Constrained App Data Caching over Edge Server Graphs in Edge Computing Environment

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

Abstract—In recent years, edge computing, as an extension of cloud computing, has emerged as a promising paradigm for powering a variety of applications demanding low latency, e.g., virtual or augmented reality, interactive gaming, real-time navigation, etc. In the edge computing environment, edge servers are deployed at base stations to offer highly-accessible computing capacities to nearby end-users, e.g., CPU, RAM, storage, etc. From a service provider's perspective, caching app data on edge servers can ensure low latency in its users' data retrieval. Given constrained cache spaces on edge servers due to their physical sizes, the optimal data caching strategy must minimize overall user latency. In this paper, we formulate this Constrained Edge Data Caching (CEDC) problem as a constrained optimization problem from the service provider's perspective and prove its $\mathcal{NP}$-hardness. We propose an optimal approach named CEDC-IP to solve this CEDC problem with the Integer Programming technique. We also provide an approximation algorithm named CEDC-A for finding approximate solutions to large-scale CEDC problems efficiently and prove its approximation ratio. CEDC-IP and CEDC-A are evaluated on a real-world data set. The results demonstrate that they significantly outperform four representative approaches.

Index Terms—edge computing, data caching, optimization, approximation 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 [1]. These devices introduce massive traffic that leads to network congestion and significantly impacts the quality of service, especially service latency, which has become the major obstacle to latency-sensitive applications such as virtual or augmented reality, interactive gaming, real-time navigation [2]. *Edge computing* is proposed to tackle this challenge, where *edge servers* are attached to base stations or access points close to users to offer them computation and storage resources at the edge of the network [3]. It is a key technology that facilitates the 5G mobile network [4]. In the edge computing environment, edge servers, each powered by one or more physical machines, are deployed at base stations or access points that are geographically close to app users. Service providers can hire computing capacities on edge servers and host their applications on edge servers (referred to as *edge apps* hereafter) to ensure low latency for their app users [5]. In the meantime, computation tasks can be offloaded from app users' devices to their nearby edge servers to reduce the computation consumption and save energy on their devices [6], [7].

As an increasing number of mobile and IoT devices begin to access data through edge servers, more app data will be delivered through edge servers for the app users. Caching those app data, especially the popular ones like viral videos and photos from service providers such as Facebook, will minimize the data retrieval delay. They can retrieve app data from edge servers rather than from the remote cloud server if those data are already available on those edge servers. Additionally, caching app data on edge servers can also considerably reduce the amount of data transferred from the cloud to app users, and consequently lower the service providers' cost of data transfer under the pay-as-you-go pricing scheme [8].

Data caching techniques have been well-studied and employed in many domains, i.e., database [9] and web [10]. Data caching has also been intensively investigated in the network domain to leverage its advantages, i.e. reducing network latency, saving bandwidth consumption, and reducing energy consumption. In the last few years, many researchers have investigated caching strategies from different perspectives, e.g., information theoretic caching [11], coded caching [12] and request routing [13]. As a new distributed computing paradigm, edge computing offers new opportunities and raises critical challenges. The fundamental objective and mechanism are to caching popular app data to ensure users' low retrieve latency is the fundamental objective for data caching. This is especially important for latency-sensitive applications, e.g., interactive web gaming such as Google Stadia, social video streaming such as TicTok, etc. Caching app data on edge servers can reduce the network traffic data significantly, thus, the traffic burden on the backbone network can be lifted [14].

- *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.*
- *J. Grundy is with Faculty of Information Technology, Monash University, Melbourne, Victoria, Australia. E-mail: john.grundy@monash.edu.*
- *H. Jin is with School of Computer Science and Technolgoy, HuaZhong University of Science and Technology, China. Email: hjin@hust.edu.cn.*
- *Q. He is with School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Victoria, Australia. E-mail: qhe@swin.edu.au.*

Manuscript received March xx, 202x; revised June xx, 202x.

From the service provider's perspective, edge data caching (EDC) aims to cache a single data on edge servers to cover all the app users in a specific area at minimum data caching cost [15], [16]. However, three major issues have not been considered properly by the existing work. **First**, a service provider may want to cache multiple data for its users in the same area. For example, YouTube may want to cache multiple popular videos requested by a lot of users. **Second**, edge servers' storage capacities are constrained and must be reserved by service providers for caching their data. Unlike cloud servers that have access to virtually unlimited storage capacities in the cloud, edge servers' storage capacities are limited due to its size limit [6], [17], [18], [19]. In the open edge computing environment, many service providers may need to hire storage capacities on the edge servers in the same area for caching their data. This causes fierce competition among service providers and makes it practically impossible for every service provider to cache a huge amount of data on every edge server. In such an environment, a common practice is for service providers to reserve storage spaces on edge servers for caching their data. Thus, cost-effectiveness is a key factor in the formulation their data caching strategies. Unlike the edge infrastructure provider who often aims to serve all the users, service providers pursue to maximize caching benefits by fully utilizing the reserved caching spaces. Full user coverage is not always mandatory. **Third**, an app user may be able to retrieve data from edge servers via multiple hops over the edge server graph, instead of just zero or one hop as constrained in [15]. Based on the edge-cloud architecture [20], adjacent edge servers deployed at different base stations can communicate with each other and transmit data via high-speed links [6], [21], [22]. Thus, an app user can access data cached on an edge server via multiple hops over the edge server graph. However, to ensure low data retrieval latency for its users, a service provider must specify its app-specific latency constraint by the maximum number of hops via which its user can retrieval data from an edge server over the edge server graph.

To summarize, in practice, a service provider usually reserves some caching spaces on edge servers - depending on its caching budget - to cache multiple popular data on edge servers in an area for its users to access under the latency constraint. An *optimal data caching strategy* must minimize its app users' data retrieval latency with constrained hired storage spaces on edge servers. In this paper, this problem is referred as the *Constrained Edge Data Caching* (CEDC) problem. We study this problem from the service provider's perspective to address the above three issues. The major contributions of this paper are:

- We model and formulate the CEDC problem as a constrained optimization problem (COP) from the service provider's perspective.
- We prove that the CEDC problem is $\mathcal{NP}$-hard based on the weighted k-set packing problem.
- We develop an optimal approach, namely CEDC-IP, for solving the CEDC problem exactly with the Integer Programming technique.
- We develop an approximation approach named CEDC-A for finding approximate solutions to large-scale CEDC problems efficiently and prove the approximation ratio.
- We conduct extensive experiments on a real-world data set to evaluate the proposed approaches against four representative approaches.

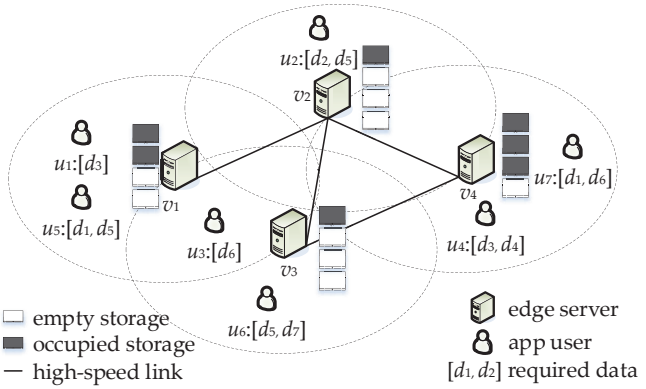


Fig. 1. An example CEDC scenario

The rest of the paper is organized as follows. Section 2 illustrates and motivates the CEDC problem with an example. Section 3 formulates the CEDC problem and proves the $\mathcal{NP}$-hardness of the CEDC problem. Section 4 proposes and analyzes the optimal approach CEDC-IP and the approximation approach CEDC-A for solving CEDC problems. In section 5, we experimentally evaluates the proposed approaches based on extensive experiments. After that, we review the related work in Section 6. In Section 7, we conclude this paper and point out the future work.

2 Motivating Example

In the edge computing environment, adjacent edge servers are deployed at base stations. The communications and data sharing between neighbour edge servers are through high-speed links [6], [21]. In a specific geographic area, edge servers and links between them can constitute an *edge server network*. This edge server network can be modeled as a graph where each edge server is represented by a node and each link is represented by an edge. Data caching in an edge server network differs from that in the conventional distributed computing environments and cloud computing environment with its three unique constraints, i.e.,*server capacity constraint*, *server coverage constraint* and *server adjacency constraint*.

Server Capacity Constraints: The storage resources on an edge server are usually limited due to its size limit [3], [15]. The competition between service providers makes it impossible for a service provider to cache all app data on every edge server. Thus, the common practice is for service providers to reserve certain cache spaces on edge servers for caching popular app data.

Server Coverage Overlaps: To avoid any blank coverage areas in a specific geographic area, the coverage areas of nearby edge servers often intersect [23], [24]. Thus, app users in an overlapping area can access any of the edge servers covering them.

Server Adjacency Constraints: An app user can retrieve a piece of app data from its nearby edge servers – we refer to as *local edge servers* hereafter – that cover the app user if the app data is cached on one of these edge servers. If the app data is not cached on any of those local edge servers, the app user can retrieve it from other edge servers that are linked to its local edge servers via multiple hops over the edge server graph – we refer to as *neighbor edge servers* – under the latency constraint. Either way, the service provider's latency constraint will ensure that it is faster than retrieving the app data from a server in the remote cloud.

From the service provider's perspective, the objective of CEDC is to minimize its users' overall data retrieval latency by caching app data with limited reserved cache spaces on edge servers in a specific area. A representative CEDC scenario is caching viral videos and photos for social web apps. Social app users such as Facebook or Instagram users access popular videos and photos shared by either their friends or public figures. Always transmitting data from the cloud to individual app users creates immense pressure on the network and increases the latency in their data retrieval, especially in areas with high user density and dynamic traffic conditions. Caching those data on edge servers brings them much closer to the users and reduce the latency in their data retrieval.

Example 1: Fig. 1 presents an example area with four edge servers, i.e., $\{v_1, ..., v_4\}$, each covering a specific geographic area. The boxes by each edge server represent the cache spaces hired by the service provider on that edge server. To illustrate the CEDC problem generically, each box in Fig. 1 can cache one piece of app data. From the service provider's perspective, caching all its popular data on every edge server in the area can easily accommodate all its users in the area. However, this is not cost-effective nor practical due to the server capacity constraint discussed above. Due to the server coverage constraint and the server adjacency constraint, the data must be cached on a number of those edge servers so that all the users in that area can retrieve the data from either their local edge servers or neighbor edge servers. For example, we assume that the service provider' latency constraint in Fig. 1 is 1 hop. This allows an app user to access any edge servers within 1 hops over the edge server graph for cached app data. Otherwise, it will have to retrieve it from the service provider's remote cloud server. For example, u_1 and u_5 can only retrieve cached data from either their local edge server v_1 or their neighbor edge server v_2. Apparently, there are multiple data caching strategies that fulfill all the three constraints. **The one that minimizes the users' overall data retrieval latency is the optimal solution to the CEDC problem**. Thus, the CEDC problem is inherently a constrained optimization problem (COP).

Model: In this paper, we quantify the optimization objective and constraints in the CEDC problem in a generic manner. For example, we model the data sizes and cache spaces by the number of data units, and the data retrieval latency by the number of hops. In Fig. 1, each piece of data to be cached is treated as 1 unit, and the total number of cache spaces on v_1 is 4 units, 2 of which are available at the moment. This way, these models can be easily extended for calculating data caching cost given the data size and a specific pricing model, e.g., caching cost per size. Let us still assume that the service provider's latency constraint in this scenario is 1 hop, and user u_7 requests for data d_1 and d_6. The data retrieval latency consists of two major components: the latency between the user and its local edge server(s), and the latency between edge servers. The former component is not avoidable and thus is not considered in the formulation of data caching strategy. Thus, u_7's data retrieval latency is 0 hop if the requested data is cached on edge server v_4, or 1 hop if it is cached on v_2 or v_3. If the data is only cached on edge server v_1, u_7 cannot retrieve the data from any edge server in this area without violating the latency constraint. Similar to the generic models for data and cache spaces, the generic latency model can be extended with different latency specifications.

The model and approaches proposed in this research are generic and applicable to various apps. In our model data are cached on edge servers in whole and we do not consider the situation where data can be partially cached, e.g., video segments. In addition, the scale of the CEDC problem in real-world scenarios can be much larger than the example presented in Fig. 1. Finding an optimal solution to such a CEDC problem is not trivial. Similar to many studies of edge computing [6], [21], [23], [25], [26], [27], [28], [29], we investigate the CEDC problem in quasi-static scenarios where the app users remain unchanged during the data retrieval, e.g., their data needs and locations. More dynamic scenarios will be investigated in our future work.

3 Problem Formulation

3.1 Problem Statement

In this research, we model the networked n edge servers in a specific area as a graph $G(V, E)$ where V is the set of nodes and E is the set of edges in G. In this graph, each node $v_i \in V$ represents an edge server, while each edge $e_t \in E$ represents an edge between two nodes in G. *In the remainder of this paper, we will speak inter-changeably of an edge server and its corresponding node in G, both denoted by v.* The notations adopted in the paper are summarized in Table 1.

As described in Section 2, we formulate the CEDC problem in a generic manner by measuring data retrieval latency by the number of hops between edge servers and data sizes and spaces by the number of data units.

Given a set of data $D = \{d_1, ..., d_k\}$ to be cached on the edge servers in the area, a data caching strategy is a vector $R = \{\langle r_1^1, ..., r_n^1\rangle, ..., \langle r_1^k, ..., r_n^k\rangle\}$, where $r_i^f \in \{0, 1\} (1 \leq i \leq n, 1 \leq f \leq k)$ denotes whether data d_f is cached on edge server v_i.

As discussed in Section 2, the service provider has reserved a finite amount of cache spaces on each edge server. Thus, the number of data cached on an edge server v_i cannot exceed its available cache spaces as_i:

$$\sum_{d_f \in D} r_i^f \leq as_i \tag{1}$$

The distance between two nodes in the graph is measured by their shortest path. As we use the number of hops to measure the data retrieval latency, the latency in an app user u's retrieval of data f is measured as follow:

$$l_u^f = \min\{l_{i,j}, r_j^f = 1, v_j \in V\}, \forall u \in U_i \tag{2}$$

TABLE 1
Summary of Notations

Notation	Description
as_i	available cache spaces on edge server i
b_u^f	maximum benefit for user u retrieving d_f
$b_{u,j}$	benefit of caching replica on server v_j for app user u
D	finite set of data
d_f	data f
E	set of links between edge servers
G	graph presenting a particular area
k	total number of data in D
L	latency constraint from service provider
$l_{i,j}$	latency from server i to server j
l_u^f	minimum latency from app user u to retrieve d_f
m	total number of app users
n	total number of edge servers
R	set of binary variables $\langle r_1^f, ..., r_n^f \rangle$ indicates whether data d_f is cached on edge servers from v_1 to v_n
r_i^f	binary variable indicating cache d_f on edge server v_i
t_u^f	binary variable indicating user u requires data d_f
U	set of app users
U_i	set of users covered by server v_i
V	set of edge servers
v_i	edge server i

where $l_{i,j}$ is the number of hops between v_i and v_j.

To evaluate and compare the effectiveness of different data caching strategies, the concept of data caching benefit is introduced here. It is calculated based on the latency reduction in user data retrieval. We use the number of hops reduced by cached data on an edge server to measure the data caching benefit. Given a latency constraint value L, the data caching benefits is 0 when the latency achieves $L+1$ hops. Denote $l_T = L+1$ as the value breaking the threshold, the following equation shows how to calculate the benefit $b_{u,j}$ produced for app user $u \in U_i$ if edge server v_j is selected to cache the data:

$$b_{u,j} = \max\{l_T - l_{i,j}, 0\} \tag{3}$$

Example 2: Take Fig. 1 as an example. Let us assume $L = 1$ and edge server v_1 is selected to cache data d_5. This way, $L_T = 2$ and we can calculate the benefits for user u_2, u_5 and u_6 to obtain d_5 from v_1, where $b_{2,1} = 1$, $b_{5,1} = 2$ and $b_{6,1} = 0$.

As discussed in Section 2, to avoid the blank area that are not covered by any edge servers, the coverage of nearby edge servers often partially overlap [3]. An app user in the overlapping area can access multiple optional local edge servers and neighbor edge servers for cached data. Thus, the data caching benefit produced by the data caching strategy for an app user u to retrieve data f is:

$$b_u^f = \max\{r_j^f \cdot b_{u,j} \cdot t_u^f, v_j \in V, t_u^f \in \{0,1\}\} \tag{4}$$

where the binary variable t_u^f indicates whether user u requires d_f.

From the service provider's perspective, the optimization objective is to maximize the total reduction in all users' overall data retrieval latency produced by its data caching strategy R, which can be converted to the maximization of the total caching benefit of all users based on (4):

$$\text{maximize } benefit(R) \tag{5}$$

3.2 Problem Hardness

In this section, we demonstrate that the COP of CEDC is $\mathcal{NP}$-hard by proving the following theorem.

Theorem 1. *The COP of CEDC is $\mathcal{NP}$-hard.*

Proof To prove this problem is $\mathcal{NP}$-hard, we first introduce the weighted k-set packing problem (WKSP). The WKSP problem is known to be $\mathcal{NP}$-hard [30]. Given a universe U_e with elements $\forall e \in U_e$, a set S of subsets of U_e and an integer number k. The subset C is a packing, where $C \subseteq S$. All sets $s \subseteq C$ are pairwise disjoint. Let $weight(s)$ be the weight of the set s and k be the maximum number of selected sets. The formulation is displayed below:

$$object : \max \sum_{s \in C} weight(s) \cdot X_s \tag{6a}$$

$$s.t. : \sum_{s \in S} X_s \leq k \tag{6b}$$

$$X_s \in \{0,1\}, \forall s \in S \tag{6c}$$

$$\sum_{e \in U_e} X_e \leq 1 \tag{6d}$$

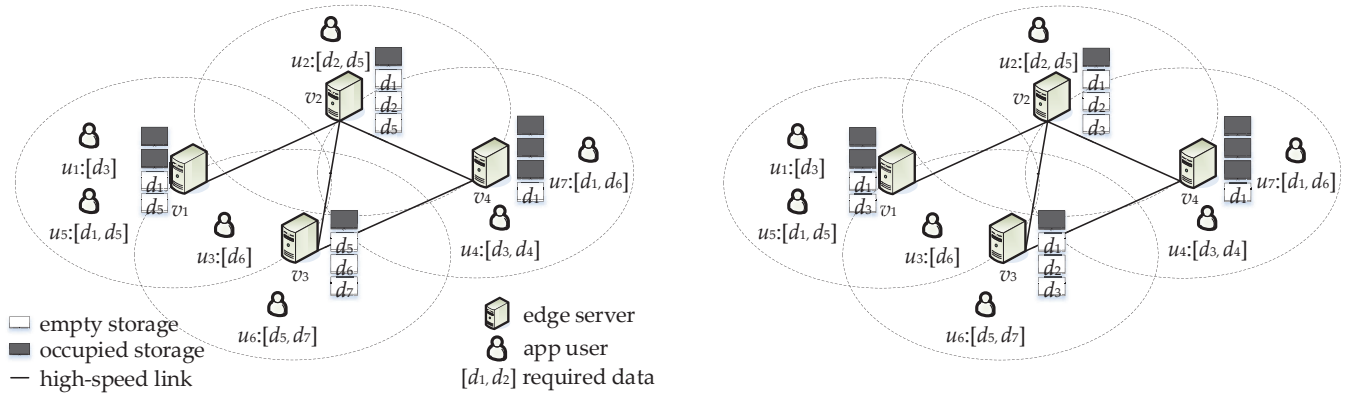
Now we prove that the WKSP problem can be reduced to an instance of the CEDC problem. We define the elements based on the data requests and the users. For example, u_1 require data $\{d_1, d_2\}$, u_2 requires $\{d_2\}$ and u_3 requires $\{d_1\}$. In this case, we can define the elements $e \in \{t_1^1, t_1^2, t_2^2, t_3^1\}$. The reduction can be done as follows: given an instance $WKSP(S, U_e, k, weights(s))$, we can construct the set of $|S|$ servers, denoted by V, and the set of $|U_e|$ users, denoted by U. Let $n = k$, we can construct an instance $CEDC(V, U, n, benefit(v))$ with the reduction above in polynomial time, where function $benefit(v_f)$ is calculated as the sum of benefit if data f is cached on edge server v. As the constraint (6b) restricts the total number of selected sets, we can project the equation $\sum_{v_i \in V} \sum_{d_f \in D} r_i^f \leq \sum_{v_i \in V} as_i$ based on (1) in the CEDC problem to that constraint, such as $benefit(v_i) = 0$ if there is no available cache spaces on edge server i. Based on (3) and (4), data f requested by user $u \in U$ can only be calculated once into the data caching benefit. Thus, constraint (6d) can be fulfilled. Moreover, the increase in the benefit produced by caching data f on edge server v can be projected to $weight(s)$. In this case, any solution R satisfying objective (6a) also satisfies objective (4).

In conclusion, any solution $\mathcal{Y}$ always satisfies the reduced CEDC problem if $\mathcal{Y}$ satisfies the WKSP problem. Therefore, the CEDC problem is reducible from the WKSP problem and it is $\mathcal{NP}$-hard.

□

4 Edge Web Data Caching Strategy Formulation

We first model the CEDC problem with the Integer Programming technique, then prove that the CEDC problem is $\mathcal{NP}$-hard. After that, we propose an approximation algorithm for finding approximate solutions to large-scale CEDC problems efficiently and prove its approximation ratio.



(a) Strategy R_1 (b) Strategy R_2

Fig. 2. Example CEDC Strategies

4.1 Optimization Approach

From the service provider's perspective, the solution to the CEDC problem must maximize the data caching benefit measured by the total reduction in all the app users' data retrieval latency in this area. In the meantime, the server capacity constraint, server coverage constraint and server adjacency constraint must be fulfilled. Thus, the CEDC problem can be modeled as a constrained optimization problem (COP). The COP model for the CEDC problem is formally expressed as follows.

For a graph $G = (V, E)$, where $V = \{v_1, .., v_n\}$, there is a matrix of variables $R = \{\langle r_1^1, ..., r_n^1\rangle, ..., \langle r_1^k, ..., r_n^k\rangle\}$, where $r_i^f \in \{0, 1\}, \forall i \in \{1, ..., n\}, \forall f \in \{1, ..., k\}$, r_i^f being 1 if a data replica of d_f is cached on the i^{th} node, 0 otherwise. The constraints for the COP model are:

$$b_u^f = \max(r_i^f \cdot b_{u,i} \cdot t_u^f) \quad \forall i \in \{1, ..., n\}, \forall u \in \{1, ..., m\}, \forall f \in \{1, ..., k\} \tag{7}$$

$$\sum_{f=1}^{k} r_i^f \leq as_i, \forall i \in \{1, ..., n\}, \forall f \in \{1, ..., k\} \tag{8}$$

Constraint family (7) is converted from (4). It ensures that every app user will always retrieve the required data from the nearest possible edge server. Constraint family (8) guarantees that the data cached on each edge servers must not exceed the available cache spaces.

As discussed in Section 3.1, there might be multiple solutions to this COP fulfilling (7) and (8).

Example 3: In Fig. 1, there are multiple possible solutions satisfying constraints (7) and (8). Assuming the latency constraint is no more than 1 hop in this scenario, two possible data caching strategies are presented in Fig. 2: $R_1 = \{\langle 1, 0, 0, 0, 1, 0, 0\rangle, \langle 1, 1, 0, 0, 1, 0, 0\rangle, \langle 0, 0, 0, 0, 1, 1, 1\rangle, \langle 1, 0, 0, 0, 0, 0, 0\rangle\}$ that caches data $\{d_1, d_5\}$ on v_1, $\{d_1, d_2, d_5\}$ on v_2, $\{d_5, d_6, d_7\}$ on v_3 and $\{d_1\}$ on v_4, and $R_2 = \{\langle 1, 0, 1, 0, 0, 0, 0\rangle, \langle 1, 1, 1, 0, 0, 0, 0\rangle, \langle 1, 1, 1, 0, 0, 0, 0\rangle, \langle 1, 0, 0, 0, 0, 0, 0\rangle\}$, caching data $\{d_1, d_3\}$ on v_1, $\{d_1, d_2, d_3\}$ on v_2, $\{d_1, d_2, d_3\}$ on v_3 and $\{d_1\}$ on v_4. Both of R_1 and R_2 fulfill constrains (7) and (8). However, the overall caching benefits produced by R_1 and R_2 are different based on (4) and (7): $benefit(R_1) = 17$ and $benefit(R_2) = 9$. Thus, the below objective function that maximizes the benefit of caching data D over G is included in the COP model to achieve the service provider's optimization objective:

$$\max \sum_{u=1}^{m} \sum_{f=1}^{k} b_u^f \tag{9}$$

The COP above can be solved with Integer Programming problem solvers, such as Gurobi[1] and IBM CPLEX Optimizer[2]. This optimal approach is named CEDC-IP hereafter. Specific caching cost models and network latency models can be easily integrated to this CEDC-IP model.

4.2 Approximation Algorithm

As the COP of CEDC is $\mathcal{NP}$-hard, finding the optimal solution to the COP is intractable in large-scale CEDC scenarios. This section presents an approximation algorithm, named CEDC-A, for finding approximate solutions to large-scale CEDC problem efficiently.

Given $V = \{v_1, ..., v_n\}$, $U = \{u_1, ..., u_m\}$ and $D = \{d_1, ..., d_f\}$, CEDC-A creates an initial candidate list , and then implements an iterative process for each initial candidate. After the above processes, CEDC-A selects the candidate solution with the maximum benefit for a service provider to cache data on selected edge servers. The pseudo code is presented in Algorithm 1, while the functions used in Algorithm 1 are presented in Algorithm 2 and 3.

In this algorithm, the decision to cache one kind of data on an edge server is treated as a $candidate$. The algorithm starts with the initialization in Lines 1-3. The algorithm initiates the set of solution candidates, C, by calculating the benefit increment of each data in each edge server's coverage area (Line 3). Each candidate has two properties, i.e., server id and data. CEDC-A always selects the candidate with maximum benefit value to cache candidate's data on candidate's server based on $c_i \in C$ (Lines 4 to 12). In the end, the solution with the highest benefits will be selected as the result of CEDC-A.

The computational complexity of functions presented in Algorithm 2 and 3 is $O(kn)$. Moreover, $initCandidates()$ can produce at most kn candidates, and the used cache

1. http://www.gurobi.com/
2. https://www.ibm.com/analytics/cplex-optimizer

Algorithm 1 CEDC-A Algorithm

Input: availableSpaces, U, V, D
$C \leftarrow \varnothing$
C = initCandidates()
for each $c_i \in C$ **do**
  as = copy(availableSpaces)
  candidate = getMaxBenefitCandidate(c_i)
  while |as| != 0 & candidate != null **do**
    $c_i \leftarrow c_i \bigcup$ candidate
    $as_{candidate.id} = as_{candidate.id} - 1$
    candidate = getMaxBenefitCandidate(c_i)
  end while
end for
$R = \arg\max_{c_i \in C}$ benefit(c_i);
return R

Algorithm 2 Function initCandidates

initCandidates():
Candidates $\leftarrow \varnothing$
for each $v_i \in V$ **do**
  if $as_i! = 0$ **then**
    for each $d_f \in D$ **do**
      Candidates $\leftarrow$ Candidates $\bigcup \{r_i^f\}$
    end for
  end if
end for
return Candidates

Algorithm 3 Function getMaxBenefitCandidate

getMaxBenefitCandidate(c_i):
candidate = null
for each $v_j \in V \bigcap \neg c_i$ **do**
  if $as_j! = 0$ **then**
    for each $d_f \in D$ **do**
      if benefit($c_i \bigcup$ candidate) $\leq$ benefit($c_i \bigcup r_j^f$) **then**
        candidate = new $Candidate()$
        candidate.id = j
        candidate.data = f
      end if
    end for
  end if
end for
return candidate

spaces on all edge servers are also at most kn. Thus, in the worst-case scenario, the computational complexity of Algorithm 1 is $O(k^2n^2)$.

Now, we prove the approximation ratio of CEDC-A, where it is the ratio of benefits produced by CEDC-A and that produced by optimal solution in the worst cases.

As function $initCandidates()$ provides the candidate list with the first cache decision, we can treat this function as the first iteration in Algorithm 1. Let OPT present the optimal solution (found by CEDC-IP) of the CEDC problem and $benefit(OPT)$ denote the benefit obtained by the OPT caching strategy. Let us assume that the order of cache decisions made by OPT is ascending in terms of cache benefit, and γ is the index of the iteration when the first edge server included in OPT's strategy but not in CEDC-A's strategy R.

Theorem 2. For each iteration $t \leq \gamma$, the following inequality is satisfied:

$$benefit(OPT) - benefit(R_{t-1}) \leq |as| \cdot \Delta b_t \tag{10}$$

where R_{t-1} is the strategy in iteration $t-1$ and Δb_t is the benefit produced by including r_t into strategy R.

Proof Since R_t selects the edge server with the maximum benefit at the t^{th} iteration, for each edge server in OPT but not in R_{t-1}, the benefit increment is at most Δb_t. Since there are a maximum of $|as|$ available cache spaces, the total benefit produced by the edge servers in OPT but not R_{t-1} is at most $|as| \cdot \Delta b_t$. Thus, the above inequality is satisfied. □

Theorem 3. For each iteration $t > \gamma$, the benefit produced by R_t fulfills:

$$benefit(R_t) \geq \left(1 - \left(1 - \frac{1}{|as|}\right)^{t-1}\right) benefit(OPT) \tag{11}$$

Proof Based on Theorem 2, the benefit achieved by S_t can be calculated by (12). Thus, we can easily prove (11) by the inductive proof. The details of the proof process are omitted here.

$$\begin{aligned} &benefit(R_t) = benefit(R_{t-1}) + \Delta b_t \\ &\geq benefit(R_{t-1}) + \frac{benefit(OPT) - benefit(R_{t-1})}{|as|} \\ &= \left(1 - \frac{1}{|as|}\right) benefit(R_{t-1}) + \frac{1}{|as|} benefit(OPT) \end{aligned} \tag{12}$$

□

Theorem 4. The approximation ratio of CEDC-A is $\frac{2}{3}\left(1 - \frac{1}{e}\right)$.

Proof Based on the cache space constraint(8), the algorithm can have at most $|as|$ iterations. Thus, when $t = |as| + 1$, we can obtain (13) based on Theorem 3:

$$benefit(R_{|as|+1}) \geq \left(1 - \frac{1}{e}\right) benefit(OPT) \tag{13}$$

However, this exceeds the constraint (8). As discussed above, the first decision in function $initCandidates()$ is treated as the first iteration.

When $|as| = 1$, the solution obtained by CEDC-A is the same as OPT. In this case, the approximation ratio is 1. When $|as| \geq 2$, the benefit increment by $\Delta b_{|as|+1}$ is less than or equal to $\frac{1}{2} benefit(R_{|as|})$. Thus, there is

$$\begin{aligned} &benefit(R_{|as|}) = benefit(R_{|as|+1}) - \Delta b_{|as|+1} \\ &\geq \left(1 - \frac{1}{e}\right) benefit(OPT) - \frac{1}{2} benefit(R_{|as|}) \end{aligned} \tag{14}$$

Based on (14), the approximation ratio of CEDC-A is $\frac{2}{3}\left(1 - \frac{1}{e}\right)$. □

5 Experimental Evaluation

We experimentally evaluate the performance of CEDC-IP and CEDC-A. All of the experiments were conducted on a Windows-10 machine equipped with Intel Core i7-8550 processor (8 CPUs, 1.80GHz) and 8GB RAM. The COP discussed in Section 4 is solved with IBM's CPLEX Optimizer.

5.1 Baseline Approaches

In the experiments, we evaluate the performance of CEDC-IP and CEDC-A against four representative approaches:

- *Request-based Collaborative Caching (RCC)*: This approach aims to minimize the overall caching cost incurred by serving the most users. This algorithm originated from the collaborative data caching approach in [31], which is implemented in the content delivery network.
- *NCCEDC-IP*: This approach finds the non-collaborative data caching optimal solution, which does not allow collaboration among edge servers. Thus, app users can only access data from their local edge servers. Other than that, it is formulated and implemented in a way similar to CEDC-IP.
- *Greedy-Connection (GC)*: This approach always selects the edge server that has the most neighbour edge servers to cache data under Constraint family (8).
- *Random*: This approach always selects the edge server randomly to cache data under Constraint family (8).

5.2 Experiment Settings

5.2.1 Experiment data

The experiments are conducted on a real-world data set, named EUA data set[3] [3], which is widely used in research on edge computing [23], [32], [33]. This data set contains the geographical locations of 125 base stations and 816 mobile users in the Melbourne CBD area. The links between edge servers are randomly generated to ensure the edge servers constitute a connected graph. In the experiments, L is set to 1 hop. The available cache spaces on each edge server are generated following a normal distribution $X \sim \mathcal{N}(\mu, \sigma^2)$, where μ is half of the number of maximum cache spaces and σ is 1.

5.2.2 Experimenting parameters

To simulate different CEDC scenarios, three parameters are varied in the experiments.

- Number of edge servers $|\mathbf{V}|$. This parameter impacts the size of graph G and varies from 4 to 14 in steps of 2.
- Number of maximum cache spaces **MS**. This parameter impacts the available cache spaces on edge edge server and varies from 2 to 10 in steps of 2.
- Number of data $|\mathbf{D}|$. The total number of data to be cached over G. This parameter varies from 3 to 8 in steps of 1.

3. https://github.com/swinedge/eua-dataset

TABLE 2
Parameter Settings

	$\|\mathbf{V}\|$	**MS**	$\|\mathbf{D}\|$
Set #1.1	4, 6, 8, 10, 12, 14	4	6
Set #1.2	10	2, 4, 6, 8, 10	6
Set #1.3	10	4	3, 4, 5, 6, 7, 8
Set #2.1	20, 30, 40, 50, 60, 70	10	15
Set #2.2	50	2, 4, 6, 8, 10	15
Set #2.3	50	10	5, 10, 15, 20, 25, 30

5.2.3 Performance Metrics

In this experiments, three metrics are employed to evaluate the performance of the approaches, one for effectiveness and one for efficiency:

- **Benefit per cache cost (**bpc**)**, measured by the total number of hops reduced divided by the data cache cost, higher the better.
- **Served request ratio per cache cost (**$SRRpc$**)**, measured by the ratio of the served data requests divided by the reserved data cache cost, higher the better.
- **Computational overhead (**$time$**)**, measured by the time taken to find the solution, the lower the better.

Table 2 summarizes the parameter settings. There are two main sets of experiments, Set #1 for small-scale experiments and Set #2 for large-scale experiments. All six approaches are implemented in Set #1, while CEDC-IP and NCCEDC-IP are not implemented in Set #2, because they cannot find an optimal solution to the $\mathcal{NP}$-hard CEDC problem in Set #2 within a reasonable time. Every time the value of a parameter varies, the experiment is repeated for 100 times and the averaged results are reported. To isolate the impact of the number of app users, we randomly select 40 covered app users and 200 covered app users from the data set in each run of the experiments in Set #1 and Set #2, respectively.

5.3 Experimental Results

Table 3 summarizes the results of experiment Set #1 and Set #2 where the best and second-best performances are marked as dark and light grey, respectively, in each column.

5.3.1 Impact of number of edge servers

The results of experiment Set #1.1 are presented in Fig. 3(a) and Fig. 4(a). It shows that the **benefit per cache cost and the served request ratio per cache cost achieved by CEDC-IP and CEDC-A outperform the other approaches significantly**. In Fig. 3(a), when the number of edge servers increases from 4 to 14, the benefit per cache cost achieved by all six approaches decreases, from 13.16 to 4.34 by 67.02% for CEDC-IP, from 12.57 to 4.20 by 66.59% for CEDC-A, from 10.99 to 3.71 by 66.24% for RCC, from 10.35 to 3.59 by 65.31% for NCCEDC-IP, from 9.54 to 3.34 by 64.99% for GC and from 7.89 to 3.06 by 61.21% for Random. The advantages of CEDC-IP are 3.77% over CEDC-A, 18.47% over RCC, 25.53% over NC, 38.12% over GC and 62.76% over Random on average. It is also shown in Fig. 3(a) that **the advantages of CEDC-IP and CEDC-A increase with the increase in**

TABLE 3
Average performance results

Approaches	Set #1.1			Set #1.2			Set #1.3		
	bpc	$SRRpc$ (%)	$Time$	bpc	$SRRpc$ (%)	$Time$	bpc	$SRRpc$ (%)	$Time$
CEDC-IP	7.4817	7.5450	1.6518	4.7040	4.6500	0.6380	5.9150	5.7500	10.6384
CEDC-A	7.2100	7.6933	0.0002	4.5840	4.7000	0.0003	5.7183	5.7233	0.0002
RCC	6.3150	6.0817	0.0001	4.1880	3.9660	0.0002	5.2250	5.2250	0.0001
NCCEDC-IP	5.9600	6.7217	0.4198	3.8820	4.1940	0.3318	4.9483	5.2517	1.1959
GC	5.4167	5.0900	0.0002	3.6700	3.4820	0.0002	4.5917	4.3300	0.0001
Random	4.5967	4.4217	0.0001	3.3580	3.1940	0.0002	4.2467	4.0433	0.0001
Approaches	Set #2.1			Set #2.2			Set #2.3		
	bpc	$SRRpc$ (%)	$Time$	bpc	$SRRpc$ (%)	$Time$	bpc	$SRRpc$ (%)	$Time$
CEDC-A	3.1288	0.5698	0.0571	3.7530	0.7534	0.0346	2.9696	0.5194	0.0833
RCC	2.7973	0.5025	0.0104	3.1547	0.5889	0.0055	2.6645	0.4590	0.0146
GC	2.5341	0.4527	0.0053	2.4462	0.4632	0.0024	2.6064	0.4550	0.0051
Random	2.3308	0.4202	0.0031	2.1924	0.4166	0.0021	2.4447	0.4298	0.0047

the number of edge servers. Fig. 4(a) shows that the served request ratio per cache cost of CEDC-A is almost the same as that of CEDC-IP. Both CEDC-IP and CEDC-A achieve much higher average served request ratio per cache than the other approaches, i.e., 0.0755 (CEDC-IP) and 0.0769 (CEDC-A) versus 0.0608 (RCC), 0.0672 (NCCEDC-IP), 0.0509 (GC) and 0.0442 (Random).

The experimental results of Set #2.1 are depicted in Fig. 5(a) and Fig. 6(a). Overall, **CEDC-A achieves the highest benefit and served request ratio per cache cost on average**. As shown in Fig. 5(a), CEDC-A outperforms RCC, GC and Random in benefit per cache cost, by 11.85%, 23.47% and 34.24%, respectively. In terms of served request ratio per cache cost, Fig. 6(a) demonstrates that the advantages of CEDC-A are 13.39% over RCC, 25.87% over GC and 35.60% over Random.

With the increase in the number of edge servers (4 to 14 in Set #1.1 and 20 to 70 in Set #2.1), the benefit and served request ratio per cache cost achieved by all approaches decrease. The reason is that, when the number of users is fixed, the maximum benefits and the total number of requests are fixed. Accordingly, the benefit and served request ratio per cache cost decrease when more cache spaces are hired by the service provider with the fixed number of users.

5.3.2 Impact of maximum cache spaces

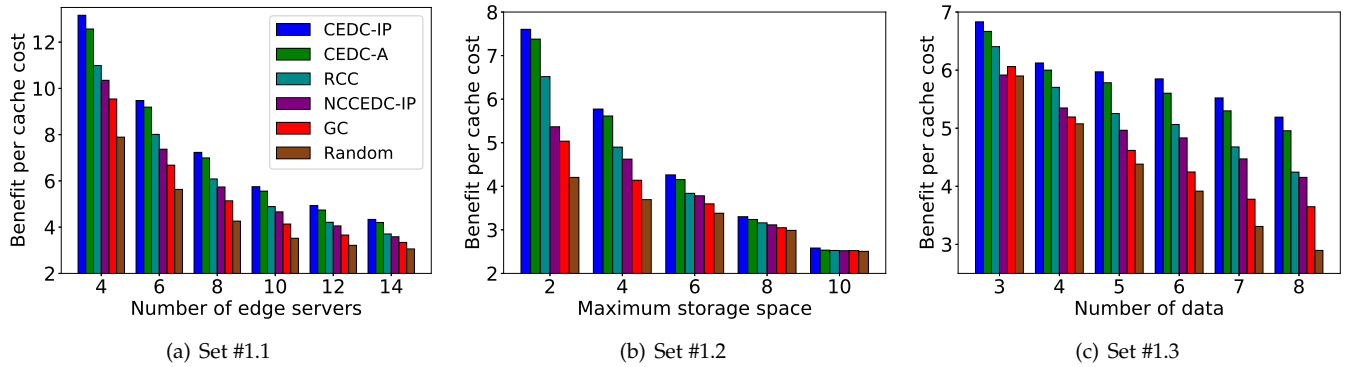
The impacts of the maximum cache spaces on the approaches are shown in Fig. 3(b), Fig. 4(b), Fig. 5(b) and Fig. 6(b). In experiment Set #1.2, **CEDC-IP achieves the highest benefit per cache cost again**, followed by CEDC-A. In terms of the served request ratio per cache cost, CEDC-IP and CEDC-A also outperform the other approaches significantly. In Fig. 3(b), both CEDC-IP and CEDC-A outperform RCC, NCCEDC-IP, GC and Random, by an average of 12.32% and 9.46%, 21.17% and 18.08%, 28.17% and 24.90%, 40.08% and 36.51%, respectively. Moreover, Fig. 4(b) shows that, for the served request ratio per cache cost, the average advantages of CEDC-A are 1.07% over CEDC-IP, 18.61% over RCC, 12.16% over NCCEDC-IP, 35.09% over GC and 47.28% over Random. With the increase in the maximum cache spaces from 2 to 10 in Set #1.2, the cache cost increases. As the number of users is fixed at 40, the maximum total benefits are fixed as well. Thus, the benefit and served request ratio per cache cost decrease for all approaches with the increase in the maximum cache spaces, while the trends and their trends are similar in Fig. 5(b) and Fig. 6(b). In Set #2.2, **CEDC-A achieves the highest benefit and served request ratio per cache cost again**. The advantages of CEDC-A are 18.97% and 27.93% over RCC, 53.42% and 62.65% over GC, 71.18% and 80.84% over Random, in terms of benefit per cache cost and served request ratio per cache cost, respectively.

Based on the results shown in the above figures, **the advantages of CEDC-IP and CEDC-A are more even more significant with fewer available cache spaces**. This indicates that CEDC-IP and CEDC-A are particularly suitable for the edge computing environment. This is because it is a highly competitive environment where the resources on edge servers available for data caching are constrained.

5.3.3 Impact of number of data

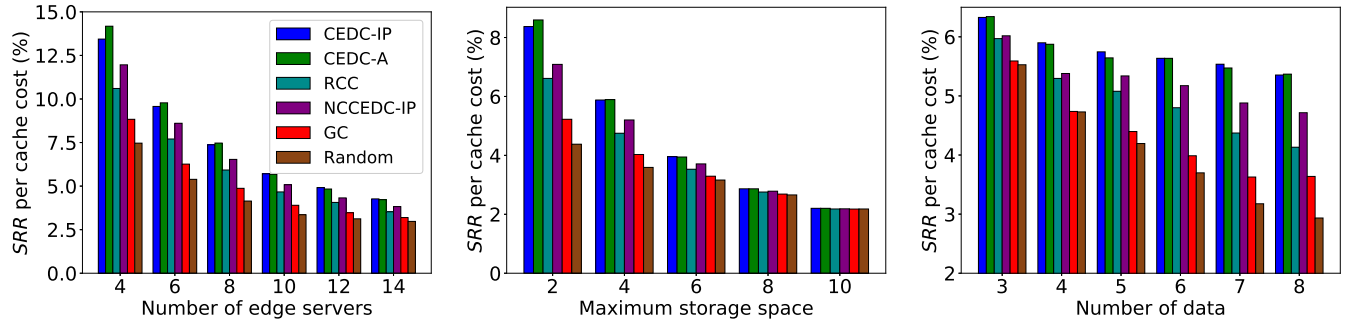
Fig. 3(c) and Fig. 4(c) depict the results obtained in Set #1.3 where the number of data to be cached varies. In terms of the benefit per cache cost and the served request ratio per cache cost, **CEDC-IP and CEDC-A outperform the other approaches with significant margins**. In terms of the benefit per cache cost, as demonstrated in Fig. 3(c), the average advantages of CEDC-IP are 3.44% over CEDC-A, 13.21% over RCC, 19.54% over NCCEDC-IP, 28.82% over GC and 39.29% over Random. In Fig. 5(c) and Fig. 6(c), **the advantage of CEDC-A is significant over the other three approaches**. On average, CEDC-A outperforms RCC by 11.45% and 13.16%, GC by 13.93% and 14.15%, Random by 21.36% and 20.85%, in the benefit per cache cost and served request ratio per cache cost, respectively.

Those results also demonstrate that **the performance of both CEDC-IP and CEDC-A decreases much slower than the other approaches** when the number of data to be cached increases. That is, they scale better with the number of data to be cached.



(a) Set #1.1 (b) Set #1.2 (c) Set #1.3

Fig. 3. Benefit per cache cost vs. parameters in Set #1



(a) Set #1.1 (b) Set #1.2 (c) Set #1.3

Fig. 4. Ratio of served request per cache cost vs. parameters in Set #1

5.3.4 Efficiency

The efficiency of an approach is evaluated by its computational overhead measured by the average time (in seconds) taken to find a solution to the CEDC problem. The results are summarized in Table 3. As shown, **CEDC-IP is much more computationally expensive than all the other approaches** in Set #1. It takes 10.6384 seconds on average in Set # 1.3. This validates the $\mathcal{NP}$-hardness of the CEDC problem - excessive computational overheads are inevitable for finding the optimal solution to large-scale CEDC problems. The other approaches, including CEDC-A, RCC, GC and random, can find a solution almost immediately in Set #1. In Set #2, the execution time increases for all approaches. In Set #2, Random takes the least time to between 0.0021 and 0.0047 seconds on average, while CEDC-A takes the most time, between 0.0346 seconds and 0.0833 seconds. CEDC-A takes more time than RCC, GC and Random. This is the performance price to pay for CEDC-A's effectiveness advantage over these approaches as shown and discussed above. With the significant advantages of CEDC-A, especially where there are more edge servers, more data, and less storage space, it is worth applying CEDC-A in the real deployment.

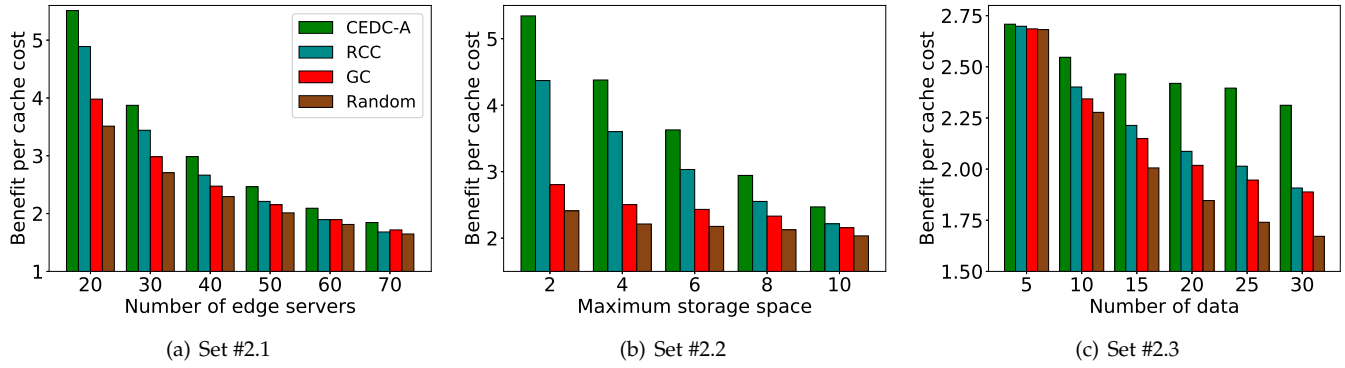
5.3.5 Conclusion

Overall, **CEDC-IP and CEDC-A outperform GC and Random significantly and consistently** in formulating cost-effective data caching strategies in different CEDC scenarios. As an approximation algorithm, the effectiveness of CEDC-A is 96.37% to 97.54% on average as high as CEDC-IP as shown by the results of experiment Set #1. In Set #2, CEDC-A outperforms all other approaches significantly at the price of slightly higher computational overheads. Thus, CEDC-IP is suitable for solving small-size CEDC problems. To solve large-scale CEDC problems, CEDC-A is more practical for its high effectiveness and efficiency in finding near-optimal solutions.

5.4 Threats to Validity

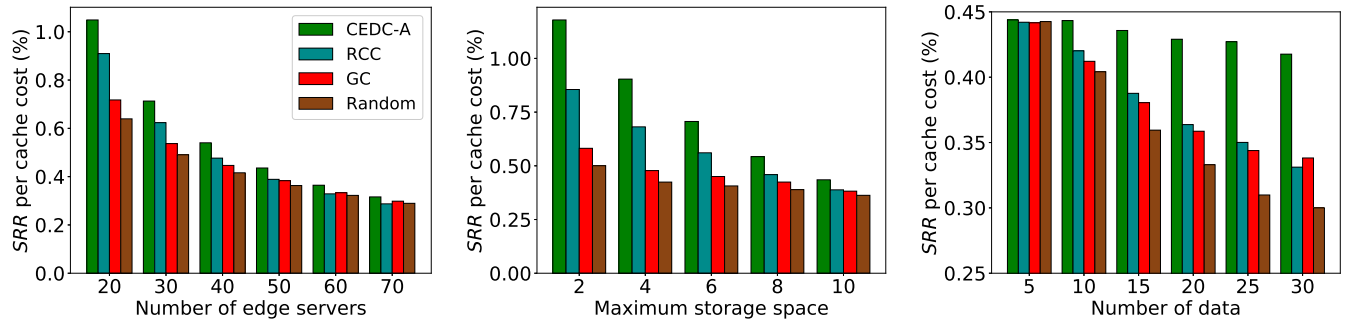
5.4.1 Construct Validity

The main threats to construct validity are the randomly generated graphs and and the four approaches used for comparison in the experiments. The graphs randomly generated in the experiments may not represent all the edge server networks in the real-world edge computing environment. To minimize this threat, the experiment is repeated for 100 times - a total of 100 graphs are randomly generated - every time the value of a setting parameter varies in the experiments. In this way, a large number of edge server networks are simulated in the experiments to provide comprehensive guidelines on the performance of our approaches in real-world scenarios. The comparison to RCC, NCCEDC-IP, GC and Random, may not suffice to comprehensively evaluate CEDC-IP and CEDC-A. To minimize this threat, three experimental parameters are varied in the experiments to simulate different CEDC scenarios. In this way, we could evaluate CEDC-IP and CEDC-A by not only the comparison with the four approaches but also by the impacts of the three varying setting parameters.



(a) Set #2.1 (b) Set #2.2 (c) Set #2.3

Fig. 5. Benefit per cache cost vs. parameters in Set #1



(a) Set #2.1 (b) Set #2.2 (c) Set #2.3

Fig. 6. Ratio of served request per cache cost vs. parameters in Set #1

5.4.2 *Threats to Internal Validity*

The main threat to the internal validity is whether the experiment setting favors CEDC-IP and CEDC-A over other approaches. To minimize this threat, we varied three parameters to simulate various CEDC scenarios so that the performance of different approaches could be compared comprehensively and fairly. Moreover, all experiments were repeated for 100 times and the results were averaged. This way, the results of experiments were somehow set up in a biased manner could be neutralized.

5.4.3 *External Validity*

The main threat here is whether CEDC-IP and CEDC-A are also applicable for other edge computing scenarios. To address this, we formulate the approaches and measure the performance in a more generic way: evaluating the effectiveness by using the number of data replicas and the number of hops for cost and benefit. This way, the exact latency model and cost model can be easily integrated into our approaches. Moreover, the widely-used real-world data set is used to evaluate all approaches. Thus, the representativeness and comprehensiveness of the evaluation are ensured, and this threat is reduced.

6 Related Work

Data caching have been extensively investigated in 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, many data caching problems are investigated in conventional distributed computing environments, i.e. web caching [10], content delivery network (CDN) [34], etc. Banerjee et al. [35] introduced a content placement strategy Greedy Caching for information-centric network based on data popularity. With popular contents cached, the approach considered the cache miss rate to decide the cached contents on the core server. Uddin et al. formulated two caching strategies for data publish-subscribe systems, including eviction-based and time-to-live-based strategy to address the space and time issues [36]. In [37], the authors focused on balancing the trade-off between latency and cost in the content-centric network, and 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. [38] proposed CloudCache, a method for managing cache, to fulfill the caching requirement of the workload and minimize cache wear-out. In [39], 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 [40]

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 [41], 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

From the perspectives of network topology and infrastructure deployment, edge computing is an extension of cloud computing with distributed computing capacities and services at the edge of the network. App users in various domains can benefit from the advantages of edge computing, e.g., interactive gaming, real-time navigation, augmented reality [2]. Offering many unique advantages, edge computing also raises various new research challenges from the service provider's perspective, e.g., edge user allocation [3], [23], edge data distribution [8], edge application deployment [33], [42], edge data integrity [43], etc.

Some researchers start to investigate data caching problems in edge computing recently. Due to the unique constraints of edge computing discussed in Section 2, the data caching strategies from cloud computing and conventional distributed computing are not applicable in the edge computing environment. Thus, new ideas and approaches were proposed in the recent years. The authors of [44] considered the costs produced during delivery and introduced an auction mechanism to find an optimal data caching solution. In [45], the authors proposed Agar, a caching system, by implementing erasure-code into data caching techniques. Considering the data popularity and network latency, Agar could find the optimal solution to cache data chunk by dynamic programming. In [46], Drolia et al. provided Cachier, a caching system for minimizing the latency of data retrieval. A coordinating mechanism was applied for balancing the work loads between edge servers and the remote cloud server. The authors of [47] investigated the caching approach over the 5G network. Both in-network caches and edge caches are involved to satisfy the time-sensitive transmission by ensuring the low latency. Similarly, Zhang et al. [48] proposed a new edge cache architecture by including caches on smart vehicles into the network caches. This approach significantly improved the resource utility and the effectiveness of this architecture. Poularakis et al. [17] studied the joint optimization of service placement and request routing in the edge computing environment. The authors of [49] proposed a hierarchical caching mechanism in the edge computing environment with consideration of wireless communication. They aimed to maximize the hitting rate by caching data in different layers including routers, base stations and mobile devices. In [50], the authors studied the budgeted service placement problem in the edge computing environment. They proposed a Lyapunov-based algorithm for minimizing the overall data retrieval latency. Deng et al. [51] also investigated the service deployment problem in the edge computing environment but tried to minimize the overall cost rather than latency. They provided a primal-dual algorithm named IDA4ReE to solve this problem under a resource constraint and performance requirement. However, these studies did not fully consider the unique constraints of edge computing, i.e. the server capacity constraint, the server coverage constraint and the server adjacency constraint.

Edge computing inherits the pay-as-you-go pricing model from cloud computing, which allows service providers to hire storage resources on edge servers from edge infrastructure providers to cache app data for their own users. Thus, both the benefit produced and the cost incurred by data caching for service providers is critical to the success of edge computing because, after all, service providers are the main customers in the edge computing environment. However, the above studies tackle the data caching problem from either the mobile network operator's or the app user's perspective. In [15], the data caching problem is firstly tackled from the service provider's perspective in the edge computing environment with the aim to cover all the app users in an area. The approach proposed in [15] can only cache data individually and does not consider the constrained cache spaces on edge servers or edge servers' ability to communicate. In this paper, we considered the these practical issues and converted the EDC problem into the constrained edge data caching (CEDC) problem. We solved the CEDC problem in a generic manner to maximize the data caching benefit with finite cache spaces on edge servers, considering the unique server capacity constraint, server coverage constraint and the server adjacency constraint in the edge computing environment.

7 Conclusion

In this paper, we formulated the new constrained edge data caching (CEDC) problem in the edge computing environment from the service provider's perspective. We proved that the CEDC problem is $\mathcal{NP}$-hard. To solve this problem, we proposed an optimal approach named CEDC-IP based on integer programming to maximum the data caching benefit measured by the overall reduction in app users' data retrieval latency with limited cache resources. As the CEDC problem is $\mathcal{NP}$-hard, we also provided an approximation approach named CEDC-A for finding approximate solutions to large-scale CEDC problems efficiently. Extensive experiments were conducted on a widely-used real-world data set to evaluate the performance of the proposed approaches. The results showed that our approaches significantly outperformed the state-of-the-art approaches in various CEDC scenarios. In our future work, we will consider the mobility of app users, real-time cache updating scenarios, security constraints and data regulation.

Acknowledgement

This research is partially funded by Australian Research Council Discovery Projects No. DP180100212, DP200102491 and Laureate Fellowship FL190100035. Qiang He is the corresponding author of this paper.

REFERENCES

[1] A. Osseiran, V. Braun, T. Hidekazu, P. Marsch, H. Schotten, H. Tullberg, M. A. Uusitalo, and M. Schellman, "The foundation of the mobile and wireless communications system for 2020 and beyond: Challenges, enablers and technology solutions," in *IEEE 77th Vehicular Technology Conference (VTC2013-Spring)*, 2013, pp. 1–5.

[2] M. Yannuzzi, F. van Lingen, A. Jain, O. L. Parellada, M. M. Flores, D. Carrera, J. L. Pérez, D. Montero, P. Chacin, A. Corsaro *et al.*, "A new era for cities with fog computing," *IEEE Internet Computing*, vol. 21, no. 2, pp. 54–67, 2017.

[3] P. Lai, Q. He, M. Abdelrazek, F. Chen, J. Hosking, J. Grundy, and Y. Yang, "Optimal edge user allocation in edge computing with variable sized vector bin packing," in *International Conference on Service-Oriented Computing*, 2018, pp. 230–245.

[4] G. Cui, Q. He, X. Xia, P. Lai, F. Chen, T. Gu, and Y. Yang, "Interference-aware saas user allocation game for edge computing," *IEEE Transactions on Cloud Computing*, 2020.

[5] 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, 2017.

[6] 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.

[7] S. Wu, C. Niu, J. Rao, H. Jin, and X. Dai, "Container-based cloud platform for mobile computation offloading," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2017, pp. 123–132.

[8] 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.

[9] K. Elhardt and R. Bayer, "A database cache for high performance and fast restart in database systems," *ACM Transactions on Database Systems (TODS)*, vol. 9, no. 4, pp. 503–525, 1984.

[10] S. Podlipnig and L. Böszörmenyi, "A survey of web cache replacement strategies," *ACM Computing Surveys (CSUR)*, vol. 35, no. 4, pp. 374–398, 2003.

[11] C.-Y. Wang, S. H. Lim, and M. Gastpar, "Information-theoretic caching: Sequential coding for computing," *IEEE Transactions on Information Theory*, vol. 62, no. 11, pp. 6393–6406, 2016.

[12] N. Karamchandani, U. Niesen, M. A. Maddah-Ali, and S. N. Diggavi, "Hierarchical coded caching," *IEEE Transactions on Information Theory*, vol. 62, no. 6, pp. 3212–3229, 2016.

[13] M. Dehghan, B. Jiang, A. Seetharam, T. He, T. Salonidis, J. Kurose, D. Towsley, and R. Sitaraman, "On the complexity of optimal request routing and content caching in heterogeneous cache networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 25, no. 3, pp. 1635–1648, 2017.

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

[15] X. Xia, F. Chen, Q. He, G. Cui, P. Lai, M. Abdelrazek, J. Grundy, and H. Jin, "Graph-based optimal data caching in edge computing," in *International Conference on Service-Oriented Computing*. Springer, 2019, pp. 477–493.

[16] ——, "Graph-based data caching optimization for edge computing," *Future generation computer systems*, vol. 113, pp. 228–239, 2020.

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

[18] H. Zhao, S. Deng, Z. Liu, J. Yin, and S. Dustdar, "Distributed redundancy scheduling for microservice-based applications at the edge," *IEEE Transactions on Services Computing*, 2020.

[19] S. Deng, Z. Xiang, P. Zhao, J. Taheri, H. Gao, J. Yin, and A. Y. Zomaya, "Dynamical resource allocation in edge for trustable internet-of-things systems: A reinforcement learning method," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 6103–6113, 2020.

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

[21] H. Guo and J. Liu, "Collaborative computation offloading for multi-access edge computing over fiber-wireless networks," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 5, pp. 4514–4526, 2018.

[22] 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.

[23] Q. He, G. Cui, X. Zhang, F. Chen, S. Deng, H. Jin, Y. Li, and Y. Yang, "A game-theoretical approach for user allocation in edge computing environment," *IEEE Transactions on Parallel and Distributed Systems*, 2019.

[24] P. Lai, Q. He, G. Cui, X. Xia, M. Abdelrazek, F. Chen, J. Hosking, J. Grundy, and Y. Yang, "Edge user allocation with dynamic quality of service," in *International Conference on Service-Oriented Computing*. Springer, 2019, pp. 86–101.

[25] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2016.

[26] W. Chen, D. Wang, and K. Li, "Multi-user multi-task computation offloading in green mobile edge cloud computing," *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 726–738, 2018.

[27] 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 Transactions on Wireless Communications*, vol. 16, no. 8, pp. 4924–4938, 2017.

[28] F. Wang, J. Xu, X. Wang, and S. Cui, "Joint offloading and computing optimization in wireless powered mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 17, no. 3, pp. 1784–1797, 2018.

[29] J. L. D. Neto, S.-y. Yu, D. F. Macedo, J. M. S. Nogueira, R. Langar, and S. Secci, "Uloof: a user level online offloading framework for mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 17, no. 11, pp. 266–2674, 2018.

[30] E. Hazan, S. Safra, and O. Schwartz, "On the complexity of approximating k-set packing," *computational complexity*, vol. 15, no. 1, pp. 20–39, 2006.

[31] 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.

[32] X. Xia, F. Chen, G. Cui, M. Abdelrazek, J. Grundy, H. Jin, and Q. He, "Budgeted data caching based on k-median in mobile edge computing," in *27th IEEE International Conference on Web Services*. IEEE, 2020, pp. 197–206.

[33] B. Li, Q. He, G. Cui, X. Xia, F. Chen, H. Jin, and Y. Yang, "Read: Robustness-oriented edge application deployment in edge computing environment," *IEEE Transactions on Services Computing*, 2020.

[34] D. S. Berger, R. K. Sitaraman, and M. Harchol-Balter, "Adaptsize: Orchestrating the hot object memory cache in a content delivery network," in *14th USENIX Symposium on Networked Systems Design and Implementation*, 2017, pp. 483–498.

[35] B. Banerjee, A. Kulkarni, and A. Seetharam, "Greedy caching: An optimized content placement strategy for information-centric networks," *Computer Networks*, vol. 140, pp. 78–91, 2018.

[36] M. Y. S. Uddin and N. Venkatasubramanian, "Edge caching for enriched notifications delivery in big active data," in *38th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 696–705.

[37] Y. Li, H. Xie, Y. Wen, and Z.-L. Zhang, "Coordinating in-network caching in content-centric networks: Model and analysis," in *33rd IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2013, pp. 62–72.

[38] D. Arteaga, J. Cabrera, J. Xu, S. Sundararaman, and M. Zhao, "Cloudcache: On-demand flash cache management for cloud computing," in *14th USENIX Conference on File and Storage Technologies(FAST)*, 2016, pp. 355–369.

[39] K. Ma, B. Yang, Z. Yang, and Z. Yu, "Segment access-aware dynamic semantic cache in cloud computing environment," *Journal of Parallel and Distributed Computing*, vol. 110, pp. 42–51, 2017.

[40] 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 and Computation: Practice and Experience*, vol. 29, no. 11, pp. 1–13, 2017.

[41] S. Tamoor-ul Hassan, S. Samarakoon, M. Bennis, M. Latva-Aho, and C. S. Hong, "Learning-based caching in cloud-aided wireless networks," *IEEE Communications Letters*, vol. 22, no. 1, pp. 137–140, 2018.

[42] F. Chen, J. Zhou, X. Xia, H. Jin, and Q. He, "Optimal application deployment in mobile edge computing environment," in *13th IEEE International Conference on Cloud Computing*. IEEE, 2020, pp. 184–192.

[43] B. Li, Q. He, F. Chen, H. Jin, Y. Xiang, and Y. Yang, "Auditing cache data integrity in the edge computing environment," *IEEE Transactions on Parallel and Distributed Systems*, 2020.

[44] 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 (ICDCS)*, 2018, pp. 388–399.

[45] R. Halalai, P. Felber, A.-M. Kermarrec, and F. Taïani, "Agar: A caching system for erasure-coded data," in *37th IEEE International Conference onDistributed Computing Systems (ICDCS)*, 2017, pp. 23–33.

[46] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, "Cachier: Edge-caching for recognition applications," in *37th IEEE International Conference onDistributed Computing Systems (ICDCS)*, 2017, pp. 276–286.

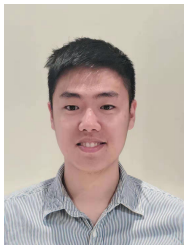
[47] X. Zhang and Q. Zhu, "Hierarchical caching for statistical qos guaranteed multimedia transmissions over 5g edge computing mobile wireless networks," *IEEE Wireless Communications*, vol. 25, no. 3, pp. 12–20, 2018.

[48] 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.

[49] X. Zhang and Q. Zhu, "Collaborative hierarchical caching over 5g edge computing mobile wireless networks," in *2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–6.

[50] J. Zhou, J. Fan, J. Wang, and J. Jia, "Dynamic service deployment for budget-constrained mobile edge computing," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 24, pp. 1–16, 2019.

[51] 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.



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.

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 of the IEEE Transactions on Software Engineering, the 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/.

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

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/.