optimal set of services as their regret and violation keep non-increasing after convergence; on the other hand, CCB exhibits a sub-linear growth in regret; (2) among the two proposed algorithms, SE converges much faster, i.e., less than 8000 time slots are needed for it to get stable whereas it takes more than 30 000 slots for EF. Moreover, we find that SE are far more computationally cost-effective than CCB and EF as it requires less than 10 min to run each simulation on our machine (2 × 2.2 GHz CPU, 8 GB Memory), whereas it takes approximately 30 min for EF and even 2 h for CCB (note that CCB requires solving the optimization problem at each time slot).

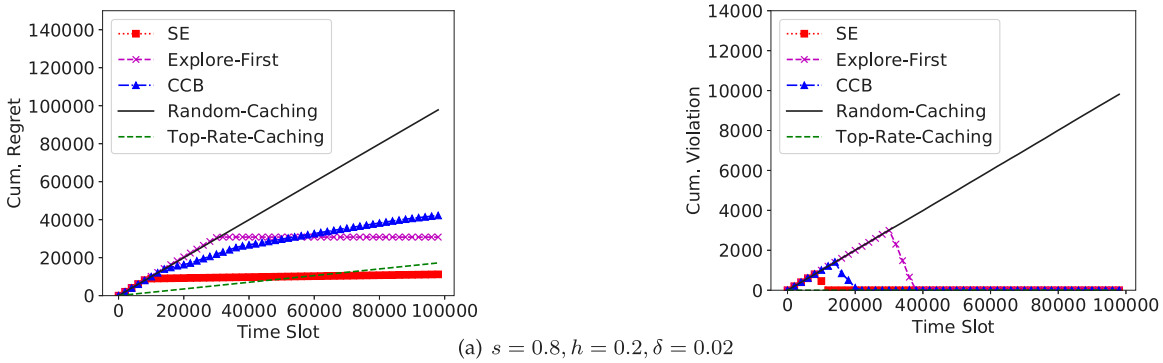
Fig. 9 gives performance of the corresponding algorithms in trace-driven simulation. As expected, we observe that both SE and EF outperform the baselines and the conclusions are consistent. Moreover, it is interesting to find that EF and SE have approximately the same convergence time under the real workload, which suggests that in practice one can adopt either of them for online service caching.

Furthermore, given the performance of CCB as shown in Fig. 4, we believe that SE and EF also outperforms when there is no service switching cost.

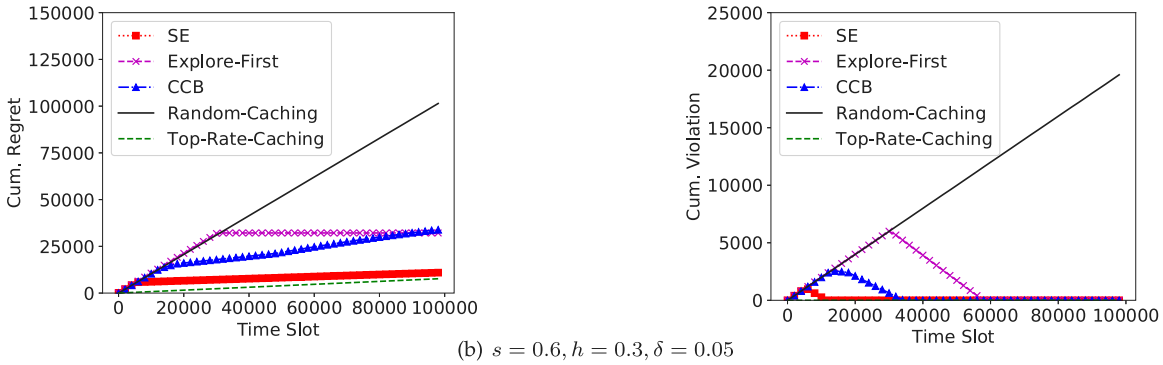
Remark. Here we further compare and analyze the advantages and application scenarios of the two algorithms. As we mentioned above, EF works by first sampling services uniformly at the same rate (with our novel sampling scheme), and then makes caching decisions based on estimates of the relevant parameters of services. It then keeps caching the same empirically optimal set of services in the exploitation phase. Obviously, this algorithm does not adapt its exploration scheduler to the history of the observed rewards. Moreover, in order to have a good performance, usually a large number of time slots is dedicated to the exploration phase, which incurs a high sampling cost. These together leads to the fact that this algorithm is particularly useful when the time horizon T is large and the system is stable. On the other hand, SE-based caching works by successively eliminating non-optimal services during the sampling process, and therefore it satisfies the so called adaptive exploration and incurs much lower sampling cost, i.e., fewer time slots are needed in the exploration/elimination phase. In other words, this algorithm converges significantly faster and achieves much better regret bounds. Based on the above reasoning, we conclude that SE-based caching is particularly suitable when the underlying system is non-stable such as a real edge computing system, where in that case both the convergence speed/rate and accuracy are important, i.e., we can divide time horizon T into multiple phases, and restart the caching algorithm when a new phase starts, so as to quickly adapt to the dynamics of the system.

4.4.2. Performance comparison with SoA

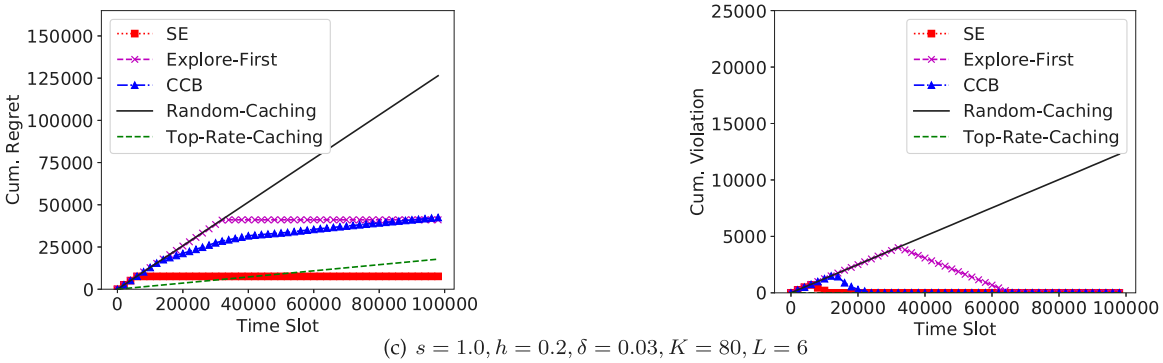
It is interesting to see the performance of our proposed schemes against existing online algorithms. To this end, we compare CCB with the recently proposed *potential*-based algorithm [20], which is a lightweight but very efficient algorithm for online content caching. Here, we model the edge-cloud system as a cache network with two nodes, where the edge server is considered as a cache node with limited storage capacity, and the cloud as a server that permanently holds all the content in system. Moreover, each service is regarded as a content, and each request for a service as a request for a content. We characterize each request for a service as a tuple (i, t_i^m, t_i^c), where i is the service requested, t_i^m and t_i^c (both are stochastic) denotes the MEC-offloading delay and cloud-computing delay, respectively. The MEC server maintains a quantity Q_i called *potential* for each service i, together with the empirical means for MEC-offloading delay T_i^m and cloud-computing delay T_i^c for each service i. Note that these quantities are initialized zero and updated whenever a new request arrives.



(a) $s = 0.8, h = 0.2, \delta = 0.02$

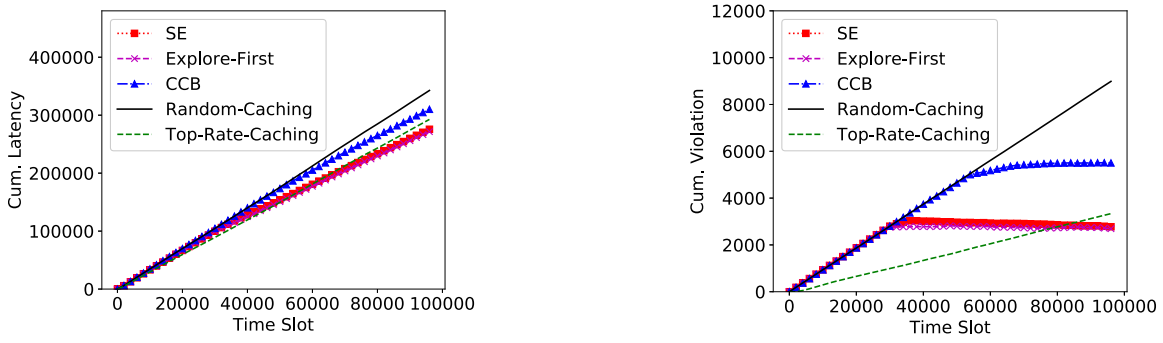


(b) $s = 0.6, h = 0.3, \delta = 0.05$



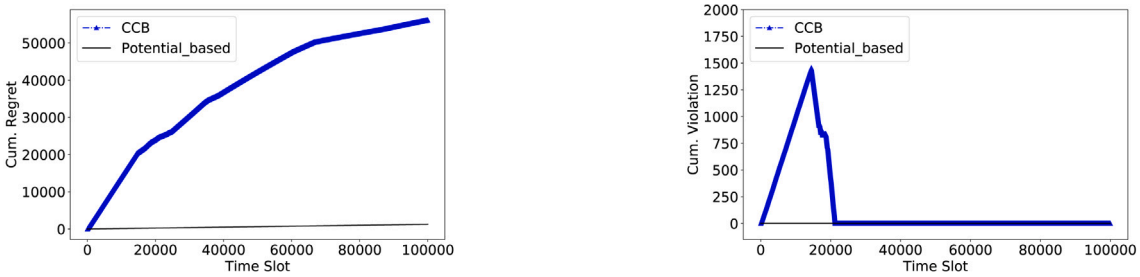
(c) $s = 1.0, h = 0.2, \delta = 0.03, K = 80, L = 6$

Fig. 8. Performance of EF, SE and baseline algorithms when there is service switching cost.



(a) Cummulative Latency (b) Cummulative Violation

Fig. 9. Performance of SE, EF and other algorithms in trace-driven simulation with switching cost: $L = 10, h = 0.2, \delta = 0.01$.



(a) Cummulative Latency (b) Cummulative Violation

Fig. 10. Performance of CCB and the *potential*-based caching algorithm in simulation with no service switch cost: $K = 100, L = 10, h = 0.2, \delta = 0.02$.

At the very beginning, there is no services cached at the MEC server. Service caching is then performed according to the following rules:

Rule 1: if a request (i, t_i^m, t_i^c) arrives at the MEC server, Q_i is updated as follows:

$$Q_i = Q_i + T_i^c - T_i^m \tag{28}$$

If i is cached, then:

$$T_i^m = \frac{T_i^m \times N_i^m + t_i^m}{N_i^m + 1}, N_i^m = N_i^m + 1 \tag{29}$$

else:

$$T_i^c = \frac{T_i^c \times N_i^c + t_i^c}{N_i^c + 1}, N_i^c = N_i^c + 1 \tag{30}$$

where N_i^m and N_i^c denotes the number of times service i is requested when it is cached at the MEC server and when it is absent, respectively.

Rule 2: if a response to request (i, t_i^m, t_i^c) from the remote cloud arrives at the MEC server and there is no room for hosting service i if it is absent, then the MEC server calculates the caching probability y_i for i based on Q_i's:

$$y_i = \frac{Q_i}{Q_i + \sum_{j \in S_c} Q_j} \tag{31}$$

where S_c is the set of services cached at the MEC server. If the decision is to cache i, then the service j with the least potential, i.e., $j = \underset{i \in S_c}{\text{argmin}}\, Q_i$, is evicted.

It can be seen from the above two rules that the potential-based caching algorithm aims at minimizing the aggregate latency for accessing the services in system. The following figures show its performance and our proposed mechanism CCB, when there is no service switch cost and when there is cost. From Fig. 10 we can see that the potential-based algorithm performs exceptionally well when there is no service switch cost, i.e., cumulative regret grows very slowly (although still linear) and the offloading rate constraint can always be satisfied. However, as depicted in Fig. 11, its performance gets poor when there is switch cost, as both the regret and violation grow linearly as time elapses. After a deep investigation, we find that this phenomenon is due to the frequent service switches incurred during the caching process by the potential-based algorithm. Based on this observation, we conclude here that any caching algorithm could perform poorly when there is service switch cost, if this cost is not taken into account when we design the algorithm.

5. Related work

The service placement problem (SPP) in Edge/Fog computing is essentially to find the available resources (nodes, links) in the network so as to optimize certain objectives (delay, energy consumption, etc.) while at the same time satisfy application requirements, resource constraints, locality constraints, etc. It has been a hot topic [21–23] in the past few years, and many approaches have emerged.

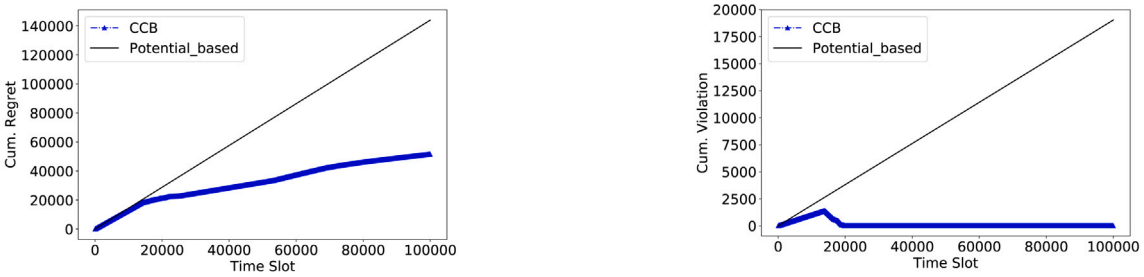
Existing solutions can be categorized into centralized [24,25] and distributed [26–28], based on the control plane design. A centralized algorithm assumes that the global information such as application demands and infrastructure resources are available, and computes a globally optimal solution. For example, Hong et al. [25] propose that a coordinator makes deployment decisions for IoT services over the fog infrastructure. The drawback of the centralized solution is that global information is generally hard to collect and the computational cost may be excessively high. On the other hand, a distributed approach relies on the local computation of each node and their collaboration to address the scalability and locality awareness issues. This approach is able to provide services that fit the local context, but generally speaking, it cannot guarantee global optimality of the solution.

The service placement problem can be addressed in an offline [29,30] and online fashion [31,32]. The offline approach requires that all the information about the system and workload are given a priori before the placement decision is computed. That is, the placement decision is made at the complie time before deployment. Examples include [29,30] that assume full knowledge of the Edge/Fog network. On the other hand, recently proposed approaches [33–35] are mainly online that the placement decisions are made during the run-time of the systems. To provide satisfactory performance, the online algorithms have to take into account the dynamic behaviors of the system. The advantage of this approach is that it is more adaptive and responsive to changes. However, it remains a challenge as how to make the best use of the system resources.

Based on whether the dynamicity of the system is handled or not, existing placement solutions can also be classified as static and dynamic [36,37]. The static approach usually assumes that the Edge/Fog infrastructure and application characteristics remain unchanged as time elapses, which is not realistic. In fact, both the two aspects are highly time-evolving as new nodes can join and leave the system due to instability of the network, the resources available can change over time based on real-life condition, and the workload varies when users' interest changes. The dynamic approaches [38,39] employ reactive strategies to deal with the dynamic nature of the infrastructure and application, in a way that new services may be deployed and existing services may be replaced/released whenever significant change is observed.

Alternatively, one can also characterize existing SPP solutions based on various aspects such as: (1) whether the mobility prediction is exploited or not for mobility and popularity caching [40,41], (2) user-centric cooperative edge caching [42,43] or network-centric non-cooperative caching, (3) intelligent handover predictions for the edge [44] and various other recent AI-based approaches like adopted reinforcement learning [45,46], (4) price congestion schemes for caching [47,48], and (5) DDPG for orchestration from an SDN perspective [49,50] and so forth.

Obviously, our algorithms belong to the category of dynamic and online solutions. The work that most close to ours is [51], where the authors address user-managed service placement problem, while in this



(a) Cummulative Latency (b) Cummulative Violation

Fig. 11. Performance of CCB and the *potential*-based caching algorithm in simulation with service switch cost: $K = 100, L = 10, h = 0.2, \delta = 0.02$.

work we address the problem from the network-side (that is, the network operators make the caching decisions instead of users). Moreover, they adopt a contextual MAB framework with a Thompson-sampling scheme for online learning, whereas we employ a Constrained-MAB framework with a novel sampling scheme that we propose to efficiently explore system dynamics.

6. Conclusion and future work

In this paper, we study the online service caching problem for a multi-access edge computing system, with the goal to minimize users' perceived latency. We formulate it as a Constrained Multi-Armed Bandits (CMAB) optimization problem, and propose three efficient algorithms — CCB, EF and SE. We show that CCB can well balance the objective and QoS constraint, whereas EF and SE can effectively learn the optimal solution when there is service switching cost. We theoretically analyze their performance by giving the bound on regret and violation, and conduct extensive simulations to validate their efficacy. Our experimental results show that these algorithms outperform baselines.

There are several interesting issues for exploration. Below we give some possible directions that we believe are important and worthy of further investigation.

(1) State-dependent CMAB formulation. We have shown that when there is service switching cost, the MEC-offloading delay depends on whether the service is cached or not, i.e., the caching state. Now if we treat each service of a particular type as an arm, then we get a state-dependent CMAB problem formulation, where the action available in the next time slot also depends on the current state of the system. It is still an open problem as how to design efficient algorithms for this type of CMAB problem, especially when the state space is huge.

(2) Heterogeneous services. We assume that the MEC server can host L services at most. This means all the services are of equal sizes. Given the limited resources of the MEC server, if services are heterogeneous in storage or memory, then at any time slot t the number of services hosted by the MEC server can be different, depending on the arms selected. This is quite different from the problem we considered in this paper. One way to handle heterogeneous services is to model the problem with Combinatorial Bandits with Knapsack Constraints (CBwK), which combines Combinatorial Bandits where a subset of arms needs to be pulled at each round, and Bandits with Knapsack Constraints where a super-arm needs to be pulled at each round but within a budget constraint. However, CBwK is not readily applicable since in addition to the knapsack constraint, we also need to ensure that the rate of tasks processed by the MEC server is no less than a preset threshold. This novel QoS constraint, however, poses significant challenges for both the design and performance analysis of efficient algorithms.

(3) Multiple instances for each service. We assume in this work that exactly one instance (e.g., VM) for each service can be hosted at the MEC server, whereas in practice there can be multiple instances, i.e., for adequate computing power or load balancing. This raises another question as how to determine the necessary number of VMs for each service, and then design efficient online learning algorithms for service caching. One possible solution is to extend the current model, i.e., by regarding each service with a particular number of instances as an arm. The key challenges here are: (1) the set of arms/actions would be huge; and (2) instead of selecting L services each time, the server capacity is now expressed as a new constraint, which further complicates the online service caching problem.

(4) Online service caching for MEC-based networks. There is a trend that multiple edge servers work collaboratively to form a shard resource pool [52–54], so as to provide reliable and elastic edge computing services. This, on one hand, provides us opportunity to leverage the power of the network for better exploration and exploitation. On the other hand, it also raises significant challenges to design online learning algorithms for the network, since MEC servers may be heterogeneous in computing power, user bases, network condition, etc. Given that there is a flurry studies on Multi-agent MAB (MA-MAB) [55–57], to the best of our knowledge, it is still unclear as how to formulate and solve the problem that we concern here, i.e., Multi-agent MAB with multiple constraints. We believe that the problem itself is interesting in the area of MAB and deserves further study.

CRediT authorship contribution statement

Weibo Chu: Conceptualization, Methodology, Writing – original draft, Writing – review & editing. **Xiaoyan Zhang:** Investigation, Software, Visualization. **Xinming Jia:** Formal analysis, Resources, Validation. **John C.S. Lui:** Writing – review & editing, Funding acquisition, Methodology, Supervision. **Zhiyong Wang:** Data curation, Software, Visualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (Grant No. 62172333) and the Natural Science Basic Research Plan in Shaanxi Province of China (Grant No. 2021JM-073). The work of John C.S. Lui was supported in part by the GRF 14215722 and RGC SRFS2122-4S02.

Appendix A. Proof of Theorem 3.1

We rely on the following lemmas to prove the theorem.

Lemma A.1 (*Azuma-Hoeffding Inequality [58]*). *Suppose* $\{Y_n : n = 0, 1, 2, 3, \ldots\}$ *is a martingale and* $|Y_n - Y_{n-1}| \le c_n$ *almost surely, then with probability at least* $1 - 2e^{-\frac{d^2}{2\sum_{j=1}^{n} c_j^2}}$, *we have:*

$$|Y_n - Y_0| \le d.$$

Lemma A.2 (*[59,60]*). *Consider* n *i.i.d random variables* $Z_1, Z_2, \ldots, Z_n$ *in* $[0, 1]$ *with expectation* z. *Let* u *denote their empirical average. Then for any* $\gamma > 0$, *with probability at least* $1 - 2e^{-\frac{1}{72}\gamma}$ *we have:*

$$|u - z| \le R(u, n)$$

where $R(u, n) = \sqrt{\frac{\gamma u}{n}} + \frac{\gamma}{n}$.

The following lemma is a corollary of Lemma A.2.

Lemma A.3 (*[14]*). *Let the empirical means* $\bar{m}_i^t$, $\bar{c}_i^t$ *and* $\bar{\beta}_i^t$ *be defined as (5) (6) (7). Then for all* i *and* t, *with probability at least* $1 - 2e^{-\frac{1}{72}\gamma}$ *we have:*

$$|\bar{m}_i^t - m_i| \le 2R(\bar{m}_i^t, N_{i,M}^t + 1) \tag{32}$$

$$|\bar{c}_i^t - c_i| \le 2R(\bar{c}_i^t, N_{i,C}^t + 1) \tag{33}$$

$$|\bar{\beta}_i^t - \beta_i| \le 2R(\bar{\beta}_i^t, t) \tag{34}$$

where $\gamma \ge 1$.

Proof of Lemma A.3. For every i and t, applying Lemma A.2, we have with probability at least $1 - 2e^{-\frac{1}{72}\gamma}$:

$$|\frac{N_{i,M}^t + 1}{N_{i,M}^t}\bar{m}_i^t - m_i| \le R(\frac{N_{i,M}^t + 1}{N_{i,M}^t}\bar{m}_i^t, N_{i,M}^t)$$

$$|\bar{m}_i^t - m_i + \frac{m_i}{N_{i,M}^t + 1}| \le \frac{N_{i,M}^t}{N_{i,M}^t + 1} R(\frac{N_{i,M}^t + 1}{N_{i,M}^t}\bar{m}_i^t, N_{i,M}^t)$$

This implies that

$$\begin{aligned}|\bar{m}_i^t - m_i| &\le \frac{N_{i,M}^t}{N_{i,M}^t + 1}(\sqrt{\frac{\gamma(N_{i,M}^t + 1)\bar{m}_i^t}{N_{i,M}^t \times N_{i,M}^t}} + \frac{\gamma}{N_{i,M}^t}) + \frac{m_i}{N_{i,M}^t + 1} \\ &= R(\bar{m}_i^t, N_{i,M}^t + 1) + \frac{m_i}{N_{i,M}^t + 1} \\ &\le 2R(\bar{m}_i^t, N_{i,M}^t + 1)\end{aligned}$$

The last inequality holds because $m_i \le 1 \le \gamma$.

Similarly, we can prove $|\bar{c}_i^t - c_i| \le 2R(\bar{c}_i^t, N_{i,C}^t + 1)$ and $|\bar{\beta}_i^t - \beta_i| \le 2R(\bar{\beta}_i^t, t)$. □

Lemma A.4. *By running CCB with* $\gamma = 72 \ln \frac{2KT}{\delta}$ *for* T *time slots, with probability at least* $1 - \delta$ *we have the following results hold simultaneously:*

$$m_i > \check{m}_i^t, \forall i \in \mathcal{K}, \forall t \le T \tag{35}$$

$$c_i > \check{c}_i^t, \forall i \in \mathcal{K}, \forall t \le T \tag{36}$$

$$\hat{\beta}_i^t > \beta_i, \forall i \in \mathcal{K}, \forall t \le T \tag{37}$$

$$|\sum_{t=1}^{T}(\sum_{i\in\mathcal{L}_t}(m_i^t - \check{m}_i^t) + \sum_{i\notin\mathcal{L}_t}(c_i^t - \check{c}_i^t))| = O((K - L)\sqrt{KT \ln \frac{2KT}{\delta}}) \tag{38}$$

$$|\sum_{t=1}^{T}\sum_{i\in\mathcal{L}_t}(\hat{\beta}_i^t - \beta_i)| = O(K\sqrt{KT \ln \frac{2KT}{\delta}}) \tag{39}$$

Proof of Lemma A.4. Denote by Q_i^t be the event such that $|\bar{m}_i^t - m_i| > 2R(\bar{m}_i^t, N_{i,M}^t + 1)$, and $\bar{Q}_i^t$ be its complement. Let $\gamma = 72 \ln \frac{2KT}{\delta}$, obviously $r \ge 1$.

From Lemma A.3 we have:

$$Pr\{Q_i^t\} < \frac{\delta}{KT}$$

taking a union bound,

$$Pr\{\cup_{i,t} Q_i^t\} \le \sum_{t=1}^{T}\sum_{i=1}^{K} Pr\{Q_i^t\} < \delta$$

therefore,

$$Pr\{\cap_{i,t} \bar{Q}_i^t\} = 1 - Pr\{\cup_{i,t} Q_i^t\} > 1 - \delta$$

The above inequality states that for all i and t, at probability at least $1 - \delta$ we have:

$$|\bar{m}_i^t - m_i| \le 2R(\bar{m}_i^t, N_{i,M}^t + 1) \tag{40}$$

It follows that at probability at least $1 - \delta$,

$$m_i > \bar{m}_i^t - 2R(\bar{m}_i^t, N_{i,M}^t + 1) = \check{m}_i^t$$

(36) and (37) can be proved in the same way.

To prove (38), define two series of random variables:

$$Z_t = \sum_{i\in\mathcal{L}_t} m_i^t - \sum_{i\in\mathcal{L}_t} m_i, \quad Y_t = \sum_{l=1}^{t} Z_l$$

It can be seen that $\{Y_n\}$ is a sequence of independent variables, and $\mathbb{E}[Z_t|\mathcal{H}_{t-1}] = \mathbb{E}[Z_t] = 0$, where $\mathcal{H}_{t-1} = (Y_{t-1}, Y_{t-2}, \ldots, Y_1)$ is the historical information up to time $t - 1$. $\{Y_n\}$ is a martingale since

$$\begin{aligned}&\mathbb{E}[Y_{t+1}|Y_t, Y_{t-1}, \ldots, Y_1] \\ =&\mathbb{E}[Y_t + Z_{t+1}|Y_t, Y_{t-1}, \ldots, Y_1] \\ =&\mathbb{E}[Y_t|Y_t, Y_{t-1}, \ldots, Y_1] + \mathbb{E}[Z_{t+1}|Y_t, Y_{t-1}, \ldots, Y_1] \\ =&Y_t + \mathbb{E}[Z_{t+1}] = Y_t + 0 = Y_t\end{aligned}$$

Moreover, $|Y_t - Y_{t-1}| = |Z_t| \le L$. Let $d = L\sqrt{2T \ln \frac{2}{\delta}}$, according to Lemma A.1, we known that with probability at least $1 - \delta$:

$$|\sum_{t=1}^{T}\sum_{i\in\mathcal{L}_t} m_i^t - \sum_{i\in\mathcal{L}_t} m_i| \le L\sqrt{2T \ln \frac{2}{\delta}} \tag{41}$$

On the other hand, for all $i \in \mathcal{L}_t$ and $t \le T$, we get

$$\begin{aligned}|\check{m}_i^t - m_i| =&|\check{m}_i^t - \bar{m}_i^t + \bar{m}_i^t - m_i| \\ \le&|\check{m}_i^t - \bar{m}_i^t| + |\bar{m}_i^t - m_i| \\ \le&2R(\bar{m}_i^t, N_{i,M}^t + 1) + |\bar{m}_i^t - m_i| \\ \le&2R(\bar{m}_i^t, N_{i,M}^t + 1) + 2R(\bar{m}_i^t, N_{i,M}^t + 1) \\ =&4R(\bar{m}_i^t, N_{i,M}^t + 1)\end{aligned}$$

where the third inequality is due to (40).

Let $\tau(i, n)$ be the time slot that arm i is played for the nth time, we have

$$\begin{aligned}|\sum_{t=1}^{T}\sum_{i\in\mathcal{L}_t}(\check{m}_i^t - m_i)| &\le \sum_{t=1}^{T}\sum_{i\in\mathcal{L}_t} 4R(\bar{m}_i^t, N_{i,M}^t + 1) \\ &= \sum_{i=1}^{K}\sum_{n=1}^{N_{i,M}^{T+1}} 4R(\bar{m}_i^{\tau(i,n)}, n) \\ &= \sum_{i=1}^{K}\sum_{n=1}^{N_{i,M}^{T+1}} 4(\sqrt{\frac{\gamma\bar{m}_i^{\tau(i,n)}}{n}} + \frac{\gamma}{n}) \\ &\le \sum_{i=1}^{K}\sum_{n=1}^{N_{i,M}^{T+1}} 4(\sqrt{\frac{\gamma}{n}} + \frac{\gamma}{n})\end{aligned}$$

Note that when n is large enough,

$$1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \longrightarrow \ln n + C$$

and for all $n \geq 1$ it can be proved that

$$1 + \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{3}} + \cdots + \frac{1}{\sqrt{n}} < 2\sqrt{n}$$

Therefore,

$$\begin{aligned}
&|\sum_{t=1}^{T}\sum_{i\in\mathcal{L}_t}(\check{m}_i^t - m_i)| \leq \sum_{i=1}^{K}\sum_{n=1}^{N_{i,\mathrm{M}}^{T+1}} 4(\sqrt{\frac{\gamma}{n}} + \frac{\gamma}{n}) \\
=&O(\sum_{i=1}^{K}(\sqrt{\gamma N_{i,\mathrm{M}}^{T+1}} + \gamma \ln N_{i,\mathrm{M}}^{T+1})) \\
=&O(\sum_{i=1}^{K}\sqrt{\gamma N_{i,\mathrm{M}}^{T+1}} + \sum_{i=1}^{K}\gamma \ln N_{i,\mathrm{M}}^{T+1}) \\
=&O(\sqrt{(\sum_{i=1}^{K}\sqrt{\gamma N_{i,\mathrm{M}}^{T+1}})^2} + \gamma \ln \prod_{i=1}^{K} N_{i,\mathrm{M}}^{T+1}) \\
\leq&O(\sqrt{\sum_{i=1}^{K}1^2\sum_{i=1}^{K}\gamma N_{i,\mathrm{M}}^{T+1}} + \gamma \ln[(\prod_{i=1}^{K} N_{i,\mathrm{M}}^{T+1})^{\frac{1}{K}}]^K) \\
\leq&O(\sqrt{K}\sqrt{\sum_{i=1}^{K}\gamma N_{i,\mathrm{M}}^{T+1}} + \gamma K \ln[(\prod_{i=1}^{K} N_{i,\mathrm{M}}^{T+1})^{\frac{1}{K}}]) \\
\leq&O(\sqrt{K}\sqrt{\sum_{i=1}^{K}\gamma N_{i,\mathrm{M}}^{T+1}} + \gamma K \ln \frac{\sum_{i=1}^{K} N_{i,\mathrm{M}}^{T+1}}{K}) \\
=&O(\sqrt{K}\sqrt{\sum_{i=1}^{K}\gamma N_{i,\mathrm{M}}^{T+1}} + \gamma K \ln \frac{LT}{K}) \\
\leq&L\sqrt{2T\ln\frac{2}{\delta}} + O(\sqrt{K}\sqrt{\sum_{i=1}^{K}\gamma N_{i,\mathrm{M}}^{T+1}} + \gamma K \ln T) \\
=&O(L\sqrt{KT\ln\frac{2KT}{\delta}})
\end{aligned}$$

where the first inequality is due to Cauchy–Schwarz inequality, and the third inequality is from inequality of arithmetic and geometric mean. The fourth equality holds since $\sum_{i=1}^{K} N_{i,\mathrm{M}}^{T+1} = LT$.

Rewriting it in a compact way, we get

$$|\sum_{t=1}^{T}\sum_{i\in\mathcal{L}_t}(\check{m}_i^t - m_i)| \leq O(L\sqrt{KT\ln\frac{2KT}{\delta}}) \tag{42}$$

By the same reasoning, we can prove that with a probability $1-\delta$,

$$|\sum_{t=1}^{T}\sum_{i\notin\mathcal{L}_t}(c_i^t - c_i)| \leq (K-L)\sqrt{2T\ln\frac{2}{\delta}} \tag{43}$$

$$|\sum_{t=1}^{T}\sum_{i\notin\mathcal{L}_t}(\check{c}_i^t - c_i)| \leq O((K-L)\sqrt{KT\ln\frac{2KT}{\delta}}) \tag{44}$$

Based on (42) (43) (44), we have

$$\begin{aligned}
&|\sum_{t=1}^{T}(\sum_{i\in\mathcal{L}_t}(m_i^t - \check{m}_i^t) + \sum_{i\notin\mathcal{L}_t}(c_i^t - \check{c}_i^t))| \\
=&|\sum_{t=1}^{T}(\sum_{i\in\mathcal{L}_t} m_i^t + \sum_{i\notin\mathcal{L}_t} c_i^t - \sum_{i\in\mathcal{L}_t}\check{m}_i^t - \sum_{i\notin\mathcal{L}_t}\check{c}_i^t)| \\
=&|\sum_{t=1}^{T}(\sum_{i\in\mathcal{L}_t} m_i^t + \sum_{i\notin\mathcal{L}_t} c_i^t + \sum_{i\in\mathcal{L}_t} m_i - \sum_{i\in\mathcal{L}_t} m_i \\
&+ \sum_{i\notin\mathcal{L}_t} c_i - \sum_{i\notin\mathcal{L}_t} c_i - \sum_{i\in\mathcal{L}_t}\check{m}_i^t - \sum_{i\notin\mathcal{L}_t}\check{c}_i^t)| \\
\leq&|\sum_{t=1}^{T}\sum_{i\in\mathcal{L}_t}(m_i^t - m_i)| + |\sum_{t=1}^{T}\sum_{i\in\mathcal{L}_t}(\check{m}_i^t - m_i)| \\
&+ |\sum_{t=1}^{T}\sum_{i\notin\mathcal{L}_t}(c_i^t - c_i)| + |\sum_{t=1}^{T}\sum_{i\notin\mathcal{L}_t}(\check{c}_i^t - c_i)| \\
\leq&L\sqrt{2T\ln\frac{2}{\delta}} + O(L\sqrt{KT\ln\frac{2KT}{\delta}}) \\
&+ (K-L)\sqrt{2T\ln\frac{2}{\delta}} + O((K-L)\sqrt{KT\ln\frac{2KT}{\delta}}) \\
=&O((K-L)\sqrt{KT\ln\frac{2KT}{\delta}})
\end{aligned}$$

the last equality holds since $L \ll K$ and $L < K - L$. (39) can be proved in the same way.

Now we prove the theorem. Let $\mathbf{x}^*$ be the optimal solution to problem (1). From (39) we know that $\mathbf{x}^*$ is also a feasible solution to problem 11, i.e., $\hat{\boldsymbol{\beta}}^{\mathbf{t}\mathrm{T}}\mathbf{x}^* \geq \boldsymbol{\beta}^{\mathrm{T}}\mathbf{x}^* \geq h$. Then for all $t \leq T$ we have:

$$\begin{aligned}
\mathbf{x}^{\mathbf{t}\mathrm{T}}\check{\mathbf{m}}^{\mathbf{t}} + (\mathbf{1}-\mathbf{x}^{\mathbf{t}})^{\mathrm{T}}\check{\mathbf{c}}^{\mathbf{t}} &\leq \mathbf{x}^{*\mathrm{T}}\check{\mathbf{m}}^{\mathbf{t}} + (\mathbf{1}-\mathbf{x}^{*})^{\mathrm{T}}\check{\mathbf{c}}^{\mathbf{t}} \\
&\leq \mathbf{x}^{*\mathrm{T}}\mathbf{m}^{\mathbf{t}} + (\mathbf{1}-\mathbf{x}^{*})^{\mathrm{T}}\mathbf{c}^{\mathbf{t}}
\end{aligned} \tag{45}$$

Combining (38) (45), we get:

$$\begin{aligned}
Reg(T) =&|\sum_{t=1}^{T}(\sum_{i\in\mathcal{L}_t} m_i^t + \sum_{i\notin\mathcal{L}_t} c_i^t) - T(\mathbf{x}^{*\mathrm{T}}\mathbf{m}^{\mathbf{t}} + (\mathbf{1}-\mathbf{x}^{*})^{\mathrm{T}}\mathbf{c}^{\mathbf{t}})| \\
\leq&|\sum_{t=1}^{T}(\sum_{i\in\mathcal{L}_t} m_i^t + \sum_{i\notin\mathcal{L}_t} c_i^t - \sum_{i\in\mathcal{L}_t}\check{m}_i^t - \sum_{i\notin\mathcal{L}_t}\check{c}_i^t)| \\
=&O((K-L)\sqrt{KT\ln\frac{2KT}{\delta}})
\end{aligned}$$

On the other hand, since for all t, $\mathbf{x}^{\mathbf{t}}$ is a feasible solution to problem 11, i.e., $\hat{\boldsymbol{\beta}}^{\mathbf{t}\mathrm{T}}\mathbf{x}^{\mathbf{t}} \geq h$, and with (39), we have:

$$\begin{aligned}
Vio(T) =&|hT - \sum_{t=1}^{T}\sum_{i\in\mathcal{L}_t}\beta_i^t|^{+} \\
\leq&|\sum_{t=1}^{T}\sum_{i\in\mathcal{L}_t}\hat{\beta}_i^t - \sum_{t=1}^{T}\sum_{i\in\mathcal{L}_t}\beta_i^t| \\
=&|\sum_{t=1}^{T}\sum_{i\in\mathcal{L}_t}(\hat{\beta}_i^t - \beta_i^t)| \\
=&O(K\sqrt{KT\ln\frac{2KT}{\delta}})
\end{aligned}$$

This completes the proof. □

Appendix B. Proof of Theorem 4.1

It suffices to consider the case when $K \mod L \neq 0$, since we can set $\alpha = 1$ when $K \mod L = 0$.

Let us focus on the time slot when the exploration phase completes, i.e., when $t = 2NK\alpha/L$. From Hoeffding inequality (Appendix A of [12]), we know that $\forall i$,

$$Pr\{|\bar{m}_{i,11} - m_{i,11}| \leq r_t(m_{i,11})\} \geq 1 - \frac{2}{T^4} \tag{46}$$

$$Pr\{|\bar{c}_i - c_i| \leq r_t(c_i)\} \geq 1 - \frac{2}{T^4} \tag{47}$$

$$Pr\{|\bar{\beta}_i - \beta_i| \leq r_t(\beta_i)\} \geq 1 - \frac{2}{T^4} \tag{48}$$

where $r_t(m_{i,11}) = \sqrt{\frac{2\ln T}{2N\alpha(K/L-1)}}$, $r_t(c_i) = \sqrt{2\ln T/N\alpha}$, and $r_t(\beta_i) = \sqrt{\frac{2L\ln T}{2NK\alpha}}$ are the radius of confidence intervals.

Define the *clean event* to be the event that (46) (47) (48) hold for all arms simultaneously, and the "*bad event*" to be its complement. To analyze the regret and violation of the algorithm, it suffices to focus

on the clean event, since the contribution of the bad event can be neglected.[2]

Denote by $\mathcal{L}_t$ be the arm set selected when the exploration phase completes (assuming at time t). If $\mathcal{L}_t = \mathcal{L}^*$, then the regret will no longer increase in the exploitation phase. On the other hand, if $\mathcal{L}_t \neq \mathcal{L}^*$, then we must have,

$$\sum_{i\notin\mathcal{L}_t}\bar{c}_i + \sum_{i\in\mathcal{L}_t}\bar{m}_{i,11} < \sum_{i\notin\mathcal{L}}^{*}\bar{c}_i + \sum_{i\in\mathcal{L}}^{*}\bar{m}_{i,11} \tag{49}$$

Since this is a clean event, we have:

$$\begin{aligned}&\sum_{i\notin\mathcal{L}_t} c_i - (K-L)r_t(c_i) + \sum_{i\in\mathcal{L}_t} m_{i,11} - Lr_t(m_{i,11})\\ \leq &\sum_{i\notin\mathcal{L}_t}\bar{c}_i + \sum_{i\in\mathcal{L}_t}\bar{m}_{i,11}\end{aligned} \tag{50}$$

and

$$\begin{aligned}&\sum_{i\notin\mathcal{L}}^{*}\bar{c}_i + \sum_{i\in\mathcal{L}}^{*}\bar{m}_{i,11}\\ \leq &\sum_{i\notin\mathcal{L}}^{*} c_i + (K-L)r_t(c_i) + \sum_{i\in\mathcal{L}}^{*} m_{i,11} + Lr_t(m_{i,11})\end{aligned} \tag{51}$$

Combining (49)(50)(51), we get:

$$\begin{aligned}&\sum_{i\notin\mathcal{L}_t}\bar{c}_i + \sum_{i\in\mathcal{L}_t}\bar{m}_{i,11} - \sum_{i\notin\mathcal{L}}^{*}\bar{c}_i - \sum_{i\in\mathcal{L}}^{*}\bar{m}_{i,11}\\ \leq &2(K-L)r_t(c_i) + 2Lr_t(m_{i,11})\end{aligned} \tag{52}$$

The above inequation states that each time slot in the exploitation phase contributes at most $2(K-L)r_t(c_i) + 2Lr_t(m_{i,11})$ to regret.

Next, we consider the regret generated in the exploration phase. Note that $c_i^t \in [0,1]$, $m_{i,11}^t \in [0,1]$, $\forall t$, and there are $2NK\alpha/L$ time slots within N segments. It follows that the regret from exploration phase can be bounded by $2NK^2\alpha/L$.

Given that the regret to time slot T is the sum of regret from the two phases, we can bound it as follows:

$$\begin{aligned}Reg(T) &\leq \Big(2(K-L)r_t(c_i) + 2Lr_t(m_{i,11})\Big)(T - \frac{2NK\alpha}{L}) + \frac{2NK^2\alpha}{L}\\ &< \Big(2(K-L)r_t(c_i) + 2Lr_t(m_{i,11})\Big)T + \frac{2NK^2\alpha}{L}\\ &< 2KTr_t(m_{i,11}) + \frac{2NK^2\alpha}{L}\end{aligned} \tag{53}$$

The last inequation holds since $r_t(c_i) < r_t(m_{i,11})$. Note that the two summands are respectively monotonically decreasing and increasing in N, to minimize the regret we can set it so that they are approximately equal. Therefore, by setting:

$$\frac{2NK^2\alpha}{L} = 2KTr_t(m_{i,11})$$

we have,

$$N = (\frac{2L^2T^2\ln T}{K^2\alpha^3})^{\frac{1}{3}} \tag{54}$$

Substituting it into (53), we get:

$$Reg(T) = O\left((\frac{K^4}{L})^{\frac{1}{3}}T^{\frac{2}{3}}(\ln T)^{\frac{1}{3}}\right).$$

Similarly, for violation we have:

$$\begin{aligned}Vio(T) &= \sum_{t=1}^{T}[h - \sum_{i\in\mathcal{L}_t}\beta_i]^+\\ &\leq \sum_{t=1}^{T}[\sum_{i\in\mathcal{L}^*}\beta_i - \sum_{i\in\mathcal{L}_t}\beta_i]^+\\ &\leq \frac{2NK\alpha}{L}L + 2LTr_t(\beta_i)\\ &< 2NK\alpha + 2LT\sqrt{\frac{\ln T}{N\alpha}}\end{aligned} \tag{55}$$

Substituting (54) into the above inequation, we get:

$$Vio(T) = O\left(K^{\frac{1}{3}}L^{\frac{2}{3}}T^{\frac{2}{3}}(\ln T)^{\frac{1}{3}}\right)$$

The proof completes.

Appendix C. Proof of Theorem 4.2

Let $r_t(m_{i,11}) = \sqrt{2\ln T/n_t(i,\mathrm{M})}$, $r_t(c_i) = \sqrt{2\ln T/n_t(i,\mathrm{C})}$. From Hoeffding inequality, we know that (46) and (47) hold.

Again, define the *clean event* to be the event that (46) (47) hold for all arms simultaneously, and the "*bad event*" to be its complement. To prove the theorem, it suffices to focus on the clean event.

For each arm $i \in \mathcal{K}$, define the gap as follows:

$$\Delta(i) := (c_{i^*} - m_{i^*,11}) - (c_i - m_{i,11})$$

where i^* is the best arm, i.e., $i^* = \underset{i\in\mathcal{K}}{\mathrm{argmax}}\{c_i - m_{i,11}\}$. Then we have,

$$Reg(t) \leq tL(c_{i^*} - m_{i^*,11}) - \sum_{\tau=1}^{t}\sum_{i\in\mathcal{L}_t}(c_i - m_{i,11})$$

Let $R(t)$ be the RHS of the above inequality, i.e., $R(t) = tL(c_{i^*} - m_{i^*,11}) - \sum_{\tau=1}^{t}\sum_{i\in\mathcal{L}_t}(c_i - m_{i,11})$, and t_j be the time slot that arm j is eliminated. Fix arm j, for all $t \leq t_j$, the confidence intervals of j and i^* must overlap at time t, therefore,

$$\begin{aligned}\Delta(j) \leq &2r_{t_j}(m_{j,11}) + 2r_{t_j}(m_{i^*,11}) + 2r_{t_j}(c_j) + 2r_{t_j}(c_{i^*})\\ \leq &4r_{t_j}(m_{j,11}) + 4r_{t_j}(m_{i^*,11})\\ = &8r_{t_j}(m_{j,11})\end{aligned} \tag{56}$$

where the second inequality is for the fact that $n_{t_j}(j,\mathrm{M}) \leq n_{t_j}(j,\mathrm{C})$, and the last equality holds since $n_{t_j}(j,\mathrm{M}) = n_{t_j}(i^*,\mathrm{M})$.

Denote by $R(t,j)$ be the contribution of arm j to regret at time slot t, we can bound it as

$$\begin{aligned}R(t,j) = &n_t(j,\mathrm{M})\cdot\Delta(j)\\ \leq &8n_t(j,\mathrm{M})\cdot\sqrt{\frac{2\ln T}{n_{t_j}(j,\mathrm{M})}}\\ = &O(\sqrt{\ln T\cdot n_{t_j}(j,\mathrm{M})})\end{aligned}$$

Therefore,

$$R(t) = \sum_{j\in\mathcal{K}}R(t,j) \leq O(\sqrt{\ln T})\sum_{j\in\mathcal{K}}\sqrt{n_{t_j}(j,\mathrm{M})}$$

Since $f(x) = \sqrt{x}$ is a concave function and $\sum_{j\in\mathcal{K}} n_{t_j}(j,\mathrm{M}) = \frac{Lt}{2}$, by Jensen's Inequality we have,

$$\frac{1}{K}\sum_{j\in\mathcal{K}}\sqrt{n_{t_j}(j,\mathrm{M})} \leq \sqrt{\frac{1}{K}\sum_{j\in\mathcal{K}}n_{t_j}(j,\mathrm{M})} \leq \sqrt{\frac{Lt}{2K}}$$

It follows that

$$Reg(t) \leq R(t) \leq O(\sqrt{\ln T})K\sqrt{\frac{Lt}{2K}} = O(\sqrt{KLt\ln T})$$

Next, we prove the bound for violation. Let $i^\star$ be the arm with the largest arrival rate, i.e., $i^\star = \underset{i\in\mathcal{K}}{\mathrm{argmax}}\,\beta_i$. Then we have,

$$Vio(T) \leq \sum_{\tau=1}^{T}(L\beta_{i^\star} - \sum_{i\in\mathcal{L}_t}\beta_i)$$

[2] The probability that bad event occurs is $O(T^{-4})$.

Denote by $V(T)$ be the RHS of the above inequality, and t_j be the time slot that arm j is eliminated. Also denote by t^1 be the time slot that the first segment ends. Define the gap as:

$$\delta(i) := \beta_{i^\star} - \beta_i$$

Obviously,

$$\delta(i) \le 2r_{t^1}(i) + 2r_{t^1}(i^\star) = 4r_{t^1}(i)$$

On the other hand, let $V(j, t_j)$ be the contribution of arm j to violation at time t_j, we have,

$$\begin{aligned} V(j,t_j) =& n_{t_j}(j,\mathrm{M}) \cdot \delta(j) \\ \le& 4n_t(j,\mathrm{M}) \cdot \sqrt{\frac{L\ln T}{aK}} \\ =& O(n_t(j,\mathrm{M}) \cdot \sqrt{\frac{L\ln T}{aK}}) \end{aligned}$$

Therefore,

$$\begin{aligned} V(T) =& \sum_{j\in\mathcal{K}} V(j,t_j) \le O(\sum_{j\in\mathcal{K}} n_t(j,\mathrm{M}) \cdot \sqrt{\frac{L\ln T}{aK}}) \\ \le& O(\frac{Lt}{2} \cdot \sqrt{\frac{L\ln T}{aK}}) \\ =& O(t\sqrt{\frac{L^3\ln T}{aK}}) \\ \le& O(t\sqrt{KL\ln T}) \end{aligned}$$

where the last inequality is due to $L <= K$. The proof completes.

Appendix D. Proof of Lemma 4.1

From (56) we known that $\Delta(j) \le 8r_{t_j}(m_{j,11})$. Since for each $j \ne i^*$, we have $n_T(j, M) = n_{t_j}(j, M)$, it follows that:

$$\Delta(j) \le O(n_T(j,M)) = O(\sqrt{\frac{\ln T}{n_T(j,M)}})$$

Therefore,

$$n_T(j,M) \le O(\frac{\ln T}{\Delta(j)^2})$$

and

$$R(T,j) = \Delta(j) \cdot n_T(j,M) \le O(\frac{\ln T}{\Delta(j)})$$

Let $\mathcal{L}^*$ be the optimal set of arms to the online caching problem, then we have:

$$R(T) = \sum_{j\notin\mathcal{L}}^{*} R(T,j) \le O(\ln T) \cdot \sum_{j\notin\mathcal{L}}^{*} \frac{1}{\Delta(j)}$$

Now let us fix some ϵ, then the regret consists of two parts: (1) each arm j with $\Delta(j) \le \epsilon$ contributes at most ϵ per time slot, for a total of $LT\epsilon$ by L arms; (2) each arm j with $\Delta(j) > \epsilon$ contributes $R(T,j) \le O(\frac{\ln T}{\Delta(j)}) \le O(\frac{\ln T}{\epsilon})$, under the clean event. Combining these two parts and under the clean event, we have:

$$R(T) \le O(LT\epsilon + \frac{\ln T}{\epsilon})$$

Note that the above inequality holds for any ϵ. To minimize the RHS, we can set $LT\epsilon = \frac{\ln T}{\epsilon}$, and this leads us to the following bound:

$$Reg(T) \le O(\sqrt{KLT\ln T})$$

The proof completes.

References

[1] A.F. Florian Scheck, A. Freyberg, 5G: A Key Requirement for Autonomous Driving—Really? https://www.kearney.com/communications-media-technology/article/-/insights/5g-a-key-requirement-for-autonomous-driving-really-.

[2] Cloud AR/VR Whitepaper, https://www.gsma.com/futurenetworks/wiki/cloud-ar-vr-whitepaper/.

[3] Y. Mao, C. You, J. Zhang, K. Huang, K.B. Letaief, A survey on mobile edge computing: The communication perspective, IEEE Commun. Surv. Tutor. 19 (4) (2017) 2322–2358.

[4] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Neal, et al., Mobile-Edge Computing Introductory Technical White Paper, in: White Paper, Mobile-Edge Computing (MEC) Industry Initiative, vol. 29, 2014, pp. 854–864.

[5] A. Alwarafy, K.A. Al-Thelaya, M. Abdallah, J. Schneider, M. Hamdi, A survey on security and privacy issues in edge-computing-assisted internet of things, IEEE Internet Things J. 8 (6) (2020) 4004–4022.

[6] S. Zhong, S. Guo, H. Yu, Q. Wang, Cooperative service caching and computation offloading in multi-access edge computing, Comput. Netw. 189 (2021) 107916.

[7] J. Xu, L. Chen, P. Zhou, Joint service caching and task offloading for mobile edge computing in dense networks, in: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, IEEE, 2018, pp. 207–215.

[8] L. Chen, C. Shen, P. Zhou, J. Xu, Collaborative service placement for edge computing in dense small cell networks, IEEE Trans. Mob. Comput. (2019).

[9] N. Yu, Q. Xie, Q. Wang, H. Du, H. Huang, X. Jia, Collaborative service placement for mobile edge computing applications, in: 2018 IEEE Global Communications Conference, GLOBECOM, IEEE, 2018, pp. 1–6.

[10] K. Poularakis, J. Llorca, A.M. Tulino, I. Taylor, L. Tassiulas, Service placement and request routing in MEC networks with storage, computation, and communication constraints, IEEE/ACM Trans. Netw. 28 (3) (2020) 1047–1060.

[11] W. Chen, Y. Wang, Y. Yuan, Combinatorial multi-armed bandit: General framework and applications, in: International Conference on Machine Learning, PMLR, 2013, pp. 151–159.

[12] A. Slivkins, et al., Introduction to multi-armed bandits, Found. Trends Mach. Learn. 12 (1–2) (2019) 1–286.

[13] R. Kleinberg, A. Slivkins, E. Upfal, Multi-armed bandits in metric spaces, in: Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing, STOC '08, 2008, pp. 681–690.

[14] K. Chen, K. Cai, L. Huang, J.C. Lui, Beyond the click-through rate: web link selection with multi-level feedback, in: Proceedings of the 27th International Joint Conference on Artificial Intelligence, 2018, pp. 3308–3314.

[15] W. Chu, P. Yu, Z. Yu, J.C. Lui, Y. Lin, Online optimal service selection, resource allocation and task offloading for multi-access edge computing: A utility-based approach, IEEE Trans. Mob. Comput. (2022).

[16] L. Dong, W. He, H. Yao, Task offloading and resource allocation for tasks with varied requirements in mobile edge computing networks, Electronics 12 (2) (2023) 366.

[17] X. Chen, L. Jiao, W. Li, X. Fu, Efficient multi-user computation offloading for mobile-edge cloud computing, IEEE/ACM Trans. Netw. 24 (5) (2015) 2795–2808.

[18] J. Huang, F. Qian, A. Gerber, Z.M. Mao, S. Sen, O. Spatscheck, A close examination of performance and power characteristics of 4G LTE networks, in: Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, 2012, pp. 225–238.

[19] A.S. Uluagac, CRAWDAD Data Set Gatech/fingerprinting (v. 2014-06-09, 2014, https://crawdad.org/gatech/fingerprinting/20140609/realtestbed/index.html.

[20] W. Chu, Z. Yu, J.C. Lui, Y. Lin, Jointly optimizing throughput and content delivery cost over lossy cache networks, IEEE Trans. Commun. 69 (6) (2021) 3846–3863.

[21] F.A. Salaht, F. Desprez, A. Lebre, An overview of service placement problem in fog and edge computing, ACM Comput. Surv. 53 (3) (2020) 1–35.

[22] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, J.P. Jue, All one needs to know about fog computing and related edge computing paradigms: A complete survey, J. Syst. Archit. 98 (2019) 289–330.

[23] C. Li, Y. Xue, J. Wang, W. Zhang, T. Li, Edge-oriented computing paradigms: A survey on architecture design and system management, ACM Comput. Surv. 51 (2) (2018) 1–34.

[24] O. Ascigil, T.K. Phan, A.G. Tasiopoulos, V. Sourlas, I. Psaras, G. Pavlou, On uncoordinated service placement in edge-clouds, in: 2017 IEEE International Conference on Cloud Computing Technology and Science, CloudCom, IEEE, 2017, pp. 41–48.

[25] H.-J. Hong, P.-H. Tsai, A.-C. Cheng, M.Y.S. Uddin, N. Venkatasubramanian, C.-H. Hsu, Supporting internet-of-things analytics in a fog computing platform, in: 2017 IEEE International Conference on Cloud Computing Technology and Science, CloudCom, IEEE, 2017, pp. 138–145.

[26] Z. Ning, P. Dong, X. Wang, S. Wang, X. Hu, S. Guo, T. Qiu, B. Hu, R.Y. Kwok, Distributed and dynamic service placement in pervasive edge computing networks, IEEE Trans. Parallel Distrib. Syst. 32 (6) (2020) 1277–1292.

[27] Z. Xu, L. Zhou, S.C.-K. Chau, W. Liang, Q. Xia, P. Zhou, Collaborate or separate? Distributed service caching in mobile edge clouds, in: IEEE INFOCOM 2020-IEEE Conference on Computer Communications, IEEE, 2020, pp. 2066–2075.

[28] P. Kayal, J. Liebeherr, Distributed service placement in fog computing: An iterative combinatorial auction approach, in: 2019 IEEE 39th International Conference on Distributed Computing Systems, ICDCS, IEEE, 2019, pp. 2145–2156.

[29] T. Nishio, R. Shinkuma, T. Takahashi, N.B. Mandayam, Service-oriented heterogeneous resource sharing for optimizing service latency in mobile cloud, in: Proceedings of the First International Workshop on Mobile Cloud Computing & Networking, 2013, pp. 19–26.
[30] V.B.C. Souza, W. Ramírez, X. Masip-Bruin, E. Marín-Tordera, G. Ren, G. Tashakor, Handling service allocation in combined fog-cloud scenarios, in: 2016 IEEE International Conference on Communications, ICC, IEEE, 2016, pp. 1–5.
[31] J. Zhao, X. Sun, Q. Li, X. Ma, Edge caching and computation management for real-time internet of vehicles: an online and distributed approach, IEEE Trans. Intell. Transp. Syst. 22 (4) (2020) 2183–2197.
[32] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, K.K. Leung, Dynamic service placement for mobile micro-clouds with predicted future costs, IEEE Trans. Parallel Distrib. Syst. 28 (4) (2016) 1002–1016.
[33] X. Li, X. Zhang, T. Huang, Asynchronous online service placement and task offloading for mobile edge computing, in: 2021 18th Annual IEEE International Conference on Sensing, Communication, and Networking, SECON, IEEE, 2021, pp. 1–9.
[34] B. Gao, Z. Zhou, F. Liu, F. Xu, B. Li, An online framework for joint network selection and service placement in mobile edge computing, IEEE Trans. Mob. Comput. (2021).
[35] T. Liu, S. Ni, X. Li, Y. Zhu, L. Kong, Y. Yang, Deep reinforcement learning based approach for online service placement and computation resource allocation in edge computing, IEEE Trans. Mob. Comput. (2022).
[36] T. Ouyang, Z. Zhou, X. Chen, Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing, IEEE J. Sel. Areas Commun. 36 (10) (2018) 2333–2345.
[37] Y. Zhang, L. Jiao, J. Yan, X. Lin, Dynamic service placement for virtual reality group gaming on mobile edge cloudlets, IEEE J. Sel. Areas Commun. 37 (8) (2019) 1881–1897.
[38] Q. Zhang, Q. Zhu, M.F. Zhani, R. Boutaba, J.L. Hellerstein, Dynamic service placement in geographically distributed clouds, IEEE J. Sel. Areas Commun. 31 (12) (2013) 762–772.
[39] L. Wang, L. Jiao, T. He, J. Li, H. Bal, Service placement for collaborative edge applications, IEEE/ACM Trans. Netw. 29 (1) (2020) 34–47.
[40] P. Yang, N. Zhang, S. Zhang, L. Yu, J. Zhang, X. Shen, Content popularity prediction towards location-aware mobile edge caching, IEEE Trans. Multimed. 21 (4) (2018) 915–929.
[41] X. Vasilakos, V.A. Siris, G.C. Polyzos, Addressing niche demand based on joint mobility prediction and content popularity caching, Comput. Netw. 110 (2016) 306–323.
[42] S. Zhang, P. He, K. Suto, P. Yang, L. Zhao, X. Shen, Cooperative edge caching in user-centric clustered mobile networks, IEEE Trans. Mob. Comput. 17 (8) (2017) 1791–1805.
[43] W. Han, A. Liu, V.K. Lau, Dual-mode user-centric open-loop cooperative caching for backhaul-limited small-cell wireless networks, IEEE Trans. Wireless Commun. 18 (1) (2018) 532–545.
[44] N. Uniyal, A. Bravalheri, X. Vasilakos, R. Nejabati, D. Simeonidou, W. Featherstone, S. Wu, D. Warren, Intelligent mobile handover prediction for zero downtime edge application mobility, in: 2021 IEEE Global Communications Conference, GLOBECOM, IEEE, 2021, pp. 1–6.
[45] C. Zhong, M.C. Gursoy, S. Velipasalar, Deep reinforcement learning-based edge caching in wireless networks, IEEE Trans. Cogn. Commun. Netw. 6 (1) (2020) 48–61.
[46] S. Chen, Z. Yao, X. Jiang, J. Yang, L. Hanzo, Multi-agent deep reinforcement learning-based cooperative edge caching for ultra-dense next-generation networks, IEEE Trans. Commun. 69 (4) (2020) 2441–2456.
[47] Z. Zheng, L. Song, Z. Han, G.Y. Li, H.V. Poor, A stackelberg game approach to proactive caching in large-scale mobile edge networks, IEEE Trans. Wireless Commun. 17 (8) (2018) 5198–5211.
[48] W. Huang, W. Chen, H.V. Poor, Request delay-based pricing for proactive caching: A stackelberg game approach, IEEE Trans. Wireless Commun. 18 (6) (2019) 2903–2918.
[49] G. Qiao, S. Leng, S. Maharjan, Y. Zhang, N. Ansari, Deep reinforcement learning for cooperative content caching in vehicular edge computing and networks, IEEE Internet Things J. 7 (1) (2019) 247–257.
[50] Z. Wang, Y. Wei, F.R. Yu, Z. Han, Utility optimization for resource allocation in multi-access edge network slicing: a twin-actor deep deterministic policy gradient approach, IEEE Trans. Wireless Commun. 21 (8) (2022) 5842–5856.
[51] T. Ouyang, R. Li, X. Chen, Z. Zhou, X. Tang, Adaptive user-managed service placement for mobile edge computing: An online learning approach, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, IEEE, 2019, pp. 1468–1476.
[52] Open Edge Computing, http://openedgecomputing.org.
[53] Openfog, https://opcfoundation.org/markets-collaboration/openfog/.
[54] V. Farhadi, F. Mehmeti, T. He, T.F. La Porta, H. Khamfroush, S. Wang, K.S. Chan, K. Poularakis, Service placement and request scheduling for data-intensive applications in edge clouds, IEEE/ACM Trans. Netw. 29 (2) (2021) 779–792.
[55] D. Vial, S. Shakkottai, R. Srikant, Robust multi-agent multi-armed bandits, in: Proceedings of the Twenty-Second International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing, 2021, pp. 161–170.
[56] P. Landgren, V. Srivastava, N.E. Leonard, Distributed cooperative decision making in multi-agent multi-armed bandits, Automatica 125 (2021) 109445.
[57] S. Hossain, E. Micha, N. Shah, Fair algorithms for multi-agent multi-armed bandits, Adv. Neural Inf. Process. Syst. 34 (2021) 24005–24017.
[58] K. Azuma, Weighted sums of certain dependent random variables, Tohoku Math. J. Sec. Ser. 19 (3) (1967) 357–367, http://dx.doi.org/10.2748/tmj/1178243286.
[59] R. Kleinberg, A. Slivkins, E. Upfal, Multi-armed bandits in metric spaces, in: Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing, 2008, pp. 681–690.
[60] A. Badanidiyuru, R. Kleinberg, A. Slivkins, Bandits with knapsacks, J. ACM 65 (3) (2018) 1–55.

Weibo Chu received the B.S. degree in software engineering in 2005 and the Ph.D. degree in control science and engineering in 2013, both from Xi'an Jiaotong University, Xi'an, China. He has participated in various research and development projects on network testing, performance evaluation and troubleshooting, and gained extensive experiences in the development of networked systems for research and engineering purposes. From 2011–2012 he worked as a visiting researcher at Microsoft Research Asia, Beijing. From 2013 he was with the School of Computer Science and Technology, Northwestern Polytechnical University. His research interests include internet measurement and modeling, traffic analysis and performance evaluation.

Xiaoyan Zhang received the B.E. degree in 2020 and M.S. degree in 2023, both in computer science from Northwestern Polytechnical University, Xi'an, China. Her research interests include task scheduling and service management for edge/cloud computing systems.

Xinming Jia received the B.E. degree from Northwestern Polytechnical University, Xi'an, China, in 2021. He is currently working toward his M.S. degree at the School of Computer Science and Technology, Northwestern Polytechnical University, Xi'an, China. His research interests include resource management and incentive mechanisms design for edge computing systems.

John C.S. Lui received the Ph.D. degree in computer science from UCLA. He is currently a professor in the Department of Computer Science and Engineering at The Chinese University of Hong Kong. His current research interests include communication networks, network/system security (e.g., cloud security, mobile security, etc.), network economics, network sciences (e.g., online social networks, information spreading, etc.), cloud computing, large-scale distributed systems and performance evaluation theory. He serves in the editorial board of IEEE/ACM Transactions on Networking, IEEE Transactions on Computers, IEEE Transactions on Parallel and Distributed Systems, Journal of Performance Evaluation and International Journal of Network Security. He was the chairman of the CSE Department from 2005 to 2011. He received various departmental teaching awards and the CUHK Vice-Chancellor's Exemplary Teaching Award. He is also a corecipient of the IFIP WG 7.3 Performance 2005 and IEEE/IFIP NOMS 2006 Best Student Paper Awards. He is an elected member of the IFIP WG 7.3, fellow of the ACM, fellow of the IEEE, and croucher senior research fellow.

Zhiyong Wang received his B.E. degree in 2021 from Huazhong University of Science and Technology, Wuhan, China. Since August 2021, he has pursued his Ph.D. degree in the Department of Computer Science & Engineering at The Chinese University of Hong Kong, Hong Kong. His research interests include bandits, reinforcement learning and their applications in computer networks and recommender systems.