Intelligent Task Caching in Edge Cloud via Bandit Learning

Yiming Miao, Yixue Hao, Hamid Gharavi, *Life Fellow, IEEE,*
Min Chen, *Fellow, IEEE,* Kai Hwang, *Life Fellow, IEEE*

Abstract—Task caching, based on edge cloud, aims to meet the latency requirements of computation-intensive and data-intensive tasks (such as augmented reality). However, current task caching strategies are generally based on the unrealistic assumption of knowing the pattern of user task requests and ignoring the fact that a task request pattern is more user specific (e.g., the mobility and personalized task demand). Moreover, it disregards the impact of task size and computing amount on the caching strategy. To investigate these issues, in this paper, we first formalize the task caching problem as a non-linear integer programming problem to minimize task latency. We then design a novel intelligent task caching algorithm based on a multi-armed bandit algorithm, called M-adaptive upper confidence bound (M-AUCB). The proposed caching strategy cannot only learn the task patterns of mobile device requests online, but can also dynamically adjust the caching strategy to incorporate the size and computing amount of each task. Moreover, we prove that the M-AUCB algorithm achieves a sublinear regret bound. The results show that, compared with other task caching schemes, the M-AUCB algorithm reduces the average task latency by at least 14.8%.

Index Terms—Edge Caching, Task Caching, Edge Cloud Computing, Bandit learning

1 Introduction

With the development of cloud computing technologies, mobile devices are capable of offloading computing tasks to a remote cloud in order to overcome the limitation of a mobile device's computing ability and battery capacity [1], [2]. In addition, the increasing popularity of applications, such as virtual reality and augmented reality, demands more computing and storage resources for mobile devices. These applications are generally delay-sensitive and computation-intensive [3]. Thus, when utilizing traditional mobile cloud computing technologies, it cannot meet the necessary requirements to offload these applications. In particular, due to long network distances and congestion of back-bone networks, offloading tasks to the cloud can cause a substantial delay that can impact the quality of service (QoS) [4].

Fortunately, by offering computing and storage capabilities on access networks, edge computing can play a crucial role in executing computing-intensive and data-intensive tasks at the network edge [5]. Thanks to a shorter distance between the edge server and mobile device, edge computing enables low delay, as well as better exploitation of users' information. Caching the tasks or contents requested by a mobile device on the edge cloud would make it possible to meet the requirements of delay-sensitive tasks [6], [7]. Given virtual reality scene rendering as an example, we can cache the scene rendering and popular videos on an edge cloud during a non-peak period that can reduce the latency of a mobile device to obtain the contents.

Specifically, there are two categories of caching in edge clouds: content caching and task caching. Content caching refers to caching contents such as popular videos on the edge cloud [8], [9]. For instance, when a mobile device requests contents, edge cloud can directly deliver the requested content to the user's device given that such content has already been cached on the edge cloud. Consequently, this reduces the latency of a mobile device to obtain its requested contents. Content caching has been widely investigated, including where to cache [10], what to cache [11], and how to cache [12], [13].

The task caching aspect of the edge cloud is concerned mainly with caching the code and the processing environment needed for task execution on the edge cloud [6], [14]. Furthermore, existing research indicates that caching a task on the edge cloud reduces the task duration as well as the energy consumption of mobile devices [15], [16]. Nonetheless, despite recent progress, the task caching strategies are still facing a number of challenges.

- *Unknown Task Demand:* Existing works in task caching either assume the task demand is known a prior [17], or using the existing prediction schemes based on the contents [18]. By comparison, we assume the task demands are unpredictable. This is because, in contrast to the content caching, a task request mode depends highly on the users operational environments (such as users' personalized task demand, time, users' location), which are hard to predict. Moreover, the network environment is dynamic and the transmission of tasks can not be predicted accurately.
- *Task Heterogeneity:* Different tasks have diverse size and required computations, resulting in different task latency. Thus, the task caching scheme needs to take into consideration both the size of a task and its computation amount.

- *Y. Miao and Y. Hao are with School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China (yimingmiao@hust.edu.cn, yixuehao@hust.edu.cn).*
- *M. Chen is with School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China and the Shenzhen Institute of Artificial Intelligence and Robotics for Society, Shenzhen 518172, China. (minchen2012@hust.edu.cn).*
- *H. Gharavi is with National Institute of Standards and Technology (NIST), Gaithersburg, MD 20899-8920 USA.(hamid.gharavi@nist.gov).*
- *K. Hwang is with the Shenzhen Institute of Artificial Intelligence and Robotics for Society, and with School of Data Science (SDS), The Chinese University of Hong Kong, Shenzhen (CUHK-SZ), Shenzhen 518172, China. (hwangkai@cuhk.edu.cn).*

- *Limited Computing and Caching Resource:* While the edge cloud has the advantage over mobile devices in terms of caching capacity and computing power, this would be at the expense of not being able to cache all types of computing tasks.

To address the above challenges, in this paper, we investigate the online task caching scheme under a realistic assumption of not knowing the task request pattern of a mobile device, while incorporating the influence of the heterogeneous task and the limited resource of the edge cloud. In our approach, we initially formalize the task caching problem on the edge cloud as a non-linear integer programming problem to minimize task latency. Then, to solve the problem we propose an intelligent task caching algorithm based on a multi-armed bandit algorithm, called M-adaptive upper confidence bound (M-AUCB). This algorithm can achieve an optimal compromise between exploration (i.e., to cache the task with unknown latency to learn the task request pattern) and exploitation (i.e., to cache the task with high estimated user demand to minimize the task latency). We further analyze the bound losses of the M-AUCB algorithm and its closeness to the optimal caching strategy (i.e., with prior knowledge of the task demand). Finally, we present the results by verifying the M-AUCB's ability to minimize the delay of the computing task.

In summary, the main contributions of this paper include:

- Formalizing the task caching problem in order to minimize the task latency as a non-linear integer programming problem. The problem will factor in the task request pattern, which is usually unknown at the edge cloud. It also incorporates the effect of the task size and computing amount.
- Developing an intelligent task caching algorithm, called M-AUCB algorithm. The proposed caching strategy is capable of learning the pattern of task request from a mobile device online. In addition, it provides an ability to adjust the caching strategy dynamically according to the size and computing amount of a task. Furthermore, we prove the boundedness of the algorithm and it is closeness to the optimal caching strategy.
- Carrying out experiments to evaluate the performance of the intelligent task caching schemes. The experimental results indicate that our scheme can reduce the average task latency by at least 14.8%.

The rest of the paper is organized as follows. In Section 2, we review related works. The system model and problem formulation are presented in Section 3. In Section 4, we give the intelligent task caching scheme. Our experimental results and discussions are given in Section 5. Finally, Section 6 presents the conclusion of the paper.

2 Related Work

With the rapid growth of mobile devices and new mobile applications (e.g., augmented reality and autonomous driving), remote cloud centric systems have difficulty in meeting the computing requirements of low-latency applications. Fortunately, with the development of edge cloud, the servers deployed on the edge of the network are close to the users, and have certain storage and computing capabilities, which can meet the application with low latency. Therefore, the code and running environment required by mobile applications can be cached in edge cloud (i.e., task caching) in advance in off-peak hours, which can achieve localized task processing and reduce the latency. For example, for the augmented reality application, visual recognition models can be cached in the edge cloud in advance, so that visual classification can be performed before the augmented information is delivered to the user.

Specifically, task caching refers to cache the code and running environment needed for task execution. Task caching is also known as task deployment, service caching and service placement. For task caching, a key issue is which tasks are cached in the edge cloud to minimize the delay for user. To solve this problem, in [14], through joint optimization of task caching and offloading, the energy efficient scheme is proposed. Furthermore, considering the limited storage, communication and computing resources of the edge cloud, it cannot cache all tasks. The authors of [20] and [19] use sub-modular optimization to give the near-optimal service placement and request scheduling scheme. For upcoming computations, Mohan et al. [21] propose an efficient task deployment scheme using the edge and fog resource. Although the limitation of storage, communication and computing resources of the edge cloud is considered in these works, it is assumed that the user's request to the task is the static request mode (i.e., the probability of the user's request to different tasks is constant and known). In practice, different users' requests for different tasks vary with time (i.e., dynamic request mode). In other words, the user's request pattern for tasks is priori unknown and time-varying.

Considering that the user's requests for tasks are priori unknown and time-varying, it is a challenge to cache which tasks in the edge cloud. In order to solve this problem, there are two schemes that exist of dynamic request pattern in content caching: (i) predict the request pattern; (ii) use an online algorithm to make decisions based on observed user's requests in the edge cloud. For the first scheme, a lot of works have designed a content caching scheme through the prediction of content popularity. However, this scheme needs a training set with known content popularity and can only learn the content popularity in the training phase. Furthermore, compared with the content, task requirements are more difficult to predict because they are more diverse and time-varying. Therefore, this scheme is not suitable for task caching.

Considering the second scheme, an optimal task caching scheme is achieved through online learning of content requests. Multi-armed bandit learning (MAB) is an effective online learning strategy and it has been widely used in wireless networks, such as content caching in edge cloud, online network slice broker and mobility management in ultra dense networks. This is because MAB can make nearly optimal online decisions for uncertain information (such as user's request pattern) by balancing exploration and exploitation, that is, by learning unknown information (i.e., exploration) and using learned information (i.e., exploration). For example, for content caching, Pascos et al. [22] designed an online gradient ascent content caching scheme for non-stationary file requests. It can minimize the learning regret and ensure the system's performance. The authors give context-aware proactive content caching using the contextual MAB algorithm in [13]. For service caching, considering that the service provider needs to pay edge cloud for service placement, Chen et al. [23] designed a spatio-temporal edge service placement scheme by using bandit learning, which can maximize the maximum utility of the service provider.

As opposed to existing works, in this paper, we model the task caching problem as a MAB problem. We not only consider task

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2020.3047417, IEEE Transactions on Network Science and Engineering

TABLE 1
Comparison of several task caching schemes

Scheme in References	Online/ Offline Learning	Size-aware	Computation -aware	Demand uncertain	Category
[19]	Offline	Yes	Yes	No	Task caching
[20]	Offline	Yes	Yes	No	Task caching
[21]	Offline	Yes	No	No	Task caching
[14]	Offline	Yes	No	No	Task caching
[22]	Online	Yes	No	Yes	Content caching
[13]	Online	Yes	No	Yes	Content caching
[23]	Online	No	No	Yes	Task caching
M-AUCB	Online	Yes	Yes	Yes	Task caching

demand as prior unknown, but also consider the heterogeneity of task, including the size and computing amount. By observing the number of user requests to the tasks in real time, our algorithm can learn the request pattern of tasks online, and adapt to the task size and computing amount. Furthermore, we give a comparison between the scheme proposed in this paper and the related works, as shown in Table 1. From the table, we can see that the M-AUCB scheme has better performance.

3 System Model and Problem Formulation

In this section, we give the system model and problem formulation. Specifically, we give the task caching model under consideration the limitation of the computing and storage capacity of edge cloud, and the unknown task request pattern of mobile devices.

3.1 System Overview

In this paper, we consider task caching in an edge computing ecosystem that includes multiple mobile devices communicating with an edge cloud over a wireless channel. To explain task caching in edge cloud more clearly, we give an example, as shown in Fig. 1. In this figure, we assume the remote cloud (i.e., cloud service) has four tasks (i.e., services). Considering the limited computing and storage capacity of the edge cloud, it can only cache one task. There are two mobile device users within the coverage of edge cloud, Alice and Bob, where Alice requests tasks 1 and 2, and Bob requests tasks 1 and 3. When task 1 is cached on edge cloud, Alice and Bob can obtain the requested task 1 through edge cloud. The requested tasks 2 and 3 need to be processed in the remote cloud. Thus, when the edge cloud receives the user's task request, it needs to decide which task to cache on the edge cloud, which can minimize the latency of task acquisition.

Specifically, each mobile device requests a computing task, e.g., video streaming, virtual reality, and/or mobile gaming. Considering that these computational tasks are computing-intensive and data-intensive tasks and the computing capacity and battery life of mobile devices are limited, we assume that mobile devices themselves cannot handle this task. Thus, similar as the works in [19], [20], in this paper we only consider task caching and processing on edge cloud or remote cloud. We assume that the edge computing system consists of N mobile devices, K tasks in the remote cloud (e.g., augmented reality) and one on the edge cloud. We denote the set of mobile devices and tasks by $\mathcal{N} = \{1, 2, \cdots, N\}$, $\mathcal{K} = \{1, 2, \cdots, K\}$, respectively. Furthermore, we describe the main notations used in this paper in Table 2.

TABLE 2
The summary table of importation notations

Notation	Meaning
$\mathcal{N}$	set of mobile devices
K	set of tasks
ω_k	computation amount of the task Q_k
s_k	input data size of the task Q_k.
a_k^t	indicates whether the task Q_k is cached on the edge cloud.
$d_{n,k}^t$	the number of request of mobile device n for the task Q_k in time slot t.
N_t	the number of mobile devices that can access the edge cloud at time slot t.
λ_k^t	the number of request for task Q_k at time slot t.
f_{ec}^k	the CPU frequency of the edge cloud assigned to the task Q_k.
f_{rc}^k	the CPU frequency of the remote cloud assigned to the task Q_k.
τ^t	the wireless transmission rate.
r_1^t	the backbone transmission rate.
r_2^t	the round-trip time to the remote cloud.
C	the maximum storage capacity of edge cloud.
F_{ec}	the maximum processing power of edge cloud.

We should point out that, compared with the remote cloud, the edge cloud has a limited computing and storage capacity. Furthermore, with respect to offloading the computation tasks, the edge cloud needs to have sufficient computing and storage resources in order to execute them. Therefore, we assume that the edge cloud cannot execute all the tasks requested by a mobile device (i.e., when a service requires handling a task, which is not cached on the edge cloud and the task cannot be executed). Under these conditions, such a task will be referred to the remote cloud for offloading and processing. Therefore, to reduce task latency as much as possible, we need to identify which task should be cached on the edge cloud. Finally, for the sake of implementation, we consider that the task caching system operates in discrete time $t = 1, 2, \cdots, T$, where T denotes the finite time horizon.

3.2 Computation Task Caching

We first give the description of the computation task where we consider an independent task caching. According to [24], [25],

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2020.3047417, IEEE Transactions on Network Science and Engineering

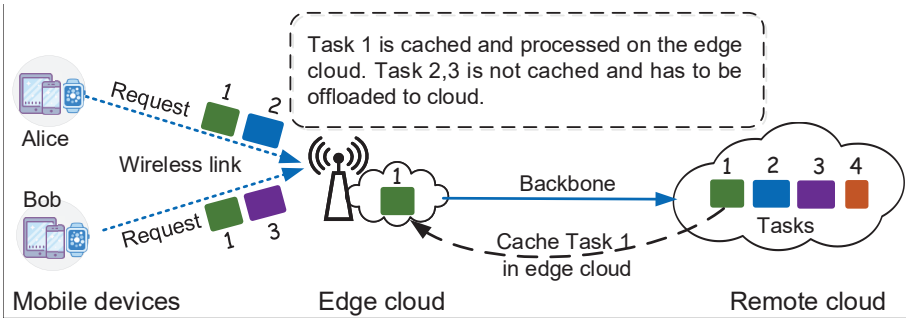


Fig. 1. An example of task caching in edge cloud.

each task, Q_k, can be described by two parameters: required computation amount ω_k and input data size s_k, where ω_k [cycles] is the computing amount of the task (i.e., the total number of CPU cycles needed to complete the task) and s_k (in bits) is the size of the computation task input data (i.e., the amount of data content, such as the processing code and data to be delivered to the edge cloud or cloud). Moreover, we can obtain the value of ω_k and s_k through profiling the task execution [26]. For example, as for video transcoding, ω_i is the computing resource needed in video transcoding and s_i is the data size of video.

Furthermore, compared to the remote cloud, which can process all computing tasks, the edge cloud has limited computing and storage capacity. Therefore, it can only cache some of the tasks. Under these assumptions, a user's task cached on the edge cloud is processed by the edge cloud. When the task is not cached on the edge cloud, it needs to be processed in the remote cloud. Thus, we define the integer task caching decision variable at time slot t as $a_k^t \in \{0, 1\}$, where,

$$a_k^t = \begin{cases} 1 & \text{The task } k \text{ is cached on edge cloud in time slot } t, \\ 0 & \text{The task } k \text{ is not cached on edge cloud in time slot } t. \end{cases} \tag{1}$$

Because of the limitation of the storage capacity of edge cloud, we assume that edge cloud has storage capacity C (in bits) that can be used to store the code and data. Therefore, task caching decisions are constrained by the following edge cloud storage capacity.

$$\sum_{k=1}^{K} a_k^t s_k \leq C, \quad \forall t. \tag{2}$$

3.3 Task Latency

Considering the mobility of users, based on the [17], we assume that the number of users connecting to the edge cloud in different time slots is different, while the number of users in the same time slot is constant due to the low mobility of users with shorter time slots. Thus, let N_t denote the number of mobile devices that can access the edge cloud at time slot t. Moreover, let $d_{n,k}^t$ denote the number of requests of mobile device n for the task Q_k at time slot t. Therefore, we can obtain the number of requests for task Q_k on edge cloud at time slot t λ_k^t is:

$$\lambda_k^t = \sum_{n=1}^{N_t} d_{n,k}^t. \tag{3}$$

Although the user's request can be predicted by well-studied learning algorithm, the number of mobile devices accessing the edge cloud N_t is not the same in different time slots due to the users' mobility. Thus, in real systems, it is difficult to predict the number of requests from edge clouds, so we assume that the number of requests from edge clouds prior is unknown.

Next, we introduce the total task latency by dividing it into the following two parts, i.e., communication latency and computation latency, as shown in the Fig. 2. Specifically, the communication latency includes a delay for offloading a computation task to the edge cloud through wireless link or remote cloud through wireless and wired link. Moreover, the communication delay from a mobile device to the edge cloud is much shorter than that to the remote cloud. For wireless link, let τ^t denote the wireless transmission rate at time slot t. For wired link, let r_1^t denote the backbone transmission rate at time slot t and r_2^t denote the round-trip time to the remote cloud at time slot t. Thus, when the task Q_k is cached on edge cloud, the communication latency is s_k/τ^t. Otherwise, the communication latency is $(s_k/r_1^t + r_2^t)$.

However, due to the dynamic nature of the network environment, the data transmission rate can not be precisely estimated. Furthermore, computation latency corresponds to the time that is required to execute a task on the edge cloud or remote cloud. Let f_{ec}^k and f_{rc}^k represent the CPU frequency of the edge cloud and remote cloud assigned to the task Q_k, respectively. Note that under the same load conditions, the CPU frequency of the cloud is usually greater than the frequency of the edge cloud. Hence, similar to [4], [25], we consider that $f_{rc}^k > f_{ec}^k$. Thus, the computation latency of task Q_k processed in the edge cloud and the remote cloud is ω_k/f_{ec}^k and ω_k/f_{rc}^k, respectively.

According to the above discussion, if a task is not cached on the edge cloud, it cannot be executed. Consequently, the computation task should be offloaded to the remote cloud. More specifically, only when $a_k^t = 1$, the computation task can be executed on the edge cloud. Otherwise ($a_k^t = 0$), the computation task will be offloaded to the remote cloud for processing. Therefore, the task latency of mobile device n at time slot t can be expressed as:

$$D_n^t(a_k^t) = \begin{cases} \sum_{k=1}^{K} d_{n,k}^t \left(\frac{w_k}{f_{ec}^k} + \frac{s_k}{\tau^t} \right) & \text{if} \quad a_k^t = 1, \\ \sum_{k=1}^{K} d_{n,k}^t \left(\frac{w_k}{f_{rc}^k} + \frac{s_k}{\tau^t} + \frac{s_k}{r_1^t} + r_2^t \right) & \text{if} \quad a_k^t = 0. \end{cases} \tag{4}$$

Furthermore, considering the limitation of edge cloud computing capacity, we assume the maximum processing power of edge cloud is F_{ec} (in CPU cycles). When tasks are cached on edge cloud, task caching decisions are limited by the following

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2020.3047417, IEEE Transactions on Network Science and Engineering

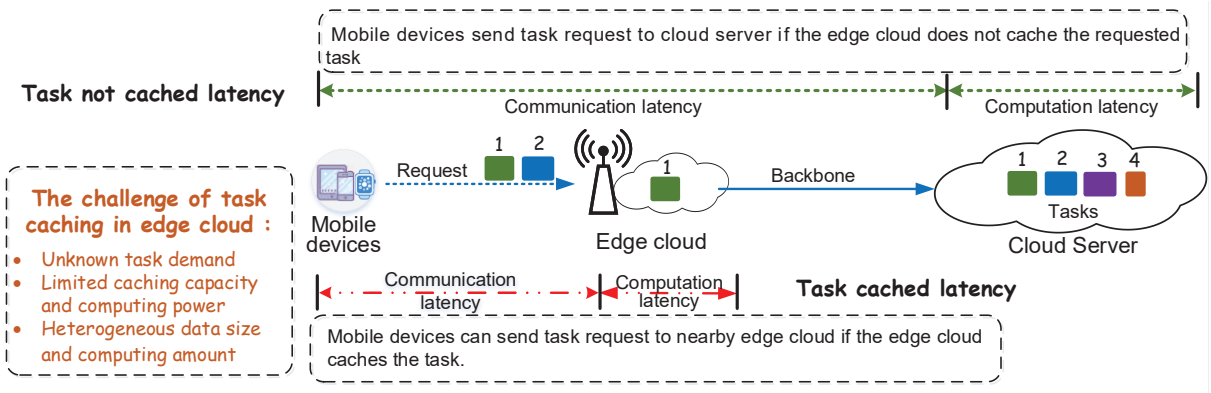


Fig. 2. The illustration of task latency model.

computational capability:

$$\sum_{k=1}^{K} a_k^t f_{ec}^k \leq F_{ec}, \quad \forall t. \tag{5}$$

3.4 Problem Formulation

For task caching decision making, our approach is based on minimizing the task latency by taking the caching and computing capacity of the edge cloud into consideration. Thus, the problem can be expressed as:

$$\mathbf{P1}: \text{minimize} \quad \frac{1}{T}\sum_{t=1}^{T}\sum_{n=1}^{N_t} D_n^t(a_k^t) \tag{6}$$

$$\text{subject to} \quad C1: \sum_{k=1}^{K} a_k^t s_k \leq C, \forall t \in \mathcal{T}. \tag{7}$$

$$C2: \sum_{k=1}^{K} a_k^t f_{ec}^k \leq F_{ec}, t \in \mathcal{T}. \tag{8}$$

$$C3: a_k^t \in \{0,1\}, \forall t \in \mathcal{T}, \forall k \in \mathcal{K} \tag{9}$$

where the objective function computes the minimal task latency. The first constraint (C1) signifies that task caching cannot exceed the maximum caching capacity. The second constraint (C2) indicates the computing resources allocated to tasks should not exceed the total computing resources of edge cloud whereas constraint (C3) shows that the task is cached on the edge cloud or not.

For the above optimization problem $P1$, assuming that we already know all the variables, the above optimization problem is a traditional 0-1 optimization problem, which can be solved by the traditional algorithm [25]. However, in practice, edge cloud does not know the request pattern of user tasks, so the traditional algorithm is not applicable. In this paper, we will use online learning strategy to solve the optimization problem.

4 Intelligent Task Caching Scheme

In order to solve the above optimization problem, we use the MAB theory to develop an intelligence task caching scheme, called the M-AUCB algorithm, and also analyze the bound of the M-AUCB algorithm.

4.1 M-AUCB Algorithm

In order to solve the optimization problem $P1$, we transform the task caching problem to the MAB problem. To explain this, we first give the description of the MAB problem. The MAB problem refers to a situation where a gambler faces with a slot machine with multiple arms, and when each arm of the machine is played, a reward from unknown statistical functions is obtained. At the beginning, the gambler does not know anything about the reward for the arms. Each time the gambler plays, he makes a decision to play one of the arms, and the machine gives him a reward. The purpose of gambler is to maximize the reward.

Lemma 1. The task caching problem ($P1$) can be match to the MAB model with new variations, i.e., (i) multi-players, (ii) limited budget (i.e., limited storage and computing capacity of edge cloud), and (iii) adaptive to the size and computing amount of the task.

Proof: The task caching problem matches the MAB model. Specifically, we first give the similarities between task caching and MAB model. Each task is equivalent to an arm. When the task is cached on the edge cloud, it is equivalent to the arm being played by the gambler. The caching agent (i.e., intelligent task caching algorithm deployed on the edge cloud) is equivalent to the gambler. And at time slot t, caching agent does not know the number of task requests and the corresponding rewards, which corresponds to the fact that the gambler does not know the benefits of each arm. In addition, when the task is cached on edge cloud, the delay can be reduced. Thus, our goal is to minimize the task latency same as maximizing the reward of MAB model.

Then, we give the differences between task caching and MAB model. (i) Edge cloud can cache multiple tasks at a time, so it is equivalent to multi-players playing the arm at the same time, which corresponds to the constraint C3 in optimization problem $P1$. (ii) Considering the limited storage and computing capacity of edge cloud, it can only cache the limited task at a time, which corresponds to the constraint C1 and C2 in optimization problem $P1$. (iii) Since different tasks have different sizes and required computation amount, we need to consider the effect of task size and computing amount on the caching strategy (i.e., the algorithm can adapt to the size and computing amount of the task). □

The main objective of the proposed MAB-based task caching scheme is to cache M tasks out of K tasks on the edge cloud within each time slot by exploiting the UCB algorithm, where M is the maximum number of tasks that satisfy the edge cloud

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2020.3047417, IEEE Transactions on Network Science and Engineering

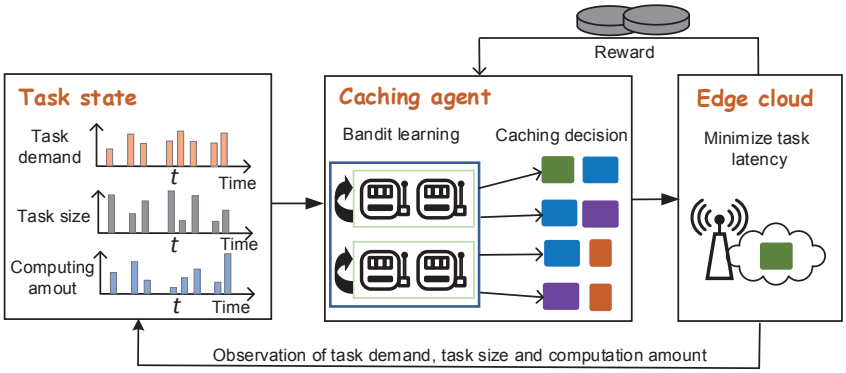


Fig. 3. The framework of intelligent task caching in edge cloud.

computing and storage capacity. Fig. 3 illustrates the framework of intelligent task caching on edge cloud. As shown in the Fig. 3, the caching agent observes the task demand, task size and computation amount, and decides a caching decision on the edge cloud using MAB theory. Then, the caching agent receives the task latency (i.e., reward) based on the objective. In this paper, our goal is to minimize the task latency, so the less reward, the better. Next, we describe the M-AUCB task caching algorithm in detail.

Given a total number of time slots T and discrete time $t \in \{1, 2, \cdots, T\}$, for time slot t, we denote $D_j^t = \sum_{n=1}^{N_t} D_n^t(a_j^t = 1)$ as the total latency of all tasks as soon as the j-th task is cached on the edge cloud within the t^{th} time slot. Thus, the D_j^t can be expressed as:

$$D_j^t = \sum_{n=1}^{N_t} D_n^t(a_j^t = 1) = \sum_{n=1}^{N_t} \left[\sum_{k=1,k\neq j}^{K} d_{n,k}^t \left(\frac{w_k}{f_{rc}^k} + \frac{s_k}{\tau^t} + \frac{s_k}{r_1^t} + r_2^t \right) + d_{n,j}^t \left(\frac{w_j}{f_{ec}^j} + \frac{s_k}{\tau^t} \right) \right]. \tag{10}$$

Next, we describe the M-AUCB algorithm. In the initialization stage, M-AUCB guarantees that each of the total K tasks will be cached on the edge cloud at least once. This is mainly to make sure that each task can be explored. After the initialization, in the t^{th} time slot, the M-AUCB algorithm calculate the average task delay cached on the edge cloud in the previous time slots (i.e., from time slot 1 to $t-1$). Accordingly, we can show the average task latency for task Q_j $\overline{D}_{t-1,j}$ as:

$$\overline{D}_{t-1,j} = \frac{\sum_{i=1}^{t-1} D_j^i}{N_{t-1,j}}, \tag{11}$$

where $N_{t,j}$ is the number of times that task Q_j has been selected in the last t time slots. Furthermore, in order to consider the impact of task size and computation amount on the task caching, we classify tasks and normalize them to $(0.5, \frac{2}{3})$ [27]. This is because when the task size is larger or the computational requirement of the task is larger, caching it will result in longer task latency. To be specific, we denote $\hat{s}_j$ and $\hat{\omega}_j$ as follows:

$$\hat{s}_j = \max(0.5 + \epsilon_1, \frac{s_j}{1.5 \max_{j\in K} s_j}), \tag{12}$$

$$\hat{\omega}_j = \max(0.5 + \epsilon_2, \frac{\omega_j}{1.5 \max_{j\in K} \omega_j}), \tag{13}$$

where the parameters ϵ_1 and ϵ_2 are constant, and $\epsilon_1, \epsilon_2 \in (0, 0.1)$. These parameters represent the sensitivity of our algorithm to the size and computation requirement of the task, e.g., when ϵ_1 is small, the algorithm will be more sensitive to the size of the task.

Second, in the caching stage, the aim is to choose M out of the K tasks in order to minimize the total task latency. Specifically, we select M tasks based on the $\hat{D}_{t,j}$, which is defined as:

$$\hat{D}_{t-1,j} = \overline{D}_{t-1,j} - \sqrt{\frac{2\hat{s}_j \hat{\omega}_j \log(Mt)}{N_{t-1,j}}}. \tag{14}$$

This formula is based on the traditional UCB arm selection formula [27]–[29]. It shows the balance between exploration and exploitation, i.e., the task caching scheme balances the exploitation of a known user's task latency in the past and the exploration of the upcoming user's request. To be specific, from the above formula, we can see that a smaller $\overline{D}_{t-1,j}$ (i.e., the average time delay of the task Q_j) or $N_{t-1,j}$ (i.e., the number of times that the task Q_j is selected) can result in a smaller $\hat{D}_{t-1,j}$. Under these conditions, the task Q_j can be easily selected. This indicates that the M-AUCB algorithm can be invoked to minimize the average task latency (i.e., exploiting a cache strategy that minimizes latency). Consequently, this formula allows the tasks which have not been explored (i.e., when $N_{t-1,j}$ is small) sufficiently to be executed. Therefore, by choosing a suitable M tasks which produces the smallest $\hat{D}_{t-1,j}$, a better task caching scheme can be exploited.

Furthermore, our algorithm can adaptively take into account the size and computing amount of each task. This is because from (14), we can observe that when the task size or the required computing amount is large, exploring this task will consume more edge cloud resources. So we reduce its number of explorations and increase exploitations frequency. Thus, heterogeneous tasks have different caching strategies.

Further details of the M-AUCB algorithm is shown in Algorithm 1 (i.e., steps 1-17). As indicated, a task j (i.e., task Q_j) is selected to be cached in each time slot by updating D_j^t and $N_{t-1,j}$. To be specific, steps 2-5 are the initialization stage. In steps 7-8, we calculate the value of $\hat{D}_{t,j}$ according to (14) for choosing a task. In step 9 and step 14, we denote $a_{t,i}$ as the i-th task chosen among K tasks in time slot t, and choose M task to minimize the task latency $\hat{D}_{t,j}$.

4.2 Regret Analysis

In this section, we analyze the regret of M-AUCB algorithm and its upper bound. Consider that user request tasks are independent of each other, thus, we assume that the losses brought by each

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2020.3047417, IEEE Transactions on Network Science and Engineering

Algorithm 1 M-AUCB Algorithm for Task Caching

Input: T, M

for $t = 1, \cdots, T$ **do**
  if Any task $j \in \mathcal{K}$ has not been cached on the edge cloud **then**
    cache task j on the edge cloud
    update $N_{t,j} = N_{t-1,j} + 1$
    update $\overline{D}_{t,j} = \frac{\overline{D}_{t-1,j} N_{t-1,j} + D_{t,j}}{N_{t,j}}$
  else
    Calculate the selection function of each candidate task $j \in \mathcal{K}$
    $\hat{D}_{t-1,j} = \overline{D}_{t-1,j} - \sqrt{\frac{2\hat{s}_j \hat{\omega}_j \log(Mt)}{N_{t-1,j}}}$
    $a_j^t = \arg\min_j \hat{D}_{t-1,j}$
    while $\sum_{j=1}^{K} a_j^t s_k \leq C$ and $\sum_{j=1}^{K} a_j^t f_{ec}^k \leq F_{ec}$ **do**
      M=1
      $a_i^t = \arg\min_{a_i^t \in \mathcal{K} \setminus \cup_{j=1}^{i-1} a_i^t} \hat{D}_{t-1,a_i^t}$
      update $N_{t,a_i^t} = N_{t-1,a_i^t} + 1$
      update $\overline{D}_{t,a_{t,i}} = \frac{\overline{D}_{t-1,a_i^t} N_{t-1,a_i^t} + D_{t,a_i^t}}{N_{t,a_i^t}}$
      update M=M+1
    end while
  end if
end for

task are independent and identically distributed (i.i.d.) over time and are independent of each other. Furthermore, we denote the expectation of D_j^t as ED_j^t) = μ_j. Furthermore, we define μ^* and j^* as:

$$\mu^* = \min_{j \in K} \mu_j, \tag{15}$$

$$j^* = \arg\min_{j \in K} \hat{D}_{t,a_i^t}, \tag{16}$$

where the j^* is the optimal caching task.

Based on the above, we define *learning regret* (i.e., the difference between the latency of the selected caching task and the minimum latency achieved by the optimal caching task) R_t as follows:

$$R_t = \sum_{j=1}^{K} N_{t,j}(D_j^t - \mu^*). \tag{17}$$

Thus, the *expected learning regret* $E(R_t)$ can be expressed as:

$$E(R_t) = \sum_{j=1}^{K} E(N_{t,j}) \Delta_j, \tag{18}$$

where $\Delta_j = \mu_j - \mu^*$ indicates the gap between the optimal caching task and task Q_j.

Furthermore, we can obtain the *expected cumulative learning regret* as follows:

$$R_T = \sum_{t=1}^{T} E(R_t) \tag{19}$$

Then, we can obtain the upper bound of the algorithm according to the following theorem:

Theorem 1. The expected cumulative learning regret of the M-AUCB algorithm has an upper bound as:

$$E(R_t) \leqslant \sum_{j=1}^{K} \left(\frac{8(\hat{s}_j \hat{\omega}_j)^2 \log(Mt)}{\Delta_j} + O(1) \right). \tag{20}$$

Proof: See Appendix A. □

From Theorem 1, we can see that the M-AUCB algorithm is bounded. Furthermore, we can obtain that as the number of task (i.e., K) and the maximum number of tasks that the edge cloud can cache (i.e., M) increase, the learning regret of the M-AUCB increases.

5 Performance Evaluation

In this section, we evaluate the learning regret, cumulative learning regret and task latency of the proposed M-ACUB algorithm through experiments.

5.1 Experiment Setup

In our experiments, we consider a system that contains an edge cloud and a set of mobile devices performing computation-intensive tasks. The edge cloud is deployed near a wireless access point (e.g., cellular base station or Wi-Fi access points). The mobile devices connect to the edge cloud via wireless channel. According to [30], we set the wireless transmission rate as $\tau_t = 1/\left(\log_2(1 + p_t h_t / \sqrt{d_t^3})\right)$, where p_t is the transmission power at time slot t, h_t is the noise power at time slot t, and d_t is the distance between user and edge cloud. The edge cloud connects to the remote cloud through the Internet. According to [25], we set the backbone transmission rate is $[2, 6]$ Mb/s and the the round-trip time is 200 ms.

For the task, we give the evaluation results by a real-world video stream analysis [31]. Specifically, the video stream includes 500 video tasks. We select Full HD video with 1920×1080 video resolution. Moreover, for the number of requests for a task, we use the real application request data set [32]. It collect data from 10208 mobile users requesting 23 mobile applications. In this experiment, we chose 20 mobile applications and assume that these applications are requests for video task (i.e., we randomly select 20 video tasks). Moreover, we assume that the mobile users are uniform distributed over the edge cloud, and the user trajectory is generated by the random movement model.

For computing resource, according to [17], we set the computing capability of the edge cloud and remote cloud to be 10 GHz and 100 GHz, respectively. Furthermore, we set the storage capacity of edge cloud to 500 GBs [19]. The caching agent deployed on the edge cloud dynamically decides which tasks to cache on the edge cloud. We run the experiment for 400 time slots (i.e., $T = 400$). For each time slot, we use data sets for 100 experiments, and calculate the average value as the experimental results. In our experiments, we focus on measuring the task latency, learning regret and cumulative learning regret.

5.2 Comparison Algorithm

The proposed M-AUCB algorithm is compared with four task caching schemes, which are briefly described below:

- *Optimal caching scheme*: In each time slot t, the optimal caching scheme is aware of the expectation of total task

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2020.3047417, IEEE Transactions on Network Science and Engineering

latency for each caching task. Under this condition we choose M tasks with the smallest task latency. In other words, the optimal caching scheme has a prior knowledge of the task demand pattern, task size and computing amount.

- *Random caching scheme* [8]: In each time slot t, the scheme randomly chooses M tasks to cache. Under this caching strategy, some tasks with fewer requests may be cached on the edge cloud, resulting in a larger task latency.
- *UCB caching scheme* [28]: In each time slot t, we use the traditional UCB caching scheme to cache one task on the edge cloud, and the other $M-1$ tasks are cached randomly. The following two equations are used to select the first task:

$$\hat{D}_{t,j} = \overline{D}_{t,j} - \sqrt{\frac{2\log(t)}{N_{t,j}}}, \tag{21}$$

$$a_i^t = \arg\min_{a_i^t} \hat{D}_{t,a_i^t}. \tag{22}$$

- *M-UCB caching scheme*: This algorithm is also proposed in this paper. The details are as follows: in each time slot t, M tasks are chosen to be cached, which is based on the number of previously cached tasks and their averaged delay. The M-UCB algorithm can sufficiently exploit caching tasks with smaller task latency, as well as exploring tasks that are less frequently cached. More specifically, we choose M tasks for caching by using the following two equations:

$$\hat{D}_{t,j} = \overline{D}_{t,j} - \sqrt{\frac{2\log(Mt)}{N_{t,j}}}, \tag{23}$$

$$a_i^t = \arg\min_{a_i^t \in \mathcal{K} \setminus \cup_{j=1}^{i-1} a_i^t} \hat{D}_{t,a_i^t}. \tag{24}$$

5.3 Performance Analysis

5.3.1 Regret analysis

In our experiments, we first evaluate the learning regret and cumulative learning regret of five different task caching schemes (including the proposed M-AUCB algorithm). The results are shown in Fig. 4. As can be seen from Fig. 4(a), when the time slot $T \leq 20$, the learning regret of all schemes is in the initial stage. When $T \geq 150$, the learning regret of the M-AUCB caching scheme becomes relatively more stable (i.e., the change of learning regret is not obvious). This can be explained by that the M-AUCB algorithm has learned the user's task request pattern after a period of exploration and exploitation.

Moreover, Fig. 4(a) and Fig. 4(b) show that the optimal caching algorithm has the minimum learning regret and cumulative learning regret, this can be explain as the optimal caching algorithm knows the expectation of total task delay when the task is caching. Furthermore, we can observe that the learning regret and cumulative learning regret brought by the proposed M-AUCB caching scheme are larger than that brought by optimal task caching scheme. This is because the optimal caching scheme assumes that the request pattern for the task is known, while M-AUCB assumes that the task request pattern is unknown to the edge cloud.

We also observe that the learning regret and cumulative learning regret brought by the M-AUCB caching scheme are slightly smaller than that brought by C-UCB caching. At the same time, it is far better than those of the UCB and random caching schemes. This is because the random caching scheme selects tasks randomly for caching at each time slot, neither considering the task request pattern, nor considering the impact of task size and computation amount on caching. Thus this caching scheme brings the biggest learning regret and cumulative learning regret. Both the M-AUCB, M-UCB and UCB learn different request pattern of tasks, but the UCB caching scheme only uses the traditional UCB algorithm to cache one task in edge cloud at each time slot, while the other m-1 tasks are still randomly selected. Although the M-UCB caching scheme selects m tasks for caching according to the number of task requests at each time slot, it is not adaptive to the task size and computing amount. Our M-AUCB algorithm not only attempts to learn the user demand patterns, but also takes into account the effect of the task size.

5.3.2 Task latency

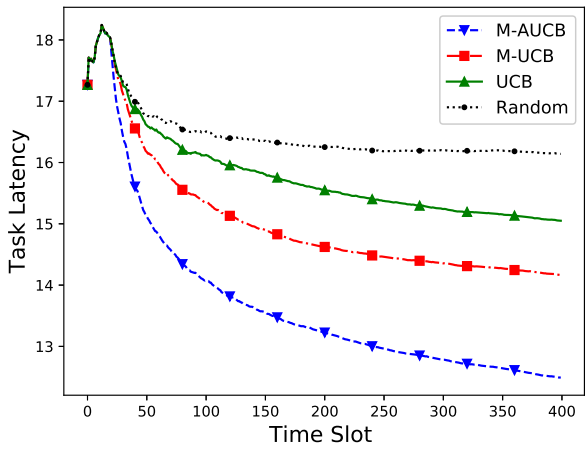


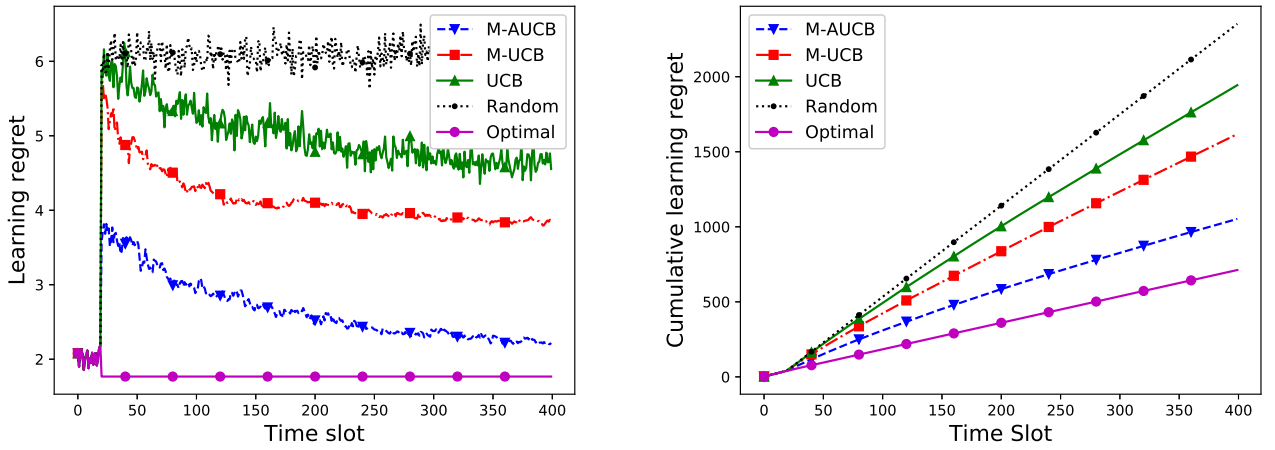
Fig. 5. Task latency analysis under different task caching schemes.

Next, we analyze the task latency under different task caching schemes. Fig. 5 depicts the task latency of each task caching scheme. We can clearly see that after the initialization stage, the task latency of each algorithm tends to gradually become stable. Furthermore, we observe that the M-AUCB caching scheme significantly reduces the task latency compared to M-UCB caching scheme, UCB caching scheme and random caching scheme. Compared to the M-UCB algorithm (i.e., the optimal baseline), the M-AUCB algorithm decreases the task delay by 14.8%. This result further shows that our caching scheme has good performance.

5.3.3 Edge cloud capacity

We also analyze the impact of edge cloud caching capacity on the task latency and learning regret. In these experiments, we run 100 experiments to produce the results and each run includes 400 time slots. Moreover, the cache capacity of the edge cloud varies from 300 Mbits to 700 Mbits. From Fig. 6(a), we can obtain the learning regret increases as the caching capacity of the edge cloud increases. This is obviously consistent with the conclusion given in Theorem 1. Furthermore, from Fig. 6(b), we can see that when the caching capacity of the edge cloud increases, the task latency decreases. This is because a larger caching capacity would allow

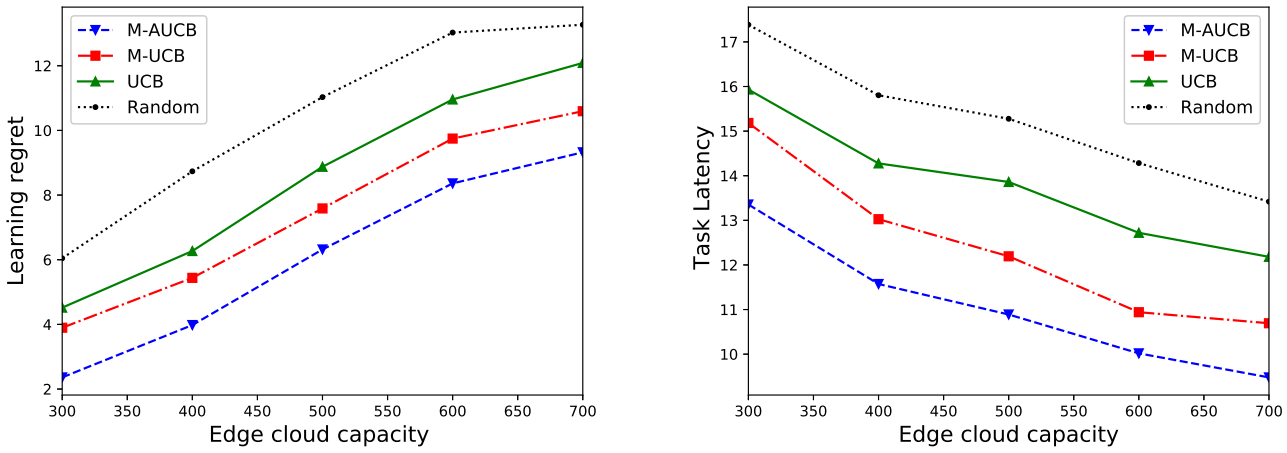
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2020.3047417, IEEE Transactions on Network Science and Engineering



(a) Learning regret under different task caching schemes.

(b) Cumulative learning regret under different caching schemes.

Fig. 4. Learning regret and cumulative learning regret under different task caching schemes.



(a) Learning regret under different task caching schemes when increasing the edge cloud capacity.

(b) Task latency under different task caching schemes when increasing the edge cloud capacity.

Fig. 6. The impact of edge cloud capacity on different task caching schemes.

more users to get their tasks through the edge cloud causing a reduction in the task latency.

5.3.4 *Number of tasks*

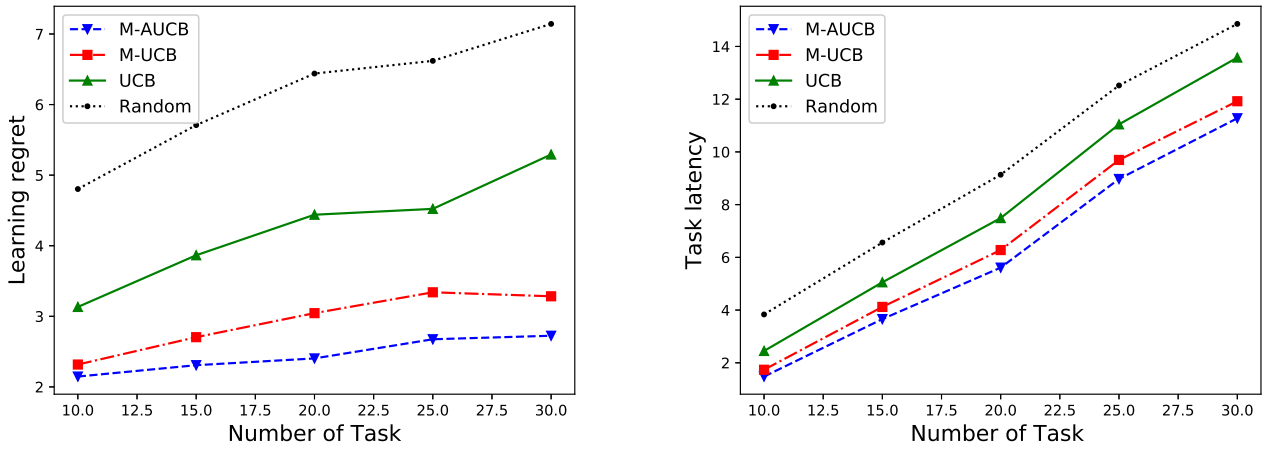
We further evaluate the impact of the task numbers on the task latency and learning regret. In these experiments, we set $T = 400$ with the number of tasks ranging from 10 to 30. From Fig. 7, we can observe that when the number of tasks increases, both the task latency and learning regret become larger. This is because when the number of tasks becomes larger, users can request more tasks when the caching capacity of edge cloud is fixed. As a consequence, some tasks that require more time cannot be cached, resulting in larger task latency and learning regret. Moreover, we find that M-AUCB algorithm is superior to other task caching scheme in different number of tasks. This indicates that our algorithm has stronger robustness when the system changes. We attribute this performance improvement to the adaptive design of the task caching scheme.

6 Conclusion and Future Work

In this paper, we first analyze the problem of task caching on the edge cloud by formalizing it under the circumstance of not having any prior knowledge of the task request pattern. Then, we propose a caching scheme, referred to as M-AUCB, which is capable of learning the task request pattern. In addition, it takes into consideration the impact of different task sizes on the edge cloud. The experimental results indicate that our proposed scheme can effectively minimize the task latency.

Though the M-AUCB task caching algorithm can not only make caching decisions online according to the number of task requests, but also adapt to the size and computing amount of the task, our model has some limitations. For example, in this paper, we design the M-AUCB task caching algorithm from the perspective of edge cloud and assume that the user obtains tasks only from one edge cloud. However, from the perspective of the user, the user can communicate with multiple edge clouds to obtain the requested task. In this case, the task caching problem is the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2020.3047417, IEEE Transactions on Network Science and Engineering



(a) Learning regret under different task caching schemes when increasing the number of tasks.

(b) Task latency under different task caching schemes when increasing the number of tasks.

Fig. 7. The impact of task numbers on different task caching schemes.

cooperative task caching. In addition, our main focus in this paper was based on designing an independent task caching. However, in the case of dependencies among tasks, our proposed algorithm can also be applied using a graphical model. In the future work, we will consider a mobile device can request tasks from multiple edge clouds, and design the cooperative task caching scheme.

ACKNOWLEDGEMENT

This work was supported by National Key R&D Program of China under Grant 2018YFC1314600, Nature Science Foundation of China under Grant 61802138, Shenzhen Institute of Artificial Intelligence and Robotics for Society, and in collaboration with the Advanced Network Technology Division (ANTD) of the National Institute of Standards and Technology (NIST), USA.

REFERENCES

[1] S. Jošilo and G. Dán, "Selfish decentralized computation offloading for mobile cloud computing in dense wireless networks," *IEEE Transactions on Mobile Computing*, vol. 18, no. 1, pp. 207–220, 2019.

[2] J. L. D. Neto, S.-Y. Yu, D. F. Macedo, M. S. Nogueira, R. Langar, S. Secci *et al.*, "Uloof: a user level online offloading framework for mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 17, no. 11, pp. 2660–2674, 2018.

[3] T. X. Tran, D. V. Le, G. Yue, and D. Pompili, "Cooperative hierarchical caching and request scheduling in a cloud radio access network," *IEEE Transactions on Mobile Computing*, vol. 17, no. 12, pp. 2729–2743, 2018.

[4] T. He, H. Khamfroush, S. Wang, T. La Porta, and S. Stein, "It's hard to share: Joint service placement and request scheduling in edge clouds with sharable and non-sharable resources," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 365–375.

[5] L. Wang, L. Jiao, T. He, J. Li, and M. Mühlhäuser, "Service entity placement for social virtual reality applications in edge computing," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 468–476.

[6] T. Zhao, I.-H. Hou, S. Wang, and K. Chan, "Red/led: An asymptotically optimal and scalable online algorithm for service caching at the edge," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 8, pp. 1857–1870, 2018.

[7] L. E. Chatzieleftheriou, M. Karaliopoulos, and I. Koutsopoulos, "Jointly optimizing content caching and recommendations in small cell networks," *IEEE Transactions on Mobile Computing*, vol. 18, no. 1, pp. 125–138, 2019.

[8] L. Qiu and G. Cao, "Popularity-aware caching increases the capacity of wireless networks," *IEEE Transactions on Mobile Computing*, 2019.

[9] L. Pu, L. Jiao, X. Chen, L. Wang, Q. Xie, and J. Xu, "Online resource allocation, content placement and request routing for cost-efficient edge caching in cloud radio access networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 8, pp. 1751–1767, 2018.

[10] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. C. Leung, "Cache in the air: Exploiting content caching and delivery techniques for 5g systems," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 131–139, 2014.

[11] K. Poularakis, G. Iosifidis, V. Sourlas, and L. Tassiulas, "Exploiting caching and multicast for 5g wireless networks," *IEEE Transactions on Wireless Communications*, vol. 15, no. 4, pp. 2995–3007, 2016.

[12] S. Zhang, P. He, K. Suto, P. Yang, L. Zhao, and X. Shen, "Cooperative edge caching in user-centric clustered mobile networks," *IEEE Transactions on Mobile Computing*, vol. 17, no. 8, pp. 1791–1805, 2018.

[13] S. Müller, O. Atan, M. van der Schaar, and A. Klein, "Context-aware proactive content caching with service differentiation in wireless networks," *IEEE Transactions on Wireless Communications*, vol. 16, no. 2, pp. 1024–1036, 2017.

[14] Y. Hao, M. Chen, L. Hu, M. S. Hossain, and A. Ghoneim, "Energy efficient task caching and offloading for mobile edge computing," *IEEE Access*, vol. 6, pp. 11 365–11 373, 2018.

[15] J. Xu, L. Chen, and 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.

[16] M. Chen, Y. Hao, L. Hu, M. S. Hossain, and A. Ghoneim, "Edge-cocaco: Toward joint optimization of computation, caching, and communication on edge cloud," *IEEE Wireless Communications*, vol. 25, no. 3, pp. 21–27, 2018.

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

[18] P. Yang, N. Zhang, S. Zhang, L. Yu, J. Zhang, and X. S. Shen, "Content popularity prediction towards location-aware mobile edge caching," *IEEE Transactions on Multimedia*, 2018.

[19] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 10–18.

[20] V. Farhadi, F. Mehmeti, T. He, T. La Porta, H. Khamfroush, S. Wang, and K. S. Chan, "Service placement and request scheduling for data-intensive applications in edge clouds," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1279–1287.

[21] N. Mohan, P. Zhou, K. Govindaraj, and J. Kangasharju, "Managing data in computational edge clouds," in *Proceedings of the Workshop on Mobile Edge Communications*. ACM, 2017, pp. 19–24.

[22] G. S. Paschos, A. Destounis, L. Vigneri, and G. Iosifidis, "Learning to

cache with no regrets," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 235–243.

[23] L. Chen, J. Xu, S. Ren, and P. Zhou, "Spatio–temporal edge service placement: A bandit learning approach," *IEEE Transactions on Wireless Communications*, vol. 17, no. 12, pp. 8388–8401, 2018.

[24] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 856–868, 2018.

[25] L. Chen, P. Zhou, L. Gao, and J. Xu, "Adaptive fog configuration for the industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4656–4664, 2018.

[26] L. Yang, J. Cao, H. Cheng, and Y. Ji, "Multi-user computation partitioning for latency sensitive mobile cloud applications," *IEEE Transactions on Computers*, vol. 64, no. 8, pp. 2253–2266, 2014.

[27] Y. Sun, X. Guo, S. Zhou, Z. Jiang, X. Liu, and Z. Niu, "Learning-based task offloading for vehicular cloud computing systems," in *2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–7.

[28] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[29] L. Chen and J. Xu, "Task offloading and replication for vehicular cloud computing: A multi-armed bandit approach," *arXiv preprint arXiv:1812.04575*, 2018.

[30] Y. Xiao and M. Krunz, "Qoe and power efficiency tradeoff for fog computing networks with fog node cooperation," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[31] A. Anjum, T. Abdullah, M. Tariq, Y. Baltaci, and N. Antonopoulos, "Video stream analysis in clouds: An object detection and classification framework for high performance video analytics," *IEEE Transactions on Cloud Computing*, 2016.

[32] S. L. Lim, P. J. Bentley, N. Kanakam, F. Ishikawa, and S. Honiden, "Investigating country differences in mobile app user behavior and challenges for software engineering," *IEEE Transactions on Software Engineering*, vol. 41, no. 1, pp. 40–64, 2014.

[33] A. Asadi, S. Müller, G. H. Sim, A. Klein, and M. Hollick, "Fml: Fast machine learning for 5g mmwave vehicular communications," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1961–1969.

[34] S. Bubeck, N. Cesa-Bianchi *et al.*, "Regret analysis of stochastic and nonstochastic multi-armed bandit problems," *Foundations and Trends® in Machine Learning*, vol. 5, no. 1, pp. 1–122, 2012.

APPENDIX A
PROOF OF THE THEOREM 1

Before providing the proof, we use the Chernoff-Hoeffding inequality to obtain the confidence interval as:

$$P\left(\overline{D}_{t,j}+\sqrt{\frac{2\hat{s}_j\hat{\omega}_j\log\left(Mt\right)}{N_{t,j}}}\leqslant\mu_j\right)\leqslant(Mt)^{-4(\hat{s}_j\hat{\omega}_j)^2},\quad(25)$$

$$P\left(\overline{D}_{t,j}-\sqrt{\frac{2\hat{s}_j\hat{\omega}_j\log\left(Mt\right)}{N_{t,j}}}\geqslant\mu_j\right)\leqslant(Mt)^{-4(\hat{s}_j\hat{\omega}_j)^2}.\quad(26)$$

Now, we present the proof. From Algorithm 1, we can see that the selection of task j (i.e., Q_j) in the t^{th} time slot, also satisfies:

$$\hat{D}_{t-1,j}\leqslant\hat{D}_{t-1,j^*}.\quad(27)$$

According to (14), it can be expressed as

$$\overline{D}_{t-1,j}-\sqrt{\frac{2\hat{s}_j\hat{\omega}_j\log\left(Mt\right)}{N_{t-1,j}}}\leqslant\overline{D}_{t-1,j^*}-\sqrt{\frac{2\hat{s}_j\hat{\omega}_j\log\left(Mt\right)}{N_{t-1,j^*}}}.\quad(28)$$

Based on the above, to satisfy $\hat{D}_{t-1,j}\leqslant\hat{D}_{t-1,j^*}$, at least one of the following three equations should be satisfied [33], [34]:

$$\mu_j-2\sqrt{\frac{2\hat{s}_j\hat{\omega}_j\log\left(Mt\right)}{N_{t-1,j}}}\leqslant\mu^*,\quad(29)$$

$$\overline{D}_{t,j}+\sqrt{\frac{2\hat{s}_j\hat{\omega}_j\log\left(Mt\right)}{N_{t-1,j}}}\leqslant\mu_j,\quad(30)$$

$$\overline{D}_{t,j^*}-\sqrt{\frac{2\hat{s}_{j^*}\hat{\omega}_{j^*}\log\left(Mt\right)}{N_{t-1,j^*}}}\geqslant\mu^*.\quad(31)$$

Since $\Delta_j=\mu_j-\mu^*$, according to (28), we can obtain the following formula:

$$N_{t-1,j}\leqslant\frac{8(\hat{s}_j\hat{\omega}_j)^2\log\left(Mt\right)}{\Delta_j^2}.\quad(32)$$

To analyze (29), (30), (31), let us denote $N_{T,j}$ as the j-th task being cached within time slot T. We can then prove that the event $\{N_{T,j}\geq\frac{8(\hat{s}_j\hat{\omega}_j)^2\log\left(Mt\right)}{\Delta_j^2}\}$ has a small probability so that each sub-optimal j-th task cannot be cached more than $\frac{8(\hat{s}_j\hat{\omega}_j)^2\log\left(Mt\right)}{\Delta_j^2}$ plus a small constant value. As for any integer u, we can obtain the following equation:

$$\begin{aligned}N_{T,j}&\leqslant u+\sum_{t=u+1}^{T}\sum_{i=1}^{C}1\left\{a_i^t=j,N_{t-1,j}\geqslant u\right\}\\&\leqslant u+\sum_{t=u+1}^{T}\\&1\left\{\exists N_{t,j}:u\leqslant N_{t,j}\leqslant t,\exists N_{t,j^*}:1\leqslant N_{t,j^*}\leqslant t,\hat{D}_{t,j}\leqslant\hat{D}_{t,j^*}\right\}\\&\leqslant u+\sum_{t=u+1}^{T}\sum_{N_{t,j}=u+1}^{t}\sum_{N_{t,j^*}=1}^{t}1\left\{\hat{D}_{t,j}\leqslant\hat{D}_{t,j^*}\right\},\end{aligned}\quad(33)$$

where $1\{\cdot\}$ is an indicator function, i.e., $1\{\cdot\}=1$ if condition is true, otherwise $1\{\cdot\}=0$. a_t^i as the i-th task is chosen from the K in time slot t.

If we assumed that $u=\frac{8(\hat{s}_j\hat{\omega}_j)^2\log\left(Mt\right)}{\Delta_j^2}$. Then, $\forall\sigma,u<\sigma<T$, and according to (31), we can obtain the following expression:

$$\mu_j-2\sqrt{\frac{2\hat{s}_j\hat{\omega}_j\log\left(Mt\right)}{\sigma}}>\mu^*.\quad(34)$$

Although we observe that (29) is not satisfied, at least one of (30) or (31) will be satisfied. For the sake of mathematical convenience, we define the symbols, $z_{t,j}$ and z_{t,j^*}:

$$z_{t,j}=\overline{D}_{t,j}+\sqrt{\frac{2\hat{s}_j\hat{\omega}_j\log\left(Mt\right)}{N_{t,j}}},$$

$$z_{t,j^*}=\overline{D}_{t,j^*}-\sqrt{\frac{2\hat{s}_{j^*}\hat{\omega}_{j^*}\log\left(Mt\right)}{N_{t,j^*}}}.$$

Therefore, (33) can be re-written as:

$$\begin{aligned}N_{T,j}\leqslant&\frac{8(\hat{s}_j\hat{\omega}_j)^2\log\left(Mt\right)}{\Delta_j^2}+\sum_{t=u+1}^{T}\sum_{N_{t,j}=u+1}^{t}\sum_{N_{t,j^*}=1}^{t}\\&\left(1\left\{z_{t,j}\leqslant\mu_j\right\}+1\left\{z_{t,j^*}\geqslant\mu^*\right\}\right).\end{aligned}\quad(35)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2020.3047417, IEEE Transactions on Network Science and Engineering

Then, according to (25) and (26), taking the expectation value of both sides of (33), we can get the average number of caching task j, $E\left(N_{T,j}\right)$, as:

$$\begin{aligned} E\left(N_{T,j}\right) &\leqslant \frac{8(\hat{s}_j\hat{\omega}_j)^2\log\left(Mt\right)}{\Delta_j^2} + \sum_{t=u+1}^{T}\sum_{N_{t,j}=u+1}^{t}\sum_{N_{t,j^*}=1}^{t} \\ &\left[P\left\{z_{t,j} \leq \mu_j\right\} + P\left\{z_{t,j^*} \geqslant \mu^*\right\}\right] \\ &\leq \frac{8(\hat{s}_j\hat{\omega}_j)^2\log\left(Mt\right)}{\Delta_j^2} + \sum_{t=u+1}^{T}\sum_{N_{t,j}=u+1}^{t}\sum_{N_{t,j^*}=1}^{t} \\ &\left[2\left(Mt\right)^{-4(\hat{s}_j\hat{\omega}_j)^2}\right] \\ &\leqslant \frac{8(\hat{s}_j\hat{\omega}_j)^2\log\left(Mt\right)}{\Delta_j^2} + 2\sum_{t=1}^{\infty} M^{-4}t^{-4(\hat{s}_j\hat{\omega}_j)^2+2}. \end{aligned} \tag{36}$$

From the definition of $\hat{s}_j$ and $\hat{\omega}_j$ in (12) and (13), we know for sure that $-4(\hat{s}_j\hat{\omega}_j)^2 + 2$ is always larger than 1. Therefore, $t^{-4(\hat{s}_j\hat{\omega}_j)^2+2}$ will converge to a finite value: $O(1)$. Therefore, we can show:

$$E\left(N_{T,j}\right) \leqslant \frac{8(\hat{s}_j\hat{\omega}_j)^2\log\left(Mt\right)}{\Delta_j^2} + O(1). \tag{37}$$

Finally, based on (18), we can find the upper bound of regret as:

$$\begin{aligned} E\left(R_{T,j}\right) &= \sum_{j=1}^{K} E\left(N_{t,j}\right)\Delta_j \\ &= \sum_{j=1}^{K}\left(\frac{8(\hat{s}_j\hat{\omega}_j)^2\log\left(Mt\right)}{\Delta_j} + O(1)\right). \end{aligned} \tag{38}$$

Therefore, we prove an upper bound of the M-AUCB algorithm.

Yiming Miao received the B.Sc. degree in College of Computer Science and Technology from Qinghai Univerisity, Xining, China in 2016. Currently, she is a Ph.D candidate in School of Computer Science and Technology at Huazhong University of Science and Technology (HUST), Wuhan, China. Her research interests include edge computing, 5G mobile communication system, Internet of Things, unmanned aerial vehicle, robotics, blockchain and wireless sensor network, etc.

Yixue Hao is an associate professor in the School of Computer Science and Technology at Huazhong University of Science and Technology. He received the Ph.D degree in computer science from Huazhong University of Science and Technology (HUST), Wuhan, China, in 2017. His current research interests include 5G network, internet of things, edge computing, edge caching and cognitive computing.

Hamid Gharavi Hamid Gharavi received the Ph.D. degree from Loughborough University, Loughborough, U.K., in 1980. He joined the Visual Communication Research Department, AT&T Bell Laboratories, Holmdel, NJ, USA, in 1982. He was then transferred to Bell Communications Research (Bellcore) after the AT&T-Bell divestiture, where he became a Consultant on video technology and a Distinguished Member of Research Staff. In 1993, he joined Loughborough University as a Professor and Chair of Communication Engineering. Since September 1998, he has been with the National Institute of Standards and Technology, U.S. Department of Commerce, Gaithersburg, MD, USA. He was a Core Member of Study Group XV (Specialist Group on Coding for Visual Telephony) of the International Communications Standardization Body CCITT (ITU-T) and a member of the IEEE 2030 Standard Working Group. His research interests include smart grid, wireless multimedia, mobile communications and wireless systems, mobile ad hoc networks, and visual communications. He received the Charles Babbage Premium Award from the Institute of Electronics and Radio Engineering in 1986, the IEEE CAS Society Darlington Best Paper Award in 1989, the Washington Academy of Science Distinguished Career in Science Award for 2017. He was a Distinguished Lecturer of the IEEE Communication Society. He has been a Guest Editor for a number of Special Issues of the proceedings of the IEEE including Smart Grids, Sensor Networks & Applications, Wireless Multimedia Communications, Advanced Automobile Technologies, and Grid Resilience. He was a TPC Co-Chair for IEEE SmartGridComm in 2010 and 2012. He was a member of the Editorial Board of proceedings of the IEEE from January 2003 to December 2008. He was Editor-in-Chief of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY and IEEE WIRELESS COMMUNICATIONS.

Min Chen is a full professor in School of Computer Science and Technology at Huazhong University of Science and Technology (HUST) since Feb. 2012. He is the director of Embedded and Pervasive Computing (EPIC) Lab, and the director of Data Engieering Institute at HUST. He is the founding Chair of IEEE Computer Society (CS) Special Technical Communities (STC) on Big Data. He was an assistant professor in School of Computer Science and Engineering at Seoul National University (SNU). He worked as a Post-Doctoral Fellow in Department of Electrical and Computer Engineering at University of British Columbia (UBC) for three years. Before joining UBC, he was a Post-Doctoral Fellow at SNU for one and half years. He received Best Paper Award from QShine 2008, IEEE ICC 2012, ICST IndustrialIoT 2016, and IEEE IWCMC 2016. He serves as associate editor for IEEE Transactions on Big Data, IEEE Network, and IEEE Trans. on Cognitive Communications and Networking, etc. He was a Series Editor for IEEE Journal on Selected Areas in Communications. He is Co-Chair of IEEE ICC 2012-Communications Theory Symposium, and Co-Chair of IEEE ICC 2013-Wireless Networks Symposium. He is General Co-Chair for IEEE CIT-2012, Tridentcom 2014, Mobimedia 2015, and Tridentcom 2017. He has 300+ publications, including 200+ SCI papers, 100+ IEEE Trans./Journal papers, 34 ESI highly cited papers and 12 ESI hot papers. He has published 12 books, including Cognitive Computing and Deep Learning (2018) with China Machine Press and Big Data Analytics for Cloud/IoT and Cognitive Computing (2017) with Wiley. His Google Scholar Citations reached 25,280+ with an h-index of 79 and i10-index of 240. His top paper was cited 3000+ times. He is an IEEE Senior Member since 2009. He was selected as Highly Cited Research at 2018. He got IEEE Communications Society Fred W. Ellersick Prize in 2017, and the IEEE Jack Neubauer Memorial Award in 2019. His research focuses on cognitive computing, 5G Networks, wearable computing, big data analytics, robotics, machine learning, deep learning, emotion detection, and mobile edge computing, etc. Min Chen is a Fellow of IEEE.

Kai Hwang is a Professor of Electrical Engineering and Computer Science, University of Southern California (USC). He received the Ph.D. from the University of California, Berkeley in 1972. Prior to joining USC in 1986, he has taught at Purdue University for 11 years. He has served as the founding Editor-in-Chief of the Journal of Parallel and Distributed Computing from 1983 to 2011. Dr. Hwang has published 8 books and 250 scientific papers. According to Google Scholars, his work was cited over 15,000 times with an h-index of 52. His most cited book on Computer Architecture and Parallel Processing was cited more than 2,300 times and his PowerTrust (IEEE-TPDS, April 2007) paper was cited over 540 times. An IEEE Life Fellow, Hwang received Lifetime Achievement Award from IEEE Cloudcom-2012 for his pioneering contributions in the field of computer architecture, parallel, distributed and cloud computing, and cyber security.