17th International Conference on Service-Oriented Computing (ICSOC2019), Oct 28-31, Toulouse, France

Graph-based Optimal Data Caching in Edge Computing

Xiaoyu Xia[1], Feifei Chen[1], Qiang He[2](✉), Guangming Cui[2], Phu Lai[2], Mohamed Abdelrazek[1], John Grundy[3], and Hai Jin[4]

[1] Deakin University, Burwood, Australia
{xiaoyu.xia,feifei.chen,mohamed.abdelrazek}@deakin.edu.au
[2] Swinburne University of Technology, Hawthorn, Australia
{qhe, gcui, tlai}@swin.edu.au
[3] Monash University, Clayton, Australia
john.grundy@monash.edu
[4] Huazhong University of Science and Technology, Wuhan, China
hjin@hust.edu.cn

Abstract. In an edge computing environment, edge servers are deployed at base stations to offer highly accessible computing capacities and services to nearby users. Data caching is thus extremely important in edge computing environments to reduce service latency. The optimal data caching strategy in the edge computing environment will minimize the data caching cost while maximizing the reduction in service latency. In this paper, we formulate this edge data caching (EDC) problem as a constrained optimization problem (COP), prove that the EDC problem is $\mathcal{NP}$-complete, propose an optimal approach named IPEDC to solve the EDC problem using the Integer Programming technique, and provide a heuristic algorithm named LGEDC to find near-optimal solutions. We have evaluated our approaches on a real-world data set and a synthesized data set. The results demonstrate that IPEDC and LGEDC significantly outperform two representative baseline approaches.

Keywords: Optimization · Edge computing · Data caching

1 Introduction

Over the last decade, the world has witnessed an exponential growth of mobile traffic over the internet, which is predicted to expand by 1,000 times over the coming decade with a huge increase in Internet of Things (IoT) connected devices [1]. The enormous network traffic load often causes network congestion that significantly impacts users' quality of experience, especially service latency. To attack this challenge, edge computing, a new distributed computing paradigm, has emerged to allow computing capacities such as CPUs, memory and storage to be distributed to *edge servers* at the edge of the cloud [2]. Each edge server is powered by one or more physical servers and deployed at base stations that are geographically close to users. Mobile and IoT app vendors can hire computing capacities on edge servers so that they can host their services to offer their app users low service latency [3]. Such services are referred to as *edge services* in the remainder of this paper.

As an increasing number of mobile devices start to access edge services, a large proportion of the rapidly growing mobile traffic will go through edge servers. Enormous data will be transmitted by edge servers. Caching data, especially popular data such as viral videos and photos from Facebook, on edge servers will minimize the latency in users' data retrieval. Users can retrieve data from a nearby edge server instead of retrieving it from the cloud if the data is cached on that edge server. This is especially important for latency-sensitive applications, e.g., gaming, navigation, augmented reality, etc. Popular data often accounts for a large percentage of the mobile traffic data over the internet. Thus, caching popular data on edge servers can significantly reduce the traffic load on the internet backbone. It is expected to reduce mobile traffic data by 35% [4]. From an app vendor's perspective, it can also considerably reduce data transfer costs by decreasing the volume of data transferred from the cloud to its users.

Given a piece of popular data, a straightforward solution is to cache it on all the edge servers in a particular area for nearby app users to access. This way, the latency in all app users' data retrieval can be minimized. However, based on the pay-as-you-go pricing model, the app vendor will need to hire substantial resources on edge servers for caching the data. This incurs excessive *caching cost* and is impractical for most, if not all, app vendors. Thus, from an app vendor's perspective, it is critical to find an optimal data caching strategy that minimizes the caching cost incurred while guaranteeing the low latency in its users' data retrieval. We refer to this data caching problem in the edge computing environment as an *edge data caching* (EDC) problem. While existing research investigates data caching in the edge computing environment from either the network infrastructure providers or users' perspectives, we make the first attempt to study the EDC problem based on graph from the app vendor's perspective.

In this work, we make the following major contributions:

– We model and formulate the EDC problem as a constrained optimization problem (COP) from the app vendor's perspective.
– We prove that the EDC problem is $\mathcal{NP}$-complete based on the minimum dominating set problem.
– We develop an optimal approach named IPEDC for solving the EDC problem with the Integer Programming technique.
– We develop a heuristic approach named LGEDC for finding near-optimal solutions to the EDC problem efficiently in large-scale scenarios.
– We evaluate our approaches against two representative baseline approaches with experiments conducted on both real-world data and synthesized data.

The rest of paper is organized as follows. Section 2 motivates this research with an example. Section 3 discusses our approaches for solving the EDC problem. Section 4 evaluates the approaches experimentally. Section 5 reviews the related work. Section 6 concludes this paper and points out future work.

2 Motivating Example

Video services accounted for 54 percent of the total internet traffic in 2017 and the ratio is expected to grow to 79 percent by 2022 [5]. Thus, a representative example of data cached on edge servers is video data. App vendors such as YouTube currently store their video data on their servers in the cloud. When a video goes viral over the internet, a large number of mobile YouTube users make requests for it. This creates immense pressure on the service in the cloud. Caching this piece of data on edge servers, especially in areas with high user density, brings it closer to the users and reduces the latency of data retrieval.

In an edge computing environment, edge servers can communicate with their neighbor edge servers and share their computing capacities and storage via high-speed links [6] (*server adjacency constraint*). This allows workloads in a particular area to be balanced across the edge servers covering that area [6]. Thus, the edge servers in a particular area can be modeled as a graph where a *node* represents an edge server and an *edge* represents the link between two edge servers. Moreover, the coverage areas of adjacent edge servers often intersect to avoid blank areas not covered by any edge servers. A user in the intersection area can connect to one of the edge servers covering this user (*server coverage constraint*).

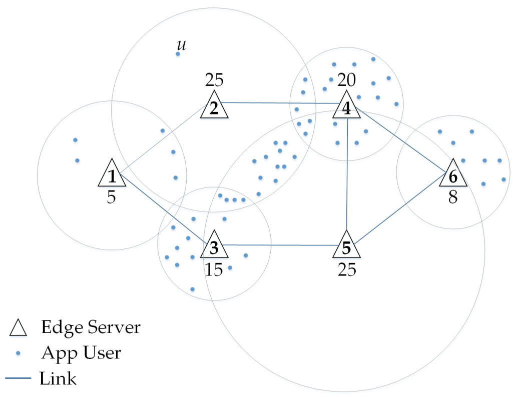


Fig. 1: An example EDC scenario

Fig. 1 presents an example area with six edge servers, i.e., $\{v_1, ..., v_6\}$, each covering a specific geographic area. The number next to each edge server is the number of app users covered by that edge server. Let us assume a YouTube video goes viral and it is predicted that a large number of mobile YouTube users in this area will request this video. Please note that there is a large body of research work available on the prediction of popular videos [7] and thus in this research we assume that the number of mobile YouTube users who will request this popular video can be predicted. From YouTube's perspective, caching this video on all the edge servers can easily accommodate all the mobile YouTube users in this area. However, it is usually not cost-effective considering that YouTube will pay for the resources on the edge servers hired for caching the data, e.g., storage and bandwidth. Thus, the data caching strategy must minimize the data caching cost and ensure 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 caching cost and data retrieval latency 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. As the first component is very small and not influenced by the data caching strategy, it is not considered in the formulation of the data caching strategy. To quantify the optimization objective and constraints in the COP in a generic manner, we measure the data caching cost using the number of cached data replicas and the data retrieval latency using the number of hops, i.e., links between edge servers. For example, the cost of caching the video on all the six edge servers in Fig. 1 is 6. The *server adjacency constraint* requires that all the users must be able to retrieve the data from an edge server less than two hops. For 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, 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, knowing the size of the data to be cached and the prices for the storage and bandwidth for caching the data, the data cache cost can be calculated based on the number of cached data replicas.

There might be multiple caching strategies that minimize the data caching cost while fulfilling the latency constraint for every app user. Different edge servers usually cover different numbers of app users, depending on the user density in their coverage areas. Thus, one of those caching strategies is to maximize the total latency reduction across all the covered app users. From YouTube's perspective, the other optimization objective is thus to maximize the benefit produced by the cached data replicas, which is measured by the total reduction in data retrieval latency for all the app users.

The model and approach proposed in this research are generic and applicable to various apps. Thus, 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 EDC problem in the 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 not trivial.

3 Our Approach

3.1 Definitions

In this research, the n edge servers in a particular area are modeled as a graph. For each edge server v_i, the graph has a corresponding node. For each pair of linked edge servers (v_i, v_j), the graph has a corresponding edge e_t. We use $G(V, E)$ to represent 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 discussed in Section 2, we formulate the EDC problem in a generic manner by measuring the data retrieval latency by the number of hops between edge servers and the data caching cost by the number of cached data replicas.

Compared with cloud's virtually unlimited computing capacities, an edge server usually has limited computing capacities due to its size limit [8, 9]. At runtime, many app vendors will need to hire the computing capacities for hosting their services and caching their data for their own app users. Thus, an app vendor is unlikely to hire a huge amount of computing capacities on an edge server for caching a lot of its data. It is more cost-effective for most, if not all, app vendors to cache the most popular data on edge servers to serve its nearby users. Thus, in this paper, we investigate the scenarios where data is processed and cached individually. The model and the approaches proposed will build the foundation for more sophisticated edge caching scenarios, e.g., caching multiple data.

Table 1: Summary of Notations

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

Given a piece of data and a set of edge servers v_i $(1 \leq i \leq n)$, a data caching strategy is a vector $R =< r_1, ..., r_n >$, where r_i $(1 \leq i \leq n)$ denotes whether the data is cached on edge server v_i:

$$r_i = \begin{cases} 0 & \text{if data is not cached on edge server } v_i \\ 1 & \text{if data is cached on edge server } v_i \end{cases} \tag{1}$$

In graph G, the distance between two nodes v_i and v_j is the number of hops on the shortest path between them. Thus, given an app user u, its data retrieval latency is measured by the number of hops between the edge server covering u and the nearest edge server with the data in its cache.

$$d_u = \min\{d_{i,j}, r_j = 1, v_j \in V\}, \forall u \in U_i \tag{2}$$

The main objective of edge data caching is to ensure a low data retrieval latency for app users. Thus, R must fulfill the *latency constraint* - every app user must be able to retrieve the data from an edge server within a certain number of hops:

$$d_u < d_T, \forall u \in U_i \tag{3}$$

As discussed in Section 2, each edge server can only communicate with its neighbors. Thus, there is $d_T = 2$. However, this can be relaxed, e.g., $d_T = 3, 4, ...$, if the high latency incurred is considered acceptable by the app vendor and new techniques enable data to be transmitted through multiple edge servers rapidly.

To evaluate and compare different data caching strategies, we use the concept of data caching benefit, which is calculated based on the reduction in data retrieval latency measured by the number of hops reduced by cached data. The caching benefit produced by caching data on v_j for user u covered by v_i denoted by $b_{u,j}$, is calculated as follows:

$$b_{u,j} = \begin{cases} 0 & \text{if } d_{i,j} \geq d_T \\ d_T - d_{i,j} & \text{if } d_{i,j} < d_T \end{cases} \tag{4}$$

In the edge computing environment, an app user $u \in U$ might be covered by multiple edge servers. App user u can retrieve the data from any of those edge servers that have the data in the cache. Thus, the data caching benefit produced by the data caching strategy for an app user u is:

$$b_u = \max\{r_j * b_{u,j}, v_j \in V\} \tag{5}$$

From the app vendor's perspective, one of the optimization objectives is to minimize the data caching cost incurred by R and measured by the number of cached data replicas:

$$\text{minimize } cost(R) \tag{6}$$

The second optimization objective is to maximize the data caching benefit, measured by the total reduction in all users' data retrieval latency produced by R based on (5):

$$\text{maximize } benefit(R) \tag{7}$$

3.2 Edge Data Caching 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.

A COP consists of a finite set of variables $X = x_1, , x_n$, with domain $D_1, ..., D_n$ listing the possible values for each variable in X, and a set of constraints $C = c_1, c_2, , c_t$ over X. A solution to a COP is an assignment of a value to each variable in X from its domain such that all constraints in C are satisfied. The COP model for the EDC problem is formally expressed as follows.

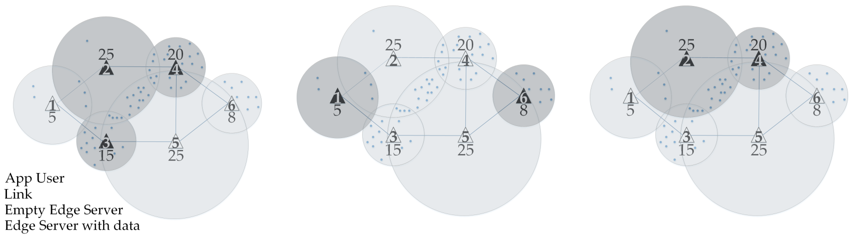
For a graph $G = (V, E)$, where $V = \{v_1, .., v_n, \}$ and $E = \{e_1, ..., e_m\}$, there are a set of variables $R = \{r_1, .., r_n\}$, where $D(r_i) = \{0, 1\}, \forall i \in \{1, ..., n\}$, r_i being 1 if the a data replica is cached on the i^{th} node, 0 otherwise. The constraints for the COP model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

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

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

Constraint family (8) is converted from (5). It ensures that every app user will always retrieve the 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 edger server within 2 hops.

There might be multiple solutions fulfilling (8) and (9). In Fig. 2(a) and Fig. 2(b), two possible data caching strategies are $R_1 = \{0, 1, 1, 1, 0, 0\}$, which caches



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

Fig. 2: Example Data Caching Strategies

the data on v_1, v_2, and v_3 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 fulfill (8) and (9). However, R_2 caches two data replicas, incurring a lower data caching cost than R_1. Thus, the below objective function that minimizes the total number of data replicas cached over G is included in the COP model 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 captured by the COP model. 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. However, compared with v_1 and v_6, v_2 and v_4 cover more app users, 39 versus 13 in total. Thus, R_3 allows more app users to retrieve the data from edge servers directly. Thus, from the app vendor's perspective, R_3 produces more caching benefits 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 (5) is included in the COP model to capture the app vendor's second optimization objective:

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

Integer Programming problem solvers, e.g., IBM CPLEX Optimizer[5] and Gurobi[6], can be employed to solve the above COP. The optimal solution is the data strategy that achieves both (10) and (11) while fulfilling (8) and (9). In this paper, objective (10) (minimize the total number of data replicas) is prioritized over objective (11) (maximize the data caching benefits) as an example for discussion. In real-world applications, objective (11) can be given a higher priority than (10) if the app vendor is willing to minimize its app users' latency at a high data caching cost.

Given multiple data to be cached over time, multiple COPs need to be solved to find one data caching strategy for each piece of data. Those COPs share the same G. Thus, the shortest distance between every two nodes in G can be pre-computed offline to facilitate rapid calculation of (2) as well as app users' cache benefits (5) at runtime.

[5] https://www.ibm.com/analytics/cplex-optimizer
[6] http://www.gurobi.com/

3.3 Problem Hardness

In this section, we demonstrate that the COP of EDC is $\mathcal{NP}$-complete by proving the following theorems.

Theorem 1. *The COP of EDC is in $\mathcal{NP}$.*

Proof. As there are $(nk + k)$ constraints in total, any solution to the COP can be validated in polynomial time by checking whether the solution satisfies the constraint group (8) and (9). Thus, the COP of EDC is in $\mathcal{NP}$.

Theorem 2. *The COP of EDC is $\mathcal{NP}$-complete.*

Proof. To prove this problem is $\mathcal{NP}$-complete, we introduce the minimum dominating set problem (MDS). MDS problem is known to be $\mathcal{NP}$-complete. Given an undirected graph $G = (V, E)$, where $|V| = n$ and $|E| = m$. The metrics $C_{n,n}$ presents the connection between vertices. $C_{i,j} = 1$ if v_i and v_j are connected, otherwise $C_{i,j} = 0$. The formulation is displayed below:

$$object : \min \sum_{i=1}^{n} v_i \tag{12a}$$

$$s.t. : v_i \in \{0, 1\}, i = \{1, .., n\} \tag{12b}$$

$$\sum_{j=1}^{n} C_{i,j} \geq 1, \forall i \in \{1, ..., n\} \tag{12c}$$

Now we prove that the minimum dominating set problem can be reduced to an instance of the EDC problem. The reduction is done in two steps: 1) let each edge server cover only one app user; 2) let each app user be covered by only one edge server. Due to the reduction, objective (8) can be ignored because the total user benefits are always the same with the same number of servers selected for caching this data. Given an instance $MDS(v, e, C_{n,n})$, we can construct an instance $EDC(r, e, B_{n,k})$ with the reduction above in polynomial time while $|r| = |v|$ and $n = k$, where $B_{n,k}$ is the benefit matrix from (4). In this case, any solution s satisfying objective (12a) and constraint (12b) also satisfies objective (10). Moreover, the constraint (12c) means that for each vertex v_i not in the solution s, there exists at least one neighbour of v_i in s. From this point, user u covered by vertex v_i can obtain benefit $b_u \geq 1$. Thus, if the solution s fulfills constraint (12c), it also fulfills constraints (8) and (9). Therefore, the COP of EDC is reducible from MDS and it is $\mathcal{NP}$-complete.

3.4 A Near-Optimal Algorithm

Finding the optimal solution to the $\mathcal{NP}$-complete EDC problem is intractable in large-scale scenarios. Thus, this section proposes a heuristic algorithm for finding a near-optimal solution to large-scale EDC problems efficiently.

A naive and straightforward heuristic is to always cache data on the edge server with the most app users. However, selecting an edge server with many neighbor edge servers allows the app users covered by those neighbor edge servers to retrieve cached data within one hop. Based on this heuristic, we present a **l**ink-oriented **g**reedy algorithm, namely *LGEDC*, that always selects the node with the most edges in G to cache the data. The pseudo code is presented in Algorithm 1. In the worst-case scenario, LGEDC selects no more than n edge servers, while the computational complexity of the function *selectMaximumEdgsServer* is $O(n)$. Thus, the computational complexity of LGEDC is $O(n^2)$.

EDC problem has two objectives (10) and (11). Here, we select the prioritized objective (10) (minimize the total number of data replicas) to calculate the approximation ratio of LGEDC. The approximation ratio can be calculated based on Theorem 3.

Theorem 3. *LGEDC is $O(n)$-approximation.*

Proof. Let us assume that the optimal solution OPT selects k edge servers to cache data. Fig. 3 presents a worst-case EDC scenario, where n edge servers are linked as a circle and each edge server covers its own group of distinct app users. In this case, Algorithm 1 will select $v_1, v_2, ..., v_{n-2}$ to cache data, as the app users in v_n covered by v_n can be served by v_1, the app users covered by v_{n-1} can be served by v_{n-2}. The solution S of LGEDC selects at most $n-2$ edge servers to cache data. Thus, there is $\frac{|S|}{|OPT|} = \frac{n-2}{k}$, and LGEDC is $O(n)$-*approximation*.

Algorithm 1 LGEDC Algorithm

```
1: Initialization:
2: $CU, S \leftarrow \varnothing$
3: End of initialization
4: repeat
5:     $v \leftarrow$ selectMaximumEdgesServer()
6:     $S \leftarrow S \cup \{v\}$
7:     $CU \leftarrow CU \cup cu_i$
8: until $CU = U$
```

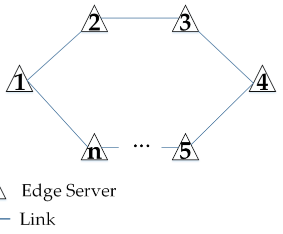


Fig. 3: The worst-case in LGEDC

4 Experimental Evaluation

We conducted two sets of experiments to evaluate the performance of IPEDC and LGEDC. The COP discussed in Section 3 is solved with IBM's CPLEX Optimizer. All the experiments are conducted on a machine equipped with Intel Core i7-8550 processor (8 CPUs, 1.8GHz) and 8GB RAM, running Windows 10.

4.1 Baseline Approaches

In these experiments, we evaluate the performance of our approaches against two representative baseline approaches, namely *Random* and *Greedy-Covered-Users*:

- *Random*: This approach randomly selects edge servers, one after another, to cache data until the latency constraint (3) is fulfilled.
- *Greedy-Covered-Users (GU)*: This approach always selects the edge server that covers the most app users to cache data until the latency constraint (3) is fulfilled.

4.2 Experimental Settings

Data Sets: Two sets of experiments are conducted, one on the public real-world EUA data set[7] [2] and the other on a synthetic data set. The latter is synthesized to simulate more general EDC scenarios. In the experiments on the synthesized

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

data set, a certain number of edge servers are randomly distributed within a particular area with app users generated also randomly. In the experiments, edges are randomly generated according to the edge density to ensure the graph is connected.

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

- The number of edge servers (n). This parameter impacts the size of graph G and varies from 10 to 40 in steps of 10.
- Edge density (d). In the second set of experiment,s given n edge servers in a particular area, a total of e edges are generated randomly according to the edge density calculated with $d = e/n$. This parameter impacts the density of graph G and varies from 1 to 3 in steps of 0.5.

Performance Metrics: Four metrics are used in the experiments for the evaluation, three for effectiveness and one for efficiency: (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; (4) Computation Overhead *time*, the lower the better.

According to (11), *cost* is calculated by summing the benefits of all app users. Thus, to stabilize the impact of the number of app users, we always select or generate a total of 100 app users in the experiments set #2.

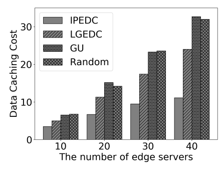
Table 2: Parameter Settings

	Number of Edge Servers	**Edge Density**	**Data Set**
Set #1	10, 20, 30, 40	1	Real-World
Set #2.1	10, 20, 30, 40	1	Synthetic
Set #2.2	10	1, 1.5, 2, 2.5, 3	Synthetic

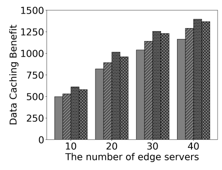
4.3 Experimental Results

The results of the experiments are shown in Fig. 4, Fig. 5 and Fig. 6, corresponding to Set #1, #2.1 and #2.2.

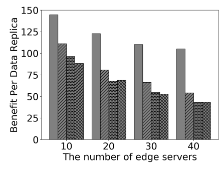
Effectiveness: Fig. 4 presents the results of experiment set #1. Overall, of all the four approaches, **IPEDC achieves the highest benefit per replica at the lowest data caching cost, while LGEDC is the second lowest in cost with the second highest in benefit per data replica**. Fig. 4(b) shows that IPEDC achieves the lowest data caching benefit. In the experiments, objective (10) is prioritized over (11). With the priority to minimize the data caching cost, retrieving data from edge servers via one hop is more preferable. Thus, IPEDC will aim for a solution that barely fulfills (9), i.e., a solution that suffices to allow the most users to retrieve data from edge servers via one hop. Fig. 4(a) shows that **the average data caching costs achieved by IPEDC and LGEDC are much lower than other two approaches**, 7.71 for IPEDC and 14.43 for LGEDC versus 19.44 for GU and 19.14 for Random. Fig. 4(a) also shows that, as the number of edge servers increases from 10 to 40, the data caching cost achieved by IPEDC increases from 3.48 to 11.14 on average, much slower than LGEDC (5.01 to 24.01), Random (6.78 to 31.98) and GU (6.57 to 32.7). Fig. 4(b) shows that the increase in the number of edge servers will increase the data caching benefits achieved by all four approaches, from 497.13 to 1164.28 for IPEDC, 530.30 to 1289.99 for LGEDC, 612.17 to 1397.02 for GU and 579.48 to 1368.11 for Random. Fig. 1(c) demonstrates **the significant advantage of IPEDC over the other approaches in achieving cost-effective data caching strategies**. On average, it outperforms LGEDC by 54.54%, GU by 84.09% and Random by 90.83%. As the number of edge servers increases, the benefits per replica achieved by all approaches decrease. The increase in the number of edge servers deployed in a specific area increases the connectivity between the edge servers. This increases app users' chances of retrieving data via one hop, which lowers the average benefit produced by each data replica.



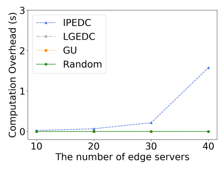
(a) *cost* vs. *n*



(b) *benefit* vs. *n*

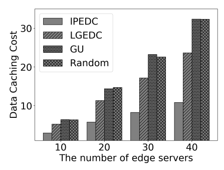


(c) *bpr* vs. *n*

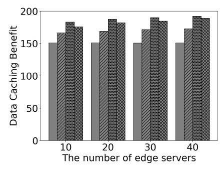


(d) *time* vs. *n*

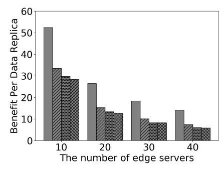
Fig. 4: Experiment Set #1



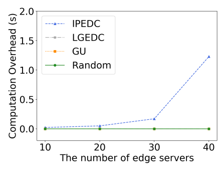
(a) *cost* vs. *n*



(b) *benefit* vs. *n*

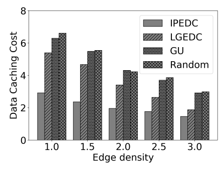


(c) *bpr* vs. *n*

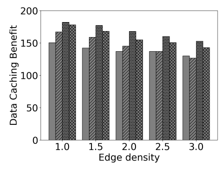


(d) *time* vs. *n*

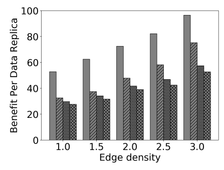
Fig. 5: Experiment Set #2.1



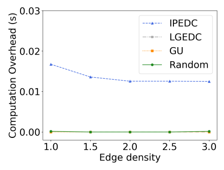
(a) *cost* vs. *d*



(b) *benefit* vs. *d*



(c) *bpr* vs. *d*



(d) *time* vs. *d*

Fig. 6: Experiment Set #2.2

Fig. 5 depicts the results from experiment Set # 2.1. Overall, **IPEDC achieves the highest data per replica at the lowest data caching cost** again. Its advantage over the other approaches is significant. In this set of experiments, the edge servers are set up in a similar way as 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. 6 shows the results in experiment Set #2.2 where the graph density varies. In terms of the average data caching cost and benefit per data replica, **IPEDC outperforms the other approaches** with large margins, 44.17% against LGEDC, 55.81% against GU and 56.77% against Random on average in data caching cost, 45.85% against LGEDC, 83.25% against GU and 89.30% 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 cost achieved by IPEDC decreases from 2.92 to 1.47. This is because the increase in the edge density allows each edge server to be linked 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 52.84 to 96.60 for IPEDC, from 32.70 to 75.25 for LGEDC, from 29.82 to 57.46 for GU and from 27.74 to 52.73 for Random.

Overall, **IPEDC significantly and consistently outperforms all other approaches, with LGEDC second**, in formulating cost-effective data caching strategies, especially in EDC scenarios where edge servers are highly connected.

Efficiency: Fig. 4(d), Fig. 5(d) and Fig. 6(d) present the average computation times taken by the four approaches to find a solution to the EDC problem. We can see in Fig. 4(d) and Fig. 5(d) that the **computation overhead of IPEDC increases rapidly** when the number of edge servers increases. When there are 40 edge servers to consider, IPEDC takes 1-2 seconds to find the optimal solution in Fig. 4(d). This excessive computation overhead is inevitable in large-scale EDC scenarios because IPEDC tries to find the optimal solution to the $\mathcal{NP}$-complete EDC problem. Thus, **IPEDC is suitable for solving EDC problems with reasonable sizes, while LGEDC is suitable for solving large-scale EDC problems**. The results in Fig. 6(d) indicates that IPEDC is also very efficient in EDC scenarios where edge servers are highly connected.

4.4 Threats to Validity

Construct Validity. The major threat to construct validity is the two baseline approaches used for comparison. Due to the innovation of the EDC problem in the edge computing environment, we chose two basic naive approaches as baselines in our evaluation. As those baseline approaches are relatively simple, IPEDC and LGEDC tend to achieve better experimental results. Thus, there is a threat that the comparison does not suffice to comprehensively evaluate IPEDC and LGEDC. To minimize this threat in the experiments, we changed two parameters, as presented in Table 2, to simulate various EDC scenarios.

In this way, we could evaluate our approaches by not only comparison to the baseline approaches, but also demonstrate how the changes in the parameters impact the performance of the approaches.

External Validity. The major threat to external validity is whether IPEDC and LGEDC can be generalized and applied in other application scenarios in the edge computing environment. To tackle this threat, we measure the performance of our approaches in a generic way - using the number of reduced hops for effectiveness evaluation and the number of data replicas for efficiency evaluation. In this way, the results of the evaluation can be interpreted based on specific models of data retrieval latency and data caching cost. In addition, we ran the experiments on a real-world data set and a synthetic data set. We also varied two parameters to vary the size and the complexity of the EDC problem. This way, the representativeness and comprehensiveness of the evaluation are ensured. The above measures allowed us to ensure that the results were generalized, which reduced the threat to external validity.

Conclusion Validity. The main threat to conclusion validity is the lack of statistical tests, e.g., chi-square tests. We could have conducted chi-square tests to draw conclusions. However, we ran the experiment for 100 times in experiments and averaged the results each time we changed a parameter. 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 [10]. For example, in experiment Set 2, there were a total of 1,300 runs. This number is not even close to the number of observation samples that concern Lin et al. in [10]. Thus, the threat to the conclusion validity due to the lack of statistical tests might be high but not significant.

5 Related Work

Edge computing is an extension of cloud computing with distributed computing resources and services at the edge of the cloud [11]. With the deployment of edge servers, the problem of computation offloading arises. It has been well studied with consideration of edge servers' energy efficiency [12], offloading cost [13] and so forth.

In the last few years, researchers have been investigating the challenges raised by data caching in the edge computing environment. Conventional approaches for data caching are not suitable in the edge computing environment and cannot be applied directly. Thus, new ideas and techniques are being proposed and investigated. An optimal auction mechanism was introduced in [14] that considers the data retrieval and delivery costs. The authors showed computationally efficient approaches for calculating the optimal decisions of cache allocation and user pays. Halalai et al. [15] proposed Agar, a caching system, from the erasure-coded perspective. They designed Agar based on a dynamic programming algorithm for optimally caching data chunks with consideration of data popularity and network latency.

Instead of data caching optimization across edge servers, some researchers study how to integrate edge servers' internal caches and external caches. In [16], the authors proposed Cachier, a system that minimizes data retrieval latency by coordinating the loading balance between edge servers and the cloud in a dynamical manner. The authors of [17] integrated in-network caching and edge caching to guarantee the quality of time-sensitive multimedia transmissions over the 5G wireless network. They also provided three hierarchical edge caching mechanisms, including a random hierarchical caching approach, a proactive hierarchical caching approach and a game-theory-based hierarchical caching approach. Zhang et al. [18] proposed an architecture to enhance edge caching by using computation resources of edge servers. They presented a caching scheme by implementing smart vehicles as edge servers to provide external caches.

To the best of our knowledge, our work is the first attempt to solve 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 maximum the data caching benefit with the server coverage constraint and the server adjacency constraint.

6 Conclusion

In this paper, we formulated the new Edge Data Caching (EDC) problem in the edge computing environment as a constrained optimization problem from the app vendor's perspective. To find an optimal solution, we proposed IPEDC, an approach based on the Integer Programming technique with two optimization objectives: 1) to minimize the data caching cost measured by the number of cached data replicas; and 2) to maximum the data caching benefit measured by the total reduction in app users' data retrieval latency. However, we also proved that the EDC problem is $\mathcal{NP}$-complete. We then provided a heuristic approach named LGEDC for finding near-optimal solutions to the EDC problem. We conducted extensive experiments based on a real-world data set and a synthetic data set to evaluate the performance of IPEDC and LGEDC in different EDC scenarios. The results demonstrate that IPEDC significantly outperforms all other approaches in formulating cost-effective EDC solutions, while LGEDC 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 our future work, we will first consider the problem of caching multiple data at the same time for an app vendor. Other issues that can be investigated include data popularity, security constraints, etc.

Acknowledgement

This research is partially funded by Australian Research Council Discovery Projects (No. DP170101932 and DP180100212).

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. 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.
3. T. X. Tran, M.-P. Hosseini, and D. Pompili, "Mobile edge computing: Recent efforts and five key research directions," *IEEE COMSOC MMTC Commun.-Frontiers*, vol. 12, no. 4, pp. 29–33, 2017.
4. M. ETSI, "Mobile edge computing - introductory technical white paper," 2014.
5. *Cisco visual networking index: global mobile data traffic forecast update, 20172022*, 2019. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-738429.html
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. A. Tatar, M. D. De Amorim, S. Fdida, and P. Antoniadis, "A survey on predicting the popularity of web content," *Journal of Internet Services and Applications*, vol. 5, no. 1, pp. 1–20, 2014.
8. M. Chen, Y. Hao, K. Lin, Z. Yuan, and L. Hu, "Label-less learning for traffic control in an edge network," *IEEE Network*, vol. 32, no. 6, pp. 8–14, 2018.
9. S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.
10. M. Lin, H. C. Lucas Jr, and G. Shmueli, "Research commentarytoo big to fail: large samples and the p-value problem," *Information Systems Research*, vol. 24, no. 4, pp. 906–917, 2013.
11. 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.
12. 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.
13. H. Yao, C. Bai, M. Xiong, D. Zeng, and Z. Fu, "Heterogeneous cloudlet deployment and user-cloudlet association toward cost effective fog computing," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 16, p. e3975, 2017.
14. 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.
15. 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.
16. 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.
17. 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.
18. 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.