Cost-Effective Data Placement in Edge Storage Systems With Erasure Code

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

Abstract—Edge computing, as a new computing paradigm, brings cloud computing's computing and storage capacities to network edge for providing low latency services for users. The networked edge servers in a specific area constitute *edge storage systems* (ESSs), where popular data can be stored to serve the users in the area. The novel ESSs raise many new opportunities as well as unprecedented challenges. Most existing studies of ESSs focus on the storage of data replicas in the system to ensure low data retrieval latency for users. However, replica-based edge storage strategies can easily incur high storage costs. It is not cost-effective to store massive replicas of large-size data, especially those that do not require real-time access at the edge, e.g., system upgrade files, popular app installation files, videos in online games. It may not even be possible due to the constrained storage resources on edge servers. In this article, we make the first attempt to investigate the use of erasure codes in cost-effective data storage at the edge. The focus is to find the optimal strategy for placing coded data blocks on the edge servers in an ESS, aiming to minimize the storage cost while serving all the users in the system. We first model this novel *Erasure Coding based Edge Data Placement* (EC-EDP) problem as an integer linear programming problem and prove its $\mathcal{NP}$-hardness. Then, we propose an optimal approach named EC-EDP-O based on integer programming. Another approximation algorithm named EC-EDP-V is proposed to address the high computation complexity of large-scale EC-EDP scenarios efficiently. The extensive experimental results demonstrate that EC-EDP-O and EC-EDP-V can save an average of 68.58% (and up to 81.16% in large-scale scenarios) storage cost compared with replica-based storage approaches.

Index Terms—Edge computing, erasure code, data placement, approximation algorithm

1 Introduction

In recent years, the world has witnessed the explosive growth of smart devices and mobile users. It is predicted that by 2021 there will be 32 billion connected mobile devices, and the global data traffic will reach 19.5 ZB per year [1]. The transmission of such massive data incurs heavy network traffic and consumes excessive network resources, leading to network issues, including service interruptions and high network latency. To tackle these challenges, edge computing has emerged as a new computing paradigm. It moves computing and storage resources onto edge servers at the edge of the cloud [2], [3]. Edge computing offers two key advantages to various online applications. First, users' data retrieval latency can be significantly reduced because they can retrieve data from their nearby edge servers rather than from the cloud. From app vendors' perspective, this ensures their users' *quality of experience* (QoE) because latency has become the key performance concern for online applications [4]. Second, service interruptions caused by network congestion can be alleviated by reducing the network traffic over the backhaul network [5]. This benefits app vendors by reducing the costs incurred by transmitting their data from the remote cloud to edge servers [6], [7].

In the edge computing environment, adjacent edge servers in an area are connected by high-speed links [8] to form an edge server network that constitutes an *edge storage system* (ESS) [3], [6], [9]. Compared with the edge-cloud architecture, ESS overcomes the single-point failure problem and performance bottleneck problem encountered [6], [10]. New challenges raised by ESS are starting to attract researchers' attention in recent years, who attempt to achieve various optimization objectives by caching data replicas on edge servers, e.g., minimum data retrieval latency [11], maximum cache hit ratio [12], [13], maximum caching benefits [14], [15], maximum caching capacity [16], [17]. A common assumption made by these replica-based approaches is that storage resources on each edge server in an ESS can always be hired on-demand or reserved in advance for caching data replicas. However, this assumption is not always realistic in a real-world edge computing environment where edge servers' storage resources are limited by the constrained physical sizes of base stations[18]. Even if it is feasible, caching massive data replicas on dense edge servers in an ESS - is often not cost-effective because the storage resources on edge servers are expensive [19]. This issue is especially critical when app vendors need to store large-size

- *Hai Jin, Ruikun Luo, Song Wu, and Zilai Zeng 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, Hubei 430074, China. E-mail: {hjin, rkluo, wusong, zilaizeng}@hust.edu.cn.*
- *Qiang He is 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 Information Technology, Deakin University, Geelong, VIC 3125, Australia. E-mail: xiaoyu.xia@deakin.edu.au.*

Manuscript received 18 May 2021; revised 7 January 2022; accepted 5 February 2022. Date of publication 24 February 2022; date of current version 10 April 2023.
This work was supported by the National Science Foundation of China (NSFC) under Grant 62032008.
(Corresponding author: Song Wu.)
Digital Object Identifier no. 10.1109/TSC.2022.3152849

data that do not require real-time access, e.g., system upgrade files, popular app installation files, videos in online games, in ESSs mainly to save on the expenses incurred by transmitting data out of the cloud for every user. A new approach is needed to enable the cost-effective storage of such *large data* in ESSs.

In this paper, we study the use of erasure coding in cost-effective storage of large data in ESSs. Under an erasure code scheme, a data $\mathcal{X}$ to be stored can be divided into M data blocks and K parity blocks. These data and parity blocks are distributed to be stored on different storage nodes (e.g., edge servers in an ESS) accessible to users. A user can retrieve M data and/or parity blocks (together referred to as *coded blocks* hereafter) from any accessible edge servers to construct $\mathcal{X}$ for use. Erasure codes have been widely employed to reduce storage costs in cloud-based storage systems [20], [21]. However, the unique characteristics that fundamentally differentiate ESSs from cloud-based storage systems render existing approaches obsolete and raise a number of new challenges. First, in the edge computing environment, the coverage of an edge server is limited. A user can only access coded blocks from edge servers that cover the user. This is the *proximity constraint* [22], [23]. A storage approach based on erasure code (referred to as EC-based storage approach hereafter) must ensure that every user in the area can retrieve enough coded blocks to construct data $\mathcal{X}$. This is the *encoding constraint*. In addition, data can be transmitted across edge servers over the edge server network topology to be delivered to users, but only within a limited number of transmission network hops [8], [18], [24]. Compared with traditional cloud storage systems, coded data blocks cannot be stored in different storage nodes arbitrarily in ESSs. This is the *transmission constraint* [6], [23], [25].

From the perspective of app vendors, the coded blocks of data stored in an ESS must be able to serve all the users at minimum storage cost while fulfilling the proximity, coverage, and transmission constraints. This problem is referred to as the *erasure coding based edge data placement* (EC-EDP) problem. This paper makes the first attempt to study this new problem, and the key contributions include:

- We formally model the EC-EDP problem as an integer linear programming problem and prove that it is $\mathcal{NP}$-hard.
- We propose EC-EDP-O, an approach for finding optimal solutions to small-scale EC-EDP problems based on integer programming solvers.
- We propose EC-EDP-V, an approximation approach, which used to find approximate solutions to large-scale EC-EDP problems with a $\ln \Theta_{limit} + 1$ approximation ratio guarantee.
- We evaluate the performance of EC-EDP-O and EC-EDP-V against five representative approaches through extensive experiments conducted on a widely-used EUA dataset.

2 Motivating Example

Since Windows-10, Microsoft has employed peer-to-peer distribution, in addition to traditional client-server distribution, to deliver large upgrade packages to its clients to reduce the network resource consumption incurred. In the meantime, app vendors like Microsoft can significantly reduce the costs of distributing such data to their clients by storing it in the ESSs facilitated at the network edge - Amazon Web Services charges up to US$0.11 to transfer 1GB data out of its S3 data storage facilities to the internet[1].

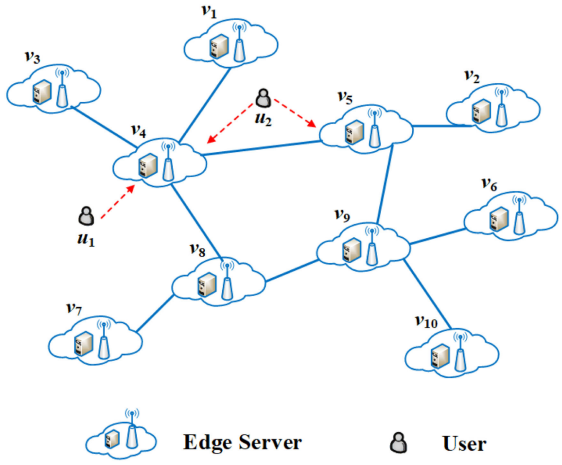


Fig. 1. Example of edge storage system.

Fig. 1 presents an example of ESS comprised of ten networked edge servers collectively serving the users in a specific area, e.g., New York CBD. A straightforward replica-based solution to the distribution of Microsoft's 1GB Windows-10 upgrade package is to store a replica on each of the ten edge servers. In the edge computing environment, a user can access the edge servers that cover the user. The distance between the user and the edge server may impact its data rate, as considered in some studies, but not the latency between them. Thus, the latencies between users and edge servers are not considered in the formulation of EC-EDP strategies in this study. In this way, all the users in the New York CBD can download the package from their nearby edge servers. However, there are two critical limitations to this solution. First, it costs Microsoft tremendously to save 10 data replicas (10GB in total) in the ESS over a long time due to the expensive storage resources on the edge servers [19], [23], [26]. Second, it does not take advantage of the collaboration of edge servers to transmit data to each other to deliver data for users [8], [18], [24]. Take the ESS presented in Fig. 1 for example. Assume that it allows data to be transmitted via two hops over the topology of edge sever network. In real-world EC-EDP 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, similar to [6], [27]. On this edge storage system, Microsoft only needs to store two replicas of its Windows-10 upgrade package (2GB in total) in the system to serve all the users in the area, e.g., v_4 and v_9, as illustrated in Fig. 2a, or v_5 and v_8.

To further reduce the storage cost of storing data in this ESS, Microsoft can first encode the upgrade package into 2

1. https://aws.amazon.com/s3/pricing/

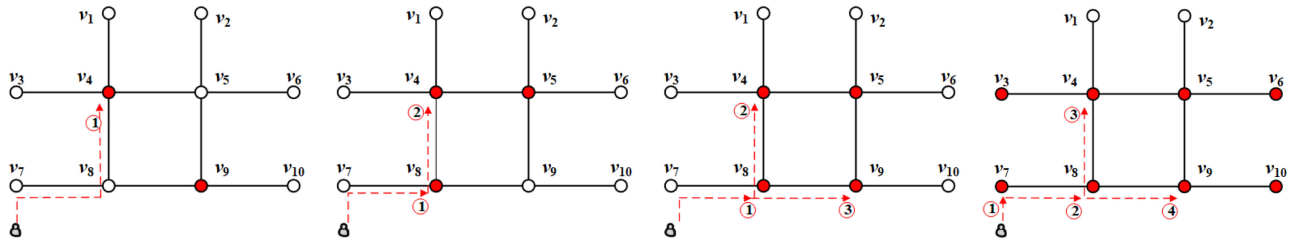


Fig. 2. Replica-based solution versus EC-EDP solutions.

data blocks (0.5GB each) and 1 parity block (0.5GB) through erasure coding to be placed on 3 of the edge servers in the ESS, e.g., v_4, v_5, and v_8, as illustrated in Fig. 2b. Under the erasure coding scheme, a user can retrieve any 2 of the three coded blocks from edge servers within two network hops to construct the upgrade package. For example, the user in Fig. 2b can retrieve a data block and a coded block from v_4 and v_8, respectively, to construct the upgrade package. In this way, Microsoft only needs to store a total of 1.5GB data in the ESS to satisfy all the user's access requests in the area, much less than the above replica-based approach.

An alternative solution is to encode the package into 3+1 coded blocks, 0.33GB each, to be placed at v_4, v_5, v_8, and v_9 as illustrated in Fig. 2c. This solution requires 1.33GB storage resources in total. Fig. 2d illustrates the third solution that encodes the package into 4 coded blocks to be placed. A total of 2GB storage resource is needed. Among the three solutions shown in Fig. 2, the $EC(3,1)$ solution presented in Fig. 2c incurs the least storage cost. Compared with replica-based storage solutions, EC-based solutions are more flexible because they require less storage occupation on individual edge servers. This is a critical advantage in the edge computing environment where the storage resources of edge servers are highly constrained and expensive [19], [23], [26].

Given an ESS, there are usually a large number of feasible EC-EDP solutions combining different data encoding and placement strategies. These solutions incur different storage costs. Meanwhile, in the real-world EC-EDP scenarios, the number of edge servers could be much larger and the network topology could be more complex. Finding the solutions to the EC-EDP problems in such scenarios is challenging. Therefore, it is important for app vendors to find the optimal one that serves all the users in the ESS at minimum storage cost. Please note that EC-based approach incurs computational overheads to users, i.e., the time taken to construct data from coded blocks [28], [29]. Thus, EC-based approaches are most suitable for storing large data that do not require real-time access but consume a large amount of network bandwidth, e.g., system upgrade files, popular app installation files, videos in online games.

3 Preliminaries

Erasure coding is widely used in the field of distributed storage system to yield low storage overhead and high reliability, such as Microsoft's Azure [20] and Facebook's F4 [30]. By applying erasure coding, a piece of data is divided into M data blocks, which are encoded into K parity blocks. The total of $M + K$ coded blocks is distributed to be stored on $M + K$ nodes. The data can be constructed from any M of the $M + K$ coded blocks [31]. Fig. 3 presents an example where $EC(3, 2)$ erasure coding ($M = 3, K = 2$) is employed to encode data $\mathcal{X}$. Data $\mathcal{X}$ is divided into three data blocks f_1, f_2, and f_3, which are encoded into two parity blocks $f_1^{'}$ and $f_2^{'}$. The five coded blocks can be distributed to be stored on different edge servers. To construct data $\mathcal{X}$, any user needs to retrieve at least three of the five coded blocks in the ESS. The encoding principle of the erasure code is to multiply the data by the coding matrix, and the decoding process is realized with the matrix inversion technique [28]. Actually, to ensure that the result of multiplication remains within a fixed size such as one byte, the results of matrix multiplication in the erasure code are obtained by mapping the matrix multiplication to a finite field [29].

In this research, we study the most general EC-EDP scenarios where at most one coded block on each edge server in the ESS. This *storage limit* generalizes the number of coded blocks that can be stored on each edge server. Allowing multiple coded blocks to be stored on each edge server will make it easier to find a storage solution but will lower the reliability of the data stored in the system. Take an extreme case for example, where all the $M + K$ coded blocks are stored on only one of the edge servers in the ESS to serve all the users. If that edge server fails, the data will become unavailable to all the users. On the contrary, if only one coded block can be stored on each edge server, the failure of an edge server does not significantly lower the reliability of the data. In fact, the ESS may still be able to serve all the users as long as they can still retrieve M coded blocks. The storage limit also generalizes our EC-EDP approach (to be presented in Section 5) by relaxing the need for app vendors to reserve a large number of storage resources on individual edge servers.

4 Model and Problem Formulation

In this section, we first formulate the EC-EDP problem and then reduce it to another classic $\mathcal{NP}$-hard problem for proving its $\mathcal{NP}$-hardness. The main notations used throughout this paper with their definitions can be found in Table 1.

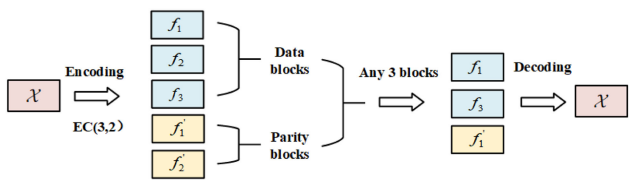


Fig. 3. EC(3,2) erasure coding scheme.

TABLE 1
Summary of Notations

Notation	Description
b_j	number of servers that user u_j can access within transmission constraint
$cost(R)$	storage cost achieved by caching strategy R
$cov(v_i)$	set of users covered by edge server v_i
$d_{i,j}$	distance between user u_j and edge server v_i
f_t	coded block of the data $\mathcal{X}$, where $t = 1, 2, \ldots, N$
F	set of coded blocks
G	graph presenting a particular area
$h_{i,w}$	distance from server v_i to server v_w
h_{limit}	maximum hops for data transmission over G
K	number of parity blocks with erasure code of $\mathcal{X}$
M	number of data blocks with erasure code of $\mathcal{X}$
N	number of total coded blocks of $\mathcal{X}$
P	number of total users
q_i	number of coded blocks stored on server v_i
$r_{i,t}$	binary variables indicating coded block d_t on edge server v_i
R	set of binary variables indicating coded blocks placement strategies
$size(f_t)$	size of coded block f_t
S	number of total edge servers
u_j	edge user j
U	set of user u_j, where $j = 1, 2, \ldots, P$
v_i	edge server i
V	set of edge server v_i, where $i = 1, 2, \ldots, S$
$\mathcal{X}$	the original data
$\alpha_{i,j}$	binary variable indicating whether u_j can access v_i within h_{limit} hops
$\beta_{i,j}$	binary variable indicating whether u_j can access coded blocks on v_i within h_{limit} hops

4.1 Problem Formulation

Similar to [6], the S networked edge servers in an ESS can be modeled as a undirected graph $G(V, E)$. In this graph G, each edge server $v_i \in V$ corresponds to a vertex, and the connection between two edge servers corresponds to an edge. In the edge computing environment, encoding multiple data for storage in an edge storage system is not cost-effective. For example, if five data are encoded as a bundle into a number of coded blocks, a user requesting one of these data will have to retrieve the coded blocks for constructing all five data. Transmitting these coded blocks will consume extra network resources. It will also take extra time for the user to construct data from the coded blocks. Thus, encoding multiple data for storage is not cost-effective in the context of this study.

Given a coded block f_t, divided from $\mathcal{X}$, and a set of edge servers v_i, a block placement decision, denoted by $r_{i,t}$, indicates whether block f_t is placed on edge server v_i

$$r_{i,t} = \begin{cases} 0 & \text{if block ft is not placed on vi} \\ 1 & \text{if block ft is placed on vi} \end{cases} \tag{1}$$

Let q_i denote the number of coded blocks placed on server v_i. It can be calculated as follow:

$$q_i = \sum_{t=1}^{N} r_{i,t}, \forall v_i \in V, \forall f_t \in F \tag{2}$$

Constraint (3) enforces the storage limit, i.e., at most one coded block needs to be stored on any edge server.

$$q_i \in \{0, 1\}, \forall v_i \in V \tag{3}$$

Let h_{limit} represent the transmission constraint introduced in Sections 1 and 2. Let $\alpha_{i,j} \in \{0, 1\}$ indicate whether user u_j can access server v_i, and b_j indicate the number of edge servers that user u_j can access without violating the transmission constraint. There is

$$\alpha_{i,j} = \begin{cases} 1 & \text{if } u_j \in cov(v_w), h_{i,w} \leq h_{limit}, v_w \in V \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

$$b_j = \sum_{i=1}^{S} \alpha_{i,j} \tag{5}$$

where $h_{w,j}$ is the distance (measured by hops) between user u_j and edge server v_w with a coded block.

To ensure that each user can retrieve adequate coded blocks for constructing data $\mathcal{X}$, M, i.e., the number of data blocks divided from $\mathcal{X}$, should not exceed the minimum number of edge servers accessible to any users $u_j \in U$ within h_{limit} hops over G. Theoretically, an erasure coding schemed divides $\mathcal{X}$ into at least 2 data blocks. Thus, there is

$$2 \leq M \leq min\{b_j | \ \forall u_j \in U\} \tag{6}$$

According to the encoding constraint, when users are covered by more than one edge server, they can access any one of these. Take Fig. 1 for example, u_2 can only directly access edge servers v_4 and v_5. Let $d_{i,j}$ denote the minimum distance from user $u_j \in U$ to server $v_i \in V$ and it can be calculated as follow:

$$d_{i,j} = min\{h_{i,w} | \ q_i = 1 \ , u_j \in cov(v_w)\}, v_i \in V \tag{7}$$

To ensure that each $u_j \in U$ can retrieve adequate coded blocks for constructing $\mathcal{X}$, there must be at least M edge servers with a coded block within h_{limit} hops over the edge server network

$$\beta_{i,j} = \begin{cases} 1 & \text{if } d_{i,j} \leq h_{limit} \\ 0 & otherwise \end{cases} \tag{8}$$

$$\sum_{i}^{S} \beta_{i,j} \geq M, \ u_j \in U, v_i \in V \tag{9}$$

where $\sum_i^s \beta_{i,j}$ is the number of coded blocks that user u_j can retrieve within h_{limit} network hops. Please note that variable $\beta_{i,j}$ is defined to ensure the feasibility of the data placement strategy by allowing users to retrieve necessary coded blocks for constructing data $\mathcal{X}$ under the transmission constraint, which is different from the definition of variable $\alpha_{i,j}$.

The optimization objective of the EC-EDP problem is to minimize the total storage cost. It is calculated based on the number and the size of the coded blocks stored in the ESS. Since matrix multiplication and inverse operations are involved in every erasure coding scheme, the size of coded blocks must be divided equally. Thus, the size of each coded block, denoted by $size(f_t) \in F$ can be calculated by $size(f_t) = size(\mathcal{X})/M$. Let $N \triangleq M + K$. Based on the storage limit, there is

$$N = \sum_{i=1}^{S} q_i, \ v_i \in V \tag{10}$$

The total storage cost incurred by an EC-EDP strategy is computed as $N \cdot (size(\mathcal{X})/M)$. Given that $size(\mathcal{X})$ is a constant specific to $\mathcal{X}$, the cost incurred by an EC-EDP strategy R can be presented as follow:

$$cost(R) = N/M \tag{11}$$

Thus, the optimization objective in the EC-EDP, i.e., to minimize the storage cost incurred, can be expressed as follow:

$$\min cost(R) \tag{12}$$

$$\text{s.t.: } (3), (6), (9)$$

.

4.2 Problem Hardness

In this section, we reduce EC-EDP to a classic $\mathcal{NP}$-hard problem, i.e., *minimum dominating set* (MDS) problem [32], for proving its $\mathcal{NP}$-hardness.

Given an undirected graph $G = (V, E)$, there is a subset of $V^{'} \subset V$. Given an arbitrary vertex $D^{'} \in V$, we always have $D^{'} \in V^{'}$ or the neighbor of $D^{'} \in V^{'}$. The set $V^{'}$ with the minimum elements can be called the minimum dominating set of graph G. For each vertex v_i, we define a binary variable y_i to denote whether it is in $V^{'}$ ($y_i = 1$) or not ($y_i = 0$). Let $\gamma(i)$ denote the vertex set composed by vertex v_i and its adjacent vertexes in G. The MDS problem can be expressed here.

$$\min \sum_{i \in V} y_i \tag{13}$$

$$s.t.: \quad \sum_{j \in \gamma(i)} y_j \geq 1, \forall i \in V \tag{13a}$$

$$y_i \in \{0, 1\}, i \in V \tag{13b}$$

The reduction from the MDS problem to the EC-EDP problem can be done as follows: 1) let the number of coded blocks M be a deterministic value; 2) let every user access a fixed edge server. Given an undirected graph $G = (V, E)$ in the MDS problem, we can find an instance of the MDS problem $MDS(V^{'}, E, ws)$, where $ws = \sum_{i \in V} y_i$. We can also construct an instance of the EDP problem $EDP(V^*, E^*, cs)$ with the reduction above where $|V^*| = |V^{'}|$ and $|E^*| = |E|$, and $cs = \sum_{j \in U, i \in V}(\beta_{i,j})$. Then, constraint (9) can be converted to $\sum_{j \in \Upsilon(i)} y_j \geq M$, where $\Upsilon(i)$ represents a vertex set comprised of vertex v_i and the vertexes within h_{limit} hops over G. We can easily see that it is equal to constraint (13a). According to (3), at most one coded block can be placed on each edge server. Thus, constraint (13b) can be fulfilled. In conclusion, any M values always satisfy the MDS problem. Thus, the EC-EDP problem is $\mathcal{NP}$-hard.

5 Approach Design

In this section, we first model the EC-EDP problem as an integer linear programming problem. Then, we propose two approaches, i.e., EC-EDP-O and EC-EDP-V. The EC-EDP-O approach is proposed to solve the small-scale EC-EDP scenarios based on integer programming. The EC-EDP-V is proposed to solve the large-scale EC-EDP scenarios with a $\ln(\Theta_{limit} + 1)$ approximation ratio guarantee.

5.1 Optimal Approach

The EC-EDP problem can be modeled as a *integer linear programming* (ILP). Given an edge server network $G = (V, E)$, where $V = \{v_1, \ldots, v_S\}$ and $E = \{e_1, \ldots, e_P\}$, let us define a set of variable $Y = \{y_1, \ldots, y_S\}$ to represent an EC-EDP strategy, where $y_i = \{0, 1\}$. If $y_i = 1$, it indicates that a coded block is placed on the edge server i, and $y_i = 0$ if not. Therefore, the formula of the ILP model for the EC-EDP problem is presented here

$$\min \quad \frac{\sum_{u_i \in U} y_i}{M} \tag{14}$$

$$\text{s.t.:} \qquad y_i \in 0, 1, \forall i \in [1, S] \tag{15}$$

$$d_{w,i} \leq h_{limit}, \forall i, w \in [1, S] \tag{16}$$

$$\sum_{i}^{S} \beta_{i,j} \geq M, \forall i \in [1, S], \forall j \in [1, P] \tag{17}$$

Constraint (16) guarantees that the users covered by edge server v_i can only retrieve coded blocks within the transmission limit. Constraint (17) is converted from (9) to guarantee that every user can retrieve adequate coded blocks to construct data $\mathcal{X}$.

Algorithm 1. EC-EDP-V

Input: $G(V, E)$, M, h_{limit}
Output: Minimum storage cost C, the best solution set S^*

Initialization:
$S^* \leftarrow \emptyset$;
$A_i \leftarrow$ servers that v_i can access within h_{limit};
$A_i^* \leftarrow$ the size of the edge server set A_i, $v_i \in V$;
End of initialization
for $M \leftarrow 2$ **to** $\arg\min A_i^*$ **do**
  initialize the number of coded blocks required per edge server $m_i \leftarrow M$;
  $C_M \leftarrow 0$;
  $S_M^* \leftarrow \emptyset$;
  while $\exists v_i \in V, m_i \neq 0$ **do**
    update the vote weight of edge server V_i by $w_i \leftarrow m_i$;
    for $v_i \in V$ **do**
      Vote for $v_j \in A_i \setminus S_M^*$ with w_j;
    end
    sort the edge servers in V by their votes;
    find v_k, i.e., the edge server with the most votes;
    $S_M^* \leftarrow S_M^* \cup \{v_k\}$;
    for $v_r \in A_k$ **do**
      $m_r \leftarrow m_r - 1$;
    end
  end
  $C_M \leftarrow |S_M^*|/M$;
  if $C_M < C$ **then**
    $C \leftarrow C_M$;
    $S^* \leftarrow S_M^*$;
  end
end
return C, S^*

This ILP can be solved by some classic widely-used integer programming solvers, such as IBM CPLEX Optimizer[2].

2. https://www.ibm.com/products/ilog-cplex-optimization-studio

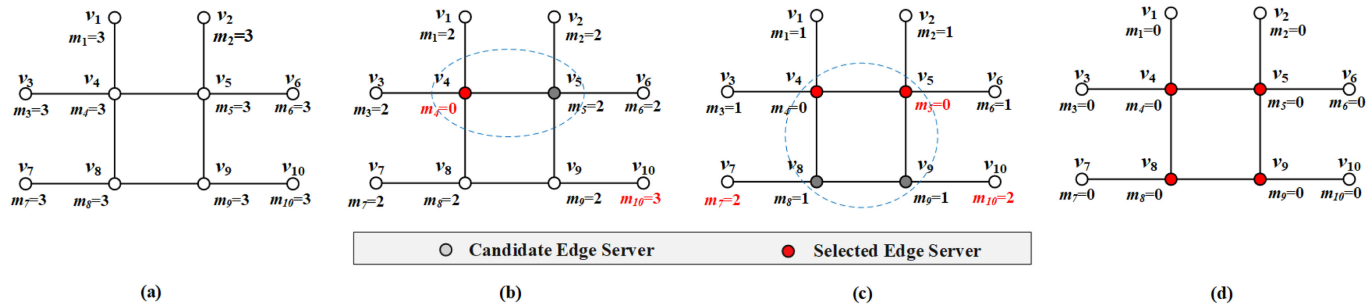


Fig. 4. Approximation process.

This optimal approach for solving the EC-EDP problem named EC-EDP-O. It uses three binary variables, i.e., $r_{i,t}$, $\alpha_{i,j}$, and $\beta_{i,j}$, and must fulfil three constraints. There are a total of $C_N^M C_S^M A_M^M$ feasible solutions. Since the number of data blocks obtained from encoding data $\mathcal{X}$ is between 2 and $S-1$, the size of the solution space is $(\frac{S!N!}{M!(N-M)!(S-M)!})^{S-1}$. It takes long time for EC-EDP-O to explore such a large solution space even in a small-scale EC-EDP scenario discussed in Section 2.

5.2 Approximation Approach

As proven in Section 4.2, the EC-EDP problem is $\mathcal{NP}$-hard. EC-EDP-O may solve small-scale EC-EDP problems. However, it is not tractable in large-scale EC-EDP scenarios. To address the complexity of solving large-scale EC-EDP scenarios, an efficient approximation approach named EC-EDP-V is proposed. Algorithm 1 presents the pseudocode of EC-EDP-V, and Fig. 4 illustrates its approximation process for finding the solution to the EC-EDP problem presented in Fig. 2.

The key idea of EC-EDP-V is to select the edge servers that produce the maximum benefits by storing coded blocks in the edge storage system. We design a new voting mechanism where EC-EDP-V adjusts the voting weight for each edge server and each edge server votes for the edge servers within its h_{limit} hops iteratively. In each iteration, the edge server with the highest number of votes will be selected. During the voting process, if multiple edge servers have the same highest number of votes, EC-EDP-V will randomly select one of them and update the vote weight for each edge server. First, the algorithm starts with an initial $S^* = \emptyset$, which is used to save the current optimal solution of EC-EDP problem (Line 2). Note that A_i $(i = 1, ..., n)$ on Line 3 is the set of edge servers within h_{limit} hops from v_i. On Line 4, A_i^* is the number of neighbor edge servers of edge server v_i within h_{limit} network hops limit. To find the final solution, the algorithm iterates for n times, one for each of the number of data blocks M, to produce n candidate EC-EDP solutions on Lines 7-21. In each iteration, the algorithm initiates the number of coded blocks needed for each edge server $m_i = M$ (Line 7), the current storage cost $C_M = 0$, and the set of selected candidate edge servers $S^* = \emptyset$ (Lines 7-9). Take the ESS presented in Fig. 4a for example. Let us assume that data can be transmitted via two network hops. Edge servers $v_1, v_2, ..., v_{10}$ are initialized with the same vote weight of 3, i.e., the number of coded blocks needed for each edge server. Then, the algorithm loops Lines 10-21. In each iteration of the loop, it assigns m_i as the vote weight w_i to each edge server $v_i \in V$ without a coded block (Line 11). Next, all the edge servers within h_{limit} hops from v_i vote for v_i with vote weight w_j (Lines 12-14). As shown in Fig. 4b, edge servers v_4 and v_5 receive 27 votes from their neighbor edge servers within 2 hops, i.e., 3 votes from each of v_1, ..., v_9 and 3 votes from each of v_1, ..., v_6 and v_8, ..., v_{10}, respectively. In this example, v_4 is chosen over v_5. After that, all the edge servers in V are sorted by the number of their votes, the algorithm selects the one (v_k) with the most votes to be included into S^*, i.e., the set candidate edge servers (Lines 15-17). Next, for each edge server $v_r \in A_k$, the number of its required coded blocks m_r decreases by 1 (Lines 18-20). In this way, as shown in Fig. 4b, when edge server v_4 is chosen, its vote weight m_4 will decrease to 0. Let us now take a look at Fig. 4c, where v_4 and v_5 are chosen for their highest votes. The vote weights of their neighbor edge servers within 2 hops, including $v_1, v_2, v_3, v_6, v_8, v_9$, and v_{10}, decrease by 1. Next, it compares the current storage cost C_M with all the candidate EC-EDP solutions. If it is lower than the current lowest storage cost, the corresponding EC-EDP solution S_M^* replaces the current best solution (Lines 22-25). As shown in Fig. 4d, the final solution contains v_4, v_5, v_8, and v_9. It achieves the lowest storage cost ratio of 1.33.

5.3 Theoretical Analysis

In this section, we theoretically analyze the approximation ratio and time complexity of the proposed approach EC-EDP-V.

5.3.1 Approximation Ratio

Given an edge server network $G = (V, E)$, let $N_{h_{limit}}(v_i)$ denote the set of edge server v_i's neighbor edge servers within h_{limit} hops, $\Theta_{h_{limit}}(G)$ denote the maximum number of $N_{h_{limit}}(v_i)$, $\lambda_{opt} = \{\lambda_{opt}^0, ..., \lambda_{opt}^{n-1}\}$ denote the optimal solution to the EC-EDP problem, and λ denote the sub- optimal solution found by EC-EDP-V. For each edge server over the network topology of edge servers, the number of its neighbor edge servers within h_{limit} network hops is less than $\Theta_{h_{limit}} + 1$. When an edge server with the most votes is included into OPT, we have the following inequality

$$n \leq (\Theta_{h_{limit}} + 1) + \Theta_{h_{limit}} \cdot (|\lambda_{OPT}| - 1) \tag{18}$$

From (18), we can infer $|\lambda_{OPT}| \geq (n-1)/\Theta_{h_{limit}}$. Let us assume that the number of remaining encoded blocks to be placed after the i-th iteration in Algorithm 1 is ψ_i with $\psi_0 = n$. Considering the i-th iteration, the optimal solution can reduce the number of coded blocks by $\psi_i - 1$. Thus, the lower bound of the number of selected edge servers in the i-th iteration by Algorithm 1 is $\lceil (\psi_i - 1)/|\lambda_{OPT}| \rceil$. Now, we can infer:

$$\begin{aligned}\psi_{i+1} &\leq \psi_i - \lceil(\psi_i - 1)/|\lambda_{OPT}|\rceil \\ &\leq \psi_i\left(1 - \frac{1}{|\lambda_{OPT}|}\right) + \frac{1}{|\lambda_{OPT}|} \end{aligned} \tag{19}$$

By the inductive proof, we can easily prove (20) based on (19). The details of the proof are omitted here.

$$\begin{aligned}\psi_i &\leq \psi_0(1 - \frac{1}{|\lambda_{OPT}|})^i + \frac{1}{|\lambda_{OPT}|}\sum_{j=0}^{i-1}\left(1 - \frac{1}{|\lambda_{OPT}|}\right)^j \\ &= (\psi_0 - 1)\left(1 - \frac{1}{|\lambda_{OPT}|}\right)^i + 1 \end{aligned} \tag{20}$$

When $i = |\lambda_{OPT}| * ln\frac{\psi_0 - 1}{|\lambda_{OPT}|}$, and the i^{th} sub-decision is made, we can obtain the number of the remaining coded blocks to be placed as follow:

$$\begin{aligned}\psi_i &\leq (\psi_0 - 1)\left(1 - \frac{1}{|\lambda_{OPT}|}\right)^i + 1 \\ &= (\psi_0 - 1)\left(1 - \frac{1}{|\lambda_{OPT}|}\right)^{|\lambda_{OPT}|ln\frac{\psi_0-1}{|\lambda_{OPT}|}} + 1 \\ &\leq (\psi_0 - 1)\left(\frac{1}{e}\right)^{ln\frac{\psi_0-1}{|\lambda_{OPT}|}} + 1 = (\psi_0 - 1)\frac{|\lambda_{OPT}|}{\psi_0 - 1} + 1 \\ &= |\lambda_{OPT}| + 1 \end{aligned} \tag{21}$$

This proves that after the $|\lambda_{OPT}| \cdot ln\frac{\psi_0 - 1}{|\lambda_{OPT}|}$-th iteration, the number of remaining coded blocks will not exceed $|\lambda_{OPT}| + 1$. Let us assume that the iterative process will end by selecting ψ_f more edge servers. The total number of selected edge servers fulfills:

$$|\lambda| = |\lambda_{OPT}| \cdot ln\frac{\psi_0 - 1}{|\lambda_{OPT}|} + \psi_f \leq ln(\Theta_{h_{limit}} + 1) \cdot |\lambda_{OPT}| \tag{22}$$

Therefore, the approximation ratio of EC-EDP-V algorithm is $ln(\Theta_{h_{limit}} + 1)$.

5.3.2 Time Complexity

Suppose an EC-EDP problem with n edge servers $V = \{v_1, v_2, ..., v_n\}$ in a geographic area. For each edge server $v_i \in V$, let p denote the average number of its neighbor edge servers within h_{limit} hops. We first analyze the time complexity of Lines 11-14 of Algorithm 1. The voting process takes $O(n)$ time because all of n edge servers will vote. The time complexity of sorting these edge servers and selecting the highest one on Line 15 at most $O(\log n)$. The upper limit of M impacts the number of inner iterations (Lines 10-21), which is determined by $\arg min|A|$. When $\arg min|A^*| \geq p$, the complexity of the overall process in the worst-case EC-EDP scenario is no more than $O((n-p)\log n)$. After the inner iteration (Lines 10-21), Algorithm 1 has obtained a total of $n-2$ candidate solutions. Therefore, the time complexity of Algorithm 1 is $O(n^2 \log n)$.

6 Evaluation

In this section, we conduct extensive experiments to evaluate the performance of EC-EDP-O and EC-EDP-V in different EC-EDP scenarios.

6.1 Settings

6.1.1 Dataset

In order to evaluate these competing approaches realistically, we conduct the experiments on the realistic EUA data set[3]. This dataset contains 1,464 real-world edge servers and 131,312 users in Metropolitan Melbourne, Australia.

6.1.2 Competing Approaches

Five representative approaches are implemented in Java 8 to be compared against EC-EDP-O and EC-EDP-V:

- *Greedy Degrees* (GD): This EC-based approach selects the edge server with the highest degree to place coded blocks each time, until all the users are covered, i.e., they can all retrieve adequate coded blocks within h_{limit} hops.
- *Random Block Placement* (RBP): This EC-based approach randomly selects one edge server at a time to place a coded block, one after another until all the users are covered.
- *LGEDC* [33]: This replica-based approach heuristically places replicas of data $\mathcal{X}$ to minimize the storage cost while covering all the users within h_{limit} hops.
- *GRED* [11]: This replica-based approach tries to spread replicas of data $\mathcal{X}$ across all the edge servers. Specifically, it first heuristically selects $\frac{n}{h_{limit}+1}$ candidate edge servers that can serve all the users with minimum data retrieval latency within h_{limit} hops. Then, from these candidate edge servers, it selects those that are connected to the fewest other candidate edge servers within h_{limit} hops until all the users are covered.
- *TMC18* [34]: This replica-based approach partitions edge servers into multiple groups with the Lagrangian method based on the number of user requests for data $\mathcal{X}$ received by individual edge servers. Then, it always places replicas of data $\mathcal{X}$ in the group with the lowest overall number of replicas of data $\mathcal{X}$ until all the users are covered.

In the implementation of EC-EDP-O, IBM's CPLEX Optimizer is employed for finding the solution by traversing all possible solutions to the ILP problem.

6.1.3 Experiment Setup

Two scales of experimental settings are conducted. Set #1 is conducted within the Melbourne CBD area to evaluate the performance of EC-EDP-O and EC-EDP-V in small-scale EC-EDP scenarios. Set #2 is conducted within Metropolitan Melbourne to evaluate EC-EDP-V in large-scale EC-EDP scenarios. To facilitate comprehensive evaluations, we simulate different EC-EDP scenarios by varying the specific values of three setting parameters, as summarized in Table 2.

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

TABLE 2
Parameter Settings

	n	d	h_{limit}
Set #1.1	10, 15, ..., 35	1.0	1
Set #1.2	20	1.0, 1.3, ..., 2.5	1
Set #1.3	20	1.0	1, 2, ..., 5
Set #2.1	50, 100, ...,250	2.0	1
Set #2.2	150	2.0, 2.6, ..., 5.0	1
Set #2.3	150	2.0	1, 2, ..., 5

- *Number of edge servers* ($n = |V|$): This parameter is the size of the edge server network G, increasing from 10 to 35 in Set #1 and from 50 to 250 in Set #2.
- *Density of edge servers* ($d = |E|/|V|$): This parameter decides the density of the edge server network G. It increases from 1 to 2.5 in Set #1, from 2.0 to 5.0 in Set #2.
- *Hop limit* (h_{limit}): This parameter is specified to enforce the transmission constraint, increasing from 1 to 5 in both Set #1 and Set #2.

6.1.4 Performance Metrics

Two metrics are employed for performance evaluation:

- *Storage cost* (*cost*). The storage cost denotes the ability of an approach to achieve the optimization objective of the EC-EDP problem. It is calculated by Eq. (11), the lower the better.
- *Computational overhead* (*time*). This metric indicates the efficiency of an approach, which is measured by the CPU computation time, the lower the better.

6.2 Experimental Results

In this section, we comprehensively present and analyze the experimental results in Set #1 and Set #2.

6.2.1 Experiment Set #1

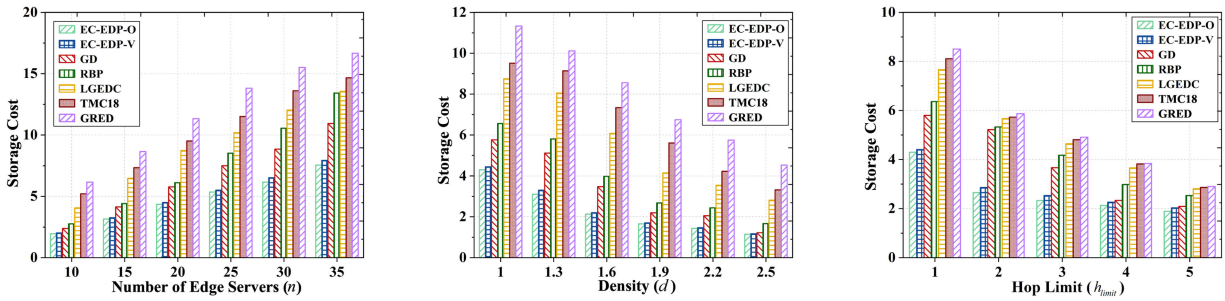
Effectiveness. *Fig. 5* illustrates the storage costs incurred by the seven approaches and impacts of the three parameters in Set #1. We can clearly see the significant advantages of EC-based approaches over replica-based approaches in minimizing storage costs. Among all the four EC-based approaches, EC-EDP-O and EC-EDP-V are the clear winners in all the cases. This illustrates the importance of leveraging the ability of edge servers to cost-effectively utilize the constrained and expensive storage resources in the ESSs. EC-EDP-O achieves the lowest storage cost in all the cases. Compared with EC-EDP-O, EC-EDP-V incurs about 3.94% more storage cost on average in Set #1. Meanwhile, EC-EDP-V incurs much less storage costs compared with GD, RBP, LGEDC, TMC18, and GRED, by 23.99%, 36.19%, 56.28%, 53.47%, and 58.29%, respectively.

Fig. 5a shows the impact of the number of edge servers n on the storage cost in Set #1.1. The storage costs incurred by the approaches increase when n increases. The storage costs incurred by LGEDC and RBP increase at higher rates compared with the other five approaches. When n increases, the scale of the EC-EDP problem increases. Accordingly, replica-based approaches need to place more data replicas to serve all the users. EC-based approaches also need to place more coded blocks to serve all the users. However, the total size of these extra coded blocks is much smaller than the extra data replicas to be placed by LGEDC, TMC18, and GRED. Among all the six approaches, EC-EDP-O always achieves the lowest storage costs, 4.01% lower than EC-EDP-V, 27.97% lower than GD, 33.35% lower than RBP, 59.43% lower than LGEDC, 53.30% lower than TMC18, and 60.06% lower than GRED on average.

Fig. 5b demonstrates the impact of the edge server density d on storage costs in Set #1.2. When d increases, the storage costs incurred by all the seven approaches decrease. The root cause is that a larger d connects each edge server to more adjacent edge servers within h_{limit} hops. Fewer coded blocks or data replicas need to be stored in the ESS to cover all the users. This immediately results in a decreases in the total storage cost incurred and indicates the importance of leveraging the collaboration of edge servers. In Set #1.2, EC-EDP-V outperforms GD, RBP, LGEDC, TMC18, and GRED by an average of 24.47%, 38.05%, 59.26%, 64.49%, and 70.98%, respectively. We can see that EC-EDP-O always achieves the lowest storage cost, 3.35% lower than EC-EDP-V on average.

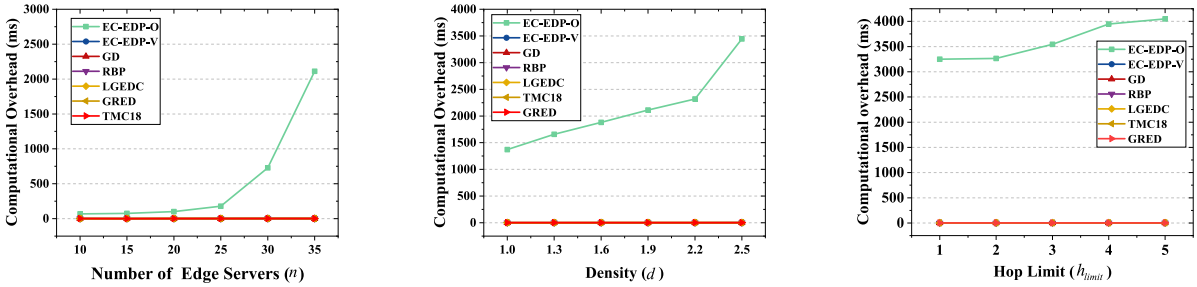
Fig. 5c shows the impact of the hop limit h_{limit} in Set #1.3. When h_{limit} increases, coded blocks or data replicas can travel via more hops to be delivered to the users. The total storage costs incurred by the approaches decrease accordingly. When h_{limit} varies from 1 to 5, EC-EDP-V outperforms GD, RBP, LGEDC, TMC18, and GRED by an average of 24.34%, 41.21%, 53.85%, 53.47%, and 58.29%, respectively. EC-EDP-O, again, achieves the lowest storage costs in all the cases, outperforming EC-EDP-V by 5.31% on average.

Efficiency. In *Fig. 6*, we can clearly see that EC-EDP-O incurs the highest computational overhead of all in the entire set of experiments. This is expected and confirms the



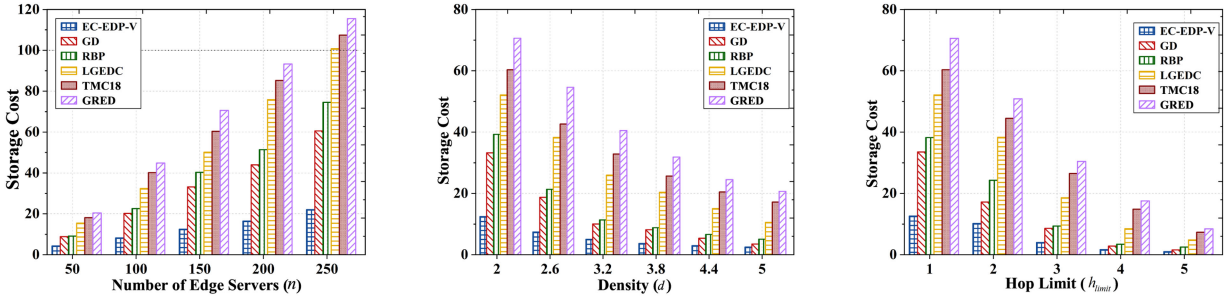
(a) Set # 1.1 (b) Set # 1.2 (c) Set # 1.3

Fig. 5. Effectiveness evaluation in Set #1.



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

Fig. 6. Efficiency evaluation in Set #1.



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

Fig. 7. Effectiveness evaluation in Set #2.

$\mathcal{NP}$-hardness of the EC-EDP problem proved in Section 4.2. As shown in *Fig. 6a*, EC-EDP-O takes an average of 2,112.40 milliseconds to find the optimal solution when $n = 35$ in Set #1.1. In the meantime, it takes up to 3,444.80 milliseconds in Set #1.2 and 4,047.70 milliseconds in Set #1.3 to find a solution, as illustrated in *Figs. 6b* and *6c*. The increase in $density$ in Set #1.2 connects the edge servers in the ESS with more edges. This rapidly increases the number of solutions that can cover all the users, taking EC-EDP-O considerably more time to find the optimal solution. In Section #1.3, as shown in *Fig. 6c*, the relaxation of h_{limit} also increases the number of solutions that can cover all the users, but less significantly compared with $density$. Interestingly, when h_{limit} increases from 4 to 5, the increase in EC-EDP-O's computational overhead is marginal. The reason is that EC-EDP-O can already cover all the users by storing coded blocks on a few of the "key" edge servers within four hops in most cases. A further increase in h_{limit} from 4 to 5 does not significantly expand the solution space for EC-EDP-O to explore, and thus does not increase its computational overhead much.

Compared with EC-EDP-O, EC-EDP-V is multiple-order-of-magnitude faster. For example, in Set #1, it takes only 1.087 milliseconds on average to find the solution, only 0.05% of what EC-EDP-O takes. In those figures, EC-EDP-O's multiple-order-of-magnitude higher computational overhead in Set #1 renders those of other approaches indistinguishable. The computational overheads of other approaches are close to 0, but not 0. Specifically, GD, RBP, LGEC, TMC18, and GRED's computation time is less than 5 milliseconds in Set #1. Therefore, *Fig. 6* does not illustrate EC-EDP-V's efficiency clearly. In the next section, without EC-EDP-O, we will illustrate and discuss the performance differences between EC-EDP-V and GD, RBP, LGEDC, TMC18, GRED in large-scale EC-EDP scenarios in Set #2 clearly.

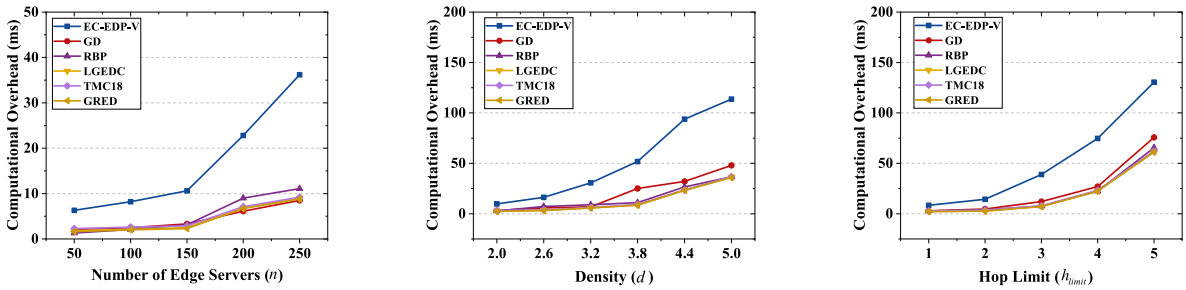
6.2.2 *Experiment Set #2*

Effectiveness. Fig. 7 demonstrates the advantages of EC-EDP-V approach in minimizing storage costs in large-scale EC-EDP scenarios. It always manages to achieve the lowest storage cost in all the cases in Set #2. Specifically, the storage cost achieved by EC-EDP-V is 55.63%, 65.7%, 79.01%, 81.06%, and 83.52% lower than GD, RBP, LGEDC, TMC18, and GRED, respectively.

As demonstrated in *Fig. 7a*, when n increases, the storage costs incurred by the EC-EDP strategies formulated by the approaches increase linearly. We can see EC-EDP-V's significant advantages over the other five approaches, i.e., 69.10%, 74.80%, 83.33%, 78.05%, and 80.24% over GD, RBP, LGEDC, TMC18, and GRED on average. The reason behind this is similar to Set #1 and thus is not repeated here. It is worth mentioning that when n reaches 250, the storage cost achieved by EC-EDP-V is only 20.23%, 19.56%, 18.23% of what is achieved by LGEDC, TMC18, and GRED. These are considerable storage cost savings and clearly show EC-EDP-V's prominent advantage over LGEDC, TMC18, and GRED in storing data cost-effectively in large-scale ESSs.

As illustrated in *Figs. 7b* and *7c*, the impacts of the increases in d and h_{limit} on storage cost in Set #2 are similar to what we observed in Set #1. Specifically, EC-EDP-V can save an average of 48.89% storage cost against GD, 61.15% against RBP, 76.58% against LGEDC, 82.86% against TMC18, and 85.61% against GRED. The underlying reasons are also similar to those in Set #1 and thus are not discussed in detail here.

Efficiency. Fig. 8 shows the computational time produced by all the approaches in Set #2. EC-EDP-V always takes more time than the other five approaches to find a solution, 170.71%, 205.59%, 241.88%, 233.26%, and 243.12% more than GD, RBP, LGEDC, TMC18, and GRED, respectively.



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

Fig. 8. Efficiency evaluation in Set#2.

Fig. 8 shows that computational overheads of all the the approaches increase gradually with n, d, and h_{limit}. Overall, EC-EDP-V scales with n, d, and h_{limit}, taking no more than 150 milliseconds. This significant advantages in minimizing storage costs over GD, RBP, LGEDC, TMC18, and GRED illustrated in *Fig. 7* make its extra computational time tolerable. When n, d or h_{limit} increases, the number of possible EC-EDP solutions that can fulfil all the constraints in the ILP model presented in Section 5.1 increases. According to Algorithm 1, it takes EC-EDP-V more iterations (Lines 10-21) to process the votes for each edge server. Thus, EC-EDP-V takes more time to complete.

6.2.3 Conclusion

The experimental results show that EC-EDP-O is a clear winner in small-scale EC-EDP scenarios, while EC-EDP-V is the best option for solving large-scale EC-EDP problems. They collectively offer a package for formulating cost-effective EC-EDP strategies in various ESSs.

7 Related Work

The edge computing paradigm enables data caching at the network edge by facilitating *edge storage systems* (ESSs) within users' close geographic proximity. ESSs offer various novel opportunities and also raise many new challenges. It has attracted widespread attention in very recent years [11], [14], [34].

Existing studies of ESSs are performed from the perspective of edge infrastructure provider, e.g., Amazon and Verizon, aiming to achieve various optimization objectives by storing or caching data and data replicas on the edge servers in an ESS. To name a few, Xie *et al.* [11] propose GRED, an efficient edge data placement algorithm that aims to balance the data retrieval workloads across the entire ESS and shorten the path for delivering data to users. Zhang *et al.* [34] explore data placement in ESSs to minimize overall data retrieval latency based on network topology, traffic distribution, and data popularity. Ren *et al.* [14] propose a cooperative edge data caching framework for ESSs that sets up cooperative caching regions to minimize data caching density and to promote data retrieval at the edge instead of from the remote cloud.

To accommodate data traffic at the network edge cost-effectively, network coding can be employed to split data into small blocks to be encoded for high data reliability and low storage occupation. Kim *et al.* [35] propose a coding framework that employ error-correcting data encoding and computation decoding to enable high data reliability in the edge computing environment. Wu *et al.* [36] introduce network coding into the mobile ad hoc network environment to minimize the energy required to transmit data between nodes. They model the physical broadcast links as a graph and construct a minimum-energy multicast tree as the optimal routing mechanism. Bulut *et al.* [37] study the erasure code based data routing problem in mobile networks and focus on parameter selection for reducing cost of message delivery. Xu *et al.* [38] propose a game theory based approach to jointly optimize the content service satisfaction degree and network throughput in edge caching systems by deploying network coding for data routing. However, these studies adopt the same assumption made for cloud storage systems, i.e., the storage nodes are fully and directly reachable to each other over high-speed links. This is, however, unrealistic in edge computing environment. In the edge computing environment, the topology of the edge server network must be properly considered.

In very recent years, researchers also start to investigate the use of ESSs from the perspective of app vendor. For example, Cao *et al.* [15] propose an auction-based approach for edge cache space allocation, aiming to maximize app vendor's caching benefits while guaranteeing the quality of services of different users. Xia *et al.* [23] propose CEDC-O, an online edge data caching algorithm, which aims to minimize app vendors' caching cost plus the data migration cost based on Lyapunov optimization. They also investigate the problem of cost-effective edge data distribution from the cloud to ESSs for app vendors [6].

It is widely acknowledged in these studies that the storage resources on edge servers are constrained and expensive [6], [39]. The competition among app vendors makes it hard and often impossible for them to hire or reserve adequate resources for storing large data. Thus, storing an app vendor's multiple data replicas in an ESS to serve users covered by different edge servers in the ESS will cost the app vendor deeply. It is in fact too expensive and too resource-demanding to be practical. Existing studies of ESSs accommodate app vendors' need for low service latency by leveraging the ability of ESSs to minimize data retrieval latency for users. There is a lack of effort in helping app vendors with storing large data in ESSs cost-effectively. In this paper, we innovatively employ erasure coding to tackle this particular challenge. The key idea is to encode data into a number of coded blocks to be placed on the edge servers in an ESS so that all the users in the ESS can be served at minimum storage cost. This problem is referred to as the EC-EDP problem in this paper.

8 Conclusion and Future Work

In this paper, we employ erasure coding to tackle the new EC-EDP problem of storing large data cost-effectively in an edge storage system, aiming to serve all the users in the system for app vendors at minimum storage cost. We first introduced, motivated, and formulated the EC-EDP problem. Then, we proposed two approaches, one for solving small-scale EC-EDP problems optimally and the other for finding approximate solutions provable performance guarantee in large-scale EC-EDP scenarios. The extensive experimental results indicate that by leveraging erasure coding and the ability of edge servers to cooperate, our approaches can formulate cost-effective EC-EDP strategies efficiently.

This study establishes the foundation for further study of the EC-EDP problem. In the future, we will study the trade-off between data reliability and storage cost in EC-based data storage at the edge.

References

[1] M. Noormohammadpour and C. S. Raghavendra, "Datacenter traffic control: Understanding techniques and tradeoffs," *IEEE Commun. Surv. Tut.*, vol. 20, no. 2, pp. 1492–1525, Apr.–Jun. 2018.

[2] Y. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing–a key technology towards 5G," *ETSI White Paper*, vol. 11, no. 11, pp. 1–16, 2015.

[3] A. Ceselli, M. Premoli, and S. Secci, "Mobile edge cloud network design optimization," *IEEE/ACM Trans. Netw.*, vol. 25, no. 3, pp. 1818–1831, Jun. 2017.

[4] H. Badri, T. Bahreini, D. Grosu, and K. Yang, "Energy-aware application placement in mobile edge computing: A stochastic optimization approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 4, pp. 909–922, Apr. 2020.

[5] A. Gharaibeh, A. Khreishah, B. Ji, and M. Ayyash, "A provably efficient online collaborative caching algorithm for multicell-coordinated systems," *IEEE Trans. Mobile Comput.*, vol. 15, no. 8, pp. 1863–1876, Aug. 2016.

[6] X. Xia, F. Chen, Q. He, J. 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.

[7] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1002–1016, Apr. 2017.

[8] Y. Li, X. Wang, X. Gan, H. Jin, L. Fu, and X. Wang, "Learning-aided computation offloading for trusted collaborative mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 19, no. 12, pp. 2833–2849, Dec. 2020.

[9] 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.*, 2021, pp. 499–515.

[10] F. Zhang *et al.*, "Online learning offloading framework for heterogeneous mobile edge computing system," *J. Parallel Distrib. Comput.*, vol. 128, pp. 167–183, 2019.

[11] J. Xie, C. Qian, D. Guo, X. Li, S. Shi, and H. Chen, "Efficient data placement and retrieval services in edge computing," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 1029–1039.

[12] T. X. Tran and D. Pompili, "Adaptive bitrate video caching and processing in mobile-edge computing networks," *IEEE Trans. Mobile Comput.*, vol. 18, no. 9, pp. 1965–1978, Sep. 2019.

[13] X. Chen, L. He, S. Xu, S. Hu, Q. Li, and G. Liu, "Hit ratio driven mobile edge caching scheme for video on demand services," in *Proc. IEEE Int. Conf. Multimedia Expo*, 2019, pp. 1702–1707.

[14] C. Ren, X. Lyu, W. Ni, H. Tian, and R. P. Liu, "Profitable cooperative region for distributed online edge caching," *"IEEE Trans. Commun."*, vol. 67, no. 7, pp. 4696–4708, Jul. 2019.

[15] X. Cao, J. Zhang, and H. V. Poor, "An optimal auction mechanism for mobile edge caching," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 388–399.

[16] F. Lyu *et al.*, "LeaD: Large-scale edge cache deployment based on spatio-temporal WiFi traffic statistics," *IEEE Trans. Mobile Comput.*, vol. 20, no. 8, pp. 2607–2623, Aug. 2021.

[17] H. Wu, F. Lyu, C. Zhou, J. Chen, L. Wang, and X. Shen, "Optimal UAV caching and trajectory in aerial-assisted vehicular networks: A learning-based approach," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 12, pp. 2783–2797, Dec. 2020.

[18] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1619–1632, Aug. 2018.

[19] Z. Xu, L. Zhou, S. C. Chau, W. Liang, Q. Xia, and P. Zhou, "Collaborate or separate? Distributed service caching in mobile edge clouds," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 2066–2075.

[20] C. Huang *et al.*, "Erasure coding in windows azure storage," in *Proc. USENIX Annu. Tech. Conf.*, 2012, pp. 15–26.

[21] K. M. Konwar, N. Prakash, N. Lynch, and M. Médard, "A layered architecture for erasure-coded consistent distributed storage," in *Proc. ACM Symp. Princ. Distrib. Comput.*, 2017, pp. 63–72.

[22] 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.

[23] 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.

[24] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service placement with provable guarantees in heterogeneous edge computing systems," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 514–522.

[25] Z. Hong, W. Chen, H. Huang, S. Guo, and Z. Zheng, "Multi-hop cooperative computation offloading for industrial IoT-edge-cloud computing environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 12, pp. 2759–2774, Dec. 2019.

[26] Q. Zhang, Q. Zhang, W. Shi, and H. Zhong, "Firework: Data processing and sharing for hybrid cloud-edge analytics," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 9, pp. 2004–2017, Sep. 2018.

[27] Q. He *et al.*, "A game-theoretical approach for mitigating edge DDoS attack," *IEEE Trans. Dependable Secure Comput.*, to be published, doi: 10.1109/TDSC.2021.3055559.

[28] H. Weatherspoon and J. D. Kubiatowicz, "Erasure coding versus replication: A quantitative comparison," in *Proc. Int. Workshop Peer-to-Peer Syst.*, 2002, pp. 328–337.

[29] X. Li, R. Li, P. P. Lee, and Y. Hu, "OpenEC: Toward unified and configurable erasure coding management in distributed storage systems," in *Proc. USENIX Conf. File Storage Technol.*, 2019, pp. 331–344.

[30] S. Muralidhar *et al.*, "F4: Facebook's warm BLOB storage system," in *Proc. USENIX Symp. Operating Syst. Des. Implementation*, 2014, pp. 383–398.

[31] H. Y. Lin and W. G. Tzeng, "A secure decentralized erasure code for distributed networked storage," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 11, pp. 1586–1594, Nov. 2010.

[32] F. Grandoni, "A note on the complexity of minimum dominating set," *J. Discrete Algorithms*, vol. 4, no. 2, pp. 209–214, 2006.

[33] X. Xia *et al.*, "Graph-based optimal data caching in edge computing," in *Proc. IEEE Conf. Serv.-Oriented Comput.*, 2019, pp. 477–493.

[34] S. Zhang, P. He, K. Suto, P. Yang, L. Zhao, and X. Shen, "Cooperative edge caching in user-centric clustered mobile networks," *IEEE Trans. Mobile Comput.*, vol. 17, no. 8, pp. 1791–1805, Aug. 2018.

[35] K. T. Kim, J. C. Wong, and M. Chiang, "Coded edge computing," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 237–246.

[36] Y. Wu, P. A. Chou, and S.-Y. Kung, "Minimum-energy multicast in mobile ad hoc networks using network coding," *IEEE Trans. Commun.*, vol. 53, no. 11, pp. 1906–1918, Nov. 2005.

[37] E. Bulut, Z. Wang, and B. K. Szymanski, "Cost efficient erasure coding based routing in delay tolerant networks," in *Proc. IEEE Int. Conf. Commun.*, 2010, pp. 1–5.

[38] D. Xu, A. Samanta, Y. Li, M. Ahmed, J. Li, and P. Hui, "Network coding for data delivery in caching at edge: Concept, model, and algorithms," *IEEE Trans. Veh. Technol*, vol. 68, no. 10, pp. 10 066–10 080, Oct. 2019.

[39] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

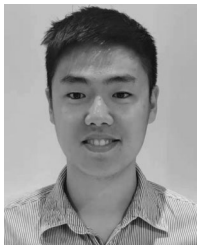
Hai Jin (Fellow, IEEE) received the PhD degree in computer engineering from the Huazhong University of Science and Technology, China, in 1994. He is currently a chair professor of computer science and engineering with the Huazhong University of Science and Technology. He was with The University of Hong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

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, China. His research interests include edge computing, parallel and distributed computing, and network storage.

Qiang He (Senior Member, IEEE) received the first PhD degree from the Swinburne University of Technology, Australia, in 2009, and the second PhD degree in computer science and engineering from the Huazhong University of Science and Technology, China, in 2010. He is currently an associate professor with Swinburne. His research interests include service computing, software engineering, cloud computing, and edge computing.

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

Zilai Zeng is currently working toward the undergraduation degree with the Huazhong University of Science and Technology, China. His research interests include edge computing, parallel and distributed computing, service computing, and cloud computing.



Xiaoyu Xia received the master's degree from The University of Melbourne, Australia, in 2015. He is currently working toward the PhD degree with Deakin University. His research interests include edge computing, parallel and distributed computing, service computing, software engineering, and cloud computing.