Enabling Balanced Data Deduplication in Mobile Edge Computing

Ruikun Luo, Hai Jin, *Fellow, IEEE*, Qiang He, *Senior Member, IEEE*, Song Wu, *Member, IEEE*, and Xiaoyu Xia, *Member, IEEE*

***Abstract*—In the *mobile edge computing* (MEC) environment, edge servers with storage and computing resources are deployed at base stations within users' geographic proximity to extend the capabilities of cloud computing to the network edge. *Edge storage system* (ESS), is comprised by connected edge servers in a specific area, which ensures low-latency services for users. However, high data storage overheads incurred by edge servers' limited storage capacities is a key challenge in ensuring the performance of applications deployed on an ESS. Data deduplication, as a classic data reduction technology, has been widely applied in cloud storage systems. It also offers a promising solution to reducing data redundancy in ESSs. However, the unique characteristics of MEC, such as edge servers' geographic distribution and coverage, render cloud data deduplication mechanisms obsolete. In addition, data distribution must be balanced over edge storage systems to accommodate future data demands, which cannot be undermined by data deduplication. Thus, *balanced edge data deduplication* (BEDD) must consider deduplication ratio, data storage benefits, and resource balance systematically under the latency constraint. In this article, we model the novel BEDD problem formally and prove its $\mathcal{NP}$-hardness. Then, we propose an optimal approach for solving the BEDD problem exactly in small-scale scenarios and a sub-optimal approach to solve large-scale BEDD problems with a theoretical performance guarantee. Extensive and comprehensive experiments conducted on a real-world dataset demonstrate the significant performance improvements of our approaches against four representative approaches.**

***Index Terms*—Data deduplication, edge storage system, mobile edge computing, optimization problem, storage resource balance.**

Manuscript received 25 February 2022; revised 15 November 2022; accepted 14 February 2023. Date of publication 22 February 2023; date of current version 16 March 2023. This work was supported in part by the National Key Research and Development Program under Grant 2022YFB4500704 and in part by the National Science Foundation of China (NSFC) under Grant 62032008. Recommended for acceptance by M. Zhao. *(Corresponding author: Qiang He.)*

Ruikun Luo, Hai Jin, and Song Wu are with the National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: rkluo@hust.edu.cn; hjin@mail.hust.edu.cn; wusong@hust.edu.cn).

Qiang He is with the National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China, and also with the Department of Computing Technologies, Swinburne University of Technology, Melbourne, VIC 3122, Australia (e-mail: qhe@swin.edu.au).

Xiaoyu Xia is with the School of Computing Technologies, RMIT University, Melbourne, VIC 3000, Australia (e-mail: xiaoyushaw@gmail.com).

Digital Object Identifier 10.1109/TPDS.2023.3247061

I. Introduction

The data produced by mobile and smart devices have grown exponentially in the last decade. Transmitting this huge amount of data to the cloud for processing consumes excessive network resources and incurs heavy network traffic. Meanwhile, the traditional centralized cloud computing architecture is failing to fulfill users' increasing low latency requirements, especially for latency-sensitive applications such as autonomous driving, AR, and VR [1], [2]. *Mobile edge computing* (MEC), as a new distributed computing paradigm, pushes cloud-like functionalities and resources to the network edge to provide users with low-latency access to applications and data on edge servers [3], [4].

Edge storage system (ESS), as an infrastructure to support edge computing-enabled applications, is comprised of connected edge servers deployed in an area [5]. Fig. 1 illustrates an ESS comprises of four connected edge servers $\{s_1, \ldots, s_4\}$ storing data $\{d_1, d_2, \ldots, d_5\}$ to serve the users in the system. Application vendors can store popular data on edge servers to reduce the latency in their users' access to these data and save the expenses incurred by transmitting large amounts of data from the cloud to their users [5], [6]. Data produced by latency-sensitive and energy-limited IoT and mobile devices can also be stored on an ESS locally for sharing or processing. Unfortunately, edge servers' storage resources are significantly limited by their small physical sizes [1], which is one of their fundamental differences from cloud servers. This *capacity constraint* limits the amount of data that can be stored on an ESS, and consequently impacts the performance of the ESS and the applications deployed on the ESS. Lots of studies have attempted to mitigate this constraint by leveraging the collaboration between edge servers [7], [8].

In the realistic MEC environment, edge servers are geographically distributed [9]. Accordingly, the data stored on edge servers, such as traffic data, shopping mall ads, often exhibit and share the same geographic characteristics. This results in data redundancy in an ESS [10], [11]. The same files generated by different users or different updates may be stored on multiple edge servers in the ESS. For example, in Fig. 1, data d_3 is stored on edge servers s_2, s_3, and s_4. As reported by [10] and [11], the similarity between users' demands on IoT and mobile data can reach up to 70%. Caching such data on edge servers without deduplication leads to significant data redundancy and storage wastes across edge servers. Given edge servers' limited storage capacities, how to reduce data redundancy is of tremendous importance in improving storage utilization on edge servers.

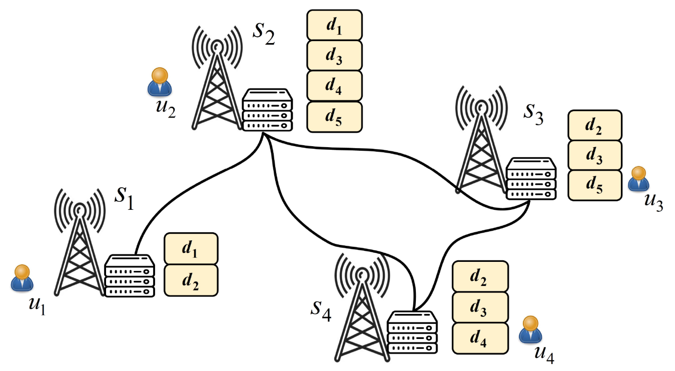


Fig. 1. Example of edge storage system. In this example, we assume that every user requests all five data. In this way, we keep the number of users minimum to present a concise and clear scenario.

An edge server can access data from other neighbor edge servers over the edge server network to serve the users within its coverage area while not violating the *latency constraint* [5], [12]. Take Fig. 1 for example and suppose that the latency constraint is one hop.[1] Edge servers s_1, s_3, and s_4 can retrieve d_1 from s_1 or s_2 to serve the users within their coverage. Thus, d_1 can be removed from s_1 to reduce data redundancy and save on storage resources. This is the foundation for *edge data deduplication* (EDD). The problem of edge data deduplication is crucial and practical because redundant data need to be removed to release edge servers' constrained storage resources. For example, content providers like YouTube and TikTok can cache videos on edge servers to fulfill users' data demands with low data retrieval latency. Many users share the same demands for popular videos at the network edge, as described in the literature [13]. Note that EDD must not violate the latency constraint - the system must still keep the ability to deliver data to corresponding users within the latency constraint. Take Fig. 1 for example, supposing that only one replica of d_2 can be retained on s_1 in the ESS and all the other replicas of d_2 are removed, under the latency constraint, mobile users u_3 and u_4 will not be able to retrieve d_2. This EDD solution is invalid.

Data deduplication has been widely employed to reduce data redundancy in central cloud storage systems [14]. However, edge data deduplication is totally different from the *cloud data deduplication* (CDD) because of MEC's unique characteristics. CDD approaches deduplicate data at the chunk level. The general idea is to split data into multiple fine-grained data chunks of fixed or variable size and then remove redundant data chunks based on chunk fingerprints. When a data request comes, a metadata server rebuilds the data based on unique chunks retrieved from different storage nodes [15]. The expensive time overhead incurred by rebuilding data from data chunks conflicts with the requirements of low data retrieval latency in the MEC environment. Therefore, EDD aims to remove duplicate data at the file level rather than the chunk level. In addition, data retrieval between edge servers must not violate the latency constraint - an edge server can only retrieve data from edge servers within its latency limitation, i.e., their nearby edge servers in the ESS [1], [16], [17]. Thus, EDD must ensure that after data deduplication, all the users can still retrieve requested data under the latency constraint.

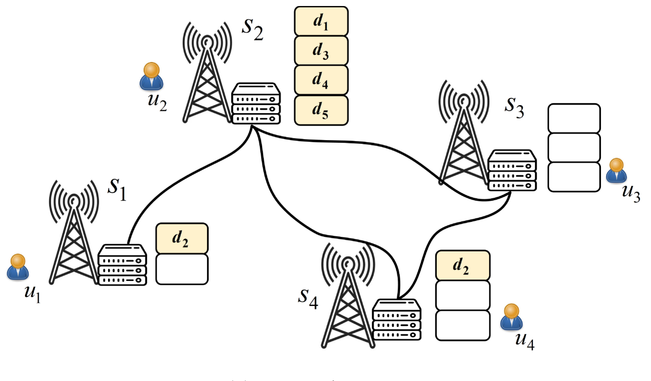
1.Latency constraint can also be specified by milliseconds, e.g., 50 ms. Here, we use hops for ease of exposition.

EDD must also balance data storage across edge servers. If an EDD approach pursues the sole objective of maximizing the deduplication ratio like CDD approaches [18], [19], it will tend to keep the data stored on edge servers that can serve the most users under the latency constraint and remove as many duplicates as possible from other edge servers. Fig. 2(a) illustrates such an EDD solution p_1 to Fig. 1. We can see that all the data on s_2 remain and all the duplicates are removed from s_1, s_3, and s_4. Such an EDD approach may overwhelm some edge servers, e.g., s_2 in Fig. 2(a) while others are underutilized over time, e.g., s_3 in Fig. 2(a). No more data can be stored on overwhelmed edge servers to accommodate future data demands. For example, in Fig. 2(a), a new data d_6 will have to be stored on $\{s_1, s_3\}$ or $\{s_1, s_4\}$ to satisfy any user's data retrieval requests in the system. Thus, EDD must consider both the data deduplication ratio and storage space balance across edge servers. Fig. 2(b) presents a solution p_2 that achieves the same deduplication ratio as p_1. It also achieves the same *data coverage* as p_1 - all the users can still access all the data in the system. Compared with p_1, p_2 balances data storage across the four edge servers so that they have spare storage to accommodate future data demands.

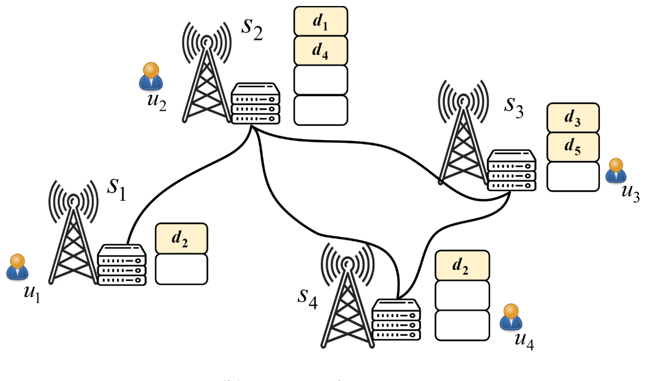
Some researchers have attempted to balance data storage across distributed nodes [20], [21]. The key idea is to measure storage space balance with a fairness index and maximize that index by evening data storage across the nodes. Unfortunately, this does not work with EDD without considering data storage benefits. Data popularity, as a significant metric in the MEC environment, varies at different locations [22]. Storing data on edge servers that can serve the most users with low latency will produce the highest data storage benefit [7]. If we take a look at Fig. 1, we can see that in many cases, storing data on s_2 tends to produce high data storage benefits because it is close to all other edge servers. Thus, EDD must not simply maximize a storage fairness index as [20], [21] without considering data storage benefits.

To summarize, EDD must consider the deduplication ratio, data storage benefits, and storage space balance jointly, as well as the latency constraint. This is challenging, and even more so in realistic EDD scenarios larger and more complex than the one presented in Fig. 1. In this paper, we study this new *balanced edge data deduplication* (BEDD) problem. Our contributions are summarised as follows:

- We motivate the BEDD problem and point out its fundamental differences from the CDD problem and the EDD problem.
- We formulate the BEDD problem comprehensively and prove its $\mathcal{NP}$-hardness theoretically.
- We design two approaches, one named BEDD-O and the other named BEDD-A. BEDD-O solves small-scale BEDD problems optimally based on integer programming. BEDD-A solves large-scale BEDD problems efficiently based on Lagrange relaxation and an improved subgradient method.
- We conduct comprehensive experiments on a wide-used EUA dataset to test the performance of BEDD-O and BEDD-A against four representative approaches.



(a) EDD solution p_1



(b) BEDD solution p_2

Fig. 2. Example of EDD and BEDD solutions with the same deduplication ratio of 0.5, and same data coverage as Fig. 1.

TABLE I
SUMMARY OF NOTATIONS

Notation	Definition
$\mathcal{B}_p$	data storage benefits produced by BEDD strategy p
D	set of data to be deduplicated
d_i	data i to be deduplicated, $d_i \in D$
h	latency constraint
$h_{k,j}$	minimum hops between s_k and s_j
$\mathcal{L}_p$	storage resource balance index produced by BEDD strategy p
ms_j	maximum storage space of edge server s_j
$N(s_j)$	s_j's neighbor edge servers while fulfilling the latency constraint
$\mathcal{O}_j$	storage resource occupied ratio of edge server s_j
p	BEDD strategy
$\mathcal{R}_i$	deduplication ratio of data d_i
$\mathcal{R}_p$	deduplication ratio produced by BEDD strategy p
r_i^j	binary variable representing whether d_i is removed from s_j
S	edge servers in edge storage system
S_{d_i}	edge servers that stores d_i before deduplication
$S_{d_i}^+$	edge servers that stores d_i after deduplication
$\hat{S}_{d_i}$	edge servers covered by S_{d_i}
$\hat{S}_{d_i}^+$	edge servers covered by $S_{d_i}^+$
s_j	jth edge server in ESS
U	users in the area

II. Problem and Model Formulation

In this section, we first formulate the BEDD problem and then prove its hardness theoretically. The main notations and their definitions are summarized in Table I.

A. Edge Data Deduplication Model

We consider an ESS denoted as $S = \{s_1, \ldots, s_n\}$, comprised of n edge servers in a specific geographic area. A set of data, denoted as $D = \{d_1, \ldots, d_m\}$, is stored on these edge servers. Let a binary variable r_j^i denote the deduplication decision, where $r_i^j = 1$ means that d_i is removed from s_j. Let h denote the latency constraint. In real-world MEC scenarios, the transmission latency between edge servers could be different. To generalize the models and approaches presented in this paper, we measure the transmission constraint by the number of hops over the edge server network, which can also be easily measured by specific milliseconds. Let S_{d_i} represent the set of edge servers that store data d_i, $\hat{S}_{d_i}$ $(S_{d_i} \subseteq \hat{S}_{d_i} \subseteq S)$ denote the data coverage of d_i:

$$\hat{S}_{d_i} = \{s_k | h_{k,j} \leq h, s_k \in S, s_j \in S_{d_i}\} \tag{1}$$

where $h_{k,j}$ is the minimum latency between s_j and s_k.

Let us employ $S_{d_i}^+$ and $S_{d_i}^- \subseteq S_{d_i}$ $(S_{d_i}^+ \cup S_{d_i}^- = S_{d_i})$ to represent the set of edge servers that still have d_i and those whose d_i are removed from S_{d_i} after deduplication, respectively:

$$S_{d_i}^+ = \{s_j |\, r_i^j = 0, s_j \in S_{d_i}\}$$
$$S_{d_i}^- = \{s_j |\, r_i^j = 1, s_j \in S_{d_i}\} \tag{2}$$

As discussed in Section I, an EDD solution must retain the same data coverage area, i.e., users covered by S_{d_i} before deduplication can still retrieve data d_i after deduplication within the data retrieval latency limitation. This is named *coverage constraint* :

$$\hat{S}_{d_i} = \hat{S}_{d_i}^+ \tag{3}$$

The deduplication ratio is the ratio of the number of edge servers whose d_i are removed to the number of edge servers that store d_i before deduplication. Thus, for each data d_i, the deduplication ratio produced by a BEDD strategy p, can be defined as $\mathcal{R}_i$:

$$\mathcal{R}_i = \frac{\sum_{s_j \in S} r_i^j}{|S_{d_i}|} \tag{4}$$

The overall deduplication ratio achieved by the BEDD strategy p is calculated as follow:

$$\mathcal{R}_p = \sum_{d_i \in D} \mathcal{R}_i = \sum_{d_i \in D} \frac{\sum_{s_j \in S} r_i^j}{|S_{d_i}|} \tag{5}$$

B. Data Storage Benefit Model

By storing data d_i, an edge server s_j produces data storage benefit by serving d_i to the users within h. When it is removed from s_j by a BEDD strategy p, some users may have to retrieve d_i from other edge servers with higher latency. For example, if d_2 is removed from s_1 and s_3 in Fig. 1, user u_1 has to retrieve d_2 from s_4 via two hops. This reduces the overall data storage benefit produced by the system. Let $\mathcal{B}_{u,d_i}$ denote the storage benefit produced by serving user u with d_i. It can be defined as follow:

$$\mathcal{B}_{u,d_i} = \max\{(h - h_{j,k}) \cdot cov(s_j), 0\} \tag{6}$$

where $cov(s_j)$ denotes the number of users covered by s_j, s_j is the edge server that covers u, and s_k is the edge server that stores d_i.

The data storage benefits produced by a data deduplication strategy p is calculated as follow:

$$\overline{\mathcal{B}}_p = \sum_{s_j \in S} \sum_{d_i \in D} \mathcal{B}_{u,d_i} \tag{7}$$

It can be normalized as follow:

$$\mathcal{B}_p = \frac{\overline{\mathcal{B}}_p}{h \cdot |U| \cdot |D|} \tag{8}$$

where $h \cdot |U| \cdot |D|$ is the storage benefit in the theoretically worst case where all the users retrieve data via h hops.

C. Storage Space Balance Model

As discussed in Section I, EDD must balance data storage across to allow spaces on individual edge servers for accommodating future data demands. Let ms_j denote the total storage spaces on s_j. Considering the heterogeneity in edge servers' storage spaces, we measure the storage occupancy rate of an edge server s_j as follow:

$$\mathcal{O}_j = \frac{\sum_{d_i \in D} (1 - r_i^j)}{ms_j} \tag{9}$$

To evaluate the storage space balance across S produced by p, we calculate the Jain's fairness index [23] based on edge servers' storage occupancy rates after p is implemented:

$$\mathcal{L}_p = \frac{|\sum_{s_j \in S} \mathcal{O}_j|^2}{|S| \sum_{s_j \in S} {\mathcal{O}_j}^2} \tag{10}$$

where $\mathcal{L}_p \in [1/|S|, 1]$ and $\mathcal{L}_p = 1$ indicates that data storage is fully balanced, i.e., all edge servers share the same storage occupancy rates.

D. Balanced Edge Data Deduplication

A BEDD strategy is evaluated based on its ability to deduplicate data, retain data storage benefits, and balance data storage. Accordingly, the optimization objective of the BEDD problem can be expressed as follow:

$$\begin{aligned} \textbf{BEDD} : \max : \alpha\mathcal{R}_p + \beta\mathcal{B}_p + \gamma\mathcal{L}_p \\ \text{s.t. } (3), (6) \end{aligned} \tag{11}$$

where α, β, and γ are the adjustable weights of the three terms ($\alpha + \beta + \gamma = 1$). They indicate the preferences for data deduplication ratio, data storage benefits, and storage space balance. For example, when the edge data size is small in general, such as text data, the benefit of a high resource balance index is usually lower than that of a high data deduplication ratio. In such cases, $\alpha > \beta$ is suggested. When duplicate data is large in general, e.g., images and videos, balancing data storage is more important than the deduplication ratio because the storage resource imbalance is more likely to occur. In such cases, a high β is recommended. For latency-sensitive applications like VR/AR, data storage benefits are more significant than the other two objectives, and a large γ is preferable.

E. Problem Hardness Proof

By reducing the BEDD problem from the classic $\mathcal{NP}$-hard *bin packing* (BP) problem [24], the $\mathcal{NP}$-hardness of the BEDD problem can be proved in this section.

Given a set of bins, $E = \{e_1, \ldots, e_x\}$, where each bin e_j has a limited capacity $size(e_j)$. Given a set of items to be packed into these x bins, $F = \{f_1, f_2, \ldots, f_y\}$, where each item f_i has a size $size(f_i)$, the goal of the BP problem is to pack all y items into a minimum number of bins. It can be formulated as follows:

$$\min \sum_{j=1}^{x} \eta_j \tag{12}$$

$$s.t. \sum_{i=1}^{y} \tau_i^j size(f_i) \leq \eta_m size(e_j) \tag{13}$$

$$size(f_i) \leq size(e_j) \tag{14}$$

$$\sum_{j=1}^{x} \tau_i^j = 1 \tag{15}$$

$$\eta_j, \tau_i^j \in \{0, 1\} \tag{16}$$

where binary variable η_j denotes whether bin e_j is used, binary variable τ_i^j denotes whether item f_i is packed into bin e_j.

Now we make the following reductions of the BEDD problem from the BP problem: 1) relax the latency constraint to the maximum latency between any two edge servers in the ESS; 2) allow only one replica of each data $d_i \in D$ to be stored in the ESS. Since the edge servers from which a data d_i can be removed are fixed, i.e., the edge servers from d_i within latency constraint, we can convert the objective of the relaxed BEDD problem to the objective that minimizes the number of edge servers that store data d_i after deduplication. Through the reduction, only one replica of each data $d_i \in D$ can be stored in the ESS. The optimization objective of the BEDD problem is the same as Objective (12) in the BP problem. Constraints (13) and (14) in the BP problem ensure that the items packed in each bin do not exceed their maximum size and the maximum size of each item does not exceed the size of the corresponding bin. This is equivalent to the range of r_i^j. Constraint (15) in the BP problem ensures that each item must be packed. This is equivalent to Constraint (3), i.e., all the edge servers covered by data d_i should still be covered after deduplication. Constraint (16) specifies the domains of η_j and τ_i^j. It is the same as the constraint for binary variable r_i^j.

In conclusion, any solution satisfying the $\mathcal{NP}$-hard BP problem can be reduced to the BEDD problem in polynomial time. The BEDD problem is thus $\mathcal{NP}$-hard.

III. Approach Design

We present BEDD-O and BEDD-A for finding optimal and sub-optimal BEDD solutions in this section.

A. Optimal Approach

The optimal solution to solve the BEDD problem aims to jointly maximize the data deduplication ratio, data storage benefits, and storage space balance while ensuring all constraints. Let binary variable $r_i^j = 1$ denote that data d_i is removed from edge server s_j, and 0 otherwise. The BEDD problem can be formulated as follows:

$$\max\ \alpha \sum_{d_i \in D} \sum_{s_j \in S} \mathcal{R}(r_i^j) + \beta \sum_{u \in U} \sum_{d_i \in D} \sum_{s_j \in S} \mathcal{B}(r_i^j, x_u^j) + \gamma \sum_{d_i \in D} \sum_{s_j \in S} \mathcal{L}(r_i^j) \tag{17}$$

$$\text{s.t.} \sum_{d_i \in D} r_i^j \le ms_j \tag{18}$$

$$\cup_{\{r_i^j = 0\}} N(s_j) = N(S_{d_i}) \tag{19}$$

where $N(s_j)$ denotes the nearby edge servers of s_j, i.e., the edge servers that can communicate with s_j within h. Objective (17) maximizes the produced deduplication ratio, data storage benefits, and storage space balance combined. Constraint (18) enforces the edge servers' capacity constraint. Constraint (19) ensures equal data coverage before and after deduplication.

BEDD-O can solve this *integer linear programming* (ILP) problem exactly by integer programming solvers, such as Gurobi.[2] The solution is an assignment of 0 or 1 to each r_i^j, where $d_i \in D$, $s_j \in S$, that maximizes the optimization objective (17) while fulfilling the capacity constraint (18), the data coverage constraint (19), and the data retrieval latency constraint (enforced by $N(s_j)$). According to the solution, the replica of d_i can be removed from edge server s_j if r_i^j is 1.

B. Sub-Optimal Approach

BEDD-O finds the optimal BEDD solution but is computationally intractable in large-scale BEDD scenarios, e.g., when the number of data to be deduplicated is large. To enable high responsiveness to the dynamic data demands in real-world MEC scenarios, we need to be able to find BEDD solutions rapidly in such scenarios. To tackle this challenge, we design an approach named BEDD-A to find sub-optimal BEDD solutions based on the *Lagrangian relaxation* (LR) method.

2.https://www.gurobi.com/products/gurobi-optimizer/

Unfortunately, LR can solve convex optimization problems only and cannot be directly applied to solve the BEDD problem in the ILP (*integer linear problem*) form. Thus, we first relax the ILP problem into a *linear programming* (LP) problem. Then, the BEDD problem can be relaxed and transformed into a Lagrange dual problem. After solving the Lagrange dual problem with our specifically-designed subgradient method, BEDD-A can make data deduplication decisions following a greedy rounding strategy based on the solved fractional solutions.

First, we relax all the binary variables into a fractional value from 0 to 1. With the relaxation, we can find that the data storage benefit $\mathcal{B}_p$ cannot be bounded by (6) as the upper bound approaches 1. Thus, we redefine a binary variable x_u^j to indicate whether user u is covered by edge server s_j and substitute $cov(s_j)$ in (6) with $\sum_{u \in U} x_u^j$. With the above relaxation, Constraint (3) can be replaced with $x_u^j h_{jk}(1 - r_i^k) \le h$. After that, the ILP model built in Section III-A can be transformed to the following forms:

$$\textbf{Relaxed BEDD} : \max : \alpha \mathcal{R}_p + \beta \mathcal{B}_p + \gamma \mathcal{L}_p \tag{20}$$

$$\textbf{s.t.} \quad x_u^j h_{jk}(1 - r_i^k) \le h \tag{21}$$

$$\max\{(h - h_{jk}) * \sum_{u \in U} x_u^j, 0\} \le |U| * \max\{|h - h_{jk}|\} \tag{22}$$

$$0 \le r_i^j \le 1, \forall d_i \in D, \forall s_j \in S \tag{23}$$

$$0 \le x_u^j \le 1, \forall u \in U, \forall s_j \in S \tag{24}$$

In the relaxed BEDD problem, we can see that (21) couples binary variables r_i^j and x_u^j while other constraints only involve one variable each. Thus, we introduce a Lagrange multiplier $\mu (\mu \ge 0)$ to (21). Then, the optimization objective of the relaxed BEDD problem can be reformulated as the Lagrange dual function below:

$$\begin{aligned}\phi(\mu) &= \max\ \alpha \mathcal{R}_p + \beta \mathcal{B}_p + \gamma \mathcal{L}_p + \mu(h - x_u^j h_{jk}(1 - r_i^k)) \\ &= \max\ \alpha \sum_{d_i \in D} \sum_{s_j \in S} \mathcal{R}(r_i^j) + \beta \sum_{u \in U} \sum_{d_i \in D} \sum_{s_j \in S} \mathcal{B}(r_i^j, x_u^j) \\ &\quad + \gamma \sum_{d_i \in D} \sum_{s_j \in S} \mathcal{L}(r_i^j) + \sum_{d_i \in D} \sum_{s_j \in S} \mu(h - x_u^j h_{jk}(1 - r_i^k))\end{aligned} \tag{25}$$

As a result, the dual problem of (25) is:

$$\min\ \phi(\mu) \tag{26}$$

Now, we can solve (25) by equivalently solving (26). Let us name it $LD(\mu)$. Note that for all the μ values, the solution to (25) is the lower bound of the relaxed BEDD problem, and the solution to (26) is the closest to the optimal solution to the primal problem.

Now we achieve $LD(\mu)$ with a subgradient method that updates the Lagrange multiplier μ by adjusting the step size iteratively. Algorithm 1 presents the pseudo code of this method. Let μ^t denote the Lagrange multiplier in the t-th iteration and the update process is expressed as follow:

$$\mu^{t+1} = \mu^t + \theta_t (h - Y(1 - r^t)) \tag{27}$$

where r^t represents the optimal solution of the $\phi(\mu^t)$ problem, $Y \triangleq x_u^j h_{jk}$ denotes the other parameters not related to r^t, θ_t is the updating step size at the t-th iteration. According to [25], the step size in the subgradient method can converge to 0. However, the convergence must not be overly fast. Otherwise, the search for the solution to the relaxed BEDD problem will easily fall into a local optimum. The convergence must not be overly slow either. Otherwise, it will take a long time for the method to converge and incur excessive time overheads. To strike a trade-off, BEDD-A adjusts the step size according to the results obtained at each iteration:

$$\theta_t = \frac{LD(\mu^t) - LD^*}{||h - Y(1 - r^t)||^2}\Delta_t \tag{28}$$

where LD^* is the upper-bound solution of the primal problem (11) and $\Delta_t \in (0, 2]$ is an adjustment factor that controls the direction of updating Lagrange multiplier and ensures algorithm converge. If $LD(\mu^t)$ is not updated to a better lower bound of problem (11), BEDD-A halves Δ_t to reduce the updating step size.

The Lagrange relaxation method is often employed to solve combinatorial optimization problems. With the relaxation, the problem can be transformed into a less constrained or an unconstrained problem. Then, the Lagrange relaxation method can find a lower-bound solution to the primal problem by updating the Lagrange multiplier iteratively. Typical applications of the Lagrange relaxation method need to estimate the value of the upper-bound solution to the primal problem and use a fixed adjustment factor of step size to update the Lagrange multiplier. These often slow down and even undermine algorithm convergence. In the BEDD problem, the upper bound LD^* cannot be estimated accurately due to constraints (21) and (22). To tackle these challenges, Algorithm 1 employs an adaptive updating mechanism with a variable-length upper bound $\mathcal{Z}_l = \mathcal{Z}_l/\sqrt{l}$ instead of a fixed $\mathcal{Z}_l$. In this way, the upper-bound solution can be adjusted in a more fine-grained manner iteratively. In general, as the number of adjustments increases, the updating step size will decrease. This adjustment mechanism allows the algorithm to converge faster than the traditional updating approach using Polyak-based step size. This will be experimentally evaluated in Section IV.

Algorithm 1 first initializes the iteration number t, the current best solutions r_{rec}^t, the Lagrange multiplier μ^0, step size θ_t, subgradient g^t, and the number of upper-bound adjustments l in Line 2. Then, it starts to update the Lagrange multiplier iteratively to find the final solution. In each iteration, a solution r^t can be calculated first by finding the minimum $LD(\mu^t)$ and the subgradient g^t can be obtained accordingly (Lines 2-3). Then, $LD(r^t)$ is compared with the last best value of function $LD(r^{t-1})$, i.e., $LD_{rec}(r^{t-1})$. If $LD(r^t)$ is smaller than $LD_{rec}(r^{t-1})$, $LD_{rec}(r^t)$ is replaced by $LD(r^t)$ and r^t is recorded as the current best solution r_{rec}^t. Otherwise, the current values of $LD_{rec}(r^t)$ and r_{rec}^t are still the best values (Lines 7-13). Next, the algorithm measures the difference between the last optimal function value and the current function value, and updates the upper bound $LD_{lev}(r^t)$ accordingly (Lines 14-20). Next, it updates the Lagrange multiplier μ^t and the step size

Algorithm 1: BEDD-A.

```
1: **Initialization**:
2: $t = 1, r_{rec}^t = 0, \mu^0 = 0, \theta_t = 0, g^t = 1, l = 0$;
3: **End of initialization**
4: **while** $\mu^t - \mu^{t-1} \geq \delta$ and $g^t \neq 0$ **do**
5:     $r^t \leftarrow \arg\min LD(\mu^t)$
6:     calculate the subgradient $g^t \leftarrow h - Y(1 - r^t)$
7:     **if** $LD(r^t) \leq LD_{rec}(r^{t-1})$ **then**
8:         $LD_{rec}(r^t) = LD(r^t)$
9:         $r_{rec}^t = r^t$
10:    **else**
11:        $LD_{rec}(r^t) = LD_{rec}(r^{t-1})$
12:        $r_{rec}^t = r_{rec}^{t-1}$
13:    **end if**
14:    **if** $LD(r^t) \leq LD_{rec}(r^{t-1}) - \mathcal{Z}_l/2$ **then**
15:        $LD_{lev}(r^t) = LD_{rec}(r^t) - \mathcal{Z}_l$
16:    **else**
17:        $LD_{lev}(r^t) = LD_{rec}(r^{t-1}) - \mathcal{Z}_l$
18:        $l = l + 1$
19:        $\mathcal{Z}_l = \mathcal{Z}_l/\sqrt{l}$
20:    **end if**
21:    update $\mu^t$ with (27)
22:    update $\theta^t$ with (28)
23:    $t \leftarrow t + 1$
24:    **if** $LD(\mu^t) - LD(\mu^{t-1}) \leq threshold$ **then**
25:        $\Delta_t = \Delta_t/2$
26:    **end if**
27: **end while**
28: **return** $r_{rec}^t$
```

adjustment factor with (27) and (28), respectively (Lines 21-22). To ensure algorithm convergence, if the improvement is less than a given threshold *threshold*, the algorithm adjusts Δ_t in Lines 24-25. The iteration stops when the updated value of the Lagrange multiplier is less than δ or the subgradient is equal to 0. Finally, in Line 28, the algorithm returns r_{rec}^t as the solution to the relaxed BEDD problem.

After running Algorithm 1, the solution r_{rec}^t may involve a set of fractional values from 0 to 1 due to the relaxation and thus is not the optimal solution to the BEDD problem. Next, BEDD-A employs a rounding algorithm for obtaining the optimal solution to the relaxed BEDD problem (20). It goes through the following steps: 1) select the $r_{max} \in \{r_{rec}^t\}$ with the highest unfixed fractional value in Line 2; 2) if r_{max} satisfies Constraints (3) and (6), fix its value to 1 and remove r_{max} from $\{r_{rec}^t\}$, otherwise 0; 3) fix other $r_i^j \in \{r_{rec}^t\}$ to 0 if it does not satisfy Constraints (3) and (6); 4) iterate steps 1-3 until all the $r_i^j \in \{r_{rec}^t\}$ are fixed.

C. Convergence Analysis

Now we analyze the convergence of BEDD-A to show that it can effectively stop. For ease of exposition, we define r^t as the solution found by Algorithm 1. The following holds:

$$LD^* = \inf_{t \in T} LD(r^t) > -\infty \tag{29}$$

Algorithm 2: Greedy Rounding.

```
1: while $r^t_{rec} \neq \emptyset$ do
2:    sort each $r^j_i$ in $\{r^t_{rec}\}$ from high to low and select the
      highest one named $r_{max}$
3:    remove $r_{max}$ from $\{r^t_{rec}\}$
4:    if $r_{max}$ satisfies constraints (3) and (6) then
5:       fix the value of $r_{max}$ to 1
6:       for each $r^j_i \in \{r^t_{rec}\}$ do
7:          if $r^j_i$ not satisfies constraints (3) and (6) then
8:             fix the value of $r^j_i$ as 0
9:             remove $r^j_i$ from $\{r^t_{rec}\}$
10:         end if
11:      end for
12:   else
13:      fix the value of $r_{max}$ to 0
14:   end if
15: end while
```

Theorem 1: When $t \to \infty$, there is $\lim_{t\to\infty} LD_{rec}(r^t) = LD^*$ and the number of upper bound adjustments $l \to \infty$ and $\lim_{l\to\infty} \mathcal{Z}_l = 0$.

Proof 1: Assuming $l \leq \infty$, from Lines 14-20 in Algorithm 1, there exists a $T \in N^*$ to make the following hold:

$$LD(r^t) \leq LD(r^{t-1}) - \frac{\mathcal{Z}_l}{2}, \forall t \geq T \tag{30}$$

Thus, there is

$$LD(r^t) \leq LD(r^{T-1}) - (t-T)\frac{\mathcal{Z}_l}{2}, \forall t \geq T \tag{31}$$

From (31), we can easily infer $\lim_{t\to\infty} LD(r^t) = -\infty$, which contradicts (29). Thus, there is $l \to \infty$. Now we employ the contradiction method to prove Theorem 1. From (29), we know that $LD_{rec}(r^t)$ decreases monotonically when t increases and has a lower bound. Thus, $\lim_{t\to\infty} LD_{rec}(r^t)$ exists and is unique. Since the solution to $LD(\mu)$ must be convex due to the duality transform [26], the following holds:

$$||r^{t+1} - r^t|| \leq ||\mu^t - r^t||, \forall t \in N^* \tag{32}$$

When $t \geq T$ and $\hat{r}^t \in L^*$, combining (32) and Lines 21-26 in Algorithm 1, we have the following:

$$\begin{aligned}||r^{k+1} - \hat{r}^t||^2 &\leq ||\mu^t - \hat{r}^t||^2 \\ &= ||r^t - \frac{\Delta_t(LD(r^t) - LD_{rec}(r^t))}{||g^t||^2} g^t - \hat{r}^t||^2 \\ &= ||r^t - \hat{r}^t||^2 - \frac{2\Delta_t(LD(r^t) - LD_{lev}(r^t))}{||g^t||^2} \cdot \\ &\quad \langle r^t - \hat{r}^t, g^t\rangle + \frac{\Delta_t^2(LD(r^t) - LD^t_{rec})^2}{||g^t||^2}\end{aligned} \tag{33}$$

Since $g^t \in \partial LD(r^t)$ and $LD(\hat{r}^t) < LD_{rec}(r^t)$, we have:

$$\langle \hat{r}^t - r^t, g^t\rangle \leq LD(\hat{r^t}) - LD(r^t) \tag{34}$$

Combining (33) and (34), the following holds:

$$\begin{aligned}||r^{t+1} - \hat{r}^t||^2 \leq ||r^t - \hat{r}^t||^2 - \Delta_t(2-\Delta_t)\cdot \\ \frac{(LD(r^t) - LD_{rec}(r^t))^2}{||g^t||^2}\end{aligned} \tag{35}$$

According to Lines 14-20 in Algorithm 1, we have:

$$(LD(r^t) - LD_{rec}(r^t))^2 \geq \frac{\mathcal{Z}_l^2(l-3)}{4l(l+1)} \geq \frac{\mathcal{Z}_l^2(t-3)}{4l(t+1)} \tag{36}$$

Combining (35) and (36), the following holds:

$$||r^{t+1} - \hat{r}^t||^2 \leq ||r^t - \hat{r}^t||^2 - \frac{\Delta_t(2-\Delta_t)\mathcal{Z}_l^2(t-3)}{4t(t+1)} \tag{37}$$

We can find that (37) holds for any $t \geq T$ so that $\sum_{t=T}^{\infty} \frac{t-3}{t(t+1)} < \infty$. This is an obvious contradiction. Therefore, Theorem 1 holds.

IV. Evaluation

Extensive and comprehensive experiments are conducted to evaluate the performance of BEDD-O and BEDD-A in different BEDD scenarios.

A. Experimental Settings

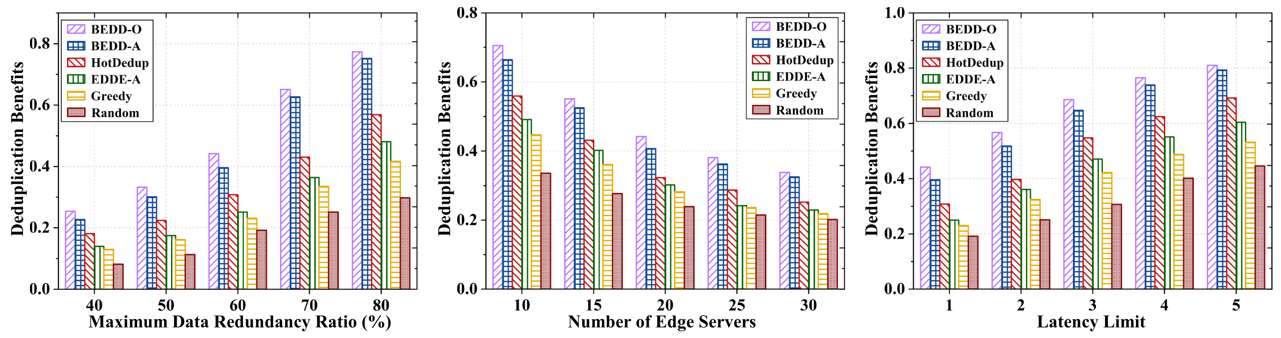
Dataset: EUA dataset, as a widely used dataset in research on mobile edge computing [1], [5], [27], contains the geographic coordinates of real-world users and edge servers in Metropolitan Melbourne, Australia.

Competing Approaches: In the experiments, BEDD-O and BEDD-A are compared with four representative approaches:

- *Random:* This baseline approach deduplicates redundant data replicas from edge servers randomly, one by one, until any constraint is violated.
- *Greedy:* This baseline approach removes redundant data replicas from the edge server with the maximum storage occupancy, one by one, until any constraint is violated.
- *HotDedup:* [11] Implemented based on the k-MST algorithm, this heuristic approach aims to maximize the deduplication ratio and data service rate based on data popularity.
- *EDDE-A:* [28] This state-of-the-art approach employs an approximation algorithm to remove redundant data, aiming to maximize storage resource savings.

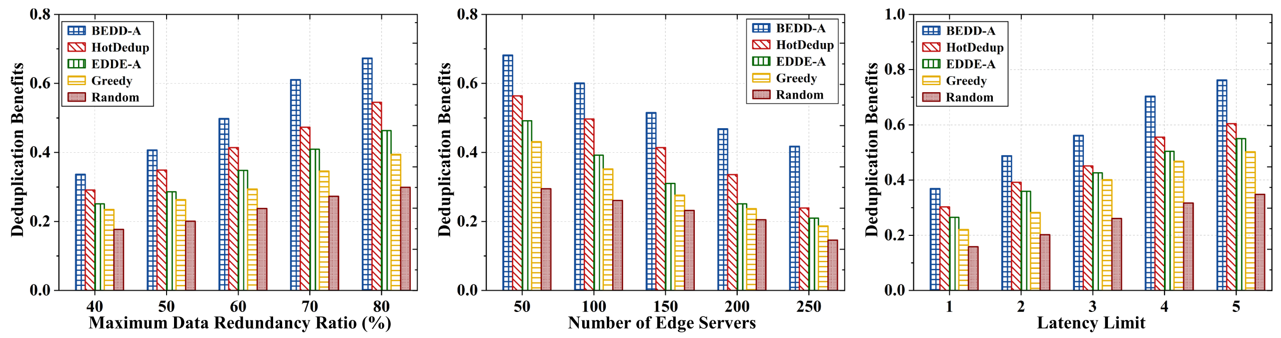
Parameter Settings: To evaluate the proposed BEDD-O and BEDD-A comprehensively, three major parameters are taken into consideration:

- *Maximum data redundancy ratio (Θ):* This parameter depicts the maximum ratio of edge servers that store the replicas of a data in the system. For example, $\Theta = 1$ denotes that every $d_i \in D$ may be stored on every edge server. This parameter varies from 0.4 to 0.8 in Set #1.1 and Set #2.1, similar to the settings in [10], [11].
- *Number of edge servers (n):* This parameter dictates the scale of a BEDD scenario. It varies from 10 to 30 in Set #1.2 and 50 to 250 in Set #2.2, respectively.
- *Latency constraint (h):* The parameter is the basis for the identification of an edge server s_j's neighbor edge servers



(a) $Benefits$ vs. Θ in Set #1.1 (b) $Benefits$ vs. n in Set #1.2 (c) $Benefits$ vs. h in Set #1.3

Fig. 3. Deduplication benefits versus parameters in Set #1. (a) $Benefits$ versus Θ in Set #1.1. (b) $Benefits$ versus n in Set #1.2. (c) $Benefits$ versus h in Set #1.3.



(a) $Benefits$ vs. Θ in Set #2.1 (b) $Benefits$ vs. n in Set #2.2 (c) $Benefits$ vs. h in Set #2.3

Fig. 4. Deduplication benefits versus parameters in Set #2.

TABLE II
PARAMETER SETTINGS. COMPARED WITH SET #1, THE SCALE OF SET #2 IS MUCH LARGER IN TERMS OF n. THIS ALLOWS US TO EVALUATE THE SCALABILITY OF BEDD-A IN SET #2

	Θ	n	h
Set #1.1	0.4, 0.5, ..., 0.8	20	1
Set #1.2	0.6	10, 15, ..., 30	1
Set #1.3	0.6	20	1, 2, ..., 5
Set #2.1	0.4, 0.5, ..., 0.8	150	2
Set #2.2	0.6	50, 100, ..., 250	2
Set #2.3	0.6	150	1, 2, ..., 5

$N(s_j)$. It can be specified in hops or milliseconds. To facilitate easy calculation of $N(s_j)$, in the experiments, it is measured by hops and varies from 1 to 5.

The specific parameter settings are summarized in Table II. To minimize the impacts of extreme cases on the results, e.g., those with overly-sparse or overly-dense edge servers, each experiment is conducted 200 times and the average values are reported when any of the above three parameters vary. For BEDD-A, the convergence threshold $threshold$ is set to 0.001. The weight factors α, β, γ used in (17) are set as $\frac{1}{3}, \frac{1}{3}, \frac{1}{3}$ to balance the importance of the optimization objectives. Considering that BEDD-O takes excessive time to find a solution in large-scale BEDD scenarios, BEDD-O is thus not included in Set #2 so that Set #2 can highlight the performance of BEDD-A.

Performance Metrics: The performance of BEDD-O and BEDD-A is evaluated based on two metrics.

- *Deduplication Benefits ($benefits$):* This metric is calculated with (11). It indicates the total benefits produced by a deduplication strategy.
- *Computational overheads ($time$):* This efficiency metric is measured by the computational time spent by the CPU to run an approach to find a solution.

B. Effectiveness Evaluation

Figs. 3 and 4 demonstrate the deduplication benefits achieved by the approaches in Set #1 and Set #2, respectively. In Set #1, BEDD-O is the winner in all the cases with slight advantages over BEDD-A. In Set #2, with BEDD-O excluded, BEDD-A wins the competition with significant advantages, outperforming HotDedup, EDDE-A, Greedy, and Random by an average of 30.51%, 57.69%, 73.06%, and 137.85%, respectively.

Impact of Data Redundancy Ratio (Θ): Figs. 3(a) and 4(a) show the impact of the maximum data redundancy ratio (Θ) on the overall benefits in Set #1 and Set #2, respectively. In Fig. 3(a), BEDD-O always achieves the highest deduplication benefits, 5.23%, 38.51%, 64.21%, 87.62%, and 124.37% higher than BEDD-A, HotDedup, EDDE-A, Greedy, and Random on average. BEDD-A seconds to BEDD-O but outperforms HotDedup, EDDE-A, Greedy, and Random significantly by 36.02%, 67.84%, 82.75%, and 146.87% on average. When n in the ESS is fixed, as Θ increases, data redundancy in the system increases in both settings. Adjacent edge servers are more likely to have duplicate data replicas. This allows the approaches to remove more duplicate date replicas without reducing the data coverage. The data storage benefit may drop but it is not as significant as the increase in the benefit produced by removing duplicate data replicas. When Θ increases, the advantages of our approaches increase. Take Fig. 4 for example. When Θ increases from 40% to 80%, its average advantage over HotDedup, EDDE-A, Greedy, and Random increases from 86.26% to 104.09% by 20.67%. The results demonstrate the importance of highly-effective EDD, especially in systems with high data redundancy.

Impact of Number of Edge Servers (n): Figs. 3(b) and 4(b) demonstrate the significant importance of n on deduplication benefits. BEDD-O again achieves the highest benefits with an average advantage of 6.42%, 34.25%, 41.51%, 48.17%, and 89.77% over BEDD-A, HotDedup, EDDE-A, Greedy, and Random in Set #1. BEDD-A is the clear winner in Set #2, with a 28.44% advantage over HotDedup, 53.29% over EDDE-A, 89.13% over Greedy, and 169.71% over Random. When n increases, its advantage increases, as shown in Fig. 3(b). With a fixed data redundancy ratio Θ, the increase in n further spreads duplicate data replicas across edge servers. It is less likely for adjacent edge servers to have duplicate data replicas. This reduces the room for EDD and decreases the deduplication benefits. In the meantime, it becomes more difficult to balance the storage resources across edge servers. This also reduces the deduplication benefits. Thus, as n increases, the deduplication benefits obtained by all approaches decrease.

Impact of Latency Constraint (h): Figs. 3(c) and 4(c) illustrate the results of Set #1.3 and Set #2.3, respectively. Again, BEDD-O outperforms all other approaches, by 5.76% against BEDD-A, 27.34% against HotDedup, 40.51% against EDDE-A, 63.91% against Greedy, and 105.03% against Random. BEDD-A achieves the second-highest deduplication benefits, 19.39% higher than HotDedup, 29.64% higher than EDDE-A, 51.80% higher than Greedy, 93.55% higher than Random in Set #1.3. In Set #2.3, BEDD-A's performance advantages are more significant than in Set #1.3. When the latency constraint h is relaxed, allowing each cached data replica to cover more users and produce high storage benefits. It also reduces the number of data replicas after EDD to keep the data coverage and makes it easier to balance storage resources across edge servers. Take d_3 in Fig. 1 as an example. When $h = 0$, d_3 must be retained on s_2, s_3, and s_4. However, if the latency constraint is relaxed to 1 hop, some replicas of d_3 can be removed to produce deduplication benefits. Thus, when h increases, the deduplication benefit increases for all the approaches in both Set #1.3 and Set #2.3.

Performance Over Time: The discussion above has focused on the deduplication ratio ($\mathcal{R}_p$) and data storage benefit ($\mathcal{B}_p$), but not the storage space balance ($\mathcal{B}_p$), i.e., the third term in the optimization objective (11) of EDD. As discussed in Section II-C, we balance storage spaces across edge servers so that they can accommodate future data demands flexibly. To demonstrate the importance of considering storage space balance in EDD, we run BEDD-A in Set #2 with $\theta = 0.6, n = 150, h = 2$ over 100 time slots and compare it with a variant of BEDD-A named EDD-A that does not consider storage space balance. The parameters of (11) are set to $\alpha = \frac{1}{2}, \beta = \frac{1}{2}, \gamma = 0$ for EDD-A. In each time slot, 100 users are randomly selected from the EUA dataset, each requesting one of the 8 data randomly. The state-of-the-art *edge data caching* (EDC) approach introduced in [7] is employed to formulate a data caching strategy for caching data replicas, aiming to maximize data storage benefit based on data popularity and storage spaces on edge servers. After that, we run BEDD-A to formulate an EDD solution and remove duplicate data replicas accordingly.

Fig. 7 shows the experimental results. BEDD-A clearly outperforms EDD-A in maximizing deduplication benefits across all the 100 time slots. On average, its deduplication benefit is 31.07% higher than EDD-A's. In addition, by balancing storage spaces across edge servers, BEDD-A manages to stabilize the deduplication benefits over time, achieving a standard deviation of 0.011 across 100 time slots, 71.27% lower than EDD-A's 0.037. As discussed in Section I, we deduplicate edge data to free up the storage resources for data caching. To demonstrate how BEDD-A achieves this objective, Fig. 7 also presents the caching benefits achieved by the EDC approach over time with the support of BEDD-A and EDD-A, respectively. We can see that with BEDD-A in place instead of EDD-A, EDC achieves much higher caching benefits, with a 48.46% advantage on average across 100 time slots. The results indicate that it is important to balance storage spaces across edge servers while performing EDD.

C. Efficiency Evaluation

Computation Time: Figs. 5 and 6 show the computation overheads taken by different approaches to find an BEDD solution in small-scale BEDD scenarios and large-scale BEDD scenarios, respectively. Unsurprisingly, $time$ taken by BEDD-O to find a solution is maximum as BEDD-O pursues to find the optimal solution to the $\mathcal{NP}$-hard BEDD problem. When Θ increases in Set #1.1, the data redundancy in the system increases, making it harder for BEDD-O to find the optimal solution. When n increases in Set #1.3, BEDD-O's computation time increases rapidly. This confirms the problem hardness of the BEDD problem is $\mathcal{NP}$-hard as proved in Section II-E. An interesting phenomenon can be presented in Fig. 5(c). When h increases from 1 to 3, the computation time taken by BEDD-O increases because it needs to explore more potential solutions to find the optimal one. As h continues to increase from 3, each data needs to be stored on only a few edge servers to cover all users in the edge storage system. Thus, the further increase in h makes BEDD-O easier to find these edge servers.

Fig. 6 demonstrates the results in Set #2, where we can observe the computation times of BEDD-A, HotDedup, Greedy, EDDE-A, and Random clearly. Among the five approaches, BEDD-A's computation time is maximum while Random takes the least computation time. BEDD-A's computation time increases only

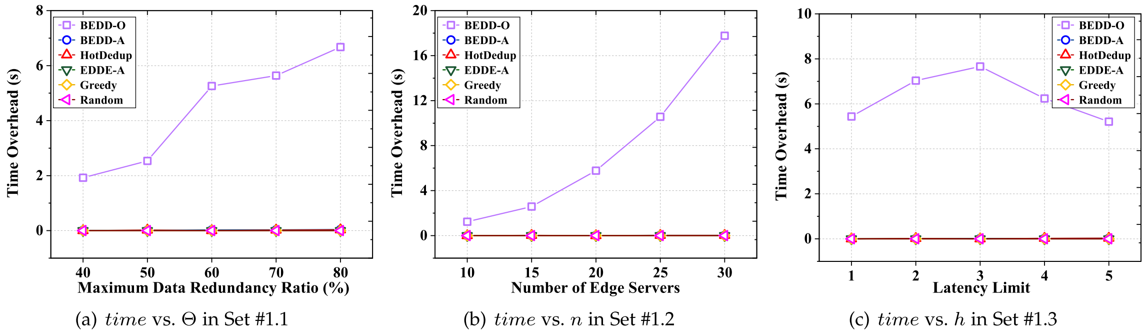


Fig. 5. Computation time versus parameters in Set #1.

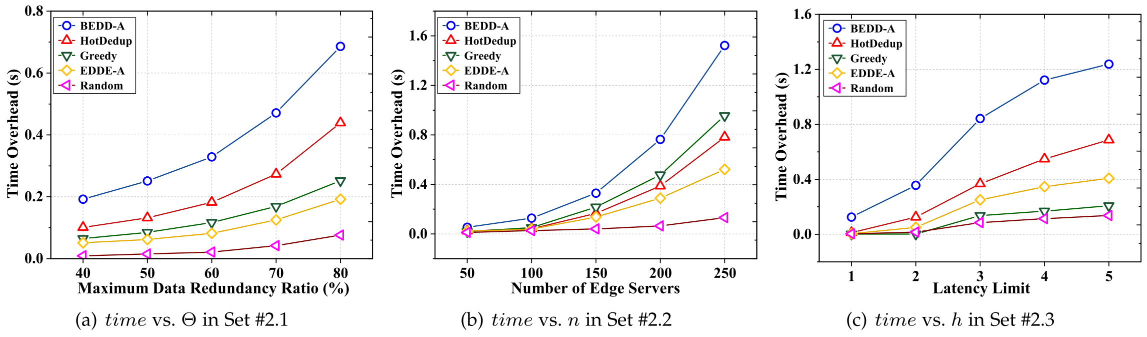


Fig. 6. Computation time versus parameters in Set #2.

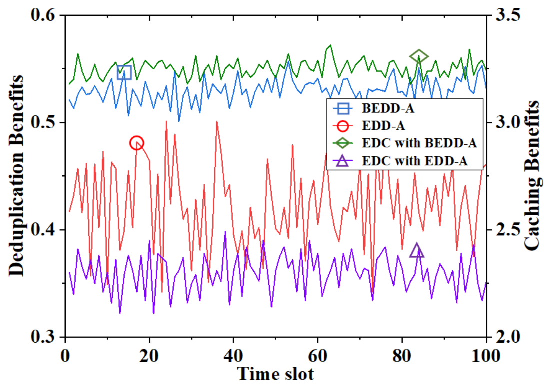


Fig. 7. Performance over time in Set #2.1 ($\Theta = 0.6$).

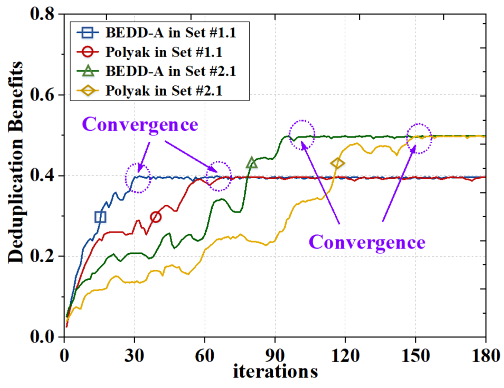


Fig. 8. Convergence of BEDD-A in Set #1.1 ($\Theta = 0.6, n = 20, h = 1$) and Set #2.2 ($\Theta = 0.6, n = 150, h = 2$).

mildly when Θ or h increases, as shown in Fig. 6(a) and (c), but significantly in Fig. 6(b). This again tells us n is the main influence in the hardness of the EDD problem. Even so, BEDD-A is highly efficient, taking only an average of 1.55 seconds to find a solution in the largest-scale experiment with 250 edge servers, as demonstrated in Fig. 6(b). Considering the significant performance improvements produced by BEDD-A, BEDD-A is the best option in large-scale BEDD scenarios.

Algorithm Convergence: As introduced in Section III-B, BEDD-A iterates to find the final solution. The number of iterations taken to find a solution as its convergence time to measure the efficiency of the approach. Fig. 8 illustrates the convergence of BEDD-A in Set #1.1 and Set #2.1, with a comparison with the classical Polyak step size based subgradient algorithm [29], a standard benchmark approach for LR-based approaches [30]. The results clearly show that BEDD-A converges much earlier than its competitors in both Set #1.1 and Set #2.2, taking 52.31% and 23.32% fewer iterations. This validates the usefulness of the variable-length upper-bound updating mechanism specifically designed for BEDD-A in Section III-B in accelerating the convergence of BEDD-A. The high efficiency of BEDD-A allows more frequent EDD to free up storage spaces on edge servers rapidly. This is particularly important in the MEC environment where data demands often vary quickly [22].

V. Related Work

As the number of smart and mobile devices has grown exponentially at an increasing pace, storing data in *edge storage systems* (ESSs) constituted by connected edge servers can provide users with low-latency data access [5]. Unfortunately, edge server's storage resources are significantly limited by their small physical size [31], [32], [33]. This sets a boundary on an ESS's storage capacity and the performance of the applications deployed on the ESS [7], [8]. To utilize ESSs cost-effectively, data deduplication offers a promising solution and may save up to 70% of an ESS's storage resources [10], [11].

Cloud Data Deduplication (CDD), as a classic data reduction technology, has been studied extensively [18], [19], [34]. The main challenge in CDD is to maximize deduplication ratio. To maximize deduplication ratio, CDD is performed at the chunk level. Specifically, data stored at different cloud nodes are partitioned into chunks so that duplicate chunks can be identified and removed across these cloud nodes. To name a few representative CDD approaches, Ni et al. [19] propose a content-defined chunking algorithm to accelerate data deduplication based on rolling hash and content locality. Fu et al. [18] propose AppDedup, an application-aware distributed deduplication framework that strikes a trade-off between scalable deduplication throughput and deduplication ratio by exploiting data similarity and data locality. What's more, few of researchers start to address the imbalance problem raised by data deduplication. Xu et al. [34] consider the read imbalance problem in cloud storage systems caused by data deduplication. They propose a heuristic algorithm to place data evenly across all the nodes with the aim to maximize read balance. They assume that any two storage nodes are reachable, which is unrealistic in the MEC environment.

Edge data deduplication (EDD) is a new problem fundamentally different from the CDD problem because of the unique constraints in the MEC environment, such as the capacity constraint, coverage constraint, latency constraint. These constraints have raised many new challenges that have attracted researchers' attention. Very recently, there is a tendency for researchers to start focusing on the problem of edge data redundancy. Li et al. [11] propose an approach named HotDedup that goes through two phases to reduce edge data redundancy. First, it employs a k-Minimum-Spanning-Tree algorithm to partition the target set of files into two subsets, one to be stored on edge nodes and the other in the cloud. Then, it identifies and removes duplicate chunks across edge nodes based on a distributed hash-chunk table. It considers the capacity constraint, but makes the same assumption as existing CDD approaches - one can retrieve any chunks from any edge nodes. In addition, it rebuilds data from chunks retrieved from edge nodes, ignoring the critical latency constraint in the MEC environment completely. A variant of HotDedup is implemented as one of the competing approaches in our experiments. The results presented and discussed in Section IV demonstrate that its performance is fairly poor in the MEC environment. Edge storage has widely acknowledged as a promising solution for ensuring low data retrieval latency and reducing backhaul network traffic.

Compared with cloud data redundancy, edge data redundancy is even a more critical problem because of edge servers' constrained storage resources. Unfortunately, it has yet to be properly solved. In this paper, we attempt to tackle this new *balanced edge data deduplication* (BEDD) problem, considering the unique constraints in the MEC environment plus the need to balance storage spaces across edge servers. Cheng et al. [9] propose a file storage strategy named Lofs, which employs a three-layer hash mapping scheme to detect data similarity, aiming to facilitate efficient data deduplication. However, this strategy does not consider data popularity, i.e, data storage benefits, which is a key characteristic in the MEC environment [11]. Thus, Lofs is not capable of balancing data retrieval latency and data deduplication ratio. To fully accommodate the unique characteristics of MEC, Luo et al. [28] propose a heuristic EDD approach to maximize data deduplication ratio. However, their approach does not consider data storage benefits and load balancing. Most of the data will be placed on edge servers with the most neighbor edge servers. This may serve all the users but does not ensure minimum data retrieval latency for them. As demonstrated in Section IV, the other critical limitation is that it will be very difficult for these edge servers to accommodate future data demands.

VI. Conclusion and Future Work

In this paper, we introduce, motivate, formulate, and solve the *balanced edge data deduplication* (BEDD) problem, taking into account the data deduplication ratio, data storage benefit, and storage space balance while fulfilling the unique constraints in the MEC environment. We proved its $\mathcal{NP}$-hardness and designed two approaches to solve small-scale and large-scale BEDD problems, respectively. Experimental results conducted on a widely used EUA dataset demonstrated the significant performance improvements of our approaches. In the future, we will study the BEDD problem further by taking the network robustness and data reliability into consideration.

References

[1] Q. He et al., "A game-theoretical approach for user allocation in edge computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 3, pp. 515–529, Mar. 2020.

[2] R. Shinkuma, T. Nishio, Y. Inagaki, and E. Oki, "Data assessment and prioritization in mobile networks for real-time prediction of spatial information using machine learning," *EURASIP J. Wireless Commun. Netw.*, vol. 2020, pp. 1–19, 2020.

[3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[4] Q. He, Z. Dong, F. Chen, S. Deng, W. Liang, and Y. Yang, "Pyramid: Enabling hierarchical neural networks with edge computing," in *Proc. ACM Web Conf.*, 2022, pp. 1860–1870.

[5] X. Xia, F. Chen, Q. He, J. Grundy, M. Abdelrazek, and H. Jin, "Online collaborative data caching in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 2, pp. 281–294, Feb. 2021.

[6] R. Luo, H. Jin, Q. He, S. Wu, and X. Xia, "Cost-effective edge server network design in mobile edge computing environment," *IEEE Trans. Sustain. Comput.*, vol. 7, no. 4, pp. 839–850, Fourth Quarter 2022.

[7] X. Xia, F. Chen, J. Grundy, M. Abdelrazek, H. Jin, and Q. He, "Constrained app data caching over edge server graphs in edge computing environment," *IEEE Trans. Services Comput.*, vol. 15, no. 5, pp. 2635–2647, Sep./Oct. 2022.

[8] Q. He et al., "A game-theoretical approach for mitigating edge DDoS attack," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 4, pp. 2333–2348, Jul./Aug. 2022.

[9] G. Cheng, D. Guo, L. Luo, J. Xia, and S. Gu, "LOFS: A lightweight online file storage strategy for effective data deduplication at network edge," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 10, pp. 2263–2276, Oct. 2022.

[10] H. Yan, X. Li, Y. Wang, and C. Jia, "Centralized duplicate removal video storage system with privacy preservation in IoT," *Sensors*, vol. 18, no. 6, 2018, Art. no. 1814.

[11] S. Li and T. Lan, "HotDedup: Managing hot data storage at network edge through optimal distributed deduplication," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 247–256.

[12] X. Xia et al., "OL-MEDC: An online approach for cost-effective data caching in mobile edge computing systems," *IEEE Trans. Mobile Comput.*, vol. 22, no. 3, pp. 1646–1658, Mar. 2023.

[13] N. Carlsson and D. Eager, "Ephemeral content popularity at the edge and implications for on-demand caching," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 6, pp. 1621–1634, Jun. 2017.

[14] Y. Shin, D. Koo, and J. Hur, "A survey of secure data deduplication schemes for cloud storage systems," *ACM Comput. Surv.*, vol. 49, no. 4, pp. 1–38, 2017.

[15] W. Xia et al., "A comprehensive study of the past, present, and future of data deduplication," *Proc. IEEE*, vol. 104, no. 9, pp. 1681–1710, Sep. 2016.

[16] H. Jin, R. Luo, Q. He, S. Wu, Z. Zeng, and X. Xia, "Cost-effective data placement in edge storage systems with erasure code," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2022.3152849.

[17] X. Xia, F. Chen, Q. He, J. C. Grundy, M. Abdelrazek, and H. Jin, "Cost-effective app data distribution in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 31–44, Jan. 2021.

[18] Y. Fu, N. Xiao, H. Jiang, G. Hu, and W. Chen, "Application-aware Big Data deduplication in cloud environment," *IEEE Trans. Cloud Comput.*, vol. 7, no. 4, pp. 921–934, Fourth Quarter 2019.

[19] F. Ni and S. Jiang, "RapidCDC: Leveraging duplicate locality to accelerate chunking in CDC-based deduplication systems," in *Proc. ACM Symp. Cloud Comput.*, 2019, pp. 220–232.

[20] C. Cai, J. Weng, X. Yuan, and C. Wang, "Enabling reliable keyword search in encrypted decentralized storage with fairness," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 1, pp. 131–144, Jan./Feb. 2021.

[21] S. Di, D. Kondo, and W. Cirne, "Host load prediction in a Google compute cloud with a Bayesian model," in *Proc. IEEE Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2012, pp. 1–11.

[22] P. Yang, N. Zhang, S. Zhang, L. Yu, J. Zhang, and X. S. Shen, "Content popularity prediction towards location-aware mobile edge caching," *IEEE Trans. Multimedia*, vol. 21, no. 4, pp. 915–929, Apr. 2019.

[23] A. B. Sediq, R. H. Gohary, R. Schoenen, and H. Yanikomeroglu, "Optimal tradeoff between sum-rate efficiency and Jain's fairness index in resource allocation," *IEEE Trans. Wireless Commun.*, vol. 12, no. 7, pp. 3496–3509, Jul. 2013.

[24] N. Karmarkar and R. M. Karp, "An efficient approximation scheme for the one-dimensional bin-packing problem," in *Proc. IEEE Annu. Symp. Found. Comput. Sci.*, 1982, pp. 312–320.

[25] A. Juttner, B. Szviatovski, I. Mécs, and Z. Rajkó, "Lagrange relaxation based method for the QoS routing problem," in *Proc. IEEE Conf. Comput. Commun.*, 2001, pp. 859–868.

[26] R. Rockafellar and R. Wets, "Stochastic convex programming: Basic duality," *Pacific J. Math.*, vol. 62, no. 1, pp. 173–195, 1976.

[27] G. Cui et al., "OL-EUA: Online user allocation for NOMA-based mobile edge computing," *IEEE Trans. Mobile Comput.*, to be published, doi: 10.1109/TMC.2021.3112941.

[28] R. Luo, H. Jin, Q. He, S. Wu, Z. Zeng, and X. Xia, "Graph-based data deduplication in mobile edge computing environment," in *Proc. Int. Conf. Serv.-Oriented Comput.*. Springer, 2021, pp. 499–515.

[29] A. Nedić and A. Ozdaglar, "Approximate primal solutions and rate analysis for dual subgradient methods," *SIAM J. Optim.*, vol. 19, no. 4, pp. 1757–1780, 2009.

[30] S. Ferdous, A. Khan, and A. Pothen, "Parallel algorithms through approximation: B-edge cover," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2018, pp. 22–33.

[31] M. Hu, L. Zhuang, D. Wu, Y. Zhou, X. Chen, and L. Xiao, "Learning driven computation offloading for asymmetrically informed edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 8, pp. 1802–1815, Aug. 2019.

[32] Z. Ning et al., "Distributed and dynamic service placement in pervasive edge computing networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 6, pp. 1277–1292, Jun. 2021.

[33] S. Deng, C. Zhang, C. Li, J. Yin, S. Dustdar, and A. Y. Zomaya, "Burst load evacuation based on dispatching and scheduling in distributed edge networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 8, pp. 1918–1932, Aug. 2021.

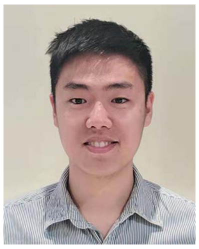
[34] M. Xu, Y. Zhu, P. P. Lee, and Y. Xu, "Even data placement for load balance in reliable distributed deduplication storage systems," in *Proc. IEEE Int. Symp. Qual. Serv.*, 2015, pp. 349–358.

Ruikun Luo received the bachelor's degree from Dalian Maritime University, China, in 2018. He is currently working toward the PhD degree with the Huazhong University of Science and Technology in China. His research interests include edge computing, service computing, cloud computing, and data storage.

Hai Jin (Fellow, IEEE) received the PhD degree in computer engineering from the Huazhong University of Science and Technology (HUST), China, in 1994. He is a chair professor of computer science and engineering with the Huazhong University of Science and Technology, China. In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz in Germany. He worked with the University of Hong Kong between 1998 and 2000, and as a visiting scholar with the University of Southern California between 1999 and 2000. He was awarded Excellent Youth Award from the National Science Foundation of China in 2001. He is a fellow the CCF, and a life member of the ACM. He has co-authored more than 20 books and published more than 900 research papers. His research interests include computer architecture, parallel and distributed computing, Big Data processing, data storage, and system security.

Qiang He (Senior Member, IEEE) received the PhD degree from the Swinburne University of Technology, Australia, in 2009, and the PhD degree from the Huazhong University of Science and Technology, China, in 2010. His research interests include edge computing, service computing, cloud computing, and software engineering. More details about his research can be found at https://sites.google.com/site/heqiang/.

Song Wu (Member, IEEE) received the PhD degree from the Huazhong University of Science and Technology (HUST), in 2003. He is a professor of computer science with HUST in China. He currently serves as the vice dean with the School of Computer Science and Technology and the vice head of Service Computing Technology and System Lab (SCTS) and the Cluster and Grid Computing Lab (CGCL), HUST. His current research interests include cloud resource scheduling and system virtualization.



Xiaoyu Xia (Member, IEEE) received the master's degree from the University of Melbourne, Australia, and the PhD degree from Deakin University, Australia. He is a lecturer with RMIT University, Australia. His research interests include edge computing, service computing, and software engineering. More details about his research can be found at https://sites.google.com/view/xiaoyuxia/.