Future Generation Computer Systems 113 (2020) 228–239

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Graph-based data caching optimization for edge computing

Xiaoyu Xia [a], Feifei Chen [a], Qiang He [b,*], Guangming Cui [b], Phu Lai [b], Mohamed Abdelrazek [a], John Grundy [c], Hai Jin [d]

[a] Deakin University, Geelong, Australia
[b] Swinburne University of Technology, Hawthorn, Australia
[c] Monash University, Clayton, Australia
[d] Huazhong University of Science and Technology, Wuhan, China

ARTICLE INFO

Article history:
Received 12 December 2019
Received in revised form 29 May 2020
Accepted 6 July 2020
Available online 13 July 2020

Keywords:
Optimization
Edge computing
Edge data caching

ABSTRACT

Edge computing has emerged as a new computing paradigm that allows computation and storage resources in the cloud to be distributed to edge servers. Those edge servers are deployed at base stations to provide nearby users with high-quality services. Thus, data caching is extremely important in ensuring low latency for service delivery in the edge computing environment. To minimize the data caching cost and maximize the reduction in service latency, we formulate this *Edge Data Caching (EDC)* problem as a constrained optimization problem in this paper. We prove the $\mathcal{NP}$-completeness of this EDC problem and provide an optimal solution named IPEDC to solve this problem based on Integer Programming. Then, we propose an approximation algorithm named AEDC to find approximate solutions with a limited bound. We conduct intensive experiments on a real-world data set and a synthesized data set to evaluate our approaches. Our results demonstrate that IPEDC and AEDC significantly outperform the four representative baseline approaches.

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. To address this issue, cloud data caching was introduced to allow users to access high-demand data, and utilize cloud computing's configurable and powerful capacities [2].

However, with the growing demand for high-quality data and lower latency, the cloud model falls short of those requirements, due to the usually unpredictable network latency and expensive bandwidth [3]. As an evident weakness of the cloud computing paradigm, it is extremely hard to reduce delay at the wide-area network scale. *Edge computing* is proposed as a new computing paradigm 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 [4]. This way, mobile and IoT app vendors (together referred to as *app vendors* hereafter) can rent computation and storage resources in the edge computing environment to host their services and cache their data on edge servers. This way, their app users can access those services or data with low latency [5]. Edge computing is also a key technology in the 5G mobile network [6].

As edge servers become the entry (first access) point for most mobile and IoT devices, the rapidly increasing internet traffic data will be transmitted through those edge servers. Caching data, especially popular data, on edge servers will significantly reduce the transmission latency in users' data retrieval. This is particularly critical for latency-sensitive applications, e.g., smart city deployment, real-time traffic navigation systems, augmented reality applications, etc. As popular data account for a large portion of internet traffic, caching popular data on edge servers can also reduce the pressure on the backbone network. It is predicted that mobile traffic will be reduced by 35% through caching data on edge servers. From an app vendor's perspective, caching data on edge servers can decrease the volume of data transferred in and out of the cloud to save the transfer costs considerably.

Given a piece of popular data, the straightforward solution is to cache it on every edge server in a specific geographic area. This way, the latency in all app users' data retrieval can be minimized. However, edge computing, as an extension of cloud computing, also employs the pay-as-you-go pricing model. App vendors need to hire storage resources on edge servers for caching this data. This solution produces a huge *caching cost*, and thus is impractical

* Corresponding author.
E-mail addresses: xiaoyu.xia@deakin.edu.au (X. Xia), feifei.chen@deakin.edu.au (F. Chen), qhe@swin.edu.au (Q. He), gcui@swin.edu.au (G. Cui), tlai@swin.edu.au (P. Lai), mohamed.abdelrazek@deakin.edu.au (M. Abdelrazek), john.grundy@monash.edu (J. Grundy), hjin@hust.edu.cn (H. Jin).

https://doi.org/10.1016/j.future.2020.07.016
0167-739X/

for most, if not all, app vendors. Thus, app vendors must find a data caching strategy to guarantee that all their app users can access the data from nearby edge servers with low latency while minimizing the cost of hired cache spaces on those edge servers. In this paper, this data caching problem in edge computing is referred to as the *edge data caching* (EDC) problem. Our previous work [7] is the first attempt to investigate this EDC problem from app vendors' perspective. This paper significantly extends [7] by providing a more thorough theoretical analysis and a new approximation algorithm to solve the EDC problem efficiently within trusted performance bounds.

In this paper, our major contributions are as follows:

- We formulate the EDC problem from the app vendors' perspective, then prove its $\mathcal{NP}$-completeness.
- We develop an optimal approach named IPEDC for finding optimal solutions to EDC problems with the Integer Programming technique.
- We develop an approximation approach named AEDC for finding near-optimal solutions to EDC problems in large-scale scenarios efficiently, and analyze its theoretical approximation ratio.
- We conduct extensive experiments on both a real-world data set and a synthesized data set to evaluate the proposed approaches against four representative approaches.

The rest of the paper is organized as follows. Section 2 presents an example to illustrate and motivate the EDC problem. Section 3 formulates the EDC problem and proves its $\mathcal{NP}$-completeness. Section 4 presents and analyzes our optimal approach and approximation approach for finding solutions to EDC problems. Section 5 experimentally evaluates the proposed approaches. Section 6 reviews the related work. Section 7 concludes this paper and points out our key future work.

2. Motivating example

Video data is a typical type of data to be cached on edge servers. According to Cisco's report, mobile video data currently accounts for more than half of the world's mobile data traffic and it will further increase by 78% by 2021 [8]. Currently, most app vendors such as YouTube store their videos on cloud servers. When a video becomes viral over the Internet – which can result in hundreds of thousands if not millions of requests – very large numbers of users will send their requests to the cloud server for this video simultaneously. This creates tremendous traffic load on the network and increases data retrieval latency. Caching these videos on selected edge servers can effectively solve these problems.

However, in edge computing, three unique constraints differentiate the EDC problem from the data caching problems in the cloud computing environment and conventional networks:

- *Server adjacency constraint*: In the edge computing environment, edge servers can communicate their neighbor edge servers via high-speed links [9]. Thus, connected edge servers can share their computation and storage resources. This way, the edge server network can be treated as a graph where edge servers are represented by nodes and the links between edge servers are represented by edges.
- *Server coverage constraint*: To avoid any blank coverage areas in a specific geographic area, the coverage areas of nearby edge servers often intersect. Thus, app users in an overlapping area can access any of the edge servers covering them.
- *Server capacity constraint*: Different from the virtually unlimited computation and storage resources available in the cloud, edge servers only have limited computation and storage resources due to their limited sizes [10,11].

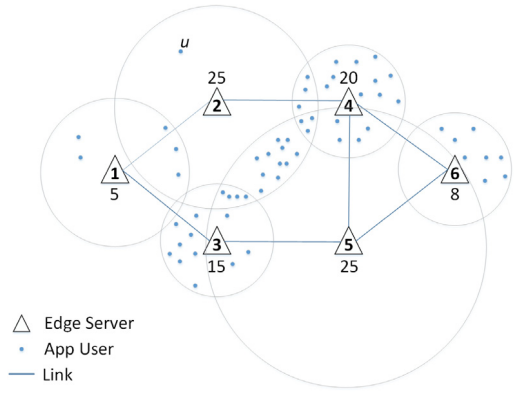


Fig. 1. An example EDC scenario.

Caching the data blocks of a data/file on multiple servers or machines is common in a large-scale cloud data center. This improves data reliability. However, in the edge computing environment, edge servers are attached to base stations that are geographically distributed. Each edge server covers the set of users within the coverage areas. Although the users within multiple edge servers' overlapping coverage area may be able to access multiple edge servers, it is usually not the case for most users. In addition, retrieving multiple data blocks from multiple edge servers and composing these data blocks into a data/file is too time-consuming in the edge computing environment where low latency is a top priority. In this paper, we have made the assumption that edge data are always cached in whole for the above reasons.

An EDC example is shown in Fig. 1. There are six edge servers in this area, with each edge server having a specific coverage area. The number next to an edge server indicates the number of users covered by that edge server. When a YouTube video becomes viral, it is predicted that a large number of YouTube users in this area will request this video. As a large amount of research effort has been made to predict video popularity [12], we assume that the number of YouTube users who will request this popular video can be predicted in this work. From YouTube's perspective, a straightforward solution to the EDC problem in this area is to cache this video on every edge server. This way, all YouTube users can access this video from edge servers. However, YouTube will need to pay for the hired resources on edge servers, such as bandwidth and storage, to cache this video. As even short videos are large, this solution is not cost-effective. Therefore, the data caching strategy must achieve the minimum data caching cost while ensuring that all the app users in this area can retrieve the video from one of the edge servers. This edge data caching (EDC) problem is inherently a Constrained Optimization Problem (COP).

The data retrieval latency and data caching cost can be evaluated using a variety of metrics. A user's data retrieval latency consists of two components: the latency between the user and its nearby edge server, and the latency between edge servers. The first component is not affected by the data caching strategy, and it is also quite small. Thus, this component is not included in the formulation of the EDC strategy. To model the COP in a more generic manner, including constraints and the optimization objective, we use the number of cached data replicas to measure the data caching cost, and the number of hops to measure the data retrieval latency. For example, the data caching cost is 6 if the video is cached on all the edge servers in Fig. 1. The *server adjacency constraint* requires that all the users must be able to retrieve the data from an edge server within one hop. For

Table 1
Summary of notations.

Notation	Description
b_u	Maximum benefit for user u
$b_{u,j}$	Benefit of caching replica on server v_j for app user u
CU	Set of users covered by the selected edge server set S
cu_i	Set of users covered by edge server v_i
$d_{i,j}$	Distance from server v_i to server v_j
d_T	Threshold of distance
d_u	Minimum distance from app user u to retrieve replica
$E = \{e_1, e_2, \ldots, e_m\}$	Finite set of links between edge servers
G	Graph presenting a particular area
$R = \{r_1, r_2, \ldots, r_n\}$	Set of binary variables indicating cache replicas on edge servers
S	Set of selected servers to cache data replica
$U = \{u_1, u_2, \ldots, u_k\}$	Finite set of users
$V = \{v_1, v_2, \ldots, v_n\}$	Finite set of edge servers

example, this constraint holds for the u in the top left corner if the video is cached on v_1, v_2 or v_4 and it does not hold if the video is only cached on v_3, v_5 and/or v_6. The rationale behind this constraint is that edge servers can communicate with their neighbor edge servers [9], but they are not designed or linked to route (potentially large) data across multiple hops. Based on the generic metrics for data caching cost and data retrieval latency, specific pricing policies and latency models can be integrated into our COP model. For example, app vendors can easily implement their own cost models and data sizes into our model.

There might exist multiple data caching solutions satisfying the latency constraint with the minimum cost. As edge servers often have different coverage radius and different user densities within their coverage areas, they usually cover different numbers of app users. Thus, the data caching solution should also reduce the maximal latency across all app users in this area. From YouTube's perspective, another optimization objective is thus to maximize the benefit produced by the cached data replicas, which is measured by the total reduction in the data retrieval latency for all the app users.

In this paper, we study quasi-static scenarios where users will not move across edge servers' coverage areas during the period of time when the EDC problem is being solved [6,9,13,14]. In highly mobile scenarios where users move across edge servers' coverage areas quickly, the app vendor can update its EDC strategy periodically or on-demand with a highly efficient EDC approach, e.g., AEDC as proposed in Section 4.2 and analyzed in Section 5.5. The model and approaches proposed in this paper are generic and applicable to various edge computing scenarios. Thus, data are cached on edge servers as a whole and we currently do not consider the situation where data can be partially cached, e.g., video segments. Also, the scale of the EDC problem in real-world scenarios can be much larger than the example presented in Fig. 1. Finding an optimal solution to a large-scale EDC problem is far from trivial.

3. Problem formulation

3.1. Problem statement

The edge servers in a particular area constitute an *edge server network*, which can be modeled as a graph where a node represents an edge server and an edge represents the link between two edge servers. Denote $G(V, E)$ as the graph, where V is the set of nodes in G and E is the set of edges in G. *In the remainder of this paper, we will speak inter-changeably of an edge server and its corresponding node in graph G, denoted as* v. The notations adopted in the paper are summarized in Table 1.

As mentioned in Section 2, this EDC problem is formulated in a generic manner: (1) using the number of data replicas to measure the data caching cost; and (2) using the number of hops between edge servers to measure the latency.

Based on the *server capacity constraint* in Section 2, edge servers only have limited resources. However, those limited resources are needed by many app vendors at the same time to host their services and cache their data for their app users. Thus, it is unlikely for one app vendor to hire most of those resources on an edge server and cache a huge amount of its data. A more cost-effective and realistic method is to cache the most popular data only for most app vendors. Therefore, this work considers the individual data caching scenarios, and builds the foundation for more sophisticated edge caching scenarios, e.g., caching multiple data.

Given a piece of data and a set of edge servers v_i ($i = \{1, \ldots, n\}$). Let $r_i \in \{0, 1\}$ be the decision indicating whether the data is cached on v_i, such that $r_i = 1$ if edge server v_i is selected to cache data. Denote the vector $R = \langle r_1, \ldots, r_n \rangle$ as the data caching solution.

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

$$d_u = \min\{d_{i,j}, r_j = 1, v_j \in V\}, \forall u \in U_i \tag{1}$$

where U_i is the set of users covered by edge server v_i.

The main objective of edge data caching is to provide high-quality services for app users with low latency. Thus, the data caching strategy R must satisfy the *server latency constraint* — the required data must be accessible from an edge server in this edge server network for every app user within a certain number of hops:

$$d_u < d_T, \forall u \in U_i \tag{2}$$

where d_T is the threshold of latency, measured by the number of hops as well.

Based on the *server adjacency constraint* discussed in Section 2, communications only occur between connected edge servers. Thus, d_T should be 2 here. However, this threshold can be relaxed if new techniques occur to allow data transmissions through multiple edge servers rapidly and the app vendor can accept the relatively high latency.

3.2. Data Caching benefit

To evaluate and compare the effectiveness of different data caching strategies, the concept of data caching benefit is introduced here, which can be calculated based on the latency reduction of user data retrieval. We use the number of hops reduced by cached data on an edge server to measure the data caching benefit. Thus, there is a negative correlation between data retrieval latency and data caching benefit. 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 data:

$$b_{u,j} = \begin{cases} d_T - d_{i,j} & \text{if } d_{i,j} < d_T \\ 0 & \text{if } d_{i,j} \geq d_T \end{cases} \tag{3}$$

As discussed in Section 2, to avoid the blank area that is not covered by any edge servers, the coverage of nearby edge servers often partially overlap. An app user in the overlapping area can access multiple edge servers, and retrieve data from any of those edge servers that the data in their cache. Thus, the data caching benefit produced by the data caching strategy for an app user u is:

$$b_u = \max\{r_j * b_{u,j}, v_j \in V\} \tag{4}$$

The data caching cost is the primary optimization objective from the app vendor's perspective. Thus, the data caching solution R must minimize this cost:

$$\text{minimize } cost(R) \tag{5}$$

With the minimum data caching cost, the optimal data caching strategy R should also maximize the data caching benefit:

$$\text{maximize } benefit(R) \tag{6}$$

In this way, we formulate this EDC problem as a constrained optimization problem.

3.3. Problem hardness

Here we prove the $\mathcal{NP}$-completeness of the EDC problem by Theorem 1.

Theorem 1. *The EDC problem is $\mathcal{NP}$-complete.*

Proof. To prove the $\mathcal{NP}$-completeness of the EDC problem, the minimum dominating set problem (MDS), one of the classic $\mathcal{NP}$-complete problems, is introduced first. Denote $G = (V, E)$ as an undirected graph where V is the set of n nodes and E is the set of m edges. Let $Con_{n,n}$ be the matrix to describe the connection between nodes such that $Con_{i,j}$ is 1 if there is an edge between node v_i and v_j. Denote S as the solution of this MDS problem. The MDS problem can be formulated as below:

$$\min \sum_{i=1}^{n} v_i \tag{7a}$$

$$s.t. : v_i \in \{0, 1\}, \forall i = \{1, .., n\} \tag{7b}$$

$$\sum_{i=1}^{n} Con_{i,j} \geq 1, \forall j \in \{1, \ldots, n\} \tag{7c}$$

Now we present the reduction process from the EDC problem to the MDS problem. The reduction consists of two parts: (1) making each app user only covered by one edge server; and (2) making only one app user in each edge server's coverage. Based on the reduction, the value of the benefit objective (6) is always same if selecting the same number of edge servers. In this case, the benefit objective is safely ignored. The instance $EDC(R, E, Ben_{n,k})$ can be constructed with the above reduction by an given instance $MDS(S, E, Con_{n,n})$ in polynomial time, where $n = k$ and $|R| = |S|$. In this $EDC(R, E, Ben_{n,k})$, the benefit matrix $Ben_{n,k}$ is calculated by (3). This way, any feasible solution S fulfilling objective (7a) and constraint (7b) also fulfills objective (5). The constraint (7c) of the MDS problem shows that if a node v_i is not selected, there is at least one neighbor of v_i selected in the solution s. Similarly, the benefit of app user $u \in U_i$ can be obtained as $b_u \geq 1$. This way, any feasible solution S fulfilling the constraint (7c) also fulfills the constraints (1) (2) (3) and (4). Therefore, the EDC problem is reducible from MDS and it is $\mathcal{NP}$-complete. □

4. Edge data caching strategy

We first propose the optimal model solved by Lexicographic Goal Programming[1] technique, then provide an efficient approximation algorithm with the analysis of its approximation ratio.

4.1. Optimal model

The EDC problem can be modeled as a constrained optimization problem (COP). One of the two optimization objectives can be prioritized over the other with the Lexicographic Goal Programming technique, depending on the app vendor's preference.

Given $G = (V, E)$, where $V = \{v_1, .., v_n, \}$ and $E = \{e_1, \ldots, e_m\}$, there is a set of variables $R = \{r_1, .., r_n\}$, where $r_i \in \{0, 1\}$, $\forall i \in \{1, \ldots, n\}$, r_i being 1 if the data replica is cached on the ith node, or 0 otherwise. The constraints for the COP model are:

$$b_u = \max\{r_i * b_{u,i}\}, \forall u \in \{1, \ldots, k\}, \forall i \in \{1, \ldots, n\} \tag{8}$$

$$1 \leq b_u \leq 2, \forall u \in \{1, \ldots, k\} \tag{9}$$

Constraint family (8) is calculated by (4), which guarantees that all the app users can retrieve data from the nearest edge server. Constraint family (9) enforces the latency constraint to ensure that every app user can retrieve the data from an edge server within one hop, which means the data is retrieved from the app user's local edge server via zero hops or neighbor edge server via one hop.

To satisfy constraint families (8) and (9), there might be multiple solutions. For example, two possible data caching solutions in Fig. 2(a) and Fig. 2(b) are $R_1 = \{0, 1, 1, 1, 0, 0\}$, which caches the data on v_2, v_3, and v_4, and $R_2 = \{1, 0, 0, 0, 0, 1\}$, which caches the data on v_1 and v_6. Both R_1 and R_2 are feasible with consideration of (8) and (9). However, the data caching cost of R_2 is less than that of R_1, where $cost(R_1) = 3$ and $cost(R_2) = 2$. To minimize the data caching cost, the below objective in the COP model is included to capture the app vendor's first optimization objective:

$$\min \sum_{i=1}^{n} r_i \tag{10}$$

The app vendor's second optimization objective also needs to be modeled in the COP. Let us assume two solutions as demonstrated in Fig. 2(b) and Fig. 2(c), $R_2 = \{1, 0, 0, 0, 0, 1\}$, which caches the data on v_1 and v_6, and $R_3 = \{0, 1, 0, 1, 0, 0\}$, which caches the data on v_2 and v_4, both fulfilling the latency constraint and achieving the app vendor's first optimization objective to minimize the data caching cost. However, compared with v_1 and v_6, v_2 and v_4 cover more app users, i.e., 39 versus 13 in total. Thus, R_3 allows more app users to retrieve the data from their local edge servers. Thus, from the app vendor's perspective, R_3 produces more caching benefits (i.e., lower retrieval latency) than R_2 at the same data caching cost. The below objective function that maximizes the data caching benefits of all app users based on (4) is included in the COP model to capture the app vendor's second optimization objective:

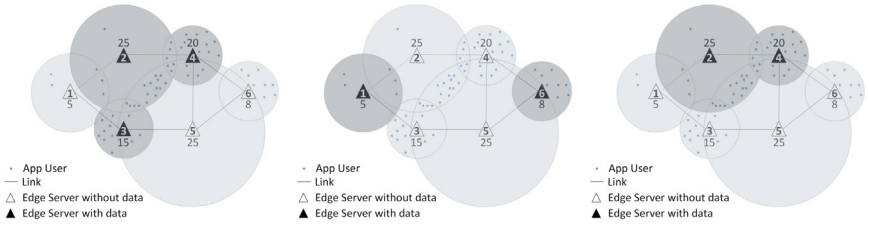
$$\max \sum_{u=1}^{k} b_u \tag{11}$$

The COP above can be solved with Integer Programming problem solvers, such as Gurobi[2] and IBM CPLEX Optimizer.[3] Here we name this optimal solution as IPEDC.

[1] https://www.ibm.com/support/knowledgecenter/en/SSSA5P_12.9.0/ilog.odms.cplex.help/CPLEX/UsrMan/topics/multiobj/multiobj_intro.html.

[2] http://www.gurobi.com/.

[3] https://www.ibm.com/analytics/cplex-optimizer.



(a) Strategy R_1 (b) Strategy R_2 (c) Strategy R_3

Fig. 2. Example data caching strategies.

4.2. Approximation algorithm

As mentioned in Section 3.3, the EDC problem is $\mathcal{NP}$-complete. It is intractable to find optimal solutions to large-scale EDC problems. In [7], we proposed a simple greedy algorithm, namely LGEDC. However, there is a significant performance gap between IPEDC and LGEDC. Thus, we develop a new approximation algorithm named AEDC to achieve higher performance with lower computation overhead than LGEDC.

Given $V = \{v_1, \ldots, v_n\}$ and $U = \{u_1, \ldots, u_m\}$, AEDC implements an iterative process for app vendors to hire edge servers to cache data replicas. The pseudo code is presented in Algorithm 1.

The algorithm starts with the initialization in Lines 1–4. To calculate the benefits of the solution provided by this algorithm, DCU is introduced to present the set of app users already covered by the solution, while CU means the set of app users that can access data via one hop. As edge servers can communicate with their neighbor edge servers, for each server $v_i \in V$, cu_i can be presented as the set of app users in the coverage of edge server v_i and their neighbor edge servers (Line 5).

In Algorithm 1, Δcu_i and Δdcu_i are used to select edge servers to cache data replicas, where Δcu_i is the number of users in cu_i but not in CU and Δdcu_i presents the number of users in dcu_i but not in DCU. For each iteration, Δcu_i and Δdcu_i are updated for each edge server $v_i \in V$. Then edge server v, which has the maximum value of Δcu, would be included into the solution set S to maximize the number of newly covered users. If there are more than one edge servers with that maximum value, AEDC selects the one with the maximum value of Δdcu to earn more benefits. This process iterates until all the app users are covered by the set of selected edge servers.

Fig. 3 presents the system state at different moments during the AEDC process for the EDC example in Fig. 1. Fig. 3(a) illustrates the initial state. Then, edge server v_5 is selected to cache a data replica because v_5 can serve the most new app users (54 users) via one hop where $\Delta CU = \{42, 41, 38, 50, 54, 46\}$ in Fig. 3(b). After the first iteration, there are 6 users not covered by the solution set $S = \{v_5\}$, and the algorithm continues the iterative process to choose another edge server. In the second iteration, both Δcu_1 and Δcu_2 are the maximal value while $\Delta CU = \{6, 6, 5, 5, 0, 0\}$. In this case, edge server v_1 is selected in Fig. 3(c), because $\Delta dcu_1 = 5 > \Delta dcu_2 = 4$. Thus, the solution of AEDC is $S = \{v_5, v_1\}$, as all users have been covered by selected edge servers. Considering the optimal solution $R_3 = \{0, 1, 0, 1, 0, 0\}$ described in Section 4.1, $Cost_{IPEDC} = Cost_{AEDC} = 2$ while $Benefits_{IPEDC} = 99 > Benefits_{AEDC} = 90$.

As the IPEDC approach is implemented based on the Integer Programming technique, IPEDC selects the optimal solution among all the possible solutions that have a total of $C_n^1 + C_n^2 + \cdots + C_n^n = 2^n$ possible results. Moreover, we prove that the COP of EDC is NP-complete in Section 3.3, thus IPEDC cannot find the optimal solution within polynomial time. In the AEDC algorithm, it can be calculated that the worst case of computational complexity is $O(n^2)$. This means that AEDC can reduce a large amount of execution time to find the solution of EDC problem, compared with IPEDC.

Algorithm 1: AEDC Algorithm.

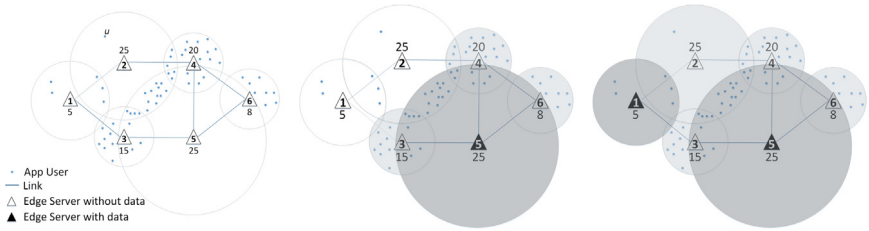
```
1: Initialization:
2: CU, DCU, S ← ∅, benefits, cost = 0
3: for each server v_i do
4:    dcu_i ← U_i, cu_i ← U_i
5: end for
6: End of initialization
7: for each v_i ∈ V do
8:    for each neighbor v_j of v_i do
9:       cu_i ← cu_i ∪ dcu_j
10:   end for
11: end for
12: repeat
13:   for each v_i ∈ V do
14:      Δcu_i = |cu_i ∩ ¬CU|
15:      Δdcu_i = |dcu_i ∩ ¬DCU|
16:   end for
17:   v ← v_0
18:   for each v_i ∈ V do
19:      if Δcu_{v_i} > Δcu_v or
         Δcu_{v_i} = Δcu_v and Δdcu_{v_i} > Δdcu_v then
20:         v ← v_i
21:      end if
22:   end for
23:   S ← S ∪ {v}
24:   CU ← CU ∪ cu_i
25:   DCU ← DCU ∪ dcu_i
26: until CU = U
27: return S
```

Now, we prove the approximation ratio of AEDC, where it is the ratio of the cost incurred by AEDC and that incurred by IPEDC in the worst case.

Theorem 2. *The approximation ratio of AEDC is $\ln\Delta + 1$, where Δ denotes the maximum number of app users that can be covered by any edge server $v \in V$ within 1 hop.*

Proof. As mentioned in Section 2, the number of data replicas is used to measure the data caching cost. For each edge server selected to cache a data replica, the cost is 1. Here we implement an amortized strategy to analyze the approximation ratio. This way, we distribute the cost 1 to newly covered users equally instead of the selected edge server. Take Fig. 3(b) as an example, the cost of each user incurred by selecting v_5 is $\frac{1}{54}$.

Based on constraint (9), all the app users should be covered by the data caching strategy. Let S_{opt} denote the optimal solution found by IPEDC. In this case, we can divide the graph G into $|S_{opt}|$ stars. Each star contains 1 edge server as the center of the star and the newly covered users as its leaves.



(a) $S = \varnothing$ (b) $S = \{v_5\}$ (c) $S = \{v_5, v_1\}$

Fig. 3. Example system state at different moments during AEDC process.

Denote $newU(v)$ as the number of newly covered app users by adding edge server v into the solution. For each edge server $v \in S_{opt}$, the corresponding star has $newU(v)$ leaves. Based on the heuristic logic of AEDC, the distributed cost is at most $\frac{1}{newU(v)}$. Otherwise, AEDC would choose another edge server. After assigning the distributed cost to newly covered app users, those app users would not produce any more costs. In the worst case, no two app users in the same star are covered together. Thus, the total distributed cost of a star is at most:

$$\frac{1}{newU(v)} + \frac{1}{newU(v)-1} + \cdots + \frac{1}{2} + \frac{1}{1}$$

Based on the harmonic series, we can obtain that the total distributed cost of a star is at most $ln(newU(v)) + 1$. Thus, the total cost produced by AEDC is at most $|S_{opt}| \cdot (ln(newU(v)) + 1)$. Moreover, for each $v \in V$, $newU(v)$ is always less than or equal to Δ. Thus, the approximation ratio of AEDC is:

$$ratio \leq \frac{|S_{opt}| \cdot (ln\Delta + 1)}{|S_{opt}|} = ln\Delta + 1 \quad \square \tag{12}$$

5. Experimental evaluation

We experimentally evaluate IPEDC and AEDC on a widely-used real-world data set and a synthetic data set, and compare their performance against four representative approaches.

5.1. Comparison approaches

In these experiments, we evaluate and compare the performance of IPEDC and AEDC against four comparison approaches, namely *LGEDC*[7], *DIP,liu2019data*, *Random* and *Greedy-Covered-Users*:

- *LGEDC*: This algorithm keeps selecting the edge server with the most links until it satisfies the latency constraint (2).
- *DIP* [15]: This approach tries to minimize the app vendor's revenue, combining caching cost and latency reduction, by caching data on edge servers without leveraging the collaboration between edge servers. Its parameter settings in the experiments are the same as [15], i.e., $[5, 25]$ for the unit cost of cache and $[0.5, 2]$ for the unit cost of latency (same as unit benefit). Accordingly, the range of the ratio between them is $[2.5, 50]$. In the EDC problem, all users must be covered. To pursue this goal, the ratio is fixed at 2.5 for DIP in the experiments to cover as many users as possible.
- *Random*: This algorithm keeps selecting the edge server randomly until it satisfies the latency constraint (2).
- *Greedy-Covered-Users*(GU): This algorithm keeps selecting the edge server with the most app users until it satisfies the latency constraint (2).

5.2. Experimental settings

Data Sets: There are two sets of experiments. The first set is conducted the public real-world Edge User Allocation (EUA) data set[4][4] with 128 edge servers and 816 app users in the Melbourne CBD area. The second set is conducted on a synthetic data set that simulates more general EDC scenarios. In the experiments on the synthesized data set, a specific number of edge servers are randomly distributed within a particular area with app users also generated randomly. In both sets of experiments, edges are randomly generated to connect the edge servers according to the edge density to ensure that the graph is connected.

Parameter Settings: To comprehensively analyze IPEDC and AEDC, we vary two parameters in the experiments to simulate different EDC scenarios, as presented in Table 2. This way, we can also analyze how the changes in the parameters impact the performance of our approaches. Each experiment is repeated 100 times every time we change a parameter, and the results are averaged:

- The total number of edge servers ($n = |V|$). In experiment Set #1 and Set #2.1, this number varies from 10 to 50 in steps of 10.
- Edge density ($d = |E|/|V|$). In experiment Set #2.2, this number varies from 1 to 3 in steps of 0.4.

Performance Metrics: In the experiments, we use four metrics to evaluate the effectiveness and efficiency of all the approaches:

1. Data Caching Cost *cost*, the lower the better;
2. Data Caching Benefit *benefit*, the higher the better;
3. Benefit per Data Replica *bpr*, the higher the better; and
4. Computation Overhead *time*, the lower the better.

To stabilize the impact of the number of app users, we always generate 100 app users in experiment Set #2.

5.3. Experimental results

Figs. 4–6 show the results of the experiments Set #1, #2.1 and #2.2, respectively.

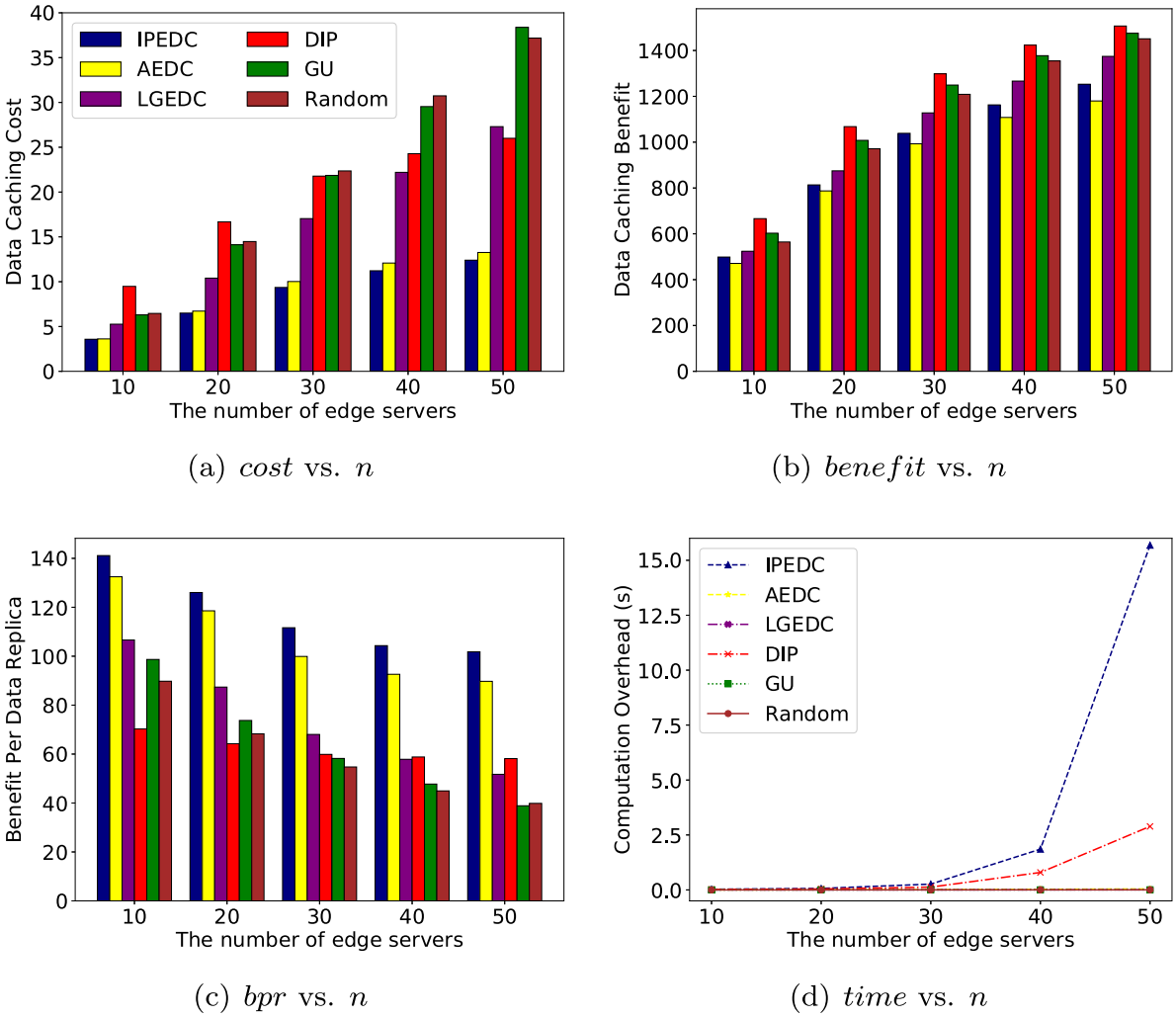
5.3.1. Effectiveness

Fig. 4 shows the results of experiment Set #1. Overall, of all the six approaches, **IPEDC achieves the highest benefit per data replica at the lowest data caching cost, while AEDC is the second lowest in cost with the second highest in benefit per data replica**. Fig. 4(b) shows that AEDC and IPEDC achieve lower data caching benefits than those four comparison approaches. With the priority to minimize the data caching cost, if an app user can retrieve data from edge servers via one hop, there is no need to do so via zero hops. Thus, IPEDC will aim for a solution

[4] https://github.com/swinedge/eua-dataset.

Table 2
Parameter settings.

	Number of edge servers	Edge density	Data set
Set #1	10, 20, 30, 40, 50	1	Real-world
Set #2.1	10, 20, 30, 40, 50	1	Synthetic
Set #2.2	30	1, 1.4, 1.8, 2.2, 2.6, 3	Synthetic



(a) *cost* vs. n (b) *benefit* vs. n

(c) *bpr* vs. n (d) *time* vs. n

Fig. 4. Experiment set #1.

that barely fulfills (9), i.e., a solution just good enough to allow as many app users as possible to retrieve data from edge servers via one hop.

Fig. 4(a) shows that **the average data caching costs achieved by IPEDC and AEDC are much lower than the other four approaches** across all five cases, i.e., 8.61 (IPEDC) and 9.14 (AEDC) versus 16.44 (LGEDC), 19.64 (DIP), 22.04 (GU) and 22.24 (Random). The average advantage of IPEDC is 5.80% against AEDC, 47.63% against LGEDC, 56.16% against DIP, 60.93% against GU, and 61.29% against Random. Fig. 4(a) also shows that, as the number of edge servers increases from 10 to 50, the data caching cost achieved by AEDC increases from 3.62 replicas to 13.26 replicas on average, similar to IPEDC (3.58 to 12.4) but much slower than LGEDC (5.26 to 27.3), DIP (9.48 to 26.0), GU (6.34 to 38.38) and Random (6.46 to 37.18). Fig. 4(b) shows that the increase in the number of edge servers will increase the data caching benefits achieved by all six approaches, from 498.68 to 1252.82 for IPEDC, 470.90 to 1179.74 for AEDC, 523.98 to 1374.32 for LGEDC, 666.24 to 1506.76 for DIP, 602.82 to 1475.24 for GU and 564.48 to 1451.04 for Random. Fig. 4(c) shows **the significant advantages of IPEDC and AEDC over the other approaches** in achieving cost-effective data caching strategies. IPEDC has the best performance, which averagely outperforms AEDC by 9.70%, LGEDC by 57.39%, DIP by 87.88% GU by 84.38% and Random by 96.57%.

Fig. 5 depicts the results of experiment Set # 2.1. Overall, **IPEDC again achieves the highest data caching benefit per replica at the lowest data caching cost**, following by AEDC. The advantages of IPEDC and AEDC over the other four approaches are significant. In this set of experiments, the edge servers are set up in a similar way as in Set #1. Therefore, the results shown in Fig. 5(a) are similar to those shown in Fig. 4. However, Fig. 5(b) shows that the **data caching benefit does not increase** with the increase in the number of edge servers. The reason is that, unlike experiment Set #1, the number of app users in experiment Set #2.1 does not increase. Thus, the data caching benefit does not increase accumulatively as in Fig. 4(b). This is also the same reason for the rapid decrease in the benefit per data replica demonstrated in Fig. 5(c).

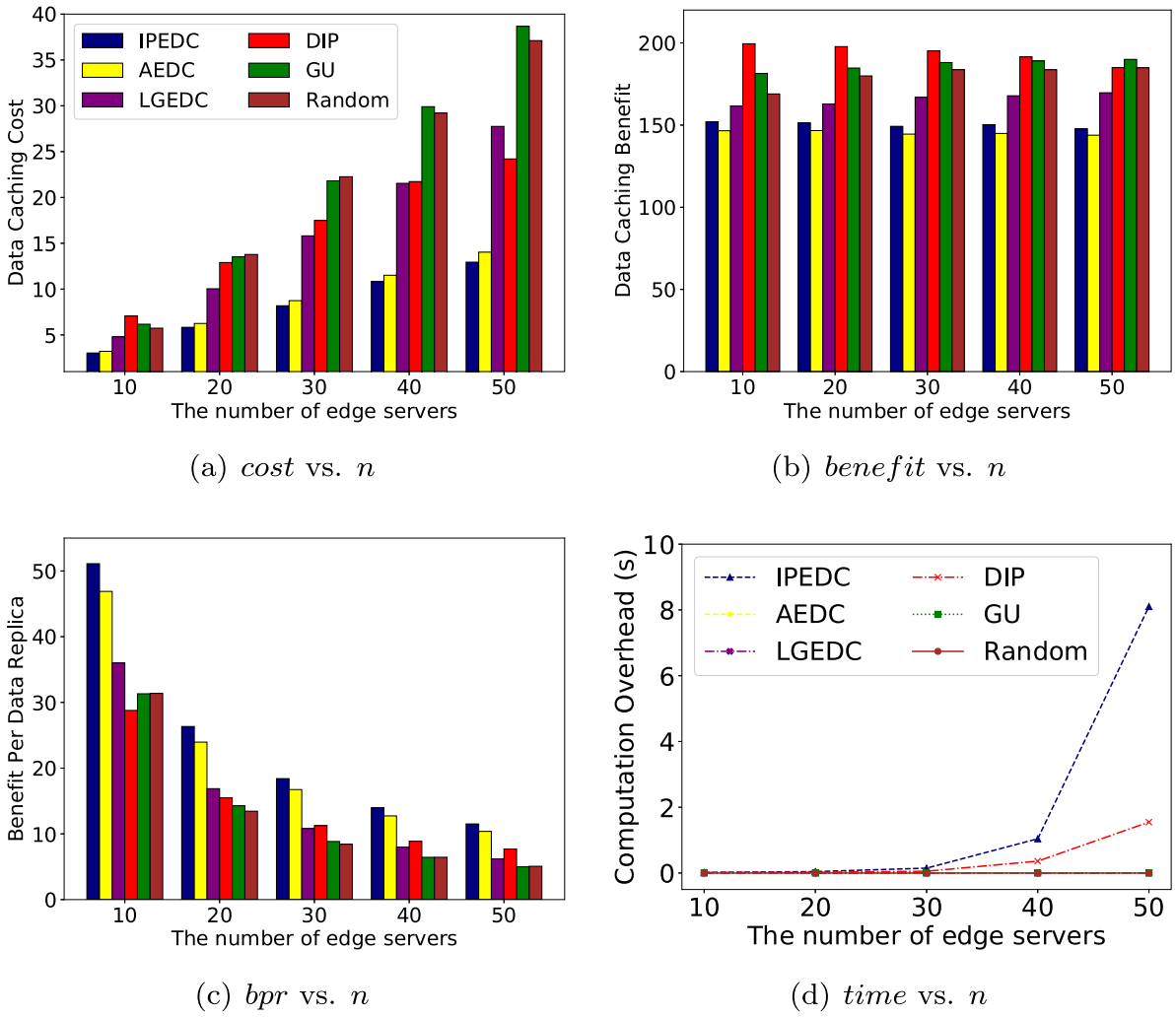


Fig. 5. Experiment set #2.1.

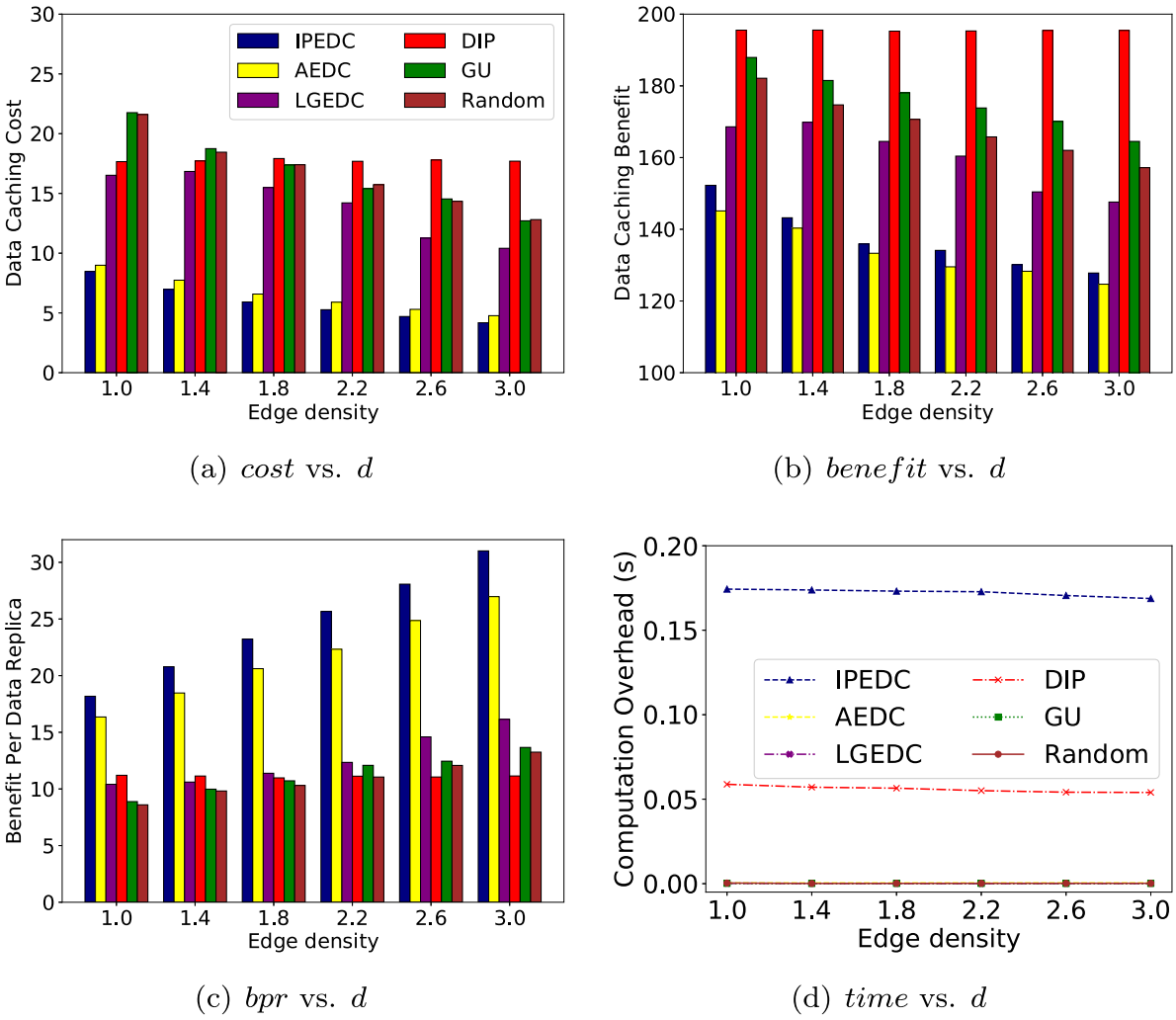
Fig. 6 shows the results in experiment Set 2.2 where the edge density varies. In terms of the average data caching cost and benefit per data replica, **IPEDC and AEDC outperform the other four approaches** with large margins, and IPEDC still has the best performance. The advantage of IPEDC is 9.62% against AEDC, 58.07% against LGEDC, 66.65% against DIP, 64.68% against GU and 64.59% against Random on average in data caching cost, while 13.38% against AEDC, 94.67% against LGEDC, 120.43% against DIP, 116.73% against GU and 125.71% against Random on average in benefit per data replica.

Interestingly, Fig. 6 shows that **the edge density impacts the approaches in a very different way** from the number of edge servers. Fig. 6(a) shows that as the edge density increases from 1.0 to 3.0, the data caching costs achieved by IPEDC and AEDC decrease from 8.47 to 4.18 and from 8.99 to 4.77 respectively. After investigating the results, we find that the increase in the edge density allows each edge server to link to more edge servers. This increases the app users' chances of retrieving data from edge servers via one hop. IPEDC does not need to cache as many data replicas to ensure that all app users are served by edge servers within one hop. As a result, the average data caching cost decreases. For the same reason, the data caching benefit decreases, as demonstrated in Fig. 6(b). The increase in the connectivity between edge servers also allows more app users to be able to retrieve data via one hop. As a result, the benefit per data replica increases, as demonstrated in Fig. 6(c), from 18.18 to 31.01 for IPEDC, from 16.35 to 26.97 for AEDC, from 10.41 to 16.15 for LGEDC, from 11.20 to 11.15 for DIP, from 8.89 to 13.67 for GU and from 8.60 to 13.25 for Random. Since DIP focuses on maximizing the benefits with consideration of caching cost, the total benefits achieved by DIP are only changed slightly when the maximum benefits are fixed in Set #2.

Overall, our **IPEDC and AEDC outperform LGEDC, DIP, GU and Random significantly and consistently** in formulating cost-effective data caching strategies. Overall, AEDC can achieve about 90% of IPEDC's performance in minimizing data caching cost and benefits per replica across all the experiments. Both IPEDC and AEDC are particularly effective in EDC scenarios where edge servers are highly connected.

5.3.2. Efficiency

Figs. 4(d), 5(d) and 6(d) present the average computation overheads of the six approaches in finding a solution to the EDC problem. We can see in Figs. 4(d) and 5(d) that the **computation overhead of IPEDC increases rapidly** when the number of edge servers increases. When there are 50 edge servers to consider, IPEDC takes more than 15 s to find the optimal solution, as shown in Fig. 4(d). Excessive computation overheads are inevitable when IPEDC is looking for the optimal solution to this NP-complete EDC problem. Thus, IPEDC is **suitable for solving EDC problems in small sizes**. To solve large-scale EDC problems, **heuristics-based approaches are more practical**, e.g., AEDC, LGEDC or GU. The



(a) *cost* vs. d (b) *benefit* vs. d

(c) *bpr* vs. d (d) *time* vs. d

Fig. 6. Experiment set #2.2.

results in Fig. 6(d) indicate that both IPEDC and AEDC are capable for handling dense graphs built based on edge servers that are highly connected. In conclusion, IPEDC can be used to solve EDC problems in small sizes while AEDC handles the large-scale ones.

5.4. Threats to validity

Threat to construct validity. The main threat to construct validity is the four comparison approaches. Due to the novelty of this edge data caching problem, we choose the greedy approach proposed in [7], the optimal benefit-based approach proposed in [15] and two basic baseline approaches to compare with our approaches in the experiments. To minimize the threat of comprehensive evaluation, two parameters are varied in the experiments to simulate different EDC scenarios. This way, we could not only evaluate IPEDC and AEDC with four comparison algorithms but also present the impacts of varying parameters on those algorithms.

Threat to external validity. The main threat here is whether IPEDC and AEDC are also suitable in 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. Moreover, the two data sets used to conduct the experiments, including a real-world one and a synthetic one. Thus, the representativeness and comprehensiveness of the evaluation are ensured, and this threat is reduced.

Threat to conclusion validity. The lack of statistical tests, e.g., chi-square tests, is the major threat to conclusion validity in our paper. To compensate this threat, we have conducted comprehensive and intensive experiments to cover various scenarios in different size and complexity. Every time a parameter changes, we repeat the experiment for 100 times and calculate the averaged results. This led to a large number of test cases, which tend to result in a small p-value in the chi-square tests and lower the practical significance of the test results [16]. For example, in experiment Set #2, there were a total of 1100 runs. This number is not even close to the number of observation samples that concern Lin et al. in [16]. Thus, the threat to the conclusion validity due to the lack of statistical tests might be high but not significant.

5.5. Real-world applications

User mobility is an important characteristic of the real-world edge computing environment. However, the structure of an edge server graph does not change when the users move in the area. It is determined by how the edge servers in the area are physically linked. Moreover, for its high efficiency, AEDC can solve the EDC problem quickly in different time slots, similar to the approaches proposed in [9,14]. In each time slot, edge servers need to update the information including users within its coverage area and their requests. Specifically, when a user no longer needs a piece of

data or has moved out of the current edge server's coverage area, this user can be removed from the current edge server's user set U_i. In the meantime, new users can be added into U_i. This way, the context can be rapidly updated at the start of each time slot. Then, the AEDC algorithm can be executed in each time slot to formulate the corresponding data caching strategies. Given its high efficiency as demonstrated in Figs. 4(d), 5(d) and 6(d), AEDC can be executed periodically or on-demand to adapt to user mobility in real-world EDC scenarios.

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 [17], content-centric networking [18], content delivery network [19], etc. Banerjee et al. [20] 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 [21], 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 [22] 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. [23] proposed CloudCache, a method for managing cache, to fulfill the caching requirement of the workload and minimize cache wear-out. In [24], 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 [25] 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 [26], the authors formulated a benefit maximization problem and created a cache replacement approach based on spatio-temporal traffic requirements. They also introduced a content clustering method for collecting popular data and clustering similar contents.

6.3. Edge data Caching

As an extension of cloud computing, edge computing distributes both computing capacities and storage resources from cloud server to edge servers [27]. With the deployment of edge servers, the problems of computation offloading and data caching occurs. The computation offloading problems have been extensively investigated from different perspectives, including edge servers' energy efficiency [28] and offloading cost [29].

Recently, there are some researchers starting to investigate the data caching problems in edge computing. As mentioned in Section 2, the data caching strategies from conventional distributed computing and cloud computing cannot be directly applied in edge computing. Thus, those researches introduced new ideas and approaches. Cao et al. proposed an optimal auction mechanism with the consideration of the costs produced during delivery and retrieval. The authors of [30] provided a caching system, namely Agar, from the erasure-coded perspective. Agar was a dynamic programming algorithm which could cache data chunks optimally with consideration of data popularity and network latency.

Instead of improving internal cache utility on edge servers, some researchers started to investigate how to combine the advantages of both internal caches and external caches. Zhang et al. [31] integrated in-network caching and edge caching to ensure the latency requirements of time-sensitive transmissions over the 5G network. The authors of [32] introduced a new edge caching architecture with improved resource utility by using smart vehicles as external edge caches.

The above studies mostly focus on cost savings. Data latency is also an important issue in edge data caching problems. Drolia et al. [33] proposed an edge caching system, namely Cachier, to minimize the data retrieval latency. They implemented a coordinating mechanism to balance the loads between the cloud server and edge servers dynamically. Liu et al. [15] studied the data caching problem in the edge computing environment with the aim to maximize the app vendor's revenue based on caching cost and latency. This approach was implemented in our experiments as DIP for comparison. However, those work ignored the collaboration between edge servers, as well as the benefit produced by the reduction in user' service latency.

Edge computing inherits the pay-as-you-go price model from cloud computing. Thus, the cost incurred for app vendors is critical to the success of edge computing because they are the main customers in the edge computing environment. However, all the above work tackles the data caching problem from network providers' or app users' perspectives, our work solves the Edge Data Caching (EDC) problem from the app vendor' perspective in the edge computing environment. We also realistically and innovatively solve the EDC problem in a generic manner to minimize the data caching cost and maximize the data caching benefit with the server coverage constraint and the server adjacency constraint.

7. Conclusion

In this paper, we formulated the new Edge Data Caching (EDC) problem as a constrained optimization problem based on the graph from the app vendor's perspective. The optimal solution of the EDC problem is to find a solution that minimizes the data caching cost and maximizes the data caching benefit. We proved that the EDC problem is $\mathcal{NP}$-complete. Then we proposed an optimal solution IPEDC solved by Integer Programming, and an approximation algorithm AEDC within a provable bound. We conducted extensive experiments based on a real-world data set and a synthetic data set to evaluate our approaches. The results demonstrate that both IPEDC and AEDC significantly outperform all other four baseline approaches in formulating cost-effective EDC solutions, while AEDC solves large-scale EDC problems efficiently.

This research has established the foundation for the EDC problem and opened up a number of research directions. In the future, we will investigate multiple data caching scenarios, partitionable data caching scenarios, users' dynamic participation and security policy. Those will allow our approaches to accommodate more sophisticated EDC scenarios.

CRediT authorship contribution statement

Xiaoyu Xia: Conceptualization, Methodology, Software, Investigation, Writing - original draft. **Feifei Chen:** Conceptualization, Methodology, Writing - review & editing, Supervision. **Qiang He:** Conceptualization, Methodology, Writing - review & editing, Supervision, Funding acquisition. **Guangming Cui:** Conceptualization, Methodology, Software. **Phu Lai:** Conceptualization, Methodology, Software. **Mohamed Abdelrazek:** Conceptualization, Writing - review & editing, Supervision, Funding acquisition. **John Grundy:** Conceptualization, Writing - review & editing, Supervision, Funding acquisition. **Hai Jin:** Conceptualization, Writing - review & editing.

Declaration of competing interest

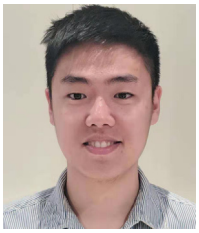
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research is partially funded by Australian Research Council Projects No. DP170101932, DP180100212 and Laureate Fellowship, Australia FL190100035.

References

[1] Afif Osseiran, Volker Braun, Taoka Hidekazu, Patrick Marsch, Hans Schotten, Hugo Tullberg, Mikko A Uusitalo, Malte 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] Anthony D. Josep, Randy Katz, Andy Konwinski, Lee Gunho, David Patterson, Ariel RabKin, A view of cloud computing, Commun. ACM 53 (4) (2010).
[3] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, Lanyu Xu, Edge computing: Vision and challenges, IEEE Internet Things J. 3 (5) (2016) 637–646.
[4] Phu Lai, Qiang He, Mohamed Abdelrazek, Feifei Chen, John Hosking, John Grundy, Yun 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.
[5] Tuyen X. Tran, Mohammad-Parsa Hosseini, Dario Pompili, Mobile edge computing: Recent efforts and five key research directions, IEEE COMSOC MMTC Commun.-Frontiers 12 (4) (2017) 29–33.
[6] Qiang He, Guangming Cui, Xuyun Zhang, Feifei Chen, Shuiguang Deng, Hai Jin, Yanhui Li, Yun Yang, A game-theoretical approach for user allocation in edge computing environment, IEEE Trans. Parallel Distrib. Syst. (2019).
[7] Xiaoyu Xia, Feifei Chen, Qiang He, Guangming Cui, Phu Lai, Mohamed Abdelrazek, John Grundy, Hai Jin, Graph-based optimal data caching in edge computing, in: International Conference on Service-Oriented Computing, Springer, 2019, pp. 477–493.
[8] VNI Cisco Mobile, Cisco visual networking index: Global mobile data traffic forecast update, 2016–2021 white paper, 2017.
[9] Lixing Chen, Sheng Zhou, Jie Xu, Computation peer offloading for energy-constrained mobile edge computing in small-cell networks, IEEE/ACM Trans. Netw. 26 (4) (2018) 1619–1632.
[10] Min Chen, Yixue Hao, Kai Lin, Zhiyong Yuan, Long Hu, Label-less learning for traffic control in an edge network, IEEE Netw. 32 (6) (2018) 8–14.
[11] Shuo Wang, Xing Zhang, Yan Zhang, Lin Wang, Juwo Yang, Wenbo Wang, A survey on mobile edge networks: Convergence of computing, caching and communications, IEEE Access 5 (2017) 6757–6779.
[12] Alexandru Tatar, Marcelo Dias De Amorim, Serge Fdida, Panayotis Antoniadis, A survey on predicting the popularity of web content, J. Internet Serv. Appl. 5 (1) (2014) 1–20.
[13] Xu Chen, Decentralized computation offloading game for mobile cloud computing, IEEE Trans. Parallel Distrib. Syst. 26 (4) (2014) 974–983.
[14] Jie Xu, Lixing Chen, Pan Zhou, Joint service caching and task offloading for mobile edge computing in dense networks, in: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, IEEE, 2018, pp. 207–215.
[15] Ying Liu, Qiang He, Dequan Zheng, Mingwei Zhang, Feifei Chen, Bin Zhang, Data caching optimization in the edge computing environment, in: 2019 IEEE International Conference on Web Services, ICWS, IEEE, 2019, pp. 99–106.
[16] Mingfeng Lin, Henry C. Lucas Jr., Galit Shmueli, Research commentary—too big to fail: large samples and the p-value problem, Inf. Syst. Res. 24 (4) (2013) 906–917.
[17] Stefan Podlipnig, Laszlo Böszörmenyi, A survey of web cache replacement strategies, ACM Comput. Surv. 35 (4) (2003) 374–398.
[18] Yonggong Wang, Zhenyu Li, Gareth Tyson, Steve Uhlig, Gaogang Xie, Optimal cache allocation for content-centric networking, in: 21st IEEE International Conference on Network Protocols, ICNP, 2013, pp. 1–10.
[19] Daniel S. Berger, Ramesh K. Sitaraman, Mor 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.
[20] Bitan Banerjee, Adita Kulkarni, Anand Seetharam, Greedy caching: An optimized content placement strategy for information-centric networks, Comput. Netw. 140 (2018) 78–91.
[21] Md Yusuf Sarwar Uddin, Nalini 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.
[22] Yanhua Li, Haiyong Xie, Yonggang Wen, Zhi-Li 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.
[23] Dulcardo Arteaga, Jorge Cabrera, Jing Xu, Swaminathan Sundararaman, Ming Zhao, CloudCache: On-demand flash cache management for cloud computing, in: 14th USENIX Conference on File and Storage Technologies, FAST, 2016, pp. 355–369.
[24] Kun Ma, Bo Yang, Zhe Yang, Ziqiang Yu, Segment access-aware dynamic semantic cache in cloud computing environment, J. Parallel Distrib. Comput. 110 (2017) 42–51.
[25] Abdel-Hameed A Badawy, Gabriel Yessin, Vikram Narayana, David Mayhew, Tarek El-Ghazawi, Optimizing thin client caches for mobile cloud computing: Design space exploration using genetic algorithms, Concurr. Comput.: Pract. Exper. 29 (11) (2017) e4048.
[26] Syed Tamoor-ul Hassan, Sumudu Samarakoon, Mehdi Bennis, Matti Latva-Aho, Choong Seon Hong, Learning-based caching in cloud-aided wireless networks, IEEE Commun. Lett. 22 (1) (2018) 137–140.
[27] Marcelo Yannuzzi, Frank van Lingen, Anuj Jain, Oriol Lluch Parellada, Manel Mendoza Flores, David Carrera, Juan Luis Pérez, Diego Montero, Pablo Chacin, Angelo Corsaro, et al., A new era for cities with fog computing, IEEE Internet Comput. 21 (2) (2017) 54–67.
[28] Feng Wang, Jie Xu, Xin Wang, Shuguang Cui, Joint offloading and computing optimization in wireless powered mobile-edge computing systems, IEEE Trans. Wireless Commun. 17 (3) (2018) 1784–1797.
[29] Hong Yao, Changmin Bai, Muzhou Xiong, Deze Zeng, Zhangjie Fu, Heterogeneous cloudlet deployment and user-cloudlet association toward cost effective fog computing, Concurr. Comput.: Pract. Exper. 29 (16) (2017) e3975.
[30] Raluca Halalai, Pascal Felber, Anne-Marie Kermarrec, François Taïani, Agar: A caching system for erasure-coded data, in: 37th IEEE International Conference OnDistributed Computing Systems, ICDCS, 2017, pp. 23–33.
[31] Xi Zhang, Qixuan Zhu, Hierarchical caching for statistical qos guaranteed multimedia transmissions over 5g edge computing mobile wireless networks, IEEE Wirel. Commun. 25 (3) (2018) 12–20.
[32] Ke Zhang, Supeng Leng, Yejun He, Sabita Maharjan, Yan Zhang, Cooperative content caching in 5g networks with mobile edge computing, IEEE Wirel. Commun. 25 (3) (2018) 80–87.
[33] Utsav Drolia, Katherine Guo, Jiaqi Tan, Rajeev Gandhi, Priya Narasimhan, Cachier: Edge-caching for recognition applications, in: 37th IEEE International Conference OnDistributed Computing Systems, ICDCS, 2017, pp. 276–286.

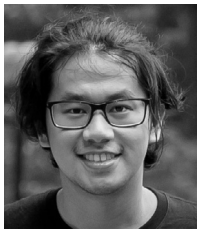


Xiaoyu Xia received his Master degree from The University of Melbourne, Australia in 2015. He is a Ph.D. candidate at Deakin University. His research interests include edge computing, service computing and software engineering.

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

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

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



Phu Lai received his M.Sc. degree in Information Technology in 2017 and is currently working toward a Ph.D. degree at Swinburne University of Technology, Australia. His research interests include software engineering, cloud computing and edge computing.

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/.

John Grundy received the B.Sc. (Hons), M.Sc., and Ph.D. degrees in computer science from the University of Auckland, New Zealand. He is currently 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/.

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 Ph.D. 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.