OL-MEDC: An Online Approach for Cost-effective Data Caching in Mobile Edge Computing Systems

Xiaoyu Xia, Feifei Chen, Qiang He, *Senior Member, IEEE*, Guangming Cui, John Grundy, *Senior Member, IEEE*, Mohamed Abdelrazek, Athman Bouguettaya, *Fellow, IEEE*, and Hai Jin, *Fellow, IEEE*

Abstract—Mobile Edge Computing (MEC) has emerged to overcome the inability of cloud computing to offer low latency services. It allows popular data to be cached on edge servers deployed within users' geographic proximity. However, the storage resources on edge servers are constrained due to their limited physical sizes. Existing studies of edge caching have predominantly focused on maximizing caching performance from the mobile network operator's perspective, e.g., maximizing data retrieval success rate, minimizing system energy consumption, balancing the overall caching workload, etc. App vendors, as key stakeholders in MEC systems, need to maximize the caching revenue, considering the cost incurred and the benefit produced. We investigate this novel Mobile Edge Data Caching (MEDC) problem from the app vendor's perspective, and prove its $\mathcal{NP}$-hardness. We then propose Online MEDC (OL-MEDC), an approach that formulates MEDC strategies for app vendors, without requiring future information about data demands. Its performance is theoretically analyzed and experimentally evaluated. The experimental results demonstrate that OL-MEDC outperforms state-of-the-art approaches by at least 20.41% on average.

Index Terms—data caching, mobile edge computing, online algorithm, cost-effective.

1 Introduction

The world is witnessing an exponential growth of mobile and Internet-of-Things devices in recent years [1]. These devices generate enormous network traffic, often increase network latency and cause network congestion. To tackle this challenge, Mobile Edge Computing (MEC) has emerged to distribute computing and storage resources at the network edge by deploying edge servers at base-stations within end-users' geographic proximity [2]. Mobile and Internet-of-Things application vendors – referred to as *app vendors* hereafter – can request the use of computing and storage resources on edge servers to deploy applications for ensuring low-latency service and data accesses for their users [3]. Some computation tasks may be offloaded from their end-devices to edge servers for reducing computation time and energy consumption [4], [5].

As the number of end-users accessing edge applications increases, it is expected that a large amount of app data will be transmitted via edge servers between remote cloud servers and users' end-devices. Caching app vendors' data on edge servers, (e.g. viral videos), will significantly reduce their users' data retrieval latency because the users can retrieve data directly from edge servers within their close geographic proximity.

Data caching techniques have been intensively studied and applied to leverage their advantages in saving bandwidth consumption, minimizing access costs and reducing network latency [6], [7], [8]. In the last few years, there has been intensive research investigating network cache in the conventional network paradigms relying on different perspectives, e.g., information-theoretic caching [9] and request routing [10]. As a new computing paradigm, MEC offers new opportunities but poses new challenges for data caching. In MEC systems, the fundamental goal is to lower users' data retrieve latency by caching popular data on edge servers [11].

Research has started in earnest to study data caching problems in MEC systems – referred to as the Mobile Edge Data Caching (MEDC) problem hereafter – from the mobile network operator's perspective with different offline optimization objectives, e.g., minimum response latency [12], minimum delay cost [13], maximum data sharing efficiency [14]. However, these studies have not systematically considered the requirements and concerns of app vendors like Facebook or Uber who posses storage resources on edge servers to cache data. In the MEC environment, app vendors and mobile network operators are two stakeholders with quite different interests. From the mobile network operator's perspective, the main objective is to optimize the overall caching performance in edge data caching scenarios, e.g., maximizing data retrieval success rate [15], minimizing system energy consumption [16], or balancing the overall caching workload [17]. In addition to mobile users, app vendors are the mobile network operators' other group of

- *X. Xia, F. Chen and M. Abdelrazek are with School of Information Technology, Deakin University, Geelong, Victoria, Australia. E-mail: xiaoyu.xia@deakin.edu.au; feifei.chen@deakin.edu.au; mohamed.abdelrazek@deakin.edu.au.*
- *Q. He and G. Cui are with School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Victoria, Australia. E-mail: qhe@swin.edu.au; gcui@swin.edu.au.*
- *J. Grundy is with Faculty of Information Technology, Monash University, Melbourne, Victoria, Australia. E-mail: john.grundy@monash.edu.*
- *A. Bouguettaya is with School of Computer Science, University of Sydney, Australia. Email: athman.bouguettaya@sydney.edu.au.*
- *H. Jin is with School of Computer Science and Technology, Huazhong University of Science and Technology, China. Email: hjin@hust.edu.cn.*

customers in the MEC environment. The key challenge for app vendors is to formulate more cost-effective caching strategies, considering both the caching benefit obtained from providing low-latency data retrieval to its app users and the caching cost incurred based on the pay-as-you-go scheme.

Given a set of requested data in an area, caching all the requested data on every individual edge server is a straightforward solution for app vendors to ensure low latency for all the users. However, edge servers' cache capacities are normally limited and expensive because of their physical size limits [2], [18]. It is difficult to cache a large amount of data for each app vendor on each edge server because of the competitions among app vendors. A common practice is to reserve cache capacities on individual edge servers in advance. Users may then benefit from low-latency data retrievals provided by their nearby edge servers, producing *caching benefits* for the app vendor. In the meantime, *caching costs* are incurred based on the pay-as-you-go scheme when data are transmitted to be cached [19]. Both caching benefits and caching costs must be considered when cost-effectively formulating MEDC strategies for app vendors. In addition, different data will need to be cached dynamically over time. Cached data may need to be flushed out to save cache spaces for more popular data. These dynamic data demands are not known in advance and must be accommodated on the fly. Thus, offline MEDC approaches are subject to low effectiveness over time.

In this paper, *we study this online MEDC problem from the app vendor's perspective for maximizing the caching revenue, considering both caching benefits and caching costs.* The main contributions include:

- We formulate this online MEDC problem and prove its $\mathcal{NP}$-hardness.
- We propose OL-MEDC (Online MEDC), an online approach for formulating cost-effective MEDC strategies over time, to solve the MEDC problem over time without requiring future information about data demands.
- We analyze the guarantee bound of OL-MEDC theoretically.
- We compare OL-MEDC with five representative approaches through extensive experiments, and show OL-MEDC's advantages over these approaches in addressing the online MEDC problem.

The organization of the paper is as follows. An example motivating the MEDC problem is provided in Section 2. In Section 3, we present the system model, formulate the MEDC problem and prove that the MEDC problem is $\mathcal{NP}$-hard. In Section 4, we present the design of OL-MEDC, and prove its performance guarantee theoretically. Section 5 evaluates OL-MEDC experimentally against five representative approaches, and analyzes the threats to the validity of the experimental evaluation. The related work is reviewed in Section 6. We conclude this study and point out the future work in Section 7.

2 Motivating Example

In an MEC environment, edge servers in an area can transmit data with other via high-speed links [3], [4]. Those edge servers and links constitute an edge server network

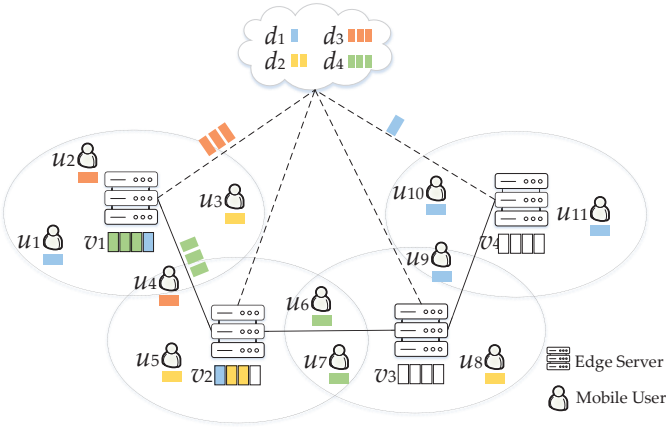


Fig. 1: An example MEDC scenario

[19]. This architecture single-point failure problem in the edge-cloud architecture, where a centralized macro base station is used to control the communications between edge servers [20]. It also avoids unpredictable network latency when edge servers can only communicate the backhaul network [19]. In fact, the edge-cloud architecture is a fully-connected edge server network. Thus, OL-MEDC can also accommodate the edge-cloud architecture.

Fig. 1 shows a typical MEC system, involving 4 edge servers, $\{v_1, ..., v_4\}$ and 11 users, $\{u_1, ..., u_{11}\}$ that request 4 data, $\{d_1, ..., d_4\}$. Compared to cloud servers, the storage resources at the network edge are constrained by edge servers' limited physical capacities. As discussed in Section 1, caching plenty of data on each edge server is usually impossible due to edge servers' constrained storage resources and the consequent competition among app vendors [2], [21]. This is the *capacity constraint*. Due to this constraint, an app vendor will not always be able to acquire the required cache spaces. Thus, app vendors normally need to reserve a certain number of cache spaces on every edge server in advance. Take Fig. 1 for example. The app vendor has reserved four cache spaces on edge server v_1. However, d_1, d_2 and d_3 are requested in the coverage area of v_1, and the cache spaces of v_1 do not suffices to cache those three kinds of data. Usually, reserving a large amount of cache spaces on an individual edge server is cost-ineffective because users' data demands for popular data vary from area to area [22]. The ability for edge servers to enable collaborative caching further reduces the app vendor's need to reserve large amounts of cache capacities on individual edge servers.

Each edge server in the MEC system has its coverage area, and only the users in this coverage area can access the edge server directly [23]. In an overlapping area covered by multiple edge servers, a user can access its local edge servers, i.e. the edge servers covering this user. For example, user u_4 in Fig. 1 can access both v_1 and v_2 directly. This is the *coverage constraint*. If a user cannot retrieve data from its local edge servers, this user can obtain the data from remote edge servers as long as the app-specific latency constraint is not violated [21], [24]. This is the *latency constraint*. Take u_6 in Fig. 1 for example. It can obtain data directly from its local edge servers v_2 and v_3, or remote edge servers v_1 and

v_4. If none of them has the requested data, the user has to retrieve it from the cloud. The main difference between these retrieval methods lies in the corresponding data retrieval latencies.

In addition to the above three unique constraints in MEC systems, the main difference between this study and existing studies is that this paper investigates the MEDC problem for app vendors, considering data dynamics in MEC systems. Please note that the cost of hiring cache spaces is fixed because cache spaces are reserved in advance by the app vendor. *The optimal MEDC strategy for app vendors should maximize the app vendor's caching revenue, i.e., caching benefit minus caching cost.* Fig. 1 shows that users u_9, u_{10} and u_{11} request data d_1. Caching d_1 on v_4 in its reserved cache spaces can reduce the data retrieval latency of those users, compared with retrieving data from the remote cloud. This produces the *caching benefit*. However, transmitting d_1 to v_4 is charged by the mobile network operator. This incurs the *caching cost*. Furthermore, the status of a dynamic real-world MEC system changes over time. For example, users may leave the system and new popular data may be requested by new users. The MEDC strategy must be updated accordingly to remain optimal, taking into account these system dynamics, the information of which is not available prior to their occurrences. Finding and implementing the global optimal MEDC strategy over a long period of time is impractical. The app vendor's MEDC must be updated over time in an online manner without the need for knowing future data dynamics in the real-world MEC system.

3 System Model and Problem Formulation

In this section, we first introduce the system architecture, then define the data retrieval latency, caching cost, and caching benefit under constrains discussed in Section 2.

3.1 System Architecture

Let V denote the set of edge servers and E denote the set of links in an MEC system, the edge server network can be modeled as a graph $G(V, E)$. The notations are summarized in Table 1.

Let D denote the data requested in the MEC system in a specific time slot t, let $s^t = \{s_1^t, ..., s_n^t\}$ denote the MEDC strategy for time slot t, where $s_i^t = \{s_{i,f}^t, \forall d_f \in D\}$, and $s_{i,f}^t$ indicates whether d_f is decided to be cached on v_i in time slot t:

$$s_{i,f}^t = \begin{cases} 0 & \text{if } d_f \text{ is not cached on } v_i \text{ in } t \\ 1 & \text{if } d_f \text{ is cached on } v_i \text{ in } t \end{cases} \quad (1)$$

Let $q_{m,d}^t$ denote whether user u_m requests data d_f in time slot t:

$$q_{m,f}^t = \begin{cases} 0 & \text{if } u_m \text{ does not request } d_f \text{ in } t \\ 1 & \text{if } u_m \text{ requests } d_f \text{ in } t \end{cases} \quad (2)$$

The data cached on v_i must not exceed the reserved spaces on v_i in any time slot:

$$\sum_{d_f \in D} s_{i,f}^t \cdot |d_f| \leq \mathcal{A}_i, \forall v_i \in V, t \in T \quad (3)$$

where $|d_f|$ is the size of d_f. This is the *cache space constraint*.

TABLE 1: Summary of Notations

Notation	Description
$\mathcal{A}_i$	reserved cache spaces on v_i
c	remote cloud server
c_{ce}	unit data transmission cost from cloud to edge server
c_{ee}	unit data transmission cost between edge servers
D	set of requested data
d_f	data f
E	set of edges in G
G	edge server network
$q_{m,f}^t$	binary variable indicating u_m requests for d_f in t
$\mathcal{S}$	data caching strategy over T
s^t	data caching strategy in t
$s_{i,f}^t$	binary variable indicating d_f will be cached on v_i at the end of t
t	time slot t
T	set of time slots
V	set of edge servers
v_i	edge server i
U	set of end-users' devices
U_j	set of end-users' devices covered by v_j
u_m	end-users' device m
γ	unit of caching benefit
$\Phi_{j,i}$	hops between edge server v_j and v_i
Φ_L	latency limit
$\Phi_{i,f}^t$	lowest latency transmitting d_f for v_i in t
$\Phi_{m,f}^t$	lowest latency of u_m retrieving d_f in t

3.2 Data Retrieval Latency

There are two main components in the data retrieval latency of a user, including the latency transmitting data between edge servers and the latency transmitting data to the user from an edge server covering this user. Since the latter is extremely low in 5G and is not influenced by the MEDC strategy, we do not consider this component in the model. Modeling the connected edge servers in the MEC system as a graph in the same way as [19], the number of hops in MEC systems is used to generically quantify data retrieval latency. The latency of user u_m retrieving data d_f in time slot t, denoted by $\Phi_{m,f}^t$, is calculated as follows:

$$\Phi_{m,f}^t = \min\{\Phi_{j,i}, s_{j,f}^t = 1, \forall v_j \in V \bigcup c\}, \forall u_m \in U_i \quad (4)$$

where v_j is the edge server caching d_f, v_i is the edge server covering u_m, and $\Phi_{j,i}$ is the minimum number of hops between v_j and v_i.

Remark: Similar to [19], [21], specific latency models can be easily integrated into OL-MEDC to accommodate various real-world MEDC scenarios. Specifically, $\Phi_{j,i}$ in (4) can be replaced with a latency function $\Phi(j, i, f)$ that represents the latency in delivering data d_f from v_j to v_i:

$$\Phi_{m,f}^t = \min\{\Phi(j, i, f), s_{j,f}^t = 1, \forall v_j \in V \bigcup c\}, \forall u_m \in U_i \quad (5)$$

Let Φ_L denote the app-specific latency constraint, i.e., the maximum data retrieval latency allowed. If $\Phi_{m,f}^t$ is higher than Φ_L, u_m will access d_f from the remote cloud.

3.3 Caching Cost

To cache a data d_f on an edge server in the MEC system, d_f can be transmitted from another edge server or from the cloud. Either way, the transmission cost occurs. The caching cost incurred by implementing an MEDC strategy s^t is the total transmission cost incurred by transmitting D according to s^t. As discussed in Section 2, the app vendor reserves cache spaces on edge servers in advance. Thus, the cost incurred by hiring cache spaces is fixed and thus does not need to be considered here.

Similar to (4), let $\Phi^t_{i,f} = min\{\Phi_{i,j}, s^{t-1}_{j,f} = 1, \forall v_j \in V\}$ denote the lowest latency in transmitting d_f to v_i in time slot t. There are two types of data transmissions: from the remote cloud server to an edge server and from an edge server to another. Let c_{ce} denote the unit cost of transmitting data from cloud to edge servers, and c_{ee} denote the unit cost of transmitting data between edge servers. If the cost incurred by transmitting d_f from the remote cloud server ($c_{ee} \cdot \Phi^t_{i,f} > c_{ce}$) is lower than that from another edge server in the MEC system ($c_{ce} < c_{ee} \cdot \Phi^t_{i,f}$), the data will be transmitted from the remote cloud to v_i. Thus, the cost of transmitting data d_f in time slot t, denoted as $\mathcal{C}(s^t)$, is:

$$\mathcal{C}(s^t) = \sum_{v_i \in V} \sum_{d_f \in D} |d_f| \cdot s^t_{i,f}(1 - s^{t-1}_{i,f}) \cdot cost^t_{i,f} \tag{6}$$

where $cost^t_{i,f} = \min\{c_{ee} \cdot \Phi^t_{i,f}, c_{ce}\}$ is the minimum cost for delivering d_f to v_i in time slot t.

Remark: Similar to latency, specific transmission cost models can be easily integrated into OL-MEDC. A cost function $c(j, i, f)$ that represents the cost of delivering d_f from v_j to v_i can be employed to calculate the actual minimum cost of delivering d_f to v_i in time slot t:

$$cost^t_{i,f} = \min\{c(j, i, f), s^t_{j,f} = 1, \forall v_j \in V \bigcup c\} \tag{7}$$

3.4 Caching Benefit

As discussed in Section 2, a user retrieves data from a local edge server covering it or a remote edge server caching the data as long as it does not violate the app-specific latency constraint Φ_L. Thus, caching benefit is yielded when a user can retrieve data in the MEC system within Φ_L. Let $\mathcal{B}^t_{m,f}$ be the caching benefit yielded for individual user u_m's retrieval of data d_f. It can be calculated as follows:

$$\mathcal{B}^t_{m,f} = \begin{cases} \max\{\Phi_L - \Phi^t_{m,f}, 0\} & \text{if } u_m \text{ retrieves } d_f \text{ from an edge server} \\ 0 & \text{if } u_m \text{ retrieves } d_f \text{ from the remote cloud} \end{cases} \tag{8}$$

Now, the caching benefit yield by an MEDC strategy in time slot t can be calculated by:

$$\mathcal{B}(s^t) = \sum_{u_m \in U} \sum_{d_f \in D} \mathcal{B}^t_{m,f} \cdot q^t_{m,f} \tag{9}$$

3.5 Problem Formulation And Hardness

Let γ denote the app vendor's priority for lowering users' data retrieval latency. A large γ indicates that the app vendor is inclined to lower network latency for its users at higher caching costs, and vice versa. Based on the cost and benefit models in Sections 3.3 and 3.4, the caching revenue produced by MEC strategy s^t, denoted by $\mathcal{P}(s^t)$, can be calculated by caching benefit $\mathcal{B}(s^t)$ minus caching cost $\mathcal{C}(s^t)$:

$$\begin{aligned} \mathcal{P}(s^t) &= \gamma \cdot \mathcal{B}(s^t) - \mathcal{C}(s^t) \\ &= \sum_{u_m \in U} \sum_{d_f \in D} \gamma \cdot \mathcal{B}^t_{m,f} \cdot q^t_{m,f} - \sum_{v_i \in V} \sum_{d_f \in D} |d_f| \cdot s^t_{i,f} \cdot \\ &\quad (1 - s^{t-1}_{i,f}) \cdot cost^t_{i,f} \end{aligned} \tag{10}$$

Now, the MEDC problem over a period of time T that consists of multiple time slots can be modeled as follows:

$$\begin{aligned} \max \quad & \lim_{T \to \infty} \sum_{t=1}^{T} \mathcal{P}(s^t) \\ s.t.: \quad & (1), (2), (3), (4) \end{aligned}$$

Now, we prove the $\mathcal{NP}$-hardness of the MEDC problem in an individual time slot t (referred to as the t-MEDC problem hereafter) with Theorem 1.

Theorem 1. *The t-MEDC problem in a time slot is $\mathcal{NP}$-hard.*

Proof To do the proof, we first introduce the weighted k-set packing (WKSP) problem, a classic $\mathcal{NP}$-hard problem. Let $\mathcal{X}$ denote an element universe with $\forall x \in \mathcal{X}$ and S denote the set of all subsets in $\mathcal{X}$. Each subset $s \in S$ covers a set of elements with weight $w(s)$. Given the limit k, the WKSP problem can be represented by:

$$\max \sum_{s \in S} w(s) \cdot c_s \tag{11a}$$

$$s.t.: \sum_{s \in S} c_s \leq k \tag{11b}$$

$$\sum_{x \in \mathcal{X}} c_x \leq 1 \tag{11c}$$

where $c_s \in \{0, 1\}$ indicates whether set $s \in S$ is included into the solution, and $c_x \in \{0, 1\}$ indicates whether element x is covered.

Now we present how to reduce a special case of the t-MEDC problem with same size data to the WKSP problem. Here we construct an 1-time-slot instance $t - MEDC(V, U, n, \mathcal{P}(s))$ based on a given instance $MEDC(V, U, n, \mathcal{P}(s))$ in the polynomial time where $|S| = |V|$, $|\mathcal{X}| = |U|$ and $n = k$. The function $\mathcal{P}(s)$ is the caching revenue produced by the MEDC strategy s based on (10). In this case, we can project the revenue function $\mathcal{P}(s)$ to $w(s)$. This way, any solution satisfying the objective of the t-MEDC problem also satisfies objective (11a). As constraint (11b) constrains the maximum number of selected sets, $\sum_{v_i \in V} \sum_{d_f \in D} s_{i,f} \leq \sum_{v_i \in V} \mathcal{A}_i$ based on (3) can be projected to (11b). Since the caching benefit of each data request d_f of user u_m can only be counted once, (11c) is satisfied. In this case, any solution satisfying constraints (3) and (4) also satisfies constraints (11b) and (11c). Thus, the WKSP problem can be reduced from the t-MEDC problem. Since the WKSP is $\mathcal{NP}$-hard, the t-MEDC problem is also $\mathcal{NP}$-hard. □

The t-MEDC problem is a special case of the MEDC problem where $T = 1$. Thus, the MEDC problem over T is also $\mathcal{NP}$-hard based on Theorem 1.

Algorithm 1 Single Time-slot MEDC Algorithm (ST-MEDC)

1: initialization
2: $\tilde{s}, \hat{s} \leftarrow \varnothing$
3: end of initialization
4: $\tilde{s}_{i,f} \leftarrow \varnothing$
5: **repeat**
6: $\tilde{s} \leftarrow \tilde{s} \bigcup \tilde{s}_{i,f}$
7: obtain the most cost-effective data caching decision $\tilde{s}_{i,f}$ under cache space constraint: $\tilde{s}_{i,f} = \arg\max\{\frac{\mathcal{B}(\tilde{s} \bigcup \tilde{s}_{i,f}) - \mathcal{B}(\tilde{s})}{|d_f|}, \forall v_i \in V, d_f \in D\}$
8: **until** no feasible decision $\tilde{s}_{i,f}$
9: $\hat{s}_{i,f} \leftarrow \varnothing$
10: **repeat**
11: $\hat{s} \leftarrow \hat{s} \bigcup \hat{s}_{i,f}$
12: obtain the data caching decision $\hat{s}_{i,f}$ with the highest benefit increase under cache space constraint: $\hat{s}_{i,f} = \arg\max\{\mathcal{B}(\hat{s} \bigcup \hat{s}_{i,f}) - \mathcal{B}(\hat{s}), \forall v_i \in V, d_f \in D\}$
13: **until** no feasible decision $\hat{s}_{i,f}$
14: **return** $\arg\max_{s \in \{\tilde{s}, \hat{s}\}} \mathcal{B}(s)$

4 Algorithm Design and Analysis

The complete information about the MEC system over time is required to solve the MEDC problem optimally. However, it is unrealistically in real-world scenarios where users' data requests usually arrive dynamically. In this section, we first present the ST-MEDC (Single Time-slot MEDC) algorithm for solving the MEDC problem in a single time slot. Then, we introduce the OL-MEDC (Online MEDC) algorithm for finding near-optimal MEDC strategies over time.

4.1 Single Time-slot MEDC Algorithm

Finding optimal solutions in large-scale MEDC scenarios is intractable, due to the $\mathcal{NP}$-hardness of the t-MEDC problem. Thus, OL-MEDC employs a heuristic algorithm named ST-MEDC (Single Time-slot MEDC) to solve the t-MEDC problem, aiming to maximize the caching benefit $\mathcal{B}(s^t)$ in individual time slots.

The pseudo-code of ST-MEDC is shown in Algorithm 1. Starting with initialization, ST-MEDC creates two initial MEDC strategy candidates $\tilde{s}$ and $\hat{s}$ (Lines 1-3). Then, it obtains $\tilde{s}$ first in an iterative manner: always including the data caching decision $\tilde{s}_{i,f}$ that produces the highest ratio of caching benefit over used caching spaces $\frac{\mathcal{B}(\tilde{s} \bigcup \tilde{s}_{i,f}) - \mathcal{B}(\tilde{s})}{|d_f|}$ into $\tilde{s}$ until the cache space constraint (3) is violated or no feasible decision $\tilde{s}_{i,f}$ can be found (Lines 4-8). Similarly, the algorithm obtains $\hat{s}$ by selecting the data caching decision $\hat{s}_{i,f}$ with the highest increase in caching benefit $\mathcal{B}(\tilde{s} \bigcup \tilde{s}_{i,f}) - \mathcal{B}(\tilde{s})$ (Lines 9-13). In Line 14, the solution with the higher total caching benefit is the final result of Algorithm 1.

The computational complexities of finding decisions in Lines 7 and 12 are at most $O(|V| \cdot |D|)$, respectively. There are $\sum_{v_i \in V} \mathcal{A}_i$ at most iterations in each of the two loops in ST-MEDC. Thus, the computational complexity of ST-MEDC is $O(2 \cdot |V| \cdot |D| \sum_{v_i \in V} \mathcal{A}_i) = O(|V|^2 \cdot |D|)$.

Algorithm 2 Online MEDC Algorithm (OL-MEDC)

1: initialization
2: $\beta = 0, t = 1, s^0 \leftarrow \varnothing, \mathcal{S} = \{s^0\}$
3: end of initialization
4: **while** $t \leq T$ **do**
5: obtain MEDC strategy s^t by Algorithm 1
6: calculate $\mathcal{B}(s^t), \mathcal{C}(s^t)$ by s^t and s^{t-1}
7: calculate $\mathcal{B}(s^{t-1})$ by s^{t-1}
8: **if** $\gamma \cdot \beta \geq k \cdot \mathcal{C}(s^t)$ **or** $\beta = 0$ **then**
9: $\beta = \mathcal{B}(s^t)$
10: **else**
11: $\beta = \beta + \mathcal{B}(s^{t-1})$
12: $s^t \leftarrow s^{t-1}$
13: **end if**
14: $\mathcal{S} \leftarrow \mathcal{S} \bigcup s^t$
15: $t = t + 1$
16: **end while**

4.2 Online MEDC Algorithm

Now we present OL-MEDC, the online algorithm for formulating cost-effective MEDC strategies over time based on ST-MEDC. The pseudo-code of OL-MEDC is shown in Algorithm 2. As discussed in Section 2 and Section 3, caching cost is an important component in the caching revenue. Updating the MEDC strategy in every time slot may incur high caching costs. Thus, OL-MEDC updates the current MEDC strategy only when it has already produced adequate benefits, i.e. $\gamma \cdot \beta > k$ times the caching cost incurred by implementing the MEDC strategy update, where β is the accumulated caching benefit produced by the current MEDC strategy since its implementation and k is a parameter specified by the app vendor based on its willingness to trade off caching cost for caching benefit. In general, a large k will tend to reduce caching costs by keeping the current MEDC strategy.

Algorithm 2 initializes the accumulated caching benefit by $\beta = 0$ and creates an initial MEDC strategy (Lines 1-3). In each time slot, Algorithm 2 first obtains an approximation solution s^t with Algorithm 1 (Line 5), then calculates the benefit produced by s^t and the caching cost incurred by updating s^{t-1} to s^t (Line 6). After that, it calculates the caching benefit produced with the current MEDC strategy unchanged (Line 7). Then, through Line 8 to Line 15, the algorithm compares the accumulated caching benefit β obtained by s^{t-1}: if $\beta > \frac{1}{\gamma}$ times $\mathcal{C}(s^t)$ or s^{t-1} is infeasible, s^{t-1} is updated by s^t. Otherwise, s^{t-1} remains and no extra caching cost incurs.

As discussed in Section 4.1, the computational complexity of ST-MEDC is $O(|V|^2 \cdot |D|)$. Thus, the computational complexity of OL-MEDC in each time slot is also $O(|V|^2 \cdot |D|)$. This indicates the high efficiency of OL-MEDC and allows it to formulate MEDC strategies rapidly over time.

4.3 Performance Analysis

Here, we first analyze the approximation ratio of ST-MEDC in terms of caching benefit, i.e., the ratio of the caching benefit produced by ST-MEDC in the worst case over that produced by the optimal solution. After that, we analyze

the competitive ratio of OL-MEDC over time, i.e., the ratio of the total caching revenue produced by OL-MEDC in the worst case over that produced by the optimal strategy, based on the approximation ratio of ST-MEDC.

In terms of caching benefit, we obtain the approximation ratio of ST-MEDC by analyzing the performance bound of $\tilde{s}$ with Theorems 2 - 5. We divide each data cache decision $\tilde{s}_{i,f}$ into $|d_f|$ sub-decisions, where each sub-decision produces $\frac{\mathcal{B}(\tilde{s}\bigcup\tilde{s}_{i,f})-\mathcal{B}(\tilde{s})}{|d_f|}$ caching benefit. Let $\tilde{s}'$ denote the set of sub-decisions based on data caching strategy $\tilde{s}$.

Let $\tilde{s}'_l$ denote the sub-decision set when the l^{th} sub-decision is included in $\tilde{s}'$. Let s^* denote the optimal solution of the t-MEDC problem, and $\mathcal{B}(s^*)$ denote the caching benefit yielded by the optimal t-MEDC strategy. The increase in the caching benefit by including the l^{th} sub-decision, denoted by $\Delta\tilde{\mathcal{B}}_l$, is at least $\frac{\mathcal{B}(s^*)-\mathcal{B}(\tilde{s}'_{l-1})}{\sum_{v_i\in V}\mathcal{A}_i}$.

Theorem 2. For the l^{th} sub-decision included in $\tilde{s}'$, the increase in benefit, $\Delta\tilde{\mathcal{B}}_l$, follows:

$$\Delta\tilde{\mathcal{B}}_l \geq \frac{\mathcal{B}(s^*)-\mathcal{B}(\tilde{s}'_{l-1})}{\sum_{v_i\in V}\mathcal{A}_i} \tag{12}$$

Proof To proof this theorem, we first divide the optimal the optimal t-MEDC strategy s^* into a set of sub-decisions s'^*, where $\mathcal{B}(s^*)=\mathcal{B}(s'^*)$. Since $\tilde{s}'_l$ selects the sub-decision with the maximum ratio of benefit when including the lth sub-decision, for the first sub-decision in s'^* but not in $\tilde{s}'_{l-1}$, it is at most $\Delta\tilde{\mathcal{B}}_l$ increased in the caching benefit. In addition, there is at most $\sum_{v_i\in V}\mathcal{A}_i$ cache spaces available in the MEC system. Thus, the total caching benefit produced by $\{s'|s'\in s'^*\bigcap\neg\tilde{s}'_{l-1}\}$ is at most $\sum_{v_i\in V}\mathcal{A}_i\cdot\Delta\tilde{\mathcal{B}}_l$. Thus, the above inequality is satisfied. □

Now, we prove that the caching benefit produced by $\tilde{s}'_l$ provided by ST-MEDC with the l^{th} sub-decision included, is at least $\left(1-\left(1-\frac{1}{\sum_{v_i\in V}\mathcal{A}_i}\right)^{l-1}\right)$ times the caching benefit produced by s'^* with Theorem 3.

Theorem 3. For each included sub-decision l, the caching benefit produced by $\tilde{s}'_l$ fulfills the following:

$$\mathcal{B}(\tilde{s}'_l)\geq\left(1-\left(1-\frac{1}{\sum_{v_i\in V}\mathcal{A}_i}\right)^{l-1}\right)\mathcal{B}(s'^*) \tag{13}$$

Proof According to Theorem 2, we can obtain the caching benefit produced by $\tilde{s}'_l$ based on (14).

$$\begin{aligned}\mathcal{B}(\tilde{s}'_l)&=\mathcal{B}(\tilde{s}'_{l-1})+\Delta\tilde{\mathcal{B}}_l\geq\mathcal{B}(\tilde{s}'_{l-1})+\frac{\mathcal{B}(s'^*)-\mathcal{B}(\tilde{s}'_{l-1})}{\sum_{v_i\in V}\mathcal{A}_i}\\&=\left(1-\frac{1}{\sum_{v_i\in V}\mathcal{A}_i}\right)\mathcal{B}(\tilde{s}'_{l-1})+\frac{1}{\sum_{v_i\in V}\mathcal{A}_i}\mathcal{B}(s'^*)\end{aligned} \tag{14}$$

This way, we can prove this theorem by the inductive proof easily, and omitted details here. □

According to Theorem 3, the lower bound of the caching benefit with the $(l+1)^{th}$ sub-decision of $\tilde{s}'$ can be calculated:

$$\mathcal{B}(\tilde{s}'_{l+1})\geq\left(1-\left(1-\frac{1}{\sum_{v_i\in V}\mathcal{A}_i}\right)^{l}\right)\mathcal{B}(s'^*) \tag{15}$$

The total amount of available cache spaces is $\sum_{v_i\in V}\mathcal{A}_i$. When all the cache spaces are occupied ($l=\sum_{v_i\in V}\mathcal{A}_i$), the caching benefit produced by strategy $\tilde{s}_{l+1}$ fulfils:

$$\begin{aligned}\mathcal{B}(\tilde{s}'_{l+1})&\geq\left(1-\left(1-\frac{1}{\sum_{v_i\in V}\mathcal{A}_i}\right)^{l}\right)\mathcal{B}(s'^*)\\&=\left(1-\left(1-\frac{1}{l}\right)^{l}\right)\mathcal{B}(s'^*)\geq\left(1-\frac{1}{e}\right)\mathcal{B}(s'^*)\end{aligned} \tag{16}$$

However, $\tilde{s}_{l+1}$ violates the cache space constraint with the $(l+1)^{th}$ sub-decision included into $\tilde{s}'_{l+1}$, where $l=\sum_{v_i\in V}\mathcal{A}_i$. Here, we prove the lower bound of the caching benefit produced by $\tilde{s}'_l$ with Theorem 4.

Theorem 4. When $l=\sum_{v_i\in V}\mathcal{A}_i$, the benefit produced by $\tilde{s}'_l$ is at least $\frac{e-1}{2e}$ times what is produced by s'^*:

$$\mathcal{B}(\tilde{s}'_l)=\mathcal{B}(\tilde{s}'_{\sum_{v_i\in V}\mathcal{A}_i})\geq\frac{e-1}{2e}\mathcal{B}(s'^*)$$

Proof According to (16), the benefit produced by $\tilde{s}'_l$ fulfills:

$$\mathcal{B}(\tilde{s}'_l)\geq\left(1-\frac{1}{e}\right)\mathcal{B}(s'^*)-\tilde{\Delta\mathcal{B}_{l+1}} \tag{17}$$

Since the increase in the caching benefit in the $l+1^{th}$ time slot $\tilde{\Delta\mathcal{B}_{l+1}}$ cannot be higher than that in the l^{th} time slot:

$$\tilde{\Delta\mathcal{B}_{l+1}}\leq\Delta\tilde{\mathcal{B}}_l\leq\mathcal{B}(\tilde{s}'_l) \tag{18}$$

Thus, we can obtain:

$$\mathcal{B}(\tilde{s}'_l)\geq\left(1-\frac{1}{e}\right)\mathcal{B}(s'^*)-\mathcal{B}(\tilde{s}'_l)\geq\frac{e-1}{2e}\mathcal{B}(s'^*) \tag{19}$$

Thus, the theorem holds. □

Please note that the MEDC strategy $\tilde{s}$ obtained by ST-MEDC considers the differentiated data sizes. Thus, the set of sub-decisions $\tilde{s}'$ derived from $\tilde{s}$ will not always contain up to $\sum_{v_i\in V}\mathcal{A}_i$ sub-decisions. Now, we analyze the approximation ratio of the t-MEDC strategy s provided by ST-MEDC in terms of the caching benefit:

Theorem 5. The caching benefit produced by ST-MEDC is at least $\frac{(1-\omega)(e-1)}{2e}$ times the caching benefit produced by the optimal solution s^*, where $\omega=\frac{|V|\cdot\min\left\{\min\left\{\mathcal{A}_i,\forall v_i\in V\right\},\max\left\{|d_f|,\forall d_f\in D\right\}\right\}}{\sum_{v_i\in V}\mathcal{A}_i}$.

Proof We first analyze the performance of $\tilde{s}$ in ST-MEDC here, to obtain the approximation ratio of ST-MEDC.

$$\begin{aligned}\mathcal{B}(\tilde{s})&\geq\mathcal{B}(\tilde{s}'_{\sum_{v_i\in V}\mathcal{A}_i})-\omega\cdot\mathcal{B}(\tilde{s}'_{\sum_{v_i\in V}\mathcal{A}_i})\\&=(1-\omega)\cdot\mathcal{B}(\tilde{s}'_{\sum_{v_i\in V}\mathcal{A}_i})\geq(1-\omega)\cdot\frac{e-1}{2e}\mathcal{B}(s'^*)\\&=\frac{(1-\omega)(e-1)}{2e}\cdot\mathcal{B}(s'^*)=\frac{(1-\omega)(e-1)}{2e}\cdot\mathcal{B}(s^*)\end{aligned} \tag{20}$$

Since the final MEDC strategy constituted by ST-MEDC always produces benefit no lower than $\tilde{s}$, we can obtain:

$$\mathcal{B}(s)=\max\{\mathcal{B}(\tilde{s}),\mathcal{B}(\hat{s})\}\geq\mathcal{B}(\tilde{s})\geq\frac{(1-\omega)(e-1)}{2e}\cdot\mathcal{B}(s^*) \tag{21}$$

Thus, the approximation ratio of ST-MEDC in terms of caching benefit is $\frac{(1-\omega)(e-1)}{2e}$. □

According to Theorem 5, we can prove the competitive ratio of OL-MEDC in terms of caching revenue:

Theorem 6. The competitive ratio of OL-MEDC is $\frac{(k-\gamma)(1-\omega)^2(e-1)^2\eta}{4ke^2}$, where $\eta = \frac{\min\left\{\mathcal{B}(s^t),\forall t\in\{1,...,T\}\right\}}{\max\left\{\mathcal{B}(s^t),\forall t\in\{1,...,T\}\right\}}$.

Proof According to Line 8 in Algorithm 2, the caching cost is at most $\frac{\gamma}{k}$ times the accumulated caching benefit β. In the worst case, the MEDC has to be updated in every time slot over T and the total caching cost incurred ($\sum_{t=1}^{T}\mathcal{C}(s^t)$) is no more than $\frac{\gamma}{k}\sum_{t=1}^{T}\mathcal{B}(s^t)$. Thus, the total caching revenue produced by $\mathcal{S} = \{s^0, s^1, ..., s^T\}$ can be calculated by:

$$\sum_{t=1}^{T}\mathcal{P}(s^t) = \sum_{t=1}^{T}(\mathcal{B}(s^t) - \mathcal{C}(s^t)) \geq (\frac{k-\gamma}{k})\cdot\sum_{t=1}^{T}\mathcal{B}(s^t) \quad (22)$$

Let $\sum_{t=1}^{T}\mathcal{P}(s^{*t})$ denote the caching revenue obtained by the offline optimal approach over T.

$$\begin{aligned}\sum_{t=1}^{T}\mathcal{P}(s^t) &\geq (\frac{k-\gamma}{k})\cdot\sum_{t=1}^{T}\frac{(1-\omega)(e-1)}{2e}\mathcal{B}^*_{min}\\ &\geq (\frac{k-\gamma}{k})\cdot\frac{(1-\omega)(e-1)}{2e}\cdot\sum_{t=1}^{T}\eta^*\cdot\mathcal{B}^*_{max}\\ &\geq (\frac{k-\gamma}{k})\cdot\left(\frac{(1-\omega)(e-1)}{2e}\right)^2\cdot\eta\cdot\sum_{t=1}^{T}\mathcal{P}(s^{*t})\end{aligned} \quad (23)$$

Thus, OL-MEDC always provides a solution that achieves $\frac{(k-\gamma)(1-\omega)^2(e-1)^2\eta}{4ke^2}$ times the optimal offline solution in terms of the caching revenue.

□

5 Experimental Evaluation

5.1 Benchmark Approaches

In the experiments, OL-MEDC is compared with five representative approaches :

- **IP-MEDC**: In each individual time slot, it provides the optimal solution $\mathcal{P}(s^t)$ to t-MEDC problem described in Section 3.5 with IBM's CPLEX Optimizer.
- **CEDC-O** [21]: This online approach aims to maximize the coverage of user requests while minimizing the system cost. As mentioned in Section 2, the app vendor needs to reserve cache spaces. Thus, the cost of hiring cache spaces is not included in this approach.
- **Request-based Collaborative Caching (RCC)** [25]: This online approach focuses on serving the most users by the MEDC strategy over time in the MEC system.
- **AEDC** [26]: This offline approach finds solutions to t-MEDC problems with the aim to approximate the maximum caching benefits obtained by IP-MEDC. In the experiments, AEDC runs in each individual time slot to obtain the results.
- **Distributed Caching Algorithm (DCA)**: This distributed algorithm originates from [27] and is enhanced to solve the online MEDC problem. Edge servers communicate and cache data collaboratively in each time slot. This algorithm maximizes the overall caching revenue by heuristically caching data on edge servers that yield the highest caching revenues.

5.2 Experimental Settings

The real-world EUA dataset[1] is used for conducting the experiments in this study. In the experiments, a total of 200 mobile users are randomly selected from the dataset to simulate users in the system. According to the experimental settings, a certain number of edge servers are randomly selected from the dataset and connected to simulate an MEC system. The latency limit Φ_L is set to 2 hops. Similar to [28], a number of these users are randomly selected to send requests for a set of data (D) in each time slot, following $\mathcal{N}(\mu, \sigma^2)$, a normal distribution, where $\mu = \frac{|M|}{2}$ and $\sigma = \frac{|M|}{4}$. The sizes of data requested are also randomly selected between 1 to the maximum reserved cache spaces in the experiments.

AWS's Snowball Edge Pricing[2] is adopted in the experiments. We set c_{ce} to \$0.016, c_{ee} to \$0.006 and γ to \$0.004. The normal distribution $\mathcal{N}(\mu', \sigma'^2)$ is used to randomly generate the reserved cache spaces on every edge server, where μ' equals to half of the maximum reserved cache spaces among all the edge servers and $\sigma' = 1$.

5.3 Parameter Settings

To evaluate OL-MEDC comprehensively, we conduct seven sets of experiments to simulate various MEDC scenarios. Set #1 aims to demonstrate and compare the performance of the six approaches over 100 time slots, i.e., $T = 100$. Set #2 aims to demonstrate the applicability of OL-MEDC in four different MEDC modes:

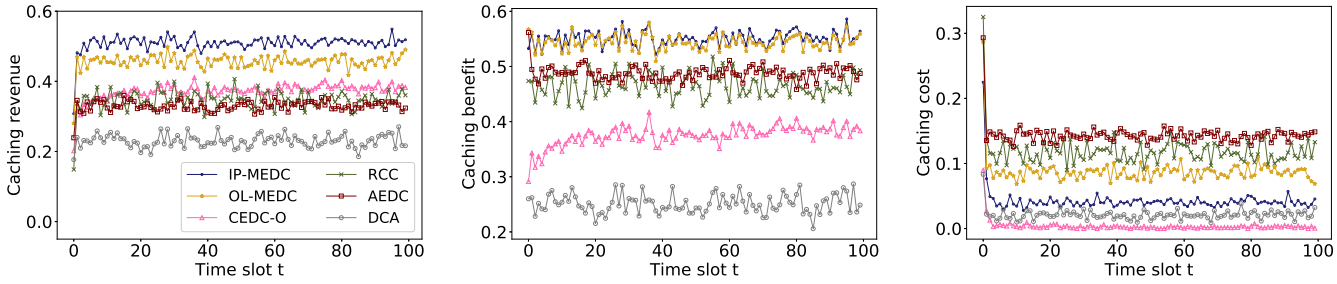
- **General Mode (GM)**. In this mode, the generic latency and cost models presented in Section 3 are applied and data demands in individual time slots follow a discrete uniform distribution across individual edge servers.
- **Zipf Mode (ZM)**. In this mode, the generic latency and cost models are applied in the same way as GM, while users' demands for different data follow a Zipf distribution, similar to [29].
- **Latency-specific Mode (LM)**. In this mode, a specific latency model is applied by randomly selecting a latency value from $(0, 2)$ for each link between two edge servers. In addition, a generic cost model is applied and data demands follow a discrete uniform distribution.
- **Cost-specific Mode (CM)**. In this mode, a specific cost model is applied by randomly selecting a cost value from $(0, 2)$ for each link between two edge servers. In addition, the generic latency model is applied and data demands follow a discrete uniform distribution.

As summarized in Table 2, we vary the value of one of the following five parameters while fixing the others in Sets #2-#7. In this way, we can evaluate OL-MEDC in different MEDC scenarios and demonstrate the impacts of the parameters. In these sets, each experiment also continues

1. https://github.com/swinedge/eua-dataset
2. https://aws.amazon.com/snowball-edge/pricing/

TABLE 2: Parameter Settings

	$Mode$	$\|V\|$	θ	MS	$\|D\|$	k
Set #1	GM	10	1.0	4	4	1
Set #2	GM, TM, LM, CM	10	1.0	4	4	1
Set #3	GM	6, 8, 10, 12, 14, 16	1.0	4	4	1
Set #4	GM	10	1.0, 1.2, 1.4, 1.6, 1.8, 2.0	4	4	1
Set #5	GM	10	1.0	2, 3, 4, 5, 6	4	1
Set #6	GM	10	1.0	4	2, 3, 4, 5, 6	1
Set #7	GM	10	1.0	4	4	1, 4, 16, 64, 256



(a) $\mathcal{P}(s^t)$ vs. t (b) $\mathcal{B}(s^t)$ vs. t (c) $\mathcal{C}(s^t)$ vs. t

Fig. 2: Set #1

for 100 time slots, is repeated 100 times when a setting parameter varies, and the average results are reported. The changed parameters in Sets #3-#7 are:

- Number of edge servers ($|V|$). This parameter varies from 6 to 16 in steps of 2 and impacts the size of the MEC system.
- Edge density (θ). Given n edge servers in the simulated MEC system, e edges are randomly generated based on $\theta = e/n$. This parameter increases from 1 to 2 in steps of 0.2.
- Maximum reserved cache spaces among all edge servers (MS). This parameter decides the maximum reserved cache spaces on the edge servers, varying from 2 to 6 in steps of 1.
- Number of data ($|D|$). This parameter increases from 2 to 6 in steps of 1 and is the number of users' requested data over T.
- Parameter k in Algorithm 2. This parameter varies among 1, 4, 16, 64, 256 and is used in Algorithm 2 to determine the updating frequency of the data caching strategy.

As discussed in Section 2, it is cost-ineffective to reserve huge cache spaces on individual edge servers. Therefore, those reserved cache spaces must not exceed the threshold MS in the experiments.

5.4 Performance Metrics

Four performance metrics are adopted in the experiments for evaluating OL-MEDC:

- Caching revenue $\mathcal{P}(s^t)$, the higher the better.
- Caching benefit $\mathcal{B}(s^t)$, the higher the better.
- Caching cost $\mathcal{C}(s^t)$, the lower the better.
- Maximum computation time, measured by seconds, the lower the better.

In the evaluation, we observe the maximum computation time of an approach across the 100 time slots to measure its efficiency and feasibility, rather than the average computation time. This is because if the approach freezes in any of the time slots due to excessive computation time, it will not be able to continue to update the MEDC strategy for the rest of T. Since DCA is a distributed algorithm, the computation time in each time slot is determined by the most time-consuming data caching decision. Please note that the efficiency results of Set #7 is not presented because k does not impact the computation time of OL-MEDC.

5.5 Effectiveness

Figs. 2 - 8 show the experimental results of all seven sets of experiments. Overall, *IP-MEDC achieves the highest average caching revenue, closely followed by OL-MEDC*. The average advantages of IP-MEDC and OL-MEDC are 29.78% and 20.41% over CEDC-O, 39.52% and 29.45% over RCC, 40.05% and 29.94% over AEDC and 149.80% and 131.76% over DCA.

Fig. 2 depicts the results of Set #1. Overall, *IP-MEDC and OL-MEDC's performance are stable over time, outperforming the other four approaches significantly in maximizing the caching revenue.* Fig. 2(a) shows that the average caching revenues achieved by IP-MEDC and OL-MEDC are 37.09% and 22.96% higher than CEDC-O, 46.64% and 31.52% higher than RCC, 54.36% and 38.45% higher than AEDC, 121.30% and 98.49% higher than DCA. The average caching revenue achieved by OL-MEDC reaches 89.69% of that achieved by IP-MEDC. This indicates the high effectiveness of OL-MEDC. The advantages of IP-MEDC and OL-MEDC in maximizing caching revenue come from their high ability to achieve high caching benefits. This can be seen in Fig.

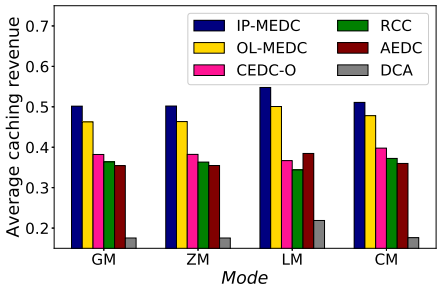


Fig. 3: Set #2

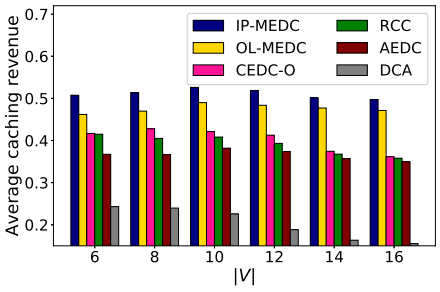


Fig. 4: Set #3

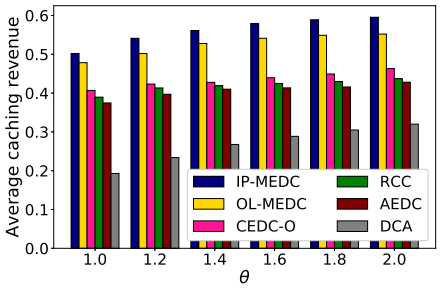


Fig. 5: Set #4

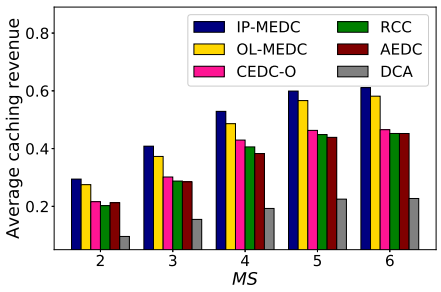


Fig. 6: Set #5

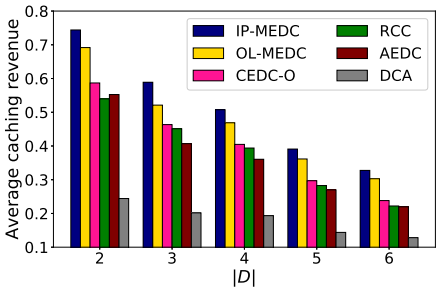


Fig. 7: Set #6

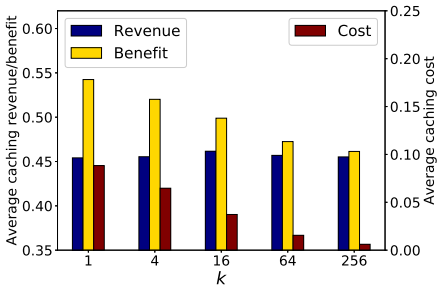


Fig. 8: Set #7

2(b). Achieving comparable caching benefits, IP-MEDC and OL-MEDC outperform CEDC-O, RCC, AEDC and DCA by large margins. Specifically, their average caching benefits are 46.98% and 45.17% higher than CDEC-O, 18.29% and 16.83% higher than RCC, 12.95% and 11.56% higher than AEDC, and 119.47% and 116.78% higher than DCA. Fig. 2(c) compares the caching costs incurred by the six approaches over T. CEDC-O incurs the lowest caching cost because CEDC-O aims to minimize the system cost. On average, the caching costs incurred by IP-MEDC and OL-MEDC are 0.0424 and 0.0881, lower than RCC and AEDC's 0.1190, 0.1583, but higher than CEDC-O and DCA's 0.0039 and 0.0212.

Fig. 3 demonstrates the average caching revenues achieved by all six approaches in Set #2. *In all four modes, IP-MEDC and OL-MEDC outperform CEDC-O, RCC, AEDC and DCA with large margins*. The average advantages of IP-MEDC and OL-MEDC are 34.87% and 24.58% over CEDC-O, 42.88% and 31.97% over RCC, 41.86% and 31.03% over AEDC and 176.39% and 155.30% over DCA. In the ZM mode where data demands follow the Zipf distribution, the six approaches achieve caching revenues similar to those achieved in other modes. This indicates the ability of OL-MEDC to accommodate data demands following different patterns. The average caching revenues achieved by IP-MEDC and OL-MEDC in the LM and CM modes are higher than those in GM and ZM modes. In the LM mode, IP-MEDC and OL-MEDC can effectively increase caching benefits by delivering data through low-latency links, which increases caching revenues. In the CM mode, they excel at minimizing caching costs by delivering data at low transmission costs, which also increases caching revenues.

Fig. 4 - Fig. 7 show the experimental results of Set #3 - #6. Those figures demonstrate that *IP-MEDC and OL-MEDC significantly outperform the other four approaches again* in terms of caching revenue. The results of Set #3 is depicted in Fig. 4 with various numbers of edge servers. With the initial increase from 6 edge servers, the caching revenues achieved by IP-MEDC and OL-MEDC increase. The reason is that more users can be served by their nearby edge servers and the caching benefits increases correspondingly. However, the caching revenues decrease when the number of edge servers exceeds 10. This is because the maximum achievable caching benefits are fixed with the fixed numbers of users. With more edge servers, there are more cache spaces in the MEC system. This increases the possibility of data transmissions, which potentially incurs extra caching costs. Thus, the caching revenues achieved by all approaches decrease in Fig. 4.

When the edge density θ increases, the caching revenue increases in Fig. 5, because each end-user has a higher chance of being served by an edge server under the latency constraint. As a result, all the approaches can achieve higher caching revenues. When the maximum reserved spaces increase from 2 to 6, the caching revenues achieved by all the fix approaches increase in Fig. 6. However, when the maximum reserved spaces increase from 5 to 6, the increase becomes much slower. This indicates that reserving a large amount of cache spaces on an individual edge server is usually cost-ineffective. Fig. 7 depicts the results of Set #6. When the number of requested data $|D|$ decreases, the caching revenues achieved by all the approaches decrease. The increase in $|D|$ leads to a higher possibility of transmitting data from the cloud because the reserved caching spaces are fixed. This way, it decreases the caching benefits and consequently the caching revenues.

Fig. 8 shows the impact of parameter k on OL-MEDC in caching revenue, benefit and cost. As discussed in Section 4.2, Algorithm 2 employs k to reduce caching costs. As shown in Fig. 8, *a larger k can indeed lower the caching cost*.

TABLE 3: Maximum computation time

	Set #1	Set #2	Set #3	Set #4	Set #5	Set #6
IP-MEDC	2.6540	2.6637	379.9838	3.3022	19.9294	7.6132
OL-MEDC	0.0095	0.0108	0.1541	0.0102	0.0087	0.0090
CDEC-O	0.3661	0.5285	3.1908	0.3836	0.6719	0.7918
RCC	0.0054	0.0061	0.0107	0.0078	0.0063	0.0075
AEDC	0.0057	0.0068	0.0129	0.0058	0.0066	0.0077
DCA	0.0005	0.0010	0.0008	0.0006	0.0005	0.0010

However, it also decreases the caching benefit in the meantime. Thus, an app vendor pursuing maximum caching benefit despite caching cost can feed a small k value to Algorithm 2. If an app vendor wants to maximize the cost-effectiveness of its MEDC strategy, it can select a k value that maximizes the caching revenue, e.g., $k = 16$ in Set #7.

5.6 Efficiency

In the experiments, we use the maximum computation time taken across the 100 time slots to evaluate the efficiency. The results are presented in Table 3. It demonstrates that *IP-MEDC takes much more computation time than others*, due to the $\mathcal{NP}$-hardness of the t-MEDC problem. In particular, the maximum computation times taken by IP-MEDC and CDEC-O are 379.9838 seconds and 3.1908 seconds in Set #5, while the maximum computation times taken by OL-MEDC, RCC, AEDC and DCA are at most 0.1541 seconds, 0.0107 seconds, 0.0129 seconds and 0.0008 seconds, respectively. This shows that *OL-MEDC is computationally feasible to deploy on large scales, real-world edge data caching problems*.

5.7 Threats to Validity

In this section, we analyze the threats to the validity of the experimental evaluation, including the threats to construct validity, internal validity and external validity.

5.7.1 Threats to Construct Validity

The main threats to construct validity in the experiments are the generated graphs and five comparison approaches. Randomly generated graphs may not always illustrate real-world scenarios precisely. To minimize this threat, the graphs are randomly generated in each execution - 100 graphs are generated when a parameter changes. Moreover, the five comparison approaches, i.e., IP-MEDC, CDEC-O, RCC, AEDC and DCA, may not suffice to evaluate OL-MEDC comprehensively. To minimize this threat, we simulate different MEDC scenarios by varying five parameters. In addition, we also evaluate the applicability of OL-MEDC in different real-world scenarios by evaluating its performance in four MEDC modes.

5.7.2 Threats to Internal Validity

For the internal validity, the main threat is the experiment settings that may favor OL-MEDC over other approaches. To tackle this threat, we simulated various MEDC scenarios by changing six parameters for comprehensively and fairly comparing the performance of all the six approaches. In addition, the experiment was repeated 100 times to obtain the averaged results when a setting parameter varies. In this way, biased results obtained in extreme experiments, e.g., those with unrealistic data request distribution or edge server distribution, are neutralized.

5.7.3 Threats to External Validity

The generalize application of OL-MEDC in different MEDC scenarios is the main threat to the external validity. In this paper, we generically modeled the MEDC problem and evaluated all approaches to reduce this threat. We employed the number of hops to measure latency. Therefore, we can easily interpret the evaluation results with specific retrieval latency and data size models. Furthermore, the experiments were conducted on a real-world dataset. In addition, we changed the complexity and size of the MEDC problem by varying the parameters in the experiment settings. In this way, we can ensure the representativeness and comprehensiveness of experimental evaluations. Thus, the threat to external validity is mitigated.

6 Related Work

Mobile edge computing (MEC) extends cloud computing by pushing computing resources and services to the network edge [30]. App vendors can deploy their services and data on edge servers to offer their users low service latency and data retrieval latency.

Existing data caching approaches for cloud computing and conventional distributed computing are rendered obsolete, due to the characteristics of MEC systems. Researchers are starting to study data caching problems in MEC systems. In [31], the authors considered the caching revenue produced during the data delivery process. They proposed an auction mechanism to find an optimal data caching solution. The authors of [32] proposed a new edge cache architecture by including caches on smart vehicles into the network caches. This approach improves the resource utility and the effectiveness of this architecture. However, these studies focus on offline approaches, which require complete information about the MEC system over time. Thus, dynamic MEDC scenarios cannot be handled by these offline approaches.

Very recently, a number of online data caching approaches have been proposed. The authors of [33] studied the joint service caching and task offloading problem in MEC. They proposed a Lyapunov-based approach, namely OREO, to achieve the minimum network latency while ensuring the low energy consumption in the long-term. Considering users' mobility, the authors of [34] propose MOREA that allocates various resources, i.e. cache spaces and CPU circles on edge servers for scheduling offloading tasks. The authors of [18] studied a micro-service deployment problem with the aim to minimize the overall cost instead of data retrieval latency. They proposed IDA4ReE, a primal-dual based algorithm, to find the solution to this problem with consideration of resource constraints and performance requirements. However, existing studies investigated data caching problems in the MEC systems mainly to complement offloading scheduling. Thus, they failed to pay sufficient attention to data caching itself, as

a unique technology aiming to reduce data retrieval latency, especially from the app vendor's perspective.

The MEDC problem was first studied from the app vendor's perspective in [26]. This research aims to cache data in an MEC system for serving all the users while minimizing the total cost. The main limitations of this study are the lack of consideration of edge servers' storage capacities and the data dynamics on MEC systems. In [35], Xia et al. considered cache space reservation in the MEDC problem. However, they ignored the dynamics in MEC systems, and only focused on caching benefit without considering caching cost. The authors of [21] provided a Lyapunov-based online algorithm, CEDC-O, for solving the MEDC problem dynamically. However, they unrealistically assumed that app vendors can always hire the needed storage spaces on edge servers without reservation, and focused solely on user coverage rather than caching revenue. The experiment results in Section 5.5 show that the caching revenue produced by CEDC-O is much lower than that produced by OL-MEDC. To the best of our knowledge, OL-MEDC is the first attempt to solve the MEDC problem efficiently for app vendors in an online manner, taking into the unique constraints and data dynamics in real-world MEC systems.

7 Conclusion

In this paper, we investigated the Mobile Edge Data Caching (MEDC) problem in MEC systems from the app vendor's perspective. We identified the major challenges and modeled the MEDC problem formally. We then proved that the MEDC problem is $\mathcal{NP}$-hard. To accommodate the dynamics of MEC systems, we proposed OL-MEDC, an online approach for formulating MEDC strategies over time with a provable performance guarantee. Through extensive experiments, the results demonstrated the advantages of OL-MEDC in maximizing the caching revenue in MEC systems, compared with representative approaches. We will consider MEC systems that allow data to be partitioned for caching in our future work.

Acknowledgement

This research is funded by Australian Research Council Discovery Projects No. DP180100212, DP200102491 and Laureate Fellowship FL190100035.

References

[1] Z. Ning, P. Dong, X. Wang, X. Hu, L. Guo, B. Hu, Y. Guo, T. Qiu, and R. Y. K. Kwok, "Mobile edge computing enabled 5g health monitoring for internet of medical things: A decentralized game theoretic approach," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 2, pp. 463–478, 2021.

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

[3] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service placement with provable guarantees in heterogeneous edge computing systems," in *IEEE Conference on Computer Communications*. IEEE, 2019, pp. 514–522.

[4] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1619–1632, 2018.

[5] Z. Ning, P. Dong, X. Wang, X. Hu, J. Liu, L. Guo, B. Hu, R. Kwok, and V. C. M. Leung, "Partial computation offloading and adaptive task scheduling for 5g-enabled vehicular networks," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2020.

[6] G. Casale, "Analyzing replacement policies in list-based caches with non-uniform access costs," in *IEEE Conference on Computer Communications*. IEEE, 2018, pp. 432–440.

[7] A. Mukhopadhyay, N. Hegde, and M. Lelarge, "Optimal content replication and request matching in large caching systems," in *IEEE Conference on Computer Communications*, 2018, pp. 288–296.

[8] K. Ji, G. Quan, and J. Tan, "Asymptotic miss ratio of lru caching with consistent hashing," in *IEEE Conference on Computer Communications*, 2018, pp. 450–458.

[9] S. H. Lim, C.-Y. Wang, and M. Gastpar, "Information-theoretic caching: The multi-user case," *IEEE Transactions on Information Theory*, vol. 63, no. 11, pp. 7018–7037, 2017.

[10] 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.

[11] S. Li and T. Lan, "Hotdedup: Managing hot data storage at network edge through optimal distributed deduplication," in *IEEE Conference on Computer Communications*. IEEE, 2020, pp. 247–256.

[12] J. Xie, D. Guo, X. Shi, H. Cai, C. Qian, and H. Chen, "A fast hybrid data sharing framework for hierarchical mobile edge computing," in *IEEE Conference on Computer Communications*. IEEE, 2020, pp. 2609–2618.

[13] T. Tran and D. Pompili, "Adaptive bitrate video caching and processing in mobile-edge computing networks," *IEEE Transactions on Mobile Computing*, pp. 1–15, 2018.

[14] G. Luo, H. Zhou, N. Cheng, Q. Yuan, J. Li, F. Yang, and X. S. Shen, "Software defined cooperative data sharing in edge computing assisted 5g-vanet," *IEEE Transactions on Mobile Computing*, 2019.

[15] D. Malak, M. Al-Shalash, and J. G. Andrews, "Optimizing content caching to maximize the density of successful receptions in device-to-device networking," *IEEE Transactions on Communications*, vol. 64, no. 10, pp. 4365–4380, 2016.

[16] F. Gabry, V. Bioglio, and I. Land, "On energy-efficient edge caching in heterogeneous networks," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3288–3298, 2016.

[17] Z. Liu, Z. Bai, Z. Liu, X. Li, C. Kim, V. Braverman, X. Jin, and I. Stoica, "Distcache: Provable load balancing for large-scale storage systems with distributed caching," in *17th {USENIX} Conference on File and Storage Technologies ({FAST} 19)*, 2019, pp. 143–157.

[18] S. Deng, Z. Xiang, J. Taheri, K. A. Mohammad, J. Yin, A. Zomaya, and S. Dustdar, "Optimal application deployment in resource constrained distributed edges," *IEEE Transactions on Mobile Computing*, 2020.

[19] X. Xia, F. Chen, Q. He, J. Grundy, M. Abdelrazek, and H. Jin, "Cost-effective app data distribution in edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 31–44, 2020.

[20] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[21] X. Xia, F. Chen, Q. He, J. Grundy, M. Abdelrazek, and H. Jin, "Online collaborative data caching in edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 281–294, 2020.

[22] 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*, vol. 21, no. 4, pp. 915–929, 2018.

[23] L. Yuan, Q. He, S. Tan, B. Li, J. Yu, F. Chen, H. Jin, and Y. Yang, "Coopedge: A decentralized blockchain-based platform for cooperative edge computing," in *Proceedings of the 30th Web Conference*, 2021.

[24] Q. He, C. Wang, G. Cui, B. Li, R. Zhou, Q. Zhou, Y. Xiang, H. Jin, and Y. Yang, "A game-theoretical approach for mitigating edge ddos attack," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[25] A. Gharaibeh, A. Khreishah, B. Ji, and M. Ayyash, "A provably efficient online collaborative caching algorithm for multicell-coordinated systems," *IEEE Transactions on Mobile Computing*, vol. 15, no. 8, pp. 1863–1876, 2016.

[26] X. Xia, F. Chen, Q. He, G. Cui, P. Lai, M. Abdelrazek, J. Grundy, and H. Jin, "Graph-based data caching optimization for edge

computing," *Future generation computer systems*, vol. 113, pp. 228–239, 2020.

[27] Y. Huang, X. Song, F. Ye, Y. Yang, and X. Li, "Fair and efficient caching algorithms and strategies for peer data sharing in pervasive edge computing environments," *IEEE Transactions on Mobile Computing*, vol. 19, no. 4, pp. 852–864, 2019.

[28] S. Chen, L. Jiao, L. Wang, and F. Liu, "An online market mechanism for edge emergency demand response via cloudlet control," in *IEEE Conference on Computer Communications*. IEEE, 2019, pp. 2566–2574.

[29] Y. Yang and J. Zhu, "Write skew and zipf distribution: Evidence and implications," *ACM transactions on Storage (TOS)*, vol. 12, no. 4, pp. 1–19, 2016.

[30] B. Gao, Z. Zhou, F. Liu, and F. Xu, "Winning at the starting line: Joint network selection and service placement for mobile edge computing," in *IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1459–1467.

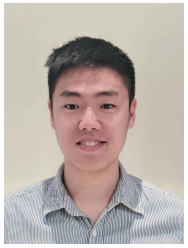
[31] X. Cao, J. Zhang, and H. V. Poor, "An optimal auction mechanism for mobile edge caching," in *38th IEEE International Conference on Distributed Computing Systems*, 2018, pp. 388–399.

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

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

[34] L. Wang, L. Jiao, J. Li, J. Gedeon, and M. Mühlhäuser, "Moera: Mobility-agnostic online resource allocation for edge computing," *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1843–1856, 2018.

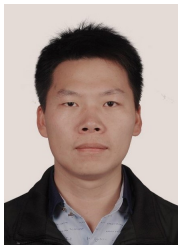
[35] X. Xia, F. Chen, J. Grundy, M. Abdelrazek, H. Jin, and Q. He, "Constrained app data caching over edge server graphs in edge computing environment," *IEEE Transactions on Services Computing*, 2021.



Xiaoyu Xia received his Master degree from The University of Melbourne, Australia in 2015. He is a PhD candidate at Deakin University. His research interests include edge computing, parallel and distributed computing, service computing, software engineering and cloud computing.

Feifei Chen received her PhD degree from Swinburne University of Technology, Australia in 2015. She is a lecturer at Deakin University. Her research interests include software engineering, cloud computing and green computing.

Qiang He received his first PhD degree from Swinburne University of Technology, Australia, in 2009 and his second PhD degree in computer science and engineering from Huazhong University of Science and Technology, China, in 2010. He is an Associate Professor at Swinburne. His research interests include service computing, software engineering, cloud computing and edge computing. More details about his research can be found at https://sites.google.com/site/heqiang/.



Guangming Cui received his Master degree from Anhui University, China, in 2018. He is a PhD candidate at Swinburne University of Technology. His research interests include software engineering, edge computing and service computing.

John C. Grundy received the BSc (Hons), MSc, and PhD degrees in computer science from the University of Auckland, New Zealand. He is currently Australian Laureate Fellow and a professor of software engineering at Monash University, Melbourne, Australia. He is an Associate Editor in Chief of IEEE Transactions on Software Engineering, and Associste Editor of Automated Software Engineering Journal and IEEE Software. His current interests include domain-specific visual languages, model-driven engineering, large-scale systems engineering, and software engineering education. More details about his research can be found at https://sites.google.com/site/johncgrundy/.

Mohamed Abdelrazek is an Associate Professor of Software Engineering and IoT at Deakin University. Before joining Deakin University in 2015, he worked as a senior research fellow at Swinburne University of Technology and Swinburne-NICTA software innovation lab (SSIL). Before 2010, he was the head of software development department at Microtech. More details about his research can be found at https://sites.google.com/site/mohamedalmorsy/.

Athman Bouguettaya is a Professor in the School of Computer Science at University of Sydney, Australia. He received his PhD in Computer Science from the University of Colorado at Boulder (USA) in 1992. He is or has been on the editorial boards of several journals including, the IEEE Transactions on Services Computing, ACM Transactions on Internet Technology, the International Journal on Next Generation Computing and VLDB Journal. He was the recipient of several federally competitive grants in Australia (e.g., ARC) and the US (e.g., NSF, NIH). He is a Fellow of the IEEE and a Distinguished Scientist of the ACM.

Hai Jin received the Ph.D. degree in computer engineering from Huazhong University of Science and Technology (HUST), Wuhan, China, in 1994. He is a Cheung Kung Scholars Chair Professor of computer science and engineering with the HUST. His research interests include computer architecture, virtualization technology, cluster computing and cloud computing, peer-to-peer computing, network storage, and network security.