Online Collaborative Data Caching in Edge Computing

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

Abstract—In the edge computing (EC) environment, edge servers are deployed at base stations to offer highly accessible computing and storage resources to nearby app users. From the app vendor's perspective, caching data on edge servers can ensure low latency in app users' retrieval of app data. However, an edge server normally owns limited resources due to its limited size. In this article, we investigate the *collaborative caching problem* in the EC environment with the aim to minimize the system cost including data caching cost, data migration cost, and quality-of-service (QoS) penalty. We model this *collaborative edge data caching problem* (CEDC) as a constrained optimization problem and prove that it is $\mathcal{NP}$-complete. We propose an online algorithm, called CEDC-O, to solve this CEDC problem during all time slots. CEDC-O is developed based on Lyapunov optimization, works online without requiring future information, and achieves provable close-to-optimal performance. CEDC-O is evaluated on a real-world data set, and the results demonstrate that it significantly outperforms four representative approaches.

Index Terms—Edge computing, data caching, online algorithm

1 Introduction

The world has witnessed an exponential growth of mobile devices including mobile phones, wearable devices, tablets, smart vehicle and Internet-of-Things (IoT) devices over the last decade [1]. The enormous network traffic often causes network congestion and increases network latency. To address this issue, edge computing (EC), a new computing paradigm, has emerged to distribute computing capacities from centralized cloud to distributed edge servers [2]. Each edge server is powered by one or more physical devices and is attached to a base station or an access point that is geographically close to app users' mobile devices. Mobile and IoT application vendors (referred to as *app vendor* hereafter) can host their apps on edge servers (referred to as *edge apps* hereafter) to ensure low latency and high-quality services for their app users by hiring computing and storage resources on edge servers [3]. Computation tasks can be offloaded from mobile devices to nearby edge servers to reduce the computation overhead and energy consumption on those mobile devices [4], [5], [6], [7]. This is a key technology that facilitates the 5G mobile network [8].

As a rapidly increasing number of app users begin to access edge apps, more mobile data will be transmitted through edge servers between the cloud and app users' mobile devices. From an app vendor's perspective, caching those data, especially popular ones like viral videos and posts from Facebook[1] and Twitter,[2] will significantly reduce network delay in app users' retrieval of app data. App users can retrieve data from nearby edge servers instead of from the remote cloud servers if the data are already cached on those edge servers. In addition, caching data on edge servers can also considerably reduce the amount of data transferred between the cloud and the mobile devices, which consequently lower app vendors' cost of data transfer under the pay-as-you-go pricing scheme.

Data caching techniques have been widely implemented in many different domains, from hardware cache, e.g., CPU [9], GPU [10], memory [11], disks [12], to software cache, e.g., web [13], database [14], etc. In the network domain, data caching has also been intensively studied to leverage its advantages in saving bandwidth consumption, reducing network latency and minimizing access costs [15], [16], [17]. In the last few years, many researchers have investigated network cache from different perspectives, e.g., cache allocation and replacement strategies [18], coded caching [19], request routing [20], and information-theoretic caching [21], [22]. As a new computing paradigm, EC offers new opportunities and raises new challenges for data caching. The fundamental objective and mechanism are to cache popular data on edge servers so that nearby app users can retrieve the cached data with low latency. This is especially

- *Xiaoyu Xia, Feifei Chen, and Mohamed Abdelrazek are with the School of Information Technology, Deakin University, Geelong, Victoria 3220, Australia. E-mail: {xiaoyu.xia, feifei.chen, mohamed.abdelrazek}@deakin.edu.au.*
- *Qiang He is with the School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Victoria 3122, Australia. E-mail: qhe@swin.edu.au.*
- *John Grundy is with the Faculty of Information Technology, Monash University, Melbourne, Victoria 3800, Australia. E-mail: john.grundy@monash.edu.*
- *Hai Jin is with the Services Computing Technology and System Lab, Big Data Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technolgoy, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China. E-mail: hjin@hust.edu.cn.*

Manuscript received 16 Jan. 2020; revised 17 June 2020; accepted 3 Aug. 2020. Date of publication 13 Aug. 2020; date of current version 24 Aug. 2020. (Corresponding author: Qiang He.) Recommended for acceptance by Y. Yang. Digital Object Identifier no. 10.1109/TPDS.2020.3016344

1. https://www.facebook.com/
2. https://www.twitch.tv/

important for latency-sensitive applications, e.g., interactive gaming, real-time navigation, augmented reality, etc. In addition, caching data on edge servers can also lift the traffic burden on the Internet backbone by reducing the amount of mobile traffic data transmitted between the cloud and app users' mobile devices significantly [23].

Given a set of popular data, from an app vendor's perspective, a straightforward solution is to cache them all on every edge server in a particular area to minimize the latency in its app users' data retrieval in that area. In addition to data latency, the app vendor also needs to consider the cost of hiring storage resources on edge servers for data caching based on the pay-as-you-go pricing model. The cost also occurs during data transmission and migration among the network. Thus, from an app vendor's perspective, it is critical to find a collaborative data caching strategy that minimizes the total system cost with limited storage spaces on edge servers while fulfilling the above mentioned constraints in the edge computing environment, including server capacity constraint, server coverage constraint and server adjacency constraint. Over time, a lot of data will be cached on the edge servers and new data will replace old data. An app vendor's hired storage resources on edge servers and its cached app data constitute an edge caching system. In the long-term, how to keep an app vendor's edge caching system stable over time across multiple time slots is the key problem to be solved in this paper.

We refer to this data caching problem in the EC environment *collaborative edge data caching* (CEDC) problem. To the best of our knowledge, this work is the first attempt to solve this CEDC problem from the app vendor's perspective. The key contributions of this work are as follows:

- We model and formulate the CEDC problem as a constrained optimization problem from the app vendor's perspective.
- We prove that the CEDC problem is $\mathcal{NP}$-complete.
- We propose an online algorithm named CEDC-O based on Lyapunov optimization to solve the CEDC problem across multiple time slots without requiring future information, and prove the performance bounds of this algorithm.
- We evaluate the performance of our algorithm by extensive simulations conducted on a real-world data set.

The rest of the paper is organized as follows. Section 2 presents an example to illustrate and motivate the CEDC problem. Section 3 presents the system model, formulates the CEDC problem and proves its $\mathcal{NP}$-completeness. Section 4 presents the CEDC-O algorithm and theoretically analyzes its performance bounds. Section 5 evaluates the CEDC-O algorithm experimentally. Section 6 reviews the related work, followed by the conclusion in Section 7.

2 Motivating Example

EC is significantly different from cloud computing which facilitates the content-centric network and the content delivery network. In the EC environment, adjacent edge servers deployed at different base stations can communicate with their neighbor edge servers and transmit data via high-speed

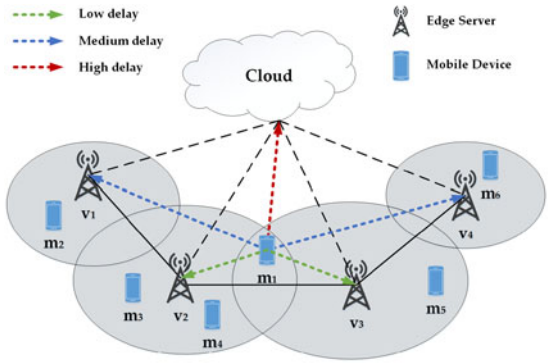


Fig. 1. An example scenario for edge computing.

links [4], [24]. App users' workloads in a particular area can be transferred and balanced across the edge servers covering that area [4]. This architecture overcomes the single-point failure problem encountered by the one with the macro base station [25]. Thus, the edge servers in a particular area can constitute a graph, namely *edge server network*, where a node represents an edge server and an edge represents the link between two edge servers.

Fig. 1 shows an example of a typical edge environment involving a set of mobile devices, $\{m_1, \ldots, m_6\}$, and edge servers, $\{v_1, \ldots, v_4\}$. Mobile devices connect to edge servers to retrieve data. Compared to cloud servers, the storage resources on an edge server are usually limited due to their limited sizes [26]. This intensifies the competition between app vendors for computing and storage resources on edge servers, making it extremely expensive and, in most cases impossible, for an app vendor to cache all its app data on every edge server. In such an environment, the common practice for an app vendor is to reserve a number of cache spaces on each edge servers for caching its most popular data. This is a fundamental difference between cloud computing and EC because the computing and storage resources available in the cloud are often assumed to be virtually unlimited. The limited resources on edge servers are referred to as the *server capacity constraint*. Furthermore, data caching in the EC environment differs from data caching in the cloud computing environment as well as other conventional distributed computing environments with its two unique constraints, i.e., *server coverage constraint* and *server adjacency constraint*:

Server coverage constraint: An edge server covers a specific geographical area so that app users' mobile devices within its coverage area can connect to it via LTE or Radio Network [27]. In a particular area, a number of edge servers are usually deployed in a distributed manner so that they can cover different geographical areas. The coverage areas of adjacent edge servers usually partially overlap to avoid blank areas not covered by any edge servers. For example, mobile device m_1 in Fig. 1 can directly access edge server v_2 and v_3 while m_2 can only access edge server v_1 directly.

Server adjacency constraint: An app user can retrieve cached app data from its nearby edge servers (referred to as *local edge servers* hereafter) if the data is cached on any of these edge servers. Otherwise, the data can be retrieved from those local edge servers' *neighbor edge servers*, i.e., edge servers that are directly linked to them via high-speed links [24]. Either way, it is faster than retrieving the data from a remote

cloud server [24]. Take m_1 in Fig. 1 as an example. This device can access the caches in its local edge servers v_2 and v_3, or its neighbor edge servers v_1 and v_4, or the remote cloud. The only difference is the data retrieval latency, which is represented by the different colors of the links in Fig. 1.

3 System Model

In this section, we first introduce the system architecture for edge data caching, then define the three components of system cost, including data caching cost, data migration cost and QoS penalty based on the constrains discussed in Section 2. The notations adopted in this section are summarized in Table 1.

3.1 System Architecture

In this research, we model the edge server network within a specific area as a graph $G(V, E)$ where $V = \{v_1, \ldots, v_n\}$ is the set of nodes and $E = \{e_1, \ldots, e_k\}$ is the set of edges in G. In this graph, each node $v \in V$ represents an edge server, while each edge $e \in E$ represents the link between two edge servers in G. *In the remainder of this paper, we will speak interchangeably of an edge server and its corresponding node in G, both denoted by v.*

To quantify the optimization objective and constraints in the CEDC problem in a generic manner, we measure the data sizes and cache spaces by the number of data units, and the data retrieval latency with the number of hops. Take Fig. 1 as an example. The cost of caching data d on all the four edge servers is 4. When data d is only cached on edge server v_3, Device 1 can retrieve the data from its local edge server v_3 via 0 hop, while Device 1 can retrieve the data from its neighbor edge server v_4 via 1 hop. This way, these models can be easily extended by integrating specific pricing models and latency models from edge infrastructure providers.

Given a set of data D required by app users in a specific area in time slot t, a data caching strategy to cover those data requests can be presented as $\lambda^t = \{\lambda_1^t, \ldots, \lambda_n^t\}$, where $\lambda_i^t = \{\lambda_{i,d}^t, \forall d \in D\}$. $\lambda_{i,d}^t$ denotes whether data d is cached on edge server v_i:

$$\lambda_{i,d}^t = \begin{cases} 0 & \text{if data } d \text{ is not cached on } v_i \text{ in time slot } t \\ 1 & \text{if data } d \text{ is cached on } v_i \text{ in time slot } t \end{cases}. \tag{1}$$

Let us denote $\tau_{m,d}^t$ as whether the request for data d from the app user's mobile device m exists in time slot t:

$$\tau_{m,d}^t = \begin{cases} 0 & \text{if the } m\text{'s request for data } d \text{ does} \\ & \text{not exist in time slot } t \\ 1 & \text{if the } m\text{'s request for data } d \text{ exists in } t \end{cases}. \tag{2}$$

Since the data requests arrive randomly in the stochastic EC environment, we model the data request arrivals as an independent and identical distribution, similar to many studies in the fields of edge computing, cloud computing and wireless networking [28], [29], [30].

TABLE 1
Notations in Our System Model

Notation	Description
A_i	available cache spaces on edge server i
$\mathcal{C}(\lambda^t)$	total system cost in time slot t
$\mathcal{C}_{\mathcal{D}}(\lambda^t)$	cost of data storage in time slot t
$\mathcal{C}_{\mathcal{M}}(\lambda^t)$	cost of data migration in time slot t
$\mathcal{C}_{\mathcal{P}}(\lambda^t)$	QoS penalty in time slot t
c_l	unit cost of data latency
c_{mc}	unit cost of data migration from cloud
c_{ms}	unit cost of data migration from edge server
c_s	unit cost of data storage
d	requested data d
D	finite set of requested data
E	finite set of links between edge servers
G	graph presenting the edge server network
$l_{i,j}$	hops between edge server v_i and v_j
$l_{i,d}^t$	lowest latency to migrate data d for v_i from the edge server network in time slot t
$l_{m,d}^t$	lowest latency of for mobile m to retrieve d from the edge server network in time slot t
$l_{\mathcal{T}}$	latency limit accepted by the app vendor
$\mathcal{L}$	long-term average latency constraint
m	mobile device m
M	set of mobile devices
M_j	set of mobile devices covered by server v_j
t	time slot t
T	infinite set of time
V	set of edge servers
v_i	edge server i
$\mathcal{X}_{i,d}^t$	binary variable indicating whether data d has been already cached on edge server v_i at the beginning of time t
$\mathcal{Y}_{i,d}^t$	binary variable indicating whether edge server i can retrieve data d from a neighbor edge server or the remote cloud
λ^t	data caching strategy in time slot t
$\lambda_{i,d}^t$	binary variable indicating whether data d will be cached on edge server v_i at the end of time t
$\tau_{m,d}^t$	binary variable indicating whether the mobile device m requests for data d in time slot t
ρ	ratio of c_{mc} over c_s
η	ratio of c_{ms} over c_s
ω	ratio of c_p over c_s

As mentioned in Section 1, the storage resources on an edge server is usually limited. Thus, the competition between app vendors usually makes it impossible for an app vendor to cache all its app data on every edge server. Thus, the number of data cached in any time slot t on each edge server v_i cannot violate the available server capacity constraint:

$$\sum_{d \in D} \lambda_{i,d}^t \leq A_i, \forall t = \{0, ..., T-1\}, i = \{1, \ldots, n\}. \tag{3}$$

3.2 Data Retrieval Latency

The data retrieval latency in the edge server network consists of two components: the latency between the device and its nearby edge server, and the latency between its local

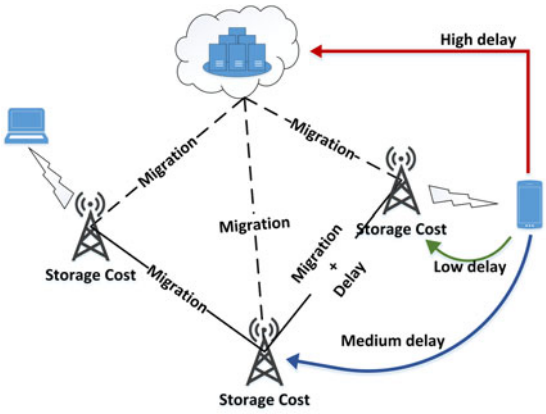


Fig. 2. System Cost: Data caching cost, migration cost and QoS penalty.

edge server and neighbor edge server. As the first component is extremely small in the 5G network and not influenced by the data caching strategy, it is not considered in the formulation of the data caching strategy. Thus the network delay in retrieving data d for the app user's mobile device m in time slot t is calculated as follows:

$$l^t_{m,d} = \min\{l_{i,j}, \lambda^t_{i,d} = 1, v_i \in V\}, \forall m \in M_j, \tag{4}$$

where v_i is the edge server caching data d and v_j is the edge server covering the app user's mobile device m, and $l_{i,j}$ is the number of hops between v_i and v_j.

We denote $l_{\mathcal{T}}$ as the latency limit specified by the app vendor. If the latency $l^t_{m,d}$ is higher than $l_{\mathcal{T}}$, mobile device m will access the data directly from the remote cloud. In this way, the delay in m's retrieval of data d from the remote cloud server c in time slot t is calculated as $l^t_{m,d} = l_{c,m}$, where $l_{c,m} > l_{\mathcal{T}}$ is the latency between c and m.

3.3 System Cost

From the app vendor's perspective, a key performance indicator for its edge caching system is the total system cost produced by the data caching strategy.

Fig. 2 depicts the key elements of the system cost model. *Data caching cost* is measured based on the storage resources hired by the app vendor in each time slot. *Data migration cost* is produced by migrating data from the cloud or the neighbor edge servers to the local edge servers. *QoS penalty* is the third component of the system cost, occurring when a user has to retrieve data from the cloud server with a high latency.

3.3.1 Data Caching Cost

It is calculated by how many cache spaces hired by the app vendor. As mentioned above, the cache spaces are measured by the number of data units. Thus, the data caching cost in time slot t can be calculated as follows:

$$\mathcal{C}_{\mathcal{D}}(\lambda^t) = \sum_{v_i \in V} \sum_{d \in D} c_s \lambda^t_{i,d}, \tag{5}$$

where c_s is the unit cost of hiring data resources on an edge server for data caching.

3.3.2 Data Migration Cost

As transferring new data from the remote cloud to an edge server or between edge servers to be cached incurs additional network delay, data migration cost is incurred and is calculated based on the number of new cached data. Here, we denote $\mathcal{X}^t_{i,d} = 0$ if data d is already cached on edge server v_i at the start of time slot t, otherwise $\mathcal{X}^t_{i,d} = 1$:

$$\mathcal{X}^t_{i,d} = 1 - \lambda^{t-1}_{i,d}. \tag{6}$$

Similar to (4), we denote $l^t_{i,d} = min\{l_{i,j}, \lambda^{t-1}_{j,d} = 1, \forall v_j \in V\}$ as the lowest latency for edge server v_i to obtain data d in time slot t, over the edge server network. We denote the unit cost of data migration from the cloud to the app user's mobile device by c_{mc}, and the unit cost of data transmission from a neighbor edge server to mobile device by c_{ms}. If the cost of data migration over the edge server network is higher than that from the remote cloud ($c_{ms} \cdot l^t_{i,d} > c_{mc}$), v_i will retrieve the data from the remote cloud directly. Here we denote the source of a requested data as $\mathcal{Y}^t_{i,d} \in \{0, 1\}$:

$$\mathcal{Y}^t_{i,d} = \begin{cases} 0 & \text{if } c_{ms} \cdot l^t_{i,d} > c_{mc} \\ 1 & \text{if } c_{ms} \cdot l^t_{i,d} \leq c_{mc} \end{cases}. \tag{7}$$

Thus, we obtain the data migration cost as follows:

$$\mathcal{C}_{\mathcal{M}}(\lambda^t) = \sum_{v_i \in V} \sum_{d \in D} \lambda^t_{i,d} \mathcal{X}^t_{i,d} ((c_{ms} \cdot l^t_{i,d} - c_{mc}) \mathcal{Y}^t_{i,d} + c_{mc}). \tag{8}$$

We denote ρ as the ratio of c_{mc} over c_s and η as the ratio of c_{ms} over c_s, then we obtain:

$$\mathcal{C}_{\mathcal{M}}(\lambda^t) = \sum_{v_i \in V} \sum_{d \in D} c_s \lambda^t_{i,d} \mathcal{X}^t_{i,d} ((\eta \cdot l^t_{i,d} - \rho) \mathcal{Y}^t_{i,d} + \rho). \tag{9}$$

3.3.3 QoS Penalty

As mentioned in Section 2, the data can be retrieved from local edge servers or neighbor edge servers. Either way, it is faster than retrieving the data from a remote cloud server [24]. Thus, the quality-of-service (QoS) is impacted significantly for users who cannot retrieve data from any available edge servers within $l_{\mathcal{T}}$. Thus, the QoS penalty occurs when a user has to retrieve data from the remote cloud server or from an edge server with a limit-violating latency. Here, we denote $\theta^t_{m,d} \in \{0, 1\}$ to indicate whether a QoS penalty is applied to m's retrieval of data d:

$$\theta^t_{m,d} = \begin{cases} 1 & \text{if } l^t_{m,d} > l_{\mathcal{T}} \\ 0 & \text{if } l^t_{m,d} \leq l_{\mathcal{T}} \end{cases}. \tag{10}$$

Please notice that $l_{c,m} > l_{\mathcal{T}}, \forall m \in M$. This way, $\theta^t_{m,d}$ is always 1.

We denote c_p as the unit cost incurred by QoS penalty, determined by the app vendor based on its priority for its app users' QoS. The QoS penalty in time slot t, as part of system cost, is calculated as:

$$\mathcal{C}_{\mathcal{P}}(\lambda^t) = \sum_{m \in M} \sum_{d \in D} c_p \theta^t_{m,d} = \sum_{m \in M} \sum_{d \in D} \omega \cdot c_s \theta^t_{m,d}, \tag{11}$$

where ω is the ratio of c_p over c_s.

3.4 Problem Formulation and Hardness

With the consideration of the system architecture in Section 3.1 and the costs presented in Section 3.3, the total system cost is calculated by summing all the aforementioned costs:

$$\begin{aligned}\mathcal{C}(\lambda^t) &= \mathcal{C}_{\mathcal{D}}(\lambda^t) + \mathcal{C}_{\mathcal{M}}(\lambda^t) + \mathcal{C}_{\mathcal{P}}(\lambda^t) \\ &= c_s\Big(\sum_{v_i\in V}\sum_{d\in D}\lambda^t_{i,d}\big(1+\mathcal{X}^t_{i,d}((\eta-\rho)\mathcal{Y}^t_{i,d}+\rho)\big)\Big) \\ &\quad + \sum_{m\in M}\sum_{d\in D}\omega\cdot c_s\theta^t_{m,d}.\end{aligned} \tag{12}$$

In a CEDC scenario, from the app vendor's perspective, it is important to minimize the system cost incurred by caching data on edge servers, migrating data across edge servers and failing to serve users on edge servers. While pursing this optimization objective, it is also necessary to stabilize the time-averaged system latency perceived by the users in the long term. Thus, an app vendor usually has a long-term average system latency constraint for requests served by edge servers, denoted by $\mathcal{L}$. Thus, the following inequality must be fulfilled:

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\frac{\sum_{m\in M}\sum_{d\in D}\theta^t_{m,d}\cdot l^t_{m,d}}{\sum_{m\in M}\sum_{d\in D}\theta^t_{m,d}} \le \mathcal{L}. \tag{13}$$

In the stochastic EC environment, data requests randomly arrive and leave [28]. Thus, the app vendor's long-term system performance usually outweighs its immediate short-term system performance. Thus, we formulate the CEDC problem over multiple time slots as a constrained optimization problem (COP):

$$\begin{aligned}\mathcal{P}_1 :& \min \lim_{T\to\infty}\sum_{t=0}^{T-1}\mathcal{C}(\lambda^t) \\ s.t. :& (1),(3),(4),(6),(13).\end{aligned}$$

Now, we demonstrate that the COP of CEDC problem in a single time slot t is $\mathcal{NP}$-complete by proving the following theorems.

Theorem 1. *The COP of CEDC in time slot t is $\mathcal{NP}$.*

Proof. As there are $(1+|M|+2|V|)|D|$ constraints in total, any solution to the COP can be validated in polynomial time by checking whether the solution satisfies the constraints (1), (3), (6) and (4). Thus, the CEDC problem is $\mathcal{NP}$.□

Theorem 2. *The COP of CEDC in time slot t is $\mathcal{NP}$-complete.*

Proof. To prove this problem is $\mathcal{NP}$-complete, we first introduce the weighted k-set packing problem. The weighted k-set packing problem is known to be $\mathcal{NP}$-complete [31]. Given a universe U with elements $\forall e \in U$, a set U' of subsets of U and an integer number k. The subset X is a packing, where $X \subseteq U'$. All sets $x \subseteq X$ are pairwise disjoint. Let $\mathcal{W}(x)$ be the weight of the set x and k be the maximum number of selected sets. The formulation is displayed below:

$$object : \max\sum_{x\in X}\mathcal{W}(x)\mathcal{T}_x \tag{14a}$$

$$s.t. : \sum_{x\in X}\mathcal{T}_x \le k \tag{14b}$$

$$\mathcal{T}_x \in \{0,1\}, \forall x \in X \tag{14c}$$

$$\sum_{e\in U}\mathcal{T}_e \le 1. \tag{14d}$$

Next, we prove that the weighted k-set packing problem can be reduced to an instance of the COP of CEDC problem. We define the elements based on the data requests from app user's mobile devices. The reduction can be done as follows: 1) adding the cloud server v_{cloud} as a node into the graph; 2) adding edges from v_{cloud} to all other nodes in the graph; 3) setting the storage cost incurred on v_{cloud} to 0; 4) setting $l_{\mathcal{T}} = \mathcal{L} = |V|$. After the above reduction, constraint (13) can be ignored. Given any instance $WeightedKSet(X, U, k, \mathcal{W}(x))$, we can construct $CEDC(V, M, n, Benefits(i, d))$ with the reduction above in polynomial time while $|X| = |V|$, $|U| = |M|$ and $n = k$. The function $Benefits(i, d)$ is calculated as the reduced QoS penalty minus the data caching cost if data d is cached on edge server v_i. As the constraint (14b) restricts the total number of selected sets, we can project (3) in the CEDC problem to that constraint. Based on (4), the latency in mobile device m's retrieval of data d is the lowest latency between m and any edge server with d in the cache. Thus, constraint (14d) can be fulfilled. Moreover, we can convert our original objective (12) to $\mathcal{C}_{\mathcal{D}}(\lambda^t) + \mathcal{C}_{\mathcal{L}}(v_{cloud}) - \sum_{v_i\in V}\sum_{d\in D}\lambda^t_{i,d}Benefits(i, d)$. Then, we can project our converted objective to the objective (14a). Thus, the COP of CEDC problem in time slot t is reducible from the weighted k-set packing problem, and it is $\mathcal{NP}$-complete. □

4 Online Caching Algorithm Design

To solve the CEDC problem optimally, the complete information about the system over all the time slots must be known. However, this cannot be realistically fulfilled for real-world scenarios. To practically fulfil the app vendor's long-term latency constraint (13), we need to convert $\mathcal{P}_1$, a non-convex problem, to a linear and convex problem. To do so, we propose an Online Collaborative Edge Data Caching (CEDC-O) algorithm based on Lyapunov optimization [32] for finding near-optimal solutions to the CEDC problem in individual time slots without future information. The notations adopted in this section are summarized in Table 2.

4.1 Online Collaborative Edge Data Caching Algorithm

We provide an online algorithm, named CEDC-O, based on Lyapunov optimization, to convert the long-term optimization problem $\mathcal{P}_2$ to optimization problems in individual time slots. The most significant characteristic of CEDC-O is that it only requires the information in the current time slot rather than the complete information in all the time slots

TABLE 2
Notations in Our Algorithm Design

Notation	Description
$\mathcal{C}$	value of system cost produced by λ
$\mathcal{C}'$	value of system cost produced by λ'
$\mathcal{C}_{opt}$	value of system cost produced by λ_{opt}
$\mathcal{C}_{min}$	smallest system cost of all possible solutions
$\mathcal{C}_{max}$	largest system cost of all possible solutions
$\mathcal{DP}(t)$	Lyapunov drift-plus-penalty function
$\mathcal{L}_{avg}(\lambda^t)$	average system latency in time slot t
$\mathcal{L}(\sigma(t))$	Lyapunov function, calculated by $\frac{1}{2}\sigma^2(t)$
$\mathcal{Q}$	constant value equal to $\frac{1}{2}\mathcal{L}^2$
$\mathcal{Q}'$	constant value equal to $Q + \gamma \cdot (\mathcal{C}_{max} - \mathcal{C}_{min})$
γ	positive parameter adjusting the trade-off between system cost $\mathcal{C}(\lambda^t)$ and the average latency $\mathcal{L}_{avg}(\lambda^t)$
λ	solution obtained by CEDC-O
λ^t	λ in time slot t
λ'	feasible solution fulfilling (21)
λ'^t	λ' in time slot t
λ^*	feasible solution fulfilling (27)
λ^{*t}	λ^* in time slot t
λ_{opt}	optimal solution to $\mathcal{P}_1$ over all time slots
λ^t_{opt}	λ_{opt} in time slot t
$\sigma(t)$	accumulated latency in time slot t
$\Delta(\sigma(t))$	Lyapunov drift function

when solving $\mathcal{P}_2$. While trying to minimize the system cost, the app vendor also needs to stabilize the system latency to ensure low-latency data access for its users. Thus, in this paper, the system metric to stabilize by CEDC-O is the time-averaged system latency perceived by the users over the long term.

Lyapunov optimization is typically applied in the communication and queuing systems. With the application of Lyapunov optimization, the problems can be formulated as problems that optimize the time averages of certain objectives subject to some time average constraints, and they can be solved with a common mathematical framework that is intimately connected to queuing theory [32]. Unlike the typical application of Lyapunov optimization that models the problem as a queuing network, we define the *accumulated latency* in Definition 1 to stabilize the system latency over time.

Definition 1 (Accumulated Latency). *Accumulated latency σ_t is the overdue delay accumulated over t time slots, calculated as:*

$$\sigma(t+1) = \max\{\sigma(t) + \mathcal{L}_{avg}(\lambda^t) - \mathcal{L}, 0\} \tag{15}$$

where $\mathcal{L}_{avg}(\lambda^t) = \frac{\sum_{m\in M}\sum_{d\in D}\theta^t_{m,d}\cdot l^t_{m,d}}{\sum_{m\in M}\sum_{d\in D}\theta^t_{m,d}}$, *and* $\sigma(0) = 0$ *because there is no latency at the very beginning.*

Based on Definition 1, the accumulated latency will increase if the latency is over $\mathcal{L}$ in the previous time slot. This can be employed as a penalty to adjust the data caching strategy to stabilize the system latency over time as specified by (13). Now, we can convert the long-term latency constraint (13) to a new constraint based on accumulated latency:

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}[\sigma(t)] \le 0. \tag{16}$$

Given (15), a Lyapunov function can be defined as $\mathcal{L}(\sigma(t)) \triangleq \frac{1}{2}\sigma^2(t)$. It indicates the system stability measured by its accumulated latency $L(\sigma(t))$. Here, the Lyapunov drift $\Delta(\sigma(t))$ is applied in each time slot to enhance the system stability:

$$\begin{aligned}
&\Delta(\sigma(t)) = \mathbb{E}[\mathcal{L}(t+1) - \mathcal{L}(t)|\sigma(t)] \\
&= \frac{1}{2}\mathbb{E}[\sigma^2(t+1) - \sigma^2(t)|\sigma(t)] \\
&= \frac{1}{2}\mathbb{E}[(\mathcal{L}_{avg}(\lambda^t) - \mathcal{L})^2|\sigma(t)] + \sigma(t)\mathbb{E}[\mathcal{L}_{avg}(\lambda^t) - \mathcal{L}|\sigma(t)] \\
&\le \mathcal{Q} + \sigma(t)\mathbb{E}[\mathcal{L}_{avg}(\lambda^t) - \mathcal{L}|\sigma(t)],
\end{aligned} \tag{17}$$

where $\mathcal{Q} = \frac{1}{2}\mathcal{L}^2$ because of $\mathcal{L}_{avg}(\lambda^t) \ge 0$.

As we obtain the upper bound of the Lyapunov drift function, we introduce the penalty in our CEDC-O algorithm based on the total cost objective (12). We denote γ as a positive parameter in Lyapunov optimization for adjusting the trade-off between the system cost $\mathcal{C}(\lambda^t)$ and the number of time slots needed to converge the time-averaged latency back to $\mathcal{L}$ when (13) is violated. Here, we introduce the Lyapunov *drift-plus-penalty* function $\mathcal{DP}(t)$, defined as:

$$\mathcal{DP}(t) = \Delta(\sigma(t)) + \gamma \cdot \mathbb{E}[\mathcal{C}(\lambda^t)|\sigma(t)]. \tag{18}$$

In each time slot, the data caching strategy is formulated to minimize the total cost $\mathcal{C}(\lambda^t)$ and to keep the system stable, and we can get the upper bound of this function by:

$$\mathcal{DP}(t) \le \mathcal{Q} + \sigma(t)\mathbb{E}[\mathcal{L}_{avg}(\lambda^t) - \mathcal{L}|\sigma(t)] + \gamma \cdot \mathbb{E}[\mathcal{C}(\lambda^t)|\sigma(t)]. \tag{19}$$

The pseudocode of the CEDC-O algorithm is presented in Algorithm 1. In each time slot, the data caching strategy is formulated by finding the optimal solution to $\mathcal{P}_2$:

$$\begin{aligned}
&\mathcal{P}_2 : \min(\mathcal{Q} + \sigma(t)(\mathcal{L}_{avg}(\lambda^t) - \mathcal{L}) + \gamma \cdot \mathcal{C}(\lambda^t)) \\
&s.t. : (1), (3), (6), (4), (15).
\end{aligned}$$

Algorithm 1. CEDC-O Algorithm

1: **Input:** $G = (V, E), M, A = \{A_1, \ldots, A_n\}, \mathcal{L}, c_s, \rho, \eta, \omega$
2: **Output:** data caching strategy $\lambda = \{\lambda^1, \ldots, \lambda^T\}$
3: $\sigma(0) = 0, t = 0$
4: **repeat**
5: Observe the data requests $\tau^t = \{\tau^t_{m,d} | \forall m \in M, d \in D\}$
6: Find the solution λ^t, where:

$$\lambda^t = \arg\min(\mathcal{Q} + \sigma(t)(\mathcal{L}_{avg}(\lambda^t) - \mathcal{L}) + \gamma \cdot \mathcal{C}(\lambda^t)) \tag{20}$$

7: Observe $\mathcal{L}_{avg}(\lambda^t)$ and update $\sigma(t)$ based on (15)
8: $t = t + 1$
9: **until** $t = T$

In Algorithm 1, no further information is required to solve $\mathcal{P}_2$ except the data requests in the current time slot. After implementing the *drift-plus-penalty* function, our CEDC-O algorithm considers the additional term $\sigma(t)$ $(\mathcal{L}_{avg}(\lambda^t) - \mathcal{L})$, while $\mathcal{Q}$ is a constant. This drift-plus-penalty helps stabilize the system's average latency $\mathcal{L}_{avg}(\lambda^t)$ around

$\mathcal{L}$. Once $\mathcal{L}_{avg}(\lambda^t)$ exceeds $\mathcal{L}$, a penalty is applied to $\mathcal{P}_2$ and drives the CEDC-O algorithm to lower the system latency. Moreover, when $\sigma(t)$ increases, minimizing $\mathcal{L}_{avg}(\lambda^t)$ is of high significance in stabilizing the system and converging $\mathcal{L}_{avg}(\lambda^t)$ to the long-term budget (16). This is validated experimentally in Section 5.

4.2 Performance Analysis

Now, we analyze the performance of the CEDC-O algorithm based on the following theorems.

Theorem 3. *The time-averaged system cost of CEDC-O algorithm is bounded by $O(\frac{1}{\gamma})$.*

Proof. Let us assume that the optimal solution to $\mathcal{P}_1$ is $\lambda_{opt} = \{\lambda^0_{opt}, \ldots, \lambda^{T-1}_{opt}\}$. The average system cost of λ_{opt} is $\mathcal{C}_{opt} = \frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}[\mathcal{C}(\lambda^t_{opt})|\sigma(t)]$. The following inequality has been proven in [32]:

$$\exists \lambda'^t, \mathbb{E}[\mathcal{L}_{avg}(\lambda'^t) - \mathcal{L}|\sigma(t)] \leq \theta, \theta \to 0^+. \tag{21}$$

The CEDC-O algorithm provides the solution that minimizes $\mathcal{P}_2$ from all feasible decisions including λ' which contains λ'^t. Thus, we can obtain:

$$E[\mathcal{C}(\lambda'^t)|\sigma(t)] \leq E[\mathcal{C}(\lambda^t_{opt})|\sigma(t)]. \tag{22}$$

After applying (21) to (19), we can obtain:

$$\begin{aligned}
&\Delta(\sigma(t)) + \gamma \cdot \mathbb{E}[\mathcal{C}(\lambda'^t)|\sigma(t)] \\
&\overset{(19)}{\leq} \sigma(t)\mathbb{E}[(\mathcal{L}_{avg}(\lambda'^t) - \mathcal{L})|\sigma(t)] + \gamma \cdot \mathbb{E}[\mathcal{C}(\lambda'^t)|\sigma(t)] + \mathcal{Q} \\
&\overset{(22)}{\leq} \sigma(t)\mathbb{E}[(\mathcal{L}_{avg}(\lambda'^t) - \mathcal{L})|\sigma(t)] + \gamma \cdot \mathbb{E}[\mathcal{C}(\lambda^t_{opt})|\sigma(t)] + \mathcal{Q} \\
&\overset{(21)}{\leq} \theta \cdot \sigma(t) + \gamma \cdot \mathbb{E}[\mathcal{C}(\lambda^t_{opt})|\sigma(t)] + \mathcal{Q} \\
&= \gamma \cdot \mathbb{E}[\mathcal{C}(\lambda^t_{opt})|\sigma(t)] + \mathcal{Q}.
\end{aligned} \tag{23}$$

Based on (17) and (23), we sum all the $\Delta(\sigma(t))$ for all time slots and get:

$$\begin{aligned}
&\mathbb{E}[L(\sigma(T)) - L(\sigma(0))] + \gamma \cdot \sum_{t=0}^{T-1}\mathbb{E}[\mathcal{C}(\lambda'^t)|\sigma(t)] \\
&\leq \gamma \cdot \sum_{t=0}^{T-1}\mathbb{E}[\mathcal{C}(\lambda^t_{opt})|\sigma(t)] + T \cdot \mathcal{Q} = \gamma \cdot T \cdot \mathcal{C}_{opt} + T \cdot \mathcal{Q}.
\end{aligned} \tag{24}$$

Denote $\mathcal{C}'$ as the value of system cost incurred by data caching strategy λ'. Considering the facts that $L(\sigma(T)) \geq 0$ and $L(\sigma(0)) = 0$, we can obtain:

$$\begin{aligned}
\mathcal{C}' &\leq \frac{1}{T}(\mathbb{E}[L(\sigma(T)) - L(\sigma(0))] + \lim_{T\to\infty}\sum_{t=0}^{T-1}\mathbb{E}[\mathcal{C}(\lambda'^t)|\sigma(t)]) \\
&\leq \mathcal{C}_{opt} + \frac{\mathcal{Q}}{\gamma}.
\end{aligned} \tag{25}$$

Based on (20), the performance $\mathcal{C}$ of solution λ provided by our CEDC-O algorithm always outperforms that of λ'. The distance of the time-averaged system cost between λ and λ_{opt} is:

$$\mathcal{C} - \mathcal{C}_{opt} \leq \mathcal{C}' - \mathcal{C}_{opt} \leq \frac{\mathcal{Q}}{\gamma}. \tag{26}$$

Thus, the time-average system cost of our CEDC-O algorithm is bounded by $O(\frac{\mathcal{Q}}{\gamma}) = O(\frac{1}{\gamma})$. □

Theorem 4. *By applying the CEDC-O algorithm, the time-average accumulated latency is bounded by $O(\gamma)$.*

Proof. Based on (15) and (16), we assume that there exist λ^{*t} and a positive value δ to fulfill:

$$\mathbb{E}[\mathcal{L}_{avg}(\lambda^{*t}) - \mathcal{L}|\sigma(t)] \leq -\delta. \tag{27}$$

Denote $\mathcal{C}_{min}$ and $\mathcal{C}_{max}$ as the smallest and largest system cost respectively. From (19), we obtain:

$$\begin{aligned}
\Delta(\sigma(t)) + \gamma \cdot \mathcal{C}_{min} \leq \mathcal{Q} + \gamma \cdot \mathcal{C}_{max} \\
+ \sigma(t)\mathbb{E}[\mathcal{L}_{avg}(\lambda^{*t}) - \mathcal{L}|\sigma(t)]
\end{aligned}. \tag{28}$$

Define $Q' = Q + \gamma \cdot (\mathcal{C}_{max} - \mathcal{C}_{min})$. We have the following:

$$\begin{aligned}
\Delta(\sigma(t)) &\leq \mathcal{Q}' + \sigma(t)\mathbb{E}[\mathcal{L}_{avg}(\lambda^{*t}) - \mathcal{L}|\sigma(t)] \\
&\overset{(27)}{\leq} \mathcal{Q}' - \delta \cdot \sigma(t).
\end{aligned} \tag{29}$$

By adding expectation to both sides of (29), we obtain:

$$\mathbb{E}[\Delta(\sigma(t))] = \mathbb{E}[L(\sigma(t)) - L(\sigma(t-1))] \leq \mathcal{Q}' - \delta \cdot \mathbb{E}[\sigma(t)]. \tag{30}$$

The time-average accumulated latency can be obtained by the sum of (30) of each time slot divided by the number of total time slots T:

$$\frac{1}{T}\lim_{T\to\infty}\sum_{t=0}^{T-1}\mathbb{E}[\sigma(t)] \leq \frac{\mathcal{Q}' - \frac{1}{T}\mathbb{E}[L(\sigma(T))]}{\delta} \leq \frac{\mathcal{Q}'}{\delta}. \tag{31}$$

Considering the fact that $O(\frac{\mathcal{Q}'}{\delta}) = O(\gamma)$, the time-average accumulated latency of CEDC-O algorithm is bounded by $O(\gamma)$. □

Based on Theorem 3, our CEDC-O algorithm finds the optimal solution to problem $\mathcal{P}_2$ when $\gamma \to \infty$. However, with the increase in γ, the accumulated latency increases. The CEDC-O algorithm then needs more time slots to converge the time-averaged system latency so that the constraint (13) can be fulfilled. The performance analysis of the CEDC-O algorithm is also experimentally validated in Section 5.

5 Simulation

We experimentally evaluate the performance of CEDC-O and the impacts of different parameters on its performance. All simulations were conducted on a Windows-10 machine.

TABLE 3
Parameter Settings

	n	$\|D\|$	MS	ω	$l_{\mathcal{T}}$	$\mathcal{L}$	γ
Set #1	8	4	5	0.25	2	2	1
Set #2	5,6,7,8,9,10	4	5	0.25	2	0.8	1
Set #3	8	2,3,4,5,6	5	0.25	2	0.8	1
Set #4	8	4	2,3,4,5,6	0.25	2	0.8	1
Set #5	8	4	5	0.15,0.20,0.25,0.30,0.35	2	0.8	1
Set #6	8	4	5	0.25	0,1,2,3,4	0.8	1
Set #7	8	4	5	0.25	2	0.4,0.8,1.2,1.6,2	1
Set #8	8	4	5	0.25	2	0.8	0.5,1.0,1.5

5.1 Settings

5.1.1 Data Set

The simulations are conducted based on the widely-used real-world EUA data set [2]. This data set contains the geographical locations of 125 cellular base stations and 816 mobile users around those base stations in the Melbourne central business district area. In all sets of simulations, a certain number of edge servers are randomly selected from the data set. In each time slot, the total number of app users' data requests is randomly generated following a normal distribution $X \sim \mathcal{N}(\mu', \sigma'^2)$, where μ' is $\frac{|M|}{2}$ and σ' is $\frac{|M|}{4}$, to cover 99.99 percent possibility. All the data requests are independently and identically distributed. The links between edge servers are randomly generated but we ensure that the edge servers constitute a connected graph.

To reflect the advantage of EC over cloud computing in terms of latency, the latency between the cloud and the app user's mobile devices in this graph area is 20 hops. This number is area-specific and does not impact the experimental results significantly as long as it is adequately large. To run the simulations realistically, we adopt AWS's Snowball Edge Pricing,[3] and set c_{mc} to $\$0.016$, c_{ms} to $\$0.008$ and c_s to $\$0.04$ caching per piece of data. The available storage space of each edge server is randomly generated separately following a normal distribution $X \sim \mathcal{N}(\mu, \sigma^2)$, where μ is the half number of maximum cache spaces and σ is 1, to build the standard normal distribution covering all the possibilities.

5.1.2 Benchmark Approaches

We compare our new CEDC-O algorithm with four representative approaches: *Delay-Oriented*, *Online-Optimal*, *Revenue-Oriented* and *Coverage-Oriented*:

- *Delay-Oriented data caching approach (DO)*: this approach always finds the optimal solution to minimize the total data latency of all users. The data caching strategy is found by this approach. Since This approach originated from [28].
- *Online-Optimal data caching approach (OO)*: this approach finds the optimal solution to the CEDC problem based on $\mathcal{P}_1$ in each individual time slot. Since (13) is a long-term latency constraint, we use (32) as a constraint to drive OO in individual time slots:

$$\frac{\sum_{m\in M}\sum_{d\in D}\theta^t_{m,d}\cdot l^t_{m,d}}{\sum_{m\in M}\sum_{d\in D}\theta^t_{m,d}} \leq \mathcal{L}. \tag{32}$$

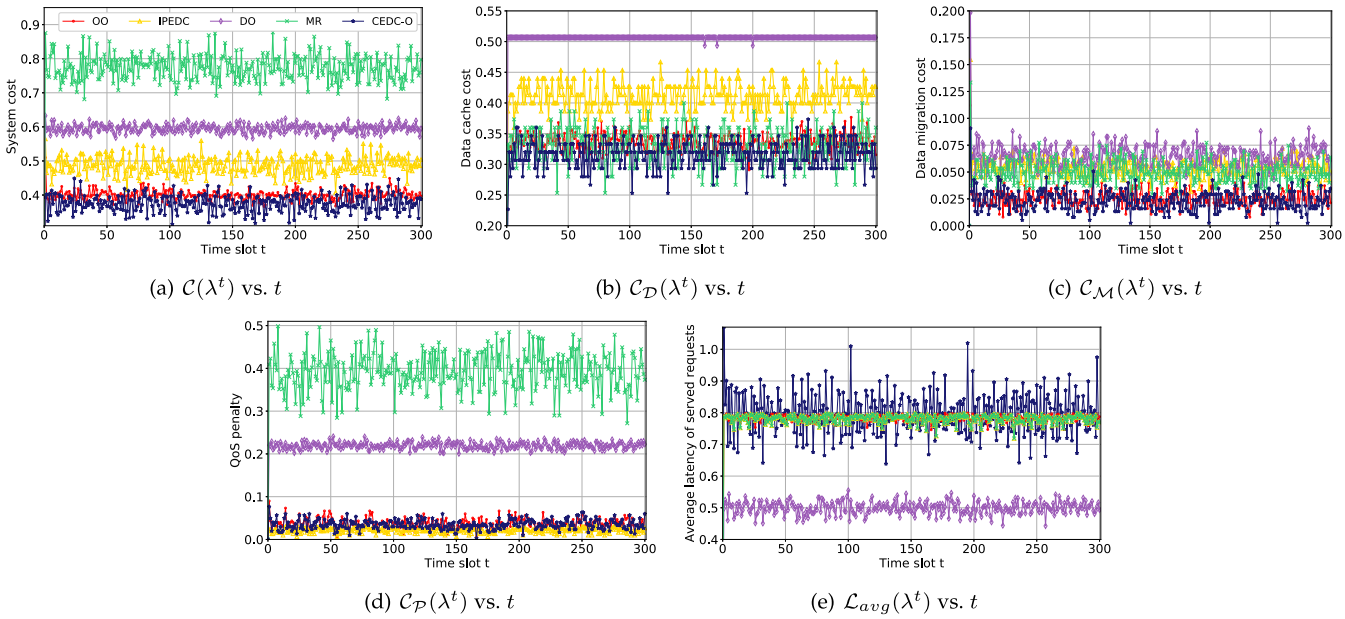
- *IPEDC* [33]: this approach minimizes the data caching cost to cover nearby users. Since some of the users may not be covered in our scenarios, we modify the original IPEDC approach to cover as many users as possible without violating the latency constraint (32).
- *Maximum Revenue data caching approach (MR)* [34]: This approach calculates the data caching revenue the benefits minus the costs produced by the data caching strategy in the EC environment. It always finds the optimal solution with the maximum data caching revenue. To perform a fair comparison in CEDC scenarios, latency constraint (32) is included into MR.

5.1.3 Parameter Settings

to analyze the performance of our CEDC-O comprehensively, we conducted seven sets of simulations to observe its performance in different CEDC scenarios. In each set of simulations except Set #1, we change one setting parameter and fix the other six. The simulation settings are summarized in Table 3. This way, we can compare the performance of the four approaches and observe how the changes in the setting parameters impact the performance of CEDC-O. The total number of time slots is 300 in all the simulations. Each time a setting parameter varies as follows, the simulation is repeated 20 times and the results are averaged:

- Number of edge servers (n). This parameter impacts the size of graph G and varies from 5 to 10 in steps of 1.
- Number of data ($|D|$). The total number of data to be cached over G, varies from 2 to 6 in steps of 1.
- Number of maximum cache spaces (MS). This parameter impacts the available cache spaces on edge server and varies from 2 to 6 in steps of 1.
- Ratio of c_p over c_s (ω) in (11). This parameter indicates the app vendor's priority for QoS and increases from 0.15, 0.20, 0.25, 0.30 to 0.35.
- Latency limit ($l_{\mathcal{T}}$). This parameter varies from 0 to 4 in steps of 1. Specifically, $l_{\mathcal{T}} = 0$ means that edge servers cannot communicate with each other - users can only access data from their local edge servers.

3. https://aws.amazon.com/snowball-edge/pricing/



(a) $\mathcal{C}(\lambda^t)$ vs. t (b) $\mathcal{C}_{\mathcal{D}}(\lambda^t)$ vs. t (c) $\mathcal{C}_{\mathcal{M}}(\lambda^t)$ vs. t

(d) $\mathcal{C}_{\mathcal{P}}(\lambda^t)$ vs. t (e) $\mathcal{L}_{avg}(\lambda^t)$ vs. t

Fig. 3. Simulation Set #1.

- Long-term latency ($\mathcal{L}$) in (15). This parameter varies from 0.4 to 2 in steps of 0.4.
- Trade-off parameter (γ) between $\mathcal{C}(\lambda^t)$ and $\sigma(t)$ in $\mathcal{P}_2$. As discussed in Section 4.2, this parameter impacts the performance bound of CEDC-O and increases from 0.5, 1.0 to 1.5.

5.1.4 Performance Metrics

In these simulations, five performance metrics are employed to evaluate all approaches:

- System cost ($\mathcal{C}(\lambda^t)$), the lower the better.
- Data caching cost ($\mathcal{C}_{\mathcal{D}}(\lambda^t)$), the lower the better.
- Data migration cost ($\mathcal{C}_{\mathcal{M}}(\lambda^t)$), the lower the better.
- QoS penalty ($\mathcal{C}_{\mathcal{P}}(\lambda^t)$), the lower the better.
- Number of served requests (Srn), the higher the better.

5.2 Performance Comparison

Fig. 3 presents the results of Set #1. Overall, of all the four approaches, *CEDC-O achieves the lowest system cost*. Fig. 3a depicts that, in term of the system cost in each time slot, the advantages of CEDC-O are 8.60 percent over OO, 23.05 percent over IPEDC, 37.04 percent over DO and 51.58 percent over MR. The advantage of CEDC-O over OO is not as significant because OO finds the optimal solution to the CEDC problem in each individual time slot. However, in 199 out of the 300 time slots in the experiments, CEDC-O achieves a system cost lower than than achieved by OO. This shows the overall advantage of CEDC-O over OO over time.

In Fig. 3b, the average data caching cost of CEDC-O is again the lowest at 0.3166, while the average data cache costs of OO, IPEDC, DO and MR are 0.3373, 0.4128, 0.5065 and 0.3314, respectively. Interestingly, the performance of DO in this figure is almost a horizontal line. DO always tries to achieve the lowest latency without considering the used cache space. Thus, it exhausts all the available cache spaces in most of the time slots.

Fig. 3c demonstrates the data migration costs of the five approaches over individual time slots. The migration costs of all the approaches are very high at the beginning, i.e., 0.0907 for CEDC-O, 0.1120 for OO, 0.1547 for IPEDC, 0.2204 for DO and 0.1333 for MR. This is because all the cached data are migrated from the cloud in time slot 0. After the data are cached on edge servers, the migration cost decreases because required data are already cached on edge servers.

Fig. 3d shows that IPEDC achieves the lowest QoS penalty, closely followed by CEDC-O and OO. The average QoS penalties of all the approaches are 0.035 for CEDC-O, 0.0373 for OO, 0.0205 for IPEDC, 0.2193 for DO and 0.6548 for MR. IPEDC focuses on covering the maximum number of users with available cache spaces. Thus, it achieves the lowest QoS penalty.

In terms of the average system latency of served requests over the edge server network, all the approaches fulfill constraint (13) as shown in Fig. 3e. The average latency is 0.7975 for CEDC-O, 0.7842 for OO, 0.7804 for IPEDC 0.4999 for DO and 0.7744 for MR. Fig. 3e also shows that the performances of OO, IPEDC and MR fluctuate slightly around 0.8. The reason is that the time-averaged latency achieved by these approaches are limited by (32).

5.3 Impact of Edge Server Number

Fig. 4 demonstrates the results of simulation Set #2, where the number of edge servers varies. Again, *CEDC-O outperforms the other four approaches* in terms of system cost per time slot, by 7.36 percent against OO, 23.14 percent against IPEDC, 46.43 percent against DO and 56.45 percent against MR. Since the number of users is determined by the number of edge servers selected from the EUA dataset, the number of users and the number of data requests increase accordingly when the number of edge servers increases. Thus, the

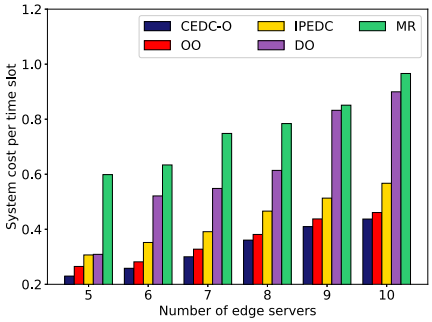


Fig. 4. Simulation Set #2.

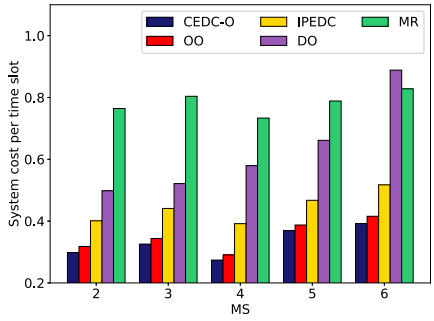


Fig. 6. Simulation Set #4.

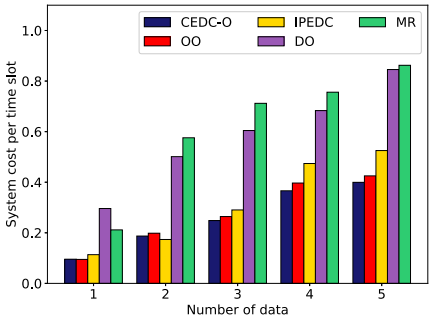


Fig. 5. Simulation Set #3.

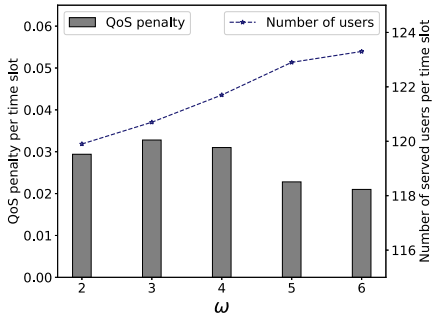


Fig. 7. Simulation Set #5.

system costs of all the approaches increases when the number of edge servers increases, as shown in Fig. 4.

5.4 Impact of Data Number

Fig. 5 depicts the results of simulation Set #3. When the number of data varies, *CEDC-O again achieves the lowest average system cost* per time slot. When the number of data increases from 1 to 5, the system costs per time slot achieved by all the five approaches increase, from 0.0956 to 0.3996 for CEDC-O, from 0.0948 to 0.4252 for OO, from 0.1136 to 0.5252 for IPEDC, from 0.2960 to 0.8456 for DO and from 0.2116 to 0.8624 for MR. With the increase in the number of data, app users are more likely to request different data from across multiple time slots. Accordingly, the average system costs achieved by all the five approaches increase.

5.5 Impact of Maximum Cache Spaces

In simulation Set #4, *CEDC-O achieves the lowest system cost per time slot* at *the lowest data cache cost per time slot*. The advantages of CEDC-O in the system cost per time slot are 5.50 percent over OO, 25.23 percent over IPEDC, 47.32 percent over DO and 57.66 percent over MR. The costs spent on data caching per time slot increase from 0.29 to 0.42 for CEDC-O, from 0.40 to 0.85 for DO and from 0.30 to 0.48 for OG, when the number of maximum cache spaces increases from 2 to 6. Different from other approaches, the system cost per time slot of DO always increases when the number of maximum cache spaces increases from 2 to 6. The reason is that DO focuses on latency optimization instead of cost optimization, and thus always exhausts the available cache spaces.

5.6 Impact of QoS Priority

Fig. 7 shows the impact of QoS priority on the performance of CEDC-O in terms of the QoS penalty and the number of served users over the edge server network. The average number of users served over the edge server network per time slot increases from 119.9 to 123.3, where ω increases from 2 to 6. The reason is that the more data cache spaces are hired to serve more users to reduce the QoS penalty. However, the QoS penalty increases from 0.0294 to 0.0328 when ω increases from 2 to 3, then decreases to 0.0210 when ω increases from 3 to 6. This is because the increase in the QoS penalty caused by the increasing ω is more than the reduction caused by hiring more cache spaces when pursuing the objective of the minimum system cost.

5.7 Impact of Latency Limit

Fig. 8 shows the QoS penalty and the number of served users over the edge server network per time slot when the latency limit l_T varies. When l_T increases from 0 to 4, the QoS penalty rapidly decreases from 0.6810 to 0, while the average number of served users increases from 72.8 to 130.9. This is because most of the users can access more edge servers when the latency limit increases. Specifically, there are 8 edge servers in Set #6, and many users can access all those edge servers within 3 hops. Thus, all the requests can be fulfilled and the QoS penalty is 0.

5.8 Impact of long-Term Latency

Fig. 9 illustrates the results of Set #7 when the long-term latency constraint varies. The system cost per time slot of CEDC-O decreases from 0.5848 to 0.2772 when $\mathcal{L}$ increases from 0.4 to 1.6. It stabilizes when $\mathcal{L}$ increases from 1,6 to 2.0. The main reason is that the same data replica can be transmitted to more users when $\mathcal{L}$ increases. In terms of the average latency of served requests $\mathcal{L}_{avg}(\lambda^t)$, it stabilizes for the same reason when $\mathcal{L}$ increases from 1,6 to 2.0.

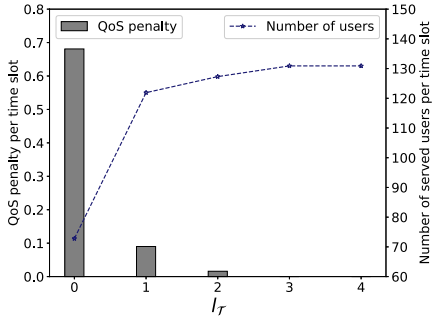


Fig. 8. Simulation Set #6.

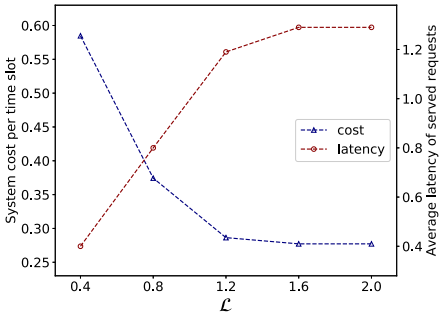


Fig. 9. Simulation Set #7.

5.9 Impact of Trade-Off Parameter

Fig. 10 shows the impact of the trade-off parameter γ in $\mathcal{P}_2$ on the time-averaged latency, which is the left side of (13). As discussed in Section 4.2, CEDC-O needs more time slots to satisfy constraint (13) when γ increases. With $\gamma = 0.5$, $\gamma = 1.0$ and $\gamma = 1.5$, the time-averaged latency achieved by CEDC-O converges back to $\mathcal{L} = 0.8$ in time slots 4, 100 and 195, respectively. In the small-scale figure, the blue line overtakes the red one in the 5th time slot. This is because of $\Delta(\sigma(t)) = 0$ in the 4th time slot and that no penalty is produced in the next time slot. In this case, the objective becomes $\min(\mathcal{Q} + \gamma \cdot \mathcal{C}(\lambda^t))$ and the time-averaged latency increases significantly in the 5th time slot. In the meantime, CEDC-O with $\gamma = 1.0$ still tries to converge the time-averaged latency back to $\mathcal{L}$. After the 5th time slot, the blue line is always lower than the red one, while the red one is lower than the yellow one. In conclusion, the system stability, ensured by the long-term latency constraint (13) (same as the accumulated latency requirement (16)), is ensured by the CEDC-O algorithm with different trade-off parameters.

5.10 Threats to Validity

5.10.1 Construct Validity

The major threat to construct validity is the four approaches used for comparison. Due to the novelty of the CEDC problem in the EC environment, DO has a different objective from CEDC and OO only considers the current time slot, while IPEDC and MR do not consider the long-term latency constraint. Thus, there is a threat that the comparison does not suffice to comprehensively evaluate CEDC-O. To minimize this threat, we enhanced IPEDC and MR by including (32) into their implementation. Moreover, we changed seven parameters, as presented in Table 3, to simulate various CEDC scenarios. In this way, we could evaluate our approaches by not only the comparison to the other approaches, but also the demonstration of how the changes in the parameters impact the performance of CEDC-O.

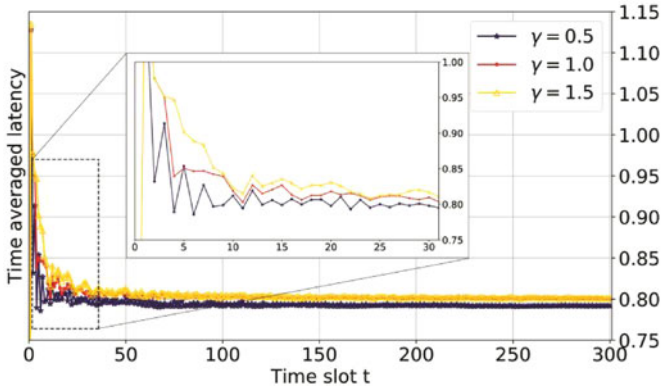


Fig. 10. Simulation Set #8.

5.10.2 External Validity

The main threat to the external validity of the evaluation is whether our approaches can be generalized and applied in other CEDC scenarios in the EC environment. To tackle this threat, We measured the performance of our approaches in a generic way. Specifically, we measured the data sizes and cache spaces by the number of data units and the data retrieval latency by the number of hops. In this way, the evaluation results can be interpreted with specific models for all cost components. In addition, we ran the simulations on a widely-used real-world data set while varying seven parameters to vary the size and the complexity of the CEDC problem. This way, the representativeness and comprehensiveness of the evaluation are ensured.

6 Related Work

Data caching have been extensively investigated in the fields of conventional distributed computing and cloud computing environments. With the popularity of edge computing, data caching in the edge computing environment is obtaining attention from researchers recently.

6.1 Conventional Distributed Data Caching

In the last few decades, there are many data caching problems investigated in conventional distributed computing environments, including web caching [13], content delivery network [35], etc. Banerjee *et al.* [36] developed a content placement strategy for information-centric network based on data popularity, namely Greedy Caching. With popular contents cached in the network, the Greedy Caching approach considered the cache miss rate at the edge to decide what contents would be cached on the core server. In [37], the authors formulated two caching strategies for data publish-subscribe systems, including eviction-based caching and time-to-live-based caching to address the space and time issues, respectively. The authors of [38] focused on balancing the trade-off between latency and cost in the content-centric network. They addressed this issue with a holistic model for provisioning the storage capability based on the network performance and the provisioning cost.

6.2 Cloud Data Caching

In the cloud computing environment, a critical problem of data caching is how to utilize cache space efficiently on cloud hosts and mobile devices. Arteaga *et al.* [39] proposed CloudCache, a method for managing cache, to fulfill the caching requirement of the workload and minimize cache wear-out. In [40], the authors presented how to use segment access-aware dynamic semantic cache in the cloud computing environment for relational databases. A cache access algorithm was introduced to consider cache exact hit, cache extended hit, cache partial hit and cache miss. The authors of [41] explored the cache design space for embedded processors with evolutionary techniques for mobile and thin client processors in the cloud computing environment. A heuristic and evolutionary method was presented to generate a near-optimal cache space design for enhancing service quality. In [42], the authors formulated a benefit maximization problem and created a cache replacement approach based on traffic requirements. They also introduced a content clustering method for collecting popular data and clustering similar contents.

6.3 Edge Data Caching

Edge computing (EC) extends cloud computing with computing resources and services geographically distributed at the edge of the cloud [43]. With the deployment of edge servers, the problem of computation offloading arises. It has been well studied with consideration of edge servers' energy efficiency, offloading cost and joint caching [28], [44].

Recently, the challenges raised by data caching are being investigated in the EC environment. Existing data caching approaches are rendered obsolete by the new characteristics of EC and thus cannot be directly applied in the EC environment. Thus, researchers are proposing and investigating new ideas and techniques for data caching in the EC environment. Cao *et al.* present an optimal auction mechanism to maximize the service provider's revenue based on cache allocation and user valuation reports. They propose computationally efficient approaches to apply the auction mechanism based on data retrieval and delivery costs. The authors of [45] propose a caching system named Cachier for recognition applications in the EC environment. Cachier coordinates the loading balance between edge servers and the cloud to minimize the data retrieval latency dynamically. However, the above approaches employ offline methods and require complete information about active users and data requests in all time slots. They cannot handle edge data caching scenarios where data and users may come and go randomly.

Instead of solving the edge data caching problem optimally in an offline manner, some researchers investigate online approaches for solving the dynamic edge data caching problems. Xu *et al.* [28] propose an online algorithm named OREO to decide service caching and task offloading. The system aims to minimize the total network latency and applies a long-term energy consumption constraint to stabilize the edge caching system. The authors of [46] propose MOREA, an online algorithm considering user mobility, to allocate different resources like caches and CPUs on edge servers for computation offloading. In [47], the authors integrate the cloud radio access network with the EC technology to schedule resources including caches and computational resources dynamically. They propose the VariedLen algorithm to maximize the mobile network provider's profit. They also extend the standard Lyapunov technology so that individual tasks can be performed across multiple time slots. However, existing works investigate edge data caching only to complement computing offloading and fail to give data caching sufficient attention as a unique technology with advantages in reducing data retrieval latency and improving the quality of services and users' experiences, especially from the app vendor's perspective who is an important stakeholder in the EC environment.

Edge computing inherits the pay-as-you-go pricing model from cloud computing, which allows app vendors to hire storage resources on edge servers from edge infrastructure providers to cache app data for their users. Thus, the cost incurred by data caching for app vendors is critical to the success of edge computing because, after all, app vendors are the main customers in the edge computing environment. To the best of our knowledge, this paper makes the first attempt to propose an approach named CEDC-O for solving the CEDC problem from the app vendor' perspective in the EC environment. By innovatively and realistically modeling the CEDC problem as a long-term optimization problem, CEDC-O can help app vendors ensure the long-term performance of their edge data caching performance.

7 Conclusion

In this paper, we studied the collaborative edge data caching (CEDC) problem. We first identified the major challenges and proposed a comprehensive cost model for this problem, where system cost is composed of data caching cost, data migration cost and QoS penalty. We also proved the $\mathcal{NP}$-completeness of the CEDC problem. We proposed CEDC-O, an online algorithm with provable performance guarantee, and evaluated its performance with extensive simulations. This research has established the foundation for the CEDC problem and opened up a number of future research directions. In our future work, we will consider dynamics on available edge server caches, user mobility and security policies.

Acknowledgments

This work was funded in part by Australian Research Council Discovery Projects (DP180100212 and DP200102491) and Laureate Fellowship FL190100035.

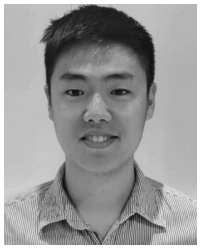
References

[1] A. Osseiran *et al.*, "The foundation of the mobile and wireless communications system for 2020 and beyond: Challenges, enablers and technology solutions," in *Proc. IEEE 77th Veh. Technol. Conf.*, 2013, pp. 1–5.

[2] P. Lai *et al.*, "Optimal edge user allocation in edge computing with variable sized vector bin packing," in *Proc. Int. Conf. Service-Oriented Comput.*, 2018, pp. 230–245.

[3] T. X. Tran, M.-P. Hosseini, and D. Pompili, "Mobile edge computing: Recent efforts and five key research directions," *IEEE COMSOC MMTC Commun.-Frontiers*, vol. 12, no. 4, pp. 29–33, Jul. 2017.

[4] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1619–1632, Aug. 2018.
[5] C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang, "Computation offloading and resource allocation in wireless cellular networks with mobile edge computing," *IEEE Trans. Wireless Commun.*, vol. 16, no. 8, pp. 4924–4938, Aug. 2017.
[6] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1397–1411, Mar. 2017.
[7] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.
[8] Q. He *et al.*, "A game-theoretical approach for user allocation in edge computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 3, pp. 515–529, Mar. 2020.
[9] P. Stenstrom, "A survey of cache coherence schemes for multiprocessors," *Computer*, vol. 23, no. 6, pp. 12–24, 1990.
[10] J. D. Owens *et al.*, "A survey of general-purpose computation on graphics hardware," in *Computer Graphics Forum*, vol. 26. Hoboken, NJ, USA: Wiley, 2007, pp. 80–113.
[11] B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk*. San Mateo, CA, USA: Morgan Kaufmann, 2010.
[12] A. J. Smith, "Disk cachemiss ratio analysis and design considerations," *ACM Trans. Comput. Syst.*, vol. 3, no. 3, pp. 161–203, 1985.
[13] S. Podlipnig and L. Böszörmenyi, "A survey of web cache replacement strategies," *ACM Comput. Surv.*, vol. 35, no. 4, pp. 374–398, 2003.
[14] K. Elhardt and R. Bayer, "A database cache for high performance and fast restart in database systems," *ACM Trans. Database Syst.*, vol. 9, no. 4, pp. 503–525, 1984.
[15] A. Mukhopadhyay, N. Hegde, and M. Lelarge, "Optimal content replication and request matching in large caching systems," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 288–296.
[16] K. Ji, G. Quan, and J. Tan, "Asymptotic miss ratio of LRU caching with consistent hashing," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 450–458.
[17] G. Casale, "Analyzing replacement policies in list-based caches with non-uniform access costs," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 432–440.
[18] K. Cho, M. Lee, K. Park, T. T. Kwon, Y. Choi, and S. Pack, "Wave: Popularity-based and collaborative in-network caching for content-oriented networks," in *Proc. IEEE Conf. Comput. Commun. Workshops*, 2012, pp. 316–321.
[19] M. A. Maddah-Ali and U. Niesen, "Decentralized coded caching attains order-optimal memory-rate tradeoff," *IEEE/ACM Trans. Netw.*, vol. 23, no. 4, pp. 1029–1040, Aug. 2015.
[20] M. Dehghan *et al.*, "On the complexity of optimal request routing and content caching in heterogeneous cache networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 3, pp. 1635–1648, Jun. 2017.
[21] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Trans. Inf. Theory*, vol. 60, no. 5, pp. 2856–2867, May 2014.
[22] C.-Y. Wang, S. H. Lim, and M. Gastpar, "Information-theoretic caching: Sequential coding for computing," *IEEE Trans. Inf. Theory*, vol. 62, no. 11, pp. 6393–6406, Nov. 2016.
[23] M. Patel *et al.*, "Mobile edge computing-introductory technical white paper," *Mobile-Edge Comput. (MEC) Industry Initiative*, White Paper, pp. 1089–7801, 2014.
[24] H. Guo and J. Liu, "Collaborative computation offloading for multi-access edge computing over fiber-wireless networks," *IEEE Trans. Veh. Technol.*, vol. 67, no. 5, pp. 4514–4526, May 2018.
[25] M. Armbrust *et al.*, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.
[26] M. Chen, Y. Hao, K. Lin, Z. Yuan, and L. Hu, "Label-less learning for traffic control in an edge network," *IEEE Netw.*, vol. 32, no. 6, pp. 8–14, Nov./Dec. 2018.
[27] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computinga key technology towards 5G," *ETSI White Paper*, vol. 11, no. 11, pp. 1–16, 2015.
[28] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 207–215.
[29] T. Zhang, F. Ren, and R. Shu, "Towards stable flow scheduling in data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 11, pp. 2627–2640, Nov. 2018.
[30] N. Abedini and S. Shakkottai, "Content caching and scheduling in wireless networks with elastic and inelastic traffic," *IEEE/ACM Trans. Netw.*, vol. 22, no. 3, pp. 864–874, Jun. 2014.
[31] E. Hazan, S. Safra, and O. Schwartz, "On the complexity of approximating k-set packing," *Comput. Complexity*, vol. 15, no. 1, pp. 20–39, 2006.
[32] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures Commun. Netw.*, vol. 3, no. 1, pp. 1–211, 2010.
[33] X. Xia *et al.*, "Graph-based optimal data caching in edge computing," in *Proc. Int. Conf. Service-Oriented Comput.*, 2019, pp. 477–493.
[34] Y. Liu, Q. He, D. Zheng, M. Zhang, F. Chen, and B. Zhang, "Data caching optimization in the edge computing environment," in *Proc. IEEE Int. Conf. Web Serv.*, 2019, pp. 99–106.
[35] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter , "AdaptSize: Orchestrating the hot object memory cache in a content delivery network," in *Proc. 14th USENIX Symp. Netw. Syst. Des. Implementation*, 2017, pp. 483–498.
[36] B. Banerjee, A. Kulkarni, and A. Seetharam, "Greedy caching: An optimized content placement strategy for information-centric networks," *Comput. Netw.*, vol. 140, pp. 78–91, 2018.
[37] M. Y. S. Uddin and N. Venkatasubramanian, "Edge caching for enriched notifications delivery in big active data," in *Proc. 38th IEEE Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 696–705.
[38] Y. Li, H. Xie, Y. Wen, and Z.-L. Zhang, "Coordinating in-network caching in content-centric networks: Model and analysis," in *Proc. 33rd IEEE Int. Conf. Distrib. Comput. Syst.*, 2013, pp. 62–72.
[39] D. Arteaga, J. Cabrera, J. Xu, S. Sundararaman, and M. Zhao, "CloudCache: On-demand flash cache management for cloud computing," in *Proc. 14th USENIX Conf. File Storage Technol.*, 2016, pp. 355–369.
[40] K. Ma, B. Yang, Z. Yang, and Z. Yu, "Segment access-aware dynamic semantic cache in cloud computing environment," *J. Parallel Distrib. Comput.*, vol. 110, pp. 42–51, 2017.
[41] A.-H. A. Badawy, G. Yessin, V. Narayana, D. Mayhew, and T. El-Ghazawi, "Optimizing thin client caches for mobile cloud computing: Design space exploration using genetic algorithms," *Concurrency Comput.: Practice Experience*, vol. 29, no. 11, 2017, Art. no. e4048.
[42] S. Tamoor-ul Hassan , S. Samarakoon, M. Bennis, M. Latva-Aho , and C. S. Hong, "Learning-based caching in cloud-aided wireless networks," *IEEE Commun. Lett.*, vol. 22, no. 1, pp. 137–140, Jan. 2018.
[43] M. Yannuzzi *et al.*, "A new era for cities with fog computing," *IEEE Internet Comput.*, vol. 21, no. 2, pp. 54–67, Mar./Apr. 2017.
[44] S. Jošilo and G. Dán, "A game theoretic analysis of selfish mobile computation offloading," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.
[45] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, "Cachier: Edge-caching for recognition applications," in *Proc. 37th IEEE Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 276–286.
[46] L. Wang, L. Jiao, J. Li, J. Gedeon, and M. Mühlhäuser, "MOERA: Mobility-agnostic online resource allocation for edge computing," *IEEE Trans. Mobile Comput.*, vol. 18, no. 8, pp. 1843–1856, Aug. 2019.
[47] X. Wang *et al.*, "Dynamic resource scheduling in mobile edge cloud with cloud radio access network," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 11, pp. 2429–2445, Nov. 2018.



Xiaoyu Xia received the master's degree from the University of Melbourne, Australia, in 2015. He is currently working toward the PhD degree at Deakin University. His research interests include edge computing, service computing, and software engineering.

Feifei Chen received the PhD degree from the Swinburne University of Technology, Australia, in 2015. She is a lecturer with Deakin University. Her research interests include software engineering, cloud computing, and green computing. For more details please visit https://sites.google.com/view/feifeichen/.

Qiang He (Senior Member, IEEE) received the first PhD degree from the Swinburne University of Technology, Australia, in 2009, and the second PhD degree in computer science and engineering from the Huazhong University of Science and Technology, China, in 2010. He is a senior lecturer with Swinburne. His research interests include service computing, software engineering, cloud computing, and edge computing. For more details please visit https://sites.google.com/site/heqiang/.

John C. Grundy (Senior Member, IEEE) received the BSc (Hons), MSc, and PhD degrees in computer science from the University of Auckland, New Zealand. He is currently an Australian Laureate fellow and a professor of software engineering with Monash University, Melbourne, Australia. He is an associate editor of the *IEEE Transactions on Software Engineering*, *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. For more details please visit https://sites.google.com/site/johncgrundy/.

Mohamed Abdelrazek received the PhD degree from Swinburne University, in 2014. He is currently an associate professor with software engineering and IoT with the School of Information Technology, Deakin University, Australia. He has more than 10 years experience in building software solutions. His research interests include software engineering, security, and artificial intelligence. For more details please visit https://sites.google.com/site/mohamedalmorsy/.

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