Graph-Based Data Deduplication in Mobile Edge Computing Environment

Ruikun Luo[1], Hai Jin[1], Qiang He[2], Song Wu[1(✉)], Zilai Zeng[1], and Xiaoyu Xia[3]

[1] National Engineering Research Center for Big Data Technology and System Services Computing Technology and System Lab, Cluster and Grid Computing Lab Huazhong University of Science and Technology, Wuhan, China
{rkluo,hjin,wusong,zilaizeng}@hust.edu.cn
[2] Swinburne University of Technology, Hawthorn, Australia
qhe@swin.edu.au
[3] Deakin University, Burwood, Australia
xiaoyu.xia@deakin.edu.au

Abstract. *Mobile edge computing* (MEC) extends cloud computing by deploying edge servers with computing and storage resources at base stations within users' geographic proximity. The networked edge servers in an area constitute an *edge storage system* (ESS), where edge servers cooperate to provide services for the users in the area. However, the potential of ESSs is challenged by edge servers' constrained storage resources due to their limited physical sizes. A straightforward method to tackle this challenge is to reduce data redundancy in the ESS. The unique characteristics and constraints in the MEC environment, e.g., edge servers' geographic coverage and distribution, render conventional data deduplication techniques designed for cloud storage systems obsolete. In this paper, we make the first attempt to study this novel *Edge Data Deduplication* (EDDE) problem. First, we model it as a constrained optimization problem with the aim to maximize data deduplication ratio under latency constraint by taking advantage of the collaboration between edge servers. Then, we prove that the EDDE problem is $\mathcal{NP}$-hard and propose an approach named EDDE-O for solving the EDDE problem optimally based on integer programming. To accommodate large-scale EDDE scenarios, we propose a $ln\alpha + 1$-approximation algorithm, namely EDDE-A, to find sub-optimal EDDE solutions efficiently. The results of extensive experiments conducted on a widely-used dataset demonstrate that EDDE-O and EDDE-A can solve the EDDE problem effectively and efficiently, outperforming four representative approaches significantly.

Keywords: Mobile edge computing · Edge data storage · Data deduplication · Integer programming · Approximation algorithm

1 Introduction

In recent years, the world has witnessed an exponential growth of network traffic produced by mobile and *internet-of-things* (IoT) services [12]. The transmission

H. Hacid et al. (Eds.): ICSOC 2021, LNCS 13121, pp. 499–515, 2021.
https://doi.org/10.1007/978-3-030-91431-8_31

of massive mobile and IoT data incurs heavy network traffic and consumes excessive network resources. In the meantime, the cloud computing paradigm is failing to fulfill various services' demand for low latency [5]. To tackle these challenges, *mobile edge computing* (MEC) as a new computing paradigm has emerged, which extends the cloud's computing and storage capabilities to the network edge in close proximity to mobile and IoT devices.

In the MEC environment, edge servers with computing and storage resources are deployed at base stations. The networked edge servers in an area constitute an *edge storage system* (ESS). Service providers like Facebook and YouTube can cache popular data on edge servers to enable low-latency data retrieval for their users [13,15]. Data produced by mobile and IoT devices can also be stored on the edge storage system to be shared or processed in real time. However, unlike cloud servers, edge servers' storage resources are highly constrained due to their limited physical sizes [5]. This unique *capacity constraint* sets an upper bound on the performance of an ESS and the services deployed on the system. It is a major challenge that service providers have never encountered before in the cloud computing environment. Many approaches have been proposed in recent years to explore the potentials of ESSs under this constraint [6,14,19].

Reducing data redundancy in the ESS is an effective way to alleviate the capacity constraint. Shared by various application vendors, as well as mobile and IoT devices, an ESS is often subject to data redundancy. For example, the real-time communication between vehicles and edge servers can lead to a large number of duplicate video frames on the same or different edge servers in an ESS. Reducing data redundancy in the ESS by removing duplicate data can effectively save on the storage resources on the system. A similar problem named data deduplication has been investigated intensively in the context of cloud storage systems with the aim to maximize data redundancy reduction [11, 18]. However, this *cloud data deduplication* (CDDE) problem is fundamentally different from the *edge data deduplication* (EDDE) problem. To reduce data redundancy, most CDDE approaches first split the data stored on all the storage nodes in the system into multiple fine-grained chunks of a specific size, e.g., 4KB and 8KB. Then, they identify and remove duplicate data chunks across all those storage nodes. A user requesting a data can, from a metadata server, retrieve the locations of all the required data chunks for building the data. In the MEC environment, a user can only access its nearby edge servers directly, i.e., edge servers that cover the user [5]. This *proximity constraint* disables all the CDDE approaches because they commonly assume that a user can access any of the storage nodes in the system. In addition, the extra time taken to build a data from data chunks undermines MEC's pursuit of low data retrieval latency. Thus, unlike CDDE that reduces data redundancy at the data chunk level, EDDE aims to reduce data redundancy at the file level by removing duplicate data across edge servers in the system.

In recent years, researchers are beginning to investigate data deduplication in the MEC environment [8,9]. However, existing studies have followed the same idea and design as CDDE approaches. Making the same assumptions as CDDE

approaches, the approaches proposed in [8,9] cannot solve the EDDE problem in the real-world MEC environment for the same reasons discussed above. In addition, these approaches have failed to leverage the ability of edge servers to communicate and transmit data over the edge server network connecting the edge servers in the ESS, which has been widely acknowledged as a promising way to enable collaboration among edge servers [6,16,19]. To serve a user's data request, the requested data can be delivered to the user from an edge server multiple hops away over the edge server network under the latency constraint. Thus, an EDDE approach is urgently needed that reduces data redundancy in an ESS at the file level under the proximity constraint and the latency constraint.

This paper makes the first attempt to study the *Edge Data Deduplication* (EDDE) problem in realistic MEC environments, with the aim to maximize data deduplication ratio while fulfilling the proximity constraint and the latency constraint. Its major contributions include:

- We motivate the EDDE problem and present its fundamental differences from the traditional data deduplication problem in cloud storage systems.
- We formulate the EDDE problem as a constrained optimization problem and prove that it is $\mathcal{NP}$-hard.
- We propose an optimal approach named EDDE-O for solving small-scale EDDE problems based on integer programming, and an approximation approach named EDDE-A for solving large-scale EDDE problems efficiently with a proven $\ln \alpha + 1$-approximation ratio.
- We comprehensively evaluate the effectiveness and efficiency of EDDE-O and EDDE-A against four representative approaches through experiments conducted on a real-world dataset.

The remainder of this paper is organized as follows. Section 2 motivates the EDDE problem with an example. Section 3 formulates the EDDE problem and theoretically analyze its $\mathcal{NP}$-hardness. Section 4 presents EDDE-O and EDDE-A in detail. Section 5 shows the experimental results of EDDE-O and EDDE-A. Section 6 reviews the related work. Section 7 summarizes this paper and points out the future work.

2 Motivating Example

Video streaming services accounted for 75% of the total internet traffic in 2017, and this proportion is expected to increase to 82% by 2022 [10]. This emphasizes the importance of data deduplication for ESSs. Figure 1(a) presents an ESS comprised of 13 edge servers $\{s_1, s_2, ..., s_{13}\}$ deployed in a specific area, e.g., Melbourne CBD. Assuming that a popular video d[1] is stored on edge servers s_1, s_2, s_4, s_5, s_{11}, and s_{13} to serve the users within the area marked by the yellow line. This area is referred to as the *data coverage* hereafter. In this example, we assume that the application-specific latency constraint is two hops - the

[1] Multiple data can be deduplicated individually and independently.

video can be delivered to a user from an edge server within two hops over the edge server network. In real-world EDDE scenarios, the latency constraint is application-specific and and the communication latency between edge servers may not always be the same. To study the EDDE problem in a generic manner, the latency constraint is measured by the number of hops over the edge server network, similar to [6,16]. Our approaches can easily handle latency constraints measured in milliseconds easily.

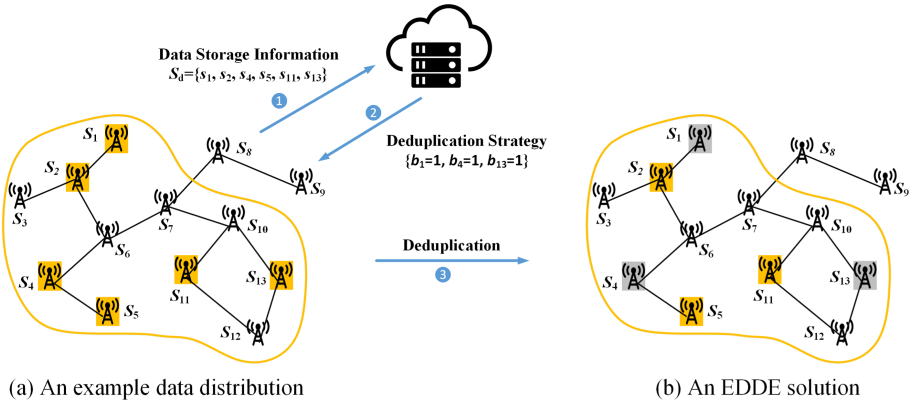


Fig. 1. Example EDDE scenario. In this example, data replicas are removed from s_1, s_4, and s_{13}. The data coverages before and after deduplication, as shown in (a) and (b), respectively, are the same.

As shown in Fig. 1(a), from the perspective of the edge infrastructure provider, e.g., T-Mobile or Amazon, this ESS does not need all the six video replicas to serve all the users within the data coverage. Some video replicas can be removed to save on system storage resources. Based on the data storage information collected from the system, an EDDE strategy can be formulated that indicates which video replicas can be removed. It will be sent to the edge servers for implementation. This process is *edge data deduplication* (EDDE). The latency constraint must not be violated - the system must still be able to deliver the video to all the users within the data coverage within 2 hops. For example, if we retain only one video replica on the system, say the one on s_1, and remove all the other video replicas, most of the users in the original data coverage will not able to retrieve the video within 2 hops. Specifically, the video can be delivered to serve only the users covered by s_1, s_2, s_3, and s_6. This EDDE solution is apparently not feasible. Figure 1(b) presents another EDDE solution that removes the data replicas on edge servers s_1, s_4, and s_{13} while keeping those on s_2, s_5, and s_{11}. As presented in Fig. 1(b), this solution offers the same data coverage as Fig. 1(a). The users within the data coverage can retrieve the video under the latency constraint. Compared with Fig. 1, the EDDE solution stores only three video replicas in the ESS, 50% fewer than Fig. 1(a). Apparently, EDDE can save

on system storage resources significantly. In the real world, the sizes of ESSs may be much larger, and there may be many possible EDDE solutions. Finding the optimal EDDE solution can save on the most system storage resources but may not be easy. An effective and efficient EDDE approach is needed.

3 Problem Statement

In this section, we formulate the EDDE problem and prove its hardness theoretically.

3.1 Problem Formulation

Let us model the n connected edge servers in an ESS as an undirected graph $G(S, E)$, where each edge server $s_i \in S$ is represented by a vertex in G and the link between two edge servers s_i and s_j is represented by an edge $e_{i,j}$ in G.

Let $S_d \subseteq S$ denote the set of edge servers where data d is stored and a_i is the binary variable indicating whether d is stored on edge server s_i:

$$a_i = \begin{cases} 0 & \text{if } d \text{ is not stored on } s_i,\ s_i \in S \\ 1 & \text{if } d \text{ is stored on } s_i\ ,\ s_i \in S \end{cases} \tag{1}$$

$$S_d = \{s_i |\ a_i = 1, s_i \in S\} \tag{2}$$

Let h denote the latency constraint, representing the maximum number of hops that data can be delivered from an edge server to a user over G. It is application-specific. A low h value indicates that a low latency is required. Let $N(s_i)$ denote the set of s_i' neighbor edge servers, i.e., those within h hops over G, and $\hat{S}_d$ ($S_d \subseteq \hat{S}_d \subseteq S$) denote the set of edge servers[2] that can retrieve d from S_d under the latency constraint:

$$N(s_i) = \{s_j |\ h_{ij} \leq h, s_j \in S\} \tag{3}$$

$$\hat{S}_d = \{N(s_i)\ |\ s_i \in S_d\} \tag{4}$$

Equation (3) is employed to identify s_i's neighbor edge servers when h is measured by the number of hops. If the latency constraint is measured in milliseconds, say 20 ms, Eq. (3) can be replaced with $N(s_i) = \{s_j |\ latency_i^j \leq 20, s_j \in S\}$, where $latency_i^j$ is the communication latency between s_i and s_j.

To represent an EDDE strategy B, let binary variable b_i denote whether d is removed from edge server $s_i \in S_d$ by B (Table 1):

$$b_i = \begin{cases} 0 & d \text{ not removed from } s_i,\ s_i \in S_d \\ 1 & d \text{ removed from } s_i,\ s_i \in S_d \end{cases} \tag{5}$$

[2] The edge server covering a user will retrieve a data from other edge servers if it does not have the data requested by the user. Thus, we refer to edge servers instead of users here for ease of exposition.

Table 1. Summary of notations

Notation	Description
a_i	Binary variable representing whether s_i has d
B	EDDE strategy
b_i	EDDE decision representing whether d is removed from s_i
d	Data to be deduplicated
E	Set of connections between edge servers
G	Graph representing connected edge servers in ESS
h_{ij}	Minimum hops from s_i to s_j
h	Latency constraint
$N(s_i)$	Set of neighbor edge servers of s_i under latency constraint
n	Number of edge servers in ESS
R	Deduplication ratio
S	Set of edge servers in ESS
S_d	Set of edge servers with d before deduplication
S_{d+}	Set of edge servers with d after deduplication
S_{d-}	Set of edge servers not with d after deduplication
$\hat{S}_d$	set of edge servers covered by S_d under latency constraint
$\hat{S}_{d+}$	Set of edge servers covered by S_{d+} under latency constraint
s_i	ith edge server in ESS

Let $S_{d+} \subseteq S_d$ denote the set of edge servers with d after d is deduplicated from S_d:

$$S_{d+} = \{s_i |\ b_i = 0, s_i \in S_d\} \tag{6}$$

Similar to S_{d+}, we employ $S_{d-} \subseteq S_d$ ($S_{d+} \cup S_{d-} = S_d$) to denote the set of edge servers where d is removed.

As illustrated and discussed in Sect. 2, over-deduplication will reduce the coverage area of S_d and stop some users from being able to retrieve d under the latency constraint. To ensure the same data coverage, the users that could retrieve data before data deduplication must also be able to retrieve it after data deduplication. This *coverage constraint* is defined below:

$$\hat{S}_d = \hat{S}_{d+} \tag{7}$$

The deduplication ratio produced by an EDDE strategy B, denoted by R, is calculated as follows:

$$R = 1 - \frac{\sum_{i=1}^{n} b_i}{\sum_{i=1}^{n} a_i} \tag{8}$$

The optimization objective of the EDDE problem, i.e., to maximize the data deduplication ratio under the latency constraint (3) and the coverage constraint

(7), can be expressed as follows:

$$\text{maximize } R \tag{9}$$

3.2 Problem Hardness

In this section, we prove the $\mathcal{NP}$-hardness of the EDDE problem by reducing it from the classical $\mathcal{NP}$-hard *uncapacitated facility location* (UFL) problem [3]. Given a weighted bipartite graph $G < F, C, E, W >$, where F represents the candidate locations for opening facilities, C represents the clients that need to be served by facilities, E represents the connections from clients to facilities, and W is the connection cost matrix from C to F. The UFL problem aims to find a set of locations, denoted as $F' \subseteq F$, for opening facilities with the minimum overall cost, including the cost of opening all the facilities in F' and the cost of connecting clients to F', while ensuring that all clients can be served. Let $cost(f)$ denote the cost of opening up a facility f. The formulation of this UFL problem can be expressed as follows:

$$\min(\sum_{f \in F'} cost(f) + \sum_{c \in C, f \in F'} x_{c,f} w_{c,f}) \tag{10}$$

$$s.t. \quad \sum_{f \in F'} w_{c,f} x_{c,f} \geq 1 \tag{11}$$

$$x_{c,f} \in \{0, 1\} \tag{12}$$

where $x_{c,f}$ is the connection decision from client c to opened facility f and $w_{c,f}$ is the cost of connecting client c to facility f.

Now we reduce the EDDE problem to the UFL problem: 1) removing edge servers not in S_d and the corresponding edges; 2) connecting each edge server and its neighbor edge servers within h hops; 3) setting the same cost of storing d on individual edge servers. This reduced EDDE problem can now be equally converted to minimize the storage cost, i.e., the cost of storing d in the system, while ensuring that all the edge servers can retrieve d within 1 hop. Since the cost of each edge is 0, the objective to maximize the data deduplication ratio in the EDDE problem is equivalent to selecting the fewest edge servers in S_d to minimize the storage cost, the same as Objective (10) in the UFL problem. Moreover, Constraint (7) is converted to cover all the edge servers in the reduced EDDE problem, equivalent to Constraint (11). Constraint (12) denotes whether client c can connect to the opened facility f. Thus, it is obvious that constraint (12) is equal to constraint (1).

In conclusion, any solution that satisfies the UFL problem can be reduced to the corresponding EDDE problem after the above discussion in polynomial time. Thus, the EDDE problem is $\mathcal{NP}$-hard.

4 EDDE Approaches

In this section, two approaches are proposed to solve the different scales of EDDE problem correspondingly.

4.1 Optimal Approach

The optimal solution to the EDDE problem must maximize the data deduplication ratio while fulfilling the same data coverage before and after deduplication under the latency constraint. As introduced in Sect. 3.1, S_d donates the set of edge servers that have data d before deduplication, and $b_i \in \{0, 1\}$ denotes whether d is removed from $s_i \in S_d$. Thus, this EDDE problem can be modeled as a *constrained optimization problem* (COP) as follows:

$$\max\ (1 - \sum_{s_i \in S_d} b_i / |S_d|) \tag{13}$$

$$h_{i,j} \leq h, \forall s_i \in S_d, s_j \in N(S_d) \tag{14a}$$

$$\cup_{\{b_i=0 | s_i \in S_d\}}\ N(s_i) = N(S_d) \tag{14b}$$

where constraint (14a) ensures the latency constraint and Constraint (14b) ensures the coverage constraint.

EDDE-O can be implemented by employing some classic integer programming solvers such as CPLEX[3] and Gurobi[4] for solving the COP presented above. The solution is an assignment of 0 or 1 to each b_i, where $s_i \in S_d$, that maximizes the data deduplication ratio (13) while fulfilling the latency constraint (14a) and the coverage constraint (14b). According to the solution, the data replicas are removed from the edge servers whose corresponding b_i values are 1.

4.2 Approximation Approach

Due to the $\mathcal{NP}$-hardness of the EDDE problem proven in Sect. 3.2, it is unrealistic to find the optimal solutions of large-scale EDDE problems. In such scenarios, it takes EDDE-O a lot of time to explore the possible solutions and find the optimal one. This can easily incur a significant delay in the implementation of edge data deduplication and lower the utilization of ESSs. Thus, this section introduces EDDE-A, an efficient approximation approach for finding sub-optimal solutions to large-scale EDDE problems efficiently. The pseudo-code of EDDE-A is presented in Algorithm 1.

In this algorithm, it first initializes the value of S_{d-}, S'_{d-}, the former for saving the EDDE solution and the latter for saving the set of candidate edge servers (Line 2). Then, it sets $R^{'}, R = 0$ to record the new deduplication ratio and the final deduplication ratio, respectively (Line 3). Next, the neighbor edge servers of $s_i (s_i \in S_d)$ can be obtained based on the latency constraint h, i.e., Eq. (3). Then, the algorithm sorts the edge servers in S_d by the number of their neighbor edge servers within h hops (Line 7). For all edge servers with the fewest neighbors, the algorithm obtains the one, denoted as s_{max}, with maximum $distance(s_j)$, i.e., the total distance from s_j to each of its neighbor edge servers in $N(s_j)$ (Lines

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

Algorithm 1. EDDE-A

Input: $G(S, E)$, S_d, h
Output: EDDE solution S_{d-};
1: **Initialization**:
2: $S_{d-}, S_{d-}^{'} \leftarrow \emptyset$
3: $R^{'}, R \leftarrow 0$
4: **End of initialization**
5: **while** $\hat{S}_{d+} \neq \hat{S}_d$ **do**
6: identify s_i's neighbor edge servers $N(s_i)$ with Eq. 3, for every $s_i \in S_d$
7: sort edge servers in S_d by $|N(s_i)|$ high to low;
8: **for** $s_j \in \arg\min_{s_i \in S_d} |N(s_i)|$ **do**
9: **for** each edge server $s_k \in N(s_j)$ **do**
10: $distance(s_j, N(s_j)) \leftarrow distance(s_j, N(s_j)) + d_{k,j}$
11: **end for**
12: **end for**
13: $s_{max} \leftarrow \arg\max\{distance(s_j, N(s_j)),\ s_j \in \arg\min_{s_i \in S_d} |N(s_i)|\ \}$
14: $S_{d-}^{'} \leftarrow S_{d-}^{'} \cup \{s_{max}\}$
15: $S_d \leftarrow S_d - s_{max}$
16: calculate $R^{'}$ with Eq. 8
17: **if** $R^{'} > R$ **then**
18: $R \leftarrow R^{'}$
19: $S_{d-} \leftarrow S_{d-}^{'}$
20: **end if**
21: **end while**
22: **return** S_{d-}

8–13). After that, s_{max} can be included into the set of candidate edge servers S'_{d-} and removed from S_d (Lines 14–15). Then, the new data deduplication ratio R' obtained by including s_{max} in S_{d-} can be calculated with Eq.(8) (Line 16). It will then be compared with the current data deduplication ratio R. If it is higher, it will replace R and S_{d-} is updated accordingly (Lines 18–21). The above process iterates until the coverage constraint is fulfilled, i.e., the set of edge servers covered by S_{d+} is equal to the set of edge servers covered by S_d (Line 5). Finally, S_{d-} is returned as the final EDDE solution. According to S_{d-}, an EDDE strategy B can be formulated by setting the corresponding $b_i = 1$ (if $\exists s_i \in S_{d-}$) or $b_i = 0$ otherwise.

Approximation Ratio. Now we analyze the approximation ratio and time complexity of EDDE-A theoretically. Let $S_{d-}^{'}(t)$ denote the set of candidate edge servers obtained by EDDE-A in the t^{th} iteration. According to Algorithm 1, whether an edge server s_j is included in $S_{d-}^{'}$ depends on $|N(s_j)|$, i.e., the number of its neighbor edge servers, and $distance(s_j, N(s_j))$, i.e., their distance from s_j. Thus, let us define $\beta_t = |N(S_{d-}^{'}(t))|/|S_{d-}^{'}(t)|$ to represent the average number of neighbor edge servers covered by each selected edge server in the t^{th} iteration. Let S_{d-}^{*} denote the optimal EDDE solution found by EDDE-O.

Compared with S^*_{d-}, the EDDE solution obtained by EDDE-A, denoted with S_{d-}, will not be able to remove more data replicas:

$$\frac{1}{\beta_t} \leq \frac{|S^{'}_{d-}(1)|}{|N(S^{'}_{d-}(t))|} \leq \frac{|S^{'}_{d-}(t)|}{|N(S^{'}_{d-}(t))|} \leq \frac{|S^*_{d-}|}{|N(S^{'}_{d-}(t))|} \tag{15}$$

Let $|S_{d-}|$ denote the number of data replicas removed by EDDE-A. After the final iteration of Algorithm 1, the number of data replicas removed by EDDE-A follows:

$$\begin{aligned} |S_{d-}| \leq & \frac{1}{\beta_1}(|N(S^{'}_{d-}(1))| - |N(S^{'}_{d-}(0))|) + \frac{1}{\beta_2}(|N(S^{'}_{d-}(2))| - |N(S^{'}_{d-}(1))|) \\ & + ... + \frac{1}{\beta_\alpha}(|N(S^{'}_{d-}(t))| - |N(S^{'}_{d-}(t-1))|) \end{aligned} \tag{16}$$

Based on Eq. (15) and Eq. (16), we can infer the following:

$$\begin{aligned} |S_{d-}| \leq & \frac{|N(S^{'}_{d-}(1))| - |N(S^{'}_{d-}(0))|}{|N(S^{'}_{d-}(1))|}|S^*_{d-}| + \frac{|N(S^{'}_{d-}(2))| - |N(S^{'}_{d-}(1))|}{|N(S^{'}_{d-}(2))|}|S^*_{d-}| \\ & + ... + \frac{|N(S^{'}_{d-}(t))| - |N(S^{'}_{d-}(t-1))|}{|N(S^{'}_{d-}(t))|}|S^*_{d-}| \end{aligned} \tag{17}$$

Let α denote the maximum number of iteration, i.e., $\alpha = |S_d|$. Based on mathematical induction, we can obtain Eq. (18):

$$|S_{d-}| \leq (\ln \alpha + 1)|S^*_{d-}| \tag{18}$$

Based on Eq. (18), we can find the approximation ratio of EDDE-A as follows:

$$\frac{R}{R^*} = \frac{|S_{d-}|/|S_d|}{|S^*_{d-}|/|S_d|} \leq \frac{(\ln \alpha + 1)|S^*_{d-}|}{|S^*_{d-}|} \leq \ln \alpha + 1 \tag{19}$$

Therefore, the approximation ratio of EDDE-A is $\ln \alpha + 1$.

Computation Complexity. Given an EDDE scenario with n edge servers $S = \{s_1, s_2, ..., s_n\}$, Algorithm 1 takes at most $O(n)$ time to find the edge servers with minimum $|N(s_i)|$ in Line 6. Then, in Lines 7–12, the algorithm selects an edge server from these edge servers based on their distance from their neighbor edge servers. The distance calculation in Line 8–9 takes $O(n^2)$ time in the worst case because the maximum number of edge servers in any $N(s_j)$ ($s_j \in S_d$) is $n-1$. Thus, the overall computation complexity of EDDE-A is $O(n^2)$.

5 Evaluation

In this section, the experiments are conducted to comprehensively evaluate our proposed two approaches, i.e., EDDE-O and EDDE-A.

5.1 Experimental Settings

Dataset. To evaluate the approaches realistically, we conduct the experiments on a widely-used real-world dataset[5] [7], which contains 1,464 edge servers with their geographic coordinates in Melbourne, Australia.

Competing Approaches. EDDE-O and EDDE-A are evaluated against the following four approaches:

- **Random:** This approach randomly removes data replicas from edge servers, one after another, until no more data replicas can be removed without violating the latency constraint or the coverage constraint.
- **Greedy:** This greedy-based approach always removes data replicas from edge servers with the fewest neighbor edge servers, one after another, until no more data replicas can be removed without violating the latency constraint or the coverage constraint.
- **EF-dedup** [9]**:** This approach originates from [9] and is adapted in the context of EDDE to remove data replicas instead of duplicate data chunks. It first creates $|S_d|$ clusters, each comprised of the neighbor edge servers of an edge server in S_d within h hops. Then, it removes data replicas within those clusters until there is one data replica within each of the clusters.
- **TSC21** [14]**:** The *edge data caching* (EDC) problem studied in [14] is slightly similar to the EDDE problem. This approach finds edge servers for storing data replicas, aiming to minimize the number of data replicas for fulfilling the latency constraint under the capacity constraint.

Parameter Settings. A set of small-scale experiments (Set #1) and a set of large-scale experiments (Set #2) are conducted. The parameter settings in the experiments are summarized in Table 2. All the experiments are conducted on a machine equipped with Intel Core i5-8400 processor (8 cores, 8 threads) and 8 GB RAM, running Windows-10. When the value of each of the following four setting parameters varies, the experiments are repeated for 200 times and the averaged value is reported.

- **Data redundancy rate (θ):** This parameter is the redundancy of data d in the ESS. Studies find that the redundancy of IoT data, e.g., multimedia and traffic video sequences is generally up to 70% [17,20]. Thus, the value of θ varies from 30% to 80% in both Set #1 and Set #2.
- **Number of edge servers (n):** This parameter decides the scale of the ESS, increasing from 10 to 30 in steps of 5 in Set#1.2, from 50 to 250 in steps of 50 in Set #2.2.
- **Edge server density (ds):** Defined as $ds = |E|/n$, this parameter is the density of the graph that represents the edge servers in the ESS. It varies from 1.0 to 2.5 in steps of 0.3 in Set #1.3, from 2.0 to 5.0 in steps of 0.6 in Set #2.3.

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

– **Latency constraint (h):** This parameter enforces the latency constraint, increasing from 1 to 5 in steps of 1 in both Set #1 and Set #2.

Table 2. Parameter settings

	θ	n	ds	h
Set # 1.1	30%, 40%, ..., 80%	20	1.0	1
Set # 1.2	60%	10, 15, ..., 30	1.0	1
Set # 1.3	60%	20	1.0, 1.3, ..., 2.5	1
Set # 1.4	60%	20	1.0	1, 2, ..., 5
Set # 2.1	30%, 40%, ..., 80%	150	2.0	1
Set # 2.2	60%	50, 100, ...,250	2.0	1
Set # 2.3	60%	150	2.0, 2.6, ..., 5.0	1
Set # 2.4	60%	150	2.0	1, 2, ..., 5

Performance Metrics

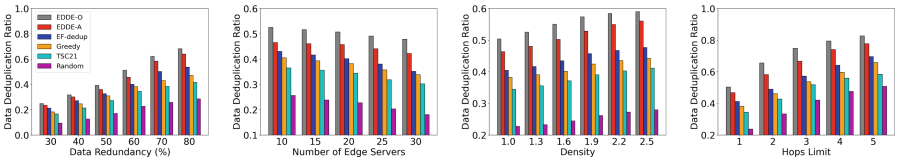
– **Data deduplication ratio (R),** calculated with (8), the higher the better.
– **Computation time,** measured by the CPU computation time that taken to find the EDDE solution by an approach, the lower the better.

5.2 Experimental Results

Effectiveness. Figures 2 and 3 show the effectiveness of the approaches in Set #1 and Set #2, respectively. Figure 2 shows that EDDE-O and EDDE-A achieve the highest and the second highest data deduplication ratios among all six approaches. Second to only EDDE-O with an average performance gap of only 8.68% across all the experiments in Set #1, EDDE-A outperforms EF-dedup, Greedy, TSC21, and Random by an average of 7.82%, 10.33%, 16.24%, and 24.87% in maximizing the data deduplication ratio. Figure 3 demonstrates EDDE-A's superior performance in maximizing data deduplication ratios in Set #2, which is 9.47%, 16.71%, 20.34%, and 32.03% higher on average than EF-dedup, Greedy, TSC21, and Random, respectively.

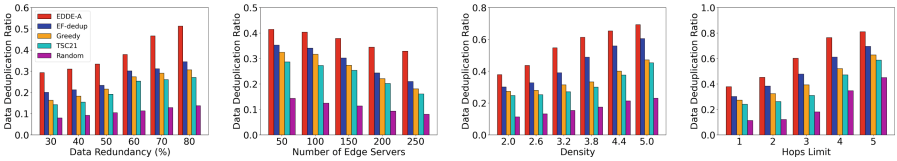
Figures 2(a) and 3(a) demonstrate the impact of data redundancy (θ) on data deduplication ratio in Set #1.1 and Set #2.1. Given a fixed number of edge servers in the ESS, a larger θ grows the number of data replicas on the ESS and the *data density* measured by the ratio of edge servers in the system with data replicas. This immediately increases the number of data replicas that can be removed without violating the latency constraint or the coverage constraint. For example, if any adjacent edge servers have duplicate data, one of them can be removed. Thus, the data deduplication ratios achieved by all approaches

increases. Figures 2(b) and 3(b) demonstrate the impact of the number of edge servers (n) on data deduplication ratio in Set #1.2 and Set #2.2. Given a fixed data redundancy rate, a larger n will further distribute data replicas across the edge servers in the ESS. This decreases the data density in the system, making it harder to remove data replicas without violating some constraints, i.e., the latency constraint and the coverage constraint. For example, data replicas are less likely to be found on adjacent edge servers. Thus, the data deduplication ratios of all approaches decrease when n increases, opposite to the impact of θ shown in Figs. 2(a) and 3(a). Figures 2(c) and 3(c) depict the results in Set #1.3 and Set #2.3 where edge server density ds varies. When the edge server density ds increases, the data deduplication ratios produced by the six approaches increase. A larger ds connects each individual edge server to connect to more other edge servers in the system. The data stored on an edge server can be delivered to users over the edge server network under the latency constraint. This indicates the importance of leveraging edge servers' ability to communicate and collaborate. Figures 2(d) and 3(d) show the impact of latency constraint (h) on data deduplication ratio in Set #1.4 and Set #2.4. As h increases, the latency constraint is relaxed. Users can retrieve data from edge servers further away. This reduces the number of data replicas needed in the system to accommodate all the users within the data coverage. Thus, more data replicas can be removed, and the average deduplication ratios produced by all approaches increase accordingly.



(a) Set # 1.1 (b) Set # 1.2 (c) Set # 1.3 (d) Set # 1.4

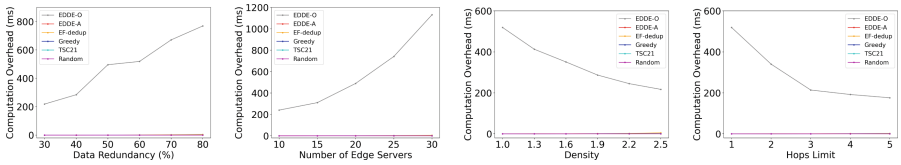
Fig. 2. Effectiveness evaluation in Set #1



(a) Set # 2.1 (b) Set # 2.2 (c) Set # 2.3 (d) Set # 2.4

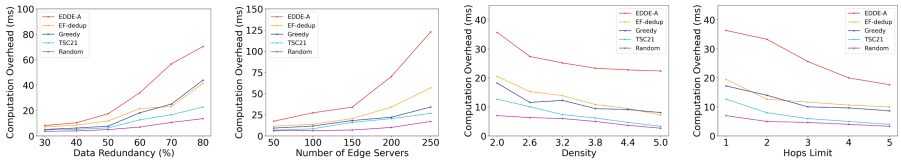
Fig. 3. Effectiveness evaluation in Set #1

Efficiency. Figures 4 and 5 demonstrate the efficiency of all approaches in Set #1 and Set #2, respectively. Figure 4 illustrates the high computation time obtained by EDDE-O in Set #1 that renders those of other approaches negligible. This high computational overheads validate the EDDE's $\mathcal{NP}$-hardness proved in Sect. 3.2. This tells us that EDDE-O is indeed not suitable for solving large-scale EDDE scenarios. Compared with EDDE-O, EDDE-A is much more efficient in solving large-scale EDDE problems. In Set #1, it takes only 1.27 ms on average to find a solution, only 0.16% of what EDDE-O takes. Please note that EDDE-O is excluded from Set #2 because it cannot find a solution within a reasonable amount of time in such large-scale EDDE scenarios. In Fig. 5, EDDE-A always takes more computation time for finding an EDDE solution than the other four competing approaches, specifically, 14.67 ms, 19.72 ms, 24.29 ms, and 28.43 ms more than EF-dedup, Greedy, TSC21, and Random, respectively. Overall, EDDE-A scales with θ and n, taking no more than 125 ms to find a solution in Set #2. Given its outstanding advantages in maximizing data deduplication ratios over EF-dedup, Greedy, TSC21, and Random, its extra computational overhead is worthwhile in most large-scale EDDE scenarios.



(a) Set # 1.1 (b) Set # 1.2 (c) Set # 1.3 (d) Set # 1.4

Fig. 4. Efficiency evaluation in Set #1



(a) Set # 2.1 (b) Set # 2.2 (c) Set # 2.3 (d) Set # 2.4

Fig. 5. Efficiency evaluation in Set #1

6 Releated Work

A large amount of data are being produced by mobile and IoT devices at the network edge, e.g., images, video frames, and locality data [12]. It has become

a trend for application vendors to cache popular data on edge servers to reduce the cost and latency incurred by transmitting data from the cloud to the network edge [16]. However, the constrained storage resources on edge servers are a major challenge to explore the potentials of edge storage systems comprised of networked edge servers [6,14,19]. Reducing data redundancy within an edge storage system can save up to 70% storage resources overall [8,17]. This can be achieved through data deduplication.

Cloud Data Deduplication. (CDDE) has been extensively studied for cloud storage systems [4,11,18]. To name a few, Dubnicki et al. [4] proposed a CDDE approach capable of deduplicating data at the data chunk level across multiple data centers based on an improved distributed hash table. Yan et al. [18] proposed a novel data deduplication approach named Z-Dedup. Z-Dedup can monitor and remove redundancy at chunk-level in compressed back-up data by exploiting some invariant information contained in the metadata compressed data. Unlike most data deduplication studies that focus on back-up data, Meister et al. [11] proposed to deduplicate data for online file systems in HPC centers with chunking strategies specifically designed based on HPC applications' data characteristics. Based on research on data deduplication, cloud service providers like Amazon and Microsoft have offered and deployed data deduplication services for their cloud storage servers [1,2].

However, specifically designed for conventional cloud storage systems, these *cloud data deduplication* (CDDE) techniques are not suitable to directly employ in edge storage systems due to the unique characteristics of the MEC environment, particularly, edge servers' geographic distribution, limited coverage, and constrained resources. In recent years, researchers are starting to investigate data deduplication in edge storage systems [8,9]. Specifically, Li et al. [9] formulated the data deduplication problem at the network edge as a clustering optimization problem. They proposed an approximate algorithm for partitioning edge servers into disjoint clusters so that CDDE approaches can be employed to deduplicate data within individual clusters. In their subsequent study [8], another approximation algorithm was proposed to take data popularity into account. However, these studies have followed the same idea of CDDE and failed to consider the unique characteristics that differ edge storage systems from cloud storage systems fundamentally, in particular, the capacity constraint, proximity constraint, and latency constraint discussed in Sect. 1 and widely acknowledged in state-of-the-art studies of MEC [5,7,16]. To facilitate EDDE, this paper makes the first attempt to motivate, model, and solve the EDDE problem with consideration of the unique characteristics of the MEC environment.

7 Conclusion and Future Work

In this paper, we formulated the *novel edge data deduplication* (EDDE) problem in the MEC environment as a constrained optimization problem. We proved that it is $\mathcal{NP}$-hard and proposed two EDDE approaches. The first one is named EDDE-O and finds optimal solutions to small-scale EDDE problems based on

integer programming. The other one is named EDDE-A and finds approximate solutions to large-scale EDDE problems efficiently. The results of extensive experiments conducted on a widely-used real-world dataset demonstrate that EDDE-O and EDDE-A can solve the EDDE problem effectively and efficiently, outperforming four representative approaches significantly.

This research has first motivated the importance to deduplicate redundancy in ESSs by fully exploring the characteristic of the MEC environment. As for further works, we will attempt to devise lightweight mechanisms for detecting data duplication and dynamic data deduplication.

Acknowledgement. We thank the anonymous reviewers for their helpful feedback. This work is supported by National Science Foundation of China under grant No.62032008.

References

1. https://docs.aws.amazon.com/fsx/latest/windowsguide/using-data-dedup.html
2. https://docs.microsoft.com/en-us/windows-server/storage/data-deduplication/overview
3. Chudak, F.A., Shmoys, D.B.: Improved approximation algorithms for the uncapacitated facility location problem. SIAM J. Comput. **33**(1), 1–25 (2003)
4. Dubnicki, C., et al.: Hydrastor: a scalable secondary storage. In: Proceedings of 7th USENIX Conference on File and Storage Technologies, vol. 9, pp. 197–210 (2009)
5. He, Q., et al.: A game-theoretical approach for user allocation in edge computing environment. IEEE Trans. Parallel Distrib. Syst. **31**(3), 515–529 (2019)
6. He, Q., et al.: A game-theoretical approach for mitigating edge DDoS attack. IEEE Trans. Dependable Secure Comput. 1 (2021). https://doi.org/10.1109/TDSC.2021.3055559
7. Lai, P., et al.: Optimal edge user allocation in edge computing with variable sized vector bin packing. In: Pahl, C., Vukovic, M., Yin, J., Yu, Q. (eds.) ICSOC 2018. LNCS, vol. 11236, pp. 230–245. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03596-9_15
8. Li, S., Lan, T.: Hotdedup: managing hot data storage at network edge through optimal distributed deduplication. In: Proceedings of 39th IEEE Conference on Computer Communications, pp. 247–256 (2020)
9. Li, S., Lan, T., Balasubramanian, B., Ra, M.R., Lee, H.W., Panta, R.: Ef-dedup: enabling collaborative data deduplication at the network edge. In: Proceedings of 39th IEEE International Conference on Distributed Computing Systems, pp. 986–996. IEEE (2019)
10. Li, T., Braud, T., Li, Y., Hui, P.: Lifecycle-aware online video caching. IEEE Trans. Mob. Comput. **20**, 2624–2636 (2020)
11. Meister, D., Kaiser, J., Brinkmann, A., Cortes, T., Kuhn, M., Kunkel, J.: A study on data deduplication in HPC storage systems. In: Proceedings of International Conference on High Performance Computing, Networking, Storage and Analysis, pp. 1–11 (2012)
12. Shinkuma, R., Nishio, T., Inagaki, Y., Oki, E.: Data assessment and prioritization in mobile networks for real-time prediction of spatial information using machine learning. EURASIP J. Wirel. Commun. Netw. **2020**(1), 1–19 (2020). https://doi.org/10.1186/s13638-020-01709-1

13. Xia, X., et al.: Budgeted data caching based on k-median in mobile edge computing. In: Proceedings of 27th IEEE International Conference on Web Services, pp. 197–206. IEEE (2020)
14. Xia, X., Chen, F., Grundy, J., Abdelrazek, M., Jin, H., He, Q.: Constrained app data caching over edge server graphs in edge computing environment. IEEE Trans. Serv. Comput. 1 (2021). https://doi.org/10.1109/TSC.2021.3062017
15. Xia, X., et al.: Graph-based optimal data caching in edge computing. In: Proceedings of 17th International Conference on Service-Oriented Computing, pp. 477–493 (2019)
16. Xia, X., Chen, F., He, Q., Grundy, J.C., Abdelrazek, M., Jin, H.: Cost-effective app data distribution in edge computing. IEEE Trans. Parallel Distrib. Syst. **32**(1), 31–44 (2020)
17. Yan, H., Li, X., Wang, Y., Jia, C.: Centralized duplicate removal video storage system with privacy preservation in IoT. Sensors **18**(6), 1814 (2018)
18. Yan, Z., Jiang, H., Tan, Y., Skelton, S., Luo, H.: Z-dedup: a case for deduplicating compressed contents in cloud. In: Proceedings of 33rd IEEE International Parallel and Distributed Processing Symposium, pp. 386–395 (2019)
19. Yuan, L., et al.: Coopedge: a decentralized blockchain-based platform for cooperative edge computing. In: Proceedings of the 30th Web Conference (2021)
20. Zhang, Y., Wu, Y., Yang, G.: Droplet: a distributed solution of data deduplication. In: Proceedings of 13th ACM/IEEE International Conference on Grid Computing, pp. 114–121 (2012)