IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 21 May 2023, accepted 27 June 2023, date of publication 3 July 2023, date of current version 7 July 2023.

*Digital Object Identifier 10.1109/ACCESS.2023.3291452*

RESEARCH ARTICLE

# A Secure Cloud-Edge Collaborative Fault-Tolerant Storage Scheme and Its Data Writing Optimization

**JUNQI CHEN[1,2], YONG WANG[1,2], MIAO YE[1,3], AND QIUXIANG JIANG[3]**

[1]School of Computer Science and Information Security, Guilin University of Electronic Technology, Guilin 541004, China
[2]Engineering Technology Research Center of Cloud Security and Cloud Service, Guilin University of Electronic Technology, Guilin 541004, China
[3]Guangxi Key Laboratory of Wireless Broadband Communication and Signal Processing, Guilin University of Electronic Technology, Guilin 541004, China

Corresponding author: Miao Ye (yemiao@guet.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 62161006 and Grant 61861013; in part by the Guangxi Innovation-Driven Development Project under Grant AA18118031; in part by the Foundation of Guangxi Key Laboratory of Wireless Broadband Communication and Signal Processing under Grant GXKL06220110 and Grant GXKL06230102; in part by the Innovation Project of Guilin University of Electronic Technology (GUET) Graduate Education under Grant 2023YCXB06; and in part by the Foundation of Key Laboratory of Cognitive Radio and Information Processing, Ministry of Education, GUET, under Grant CRKL220103.

**ABSTRACT** Fueled by the explosive growth of edge smart IoT devices, edge storage systems (ESS) have emerged as a new paradigm to support the efficient access of massive edge data. ESS can greatly alleviate the burden of cloud center and enhance the Quality of Experience (QoE) for users. However, despite the remarkable progress of ESS, it still faces the challenges of how to improve the systems fault tolerance ability and efficiency. Therefore, designing a secure and efficient fault-tolerant storage scheme is urgent and indispensable. Unfortunately, existing fault-tolerant schemes for ESS still retain various drawbacks, including: high edge storage overhead, hard to protect the edge data privacy and low data writing performance. Motivated by this, we propose a secure cloud-edge collaborative fault-tolerant storage scheme and its data writing optimization method. Precisely, we first propose a Hierarchical Cloud-Edge Collaborative Fault-Tolerant Storage (HCEFT) model to achieve system robustness, low edge storage overhead, and edge data privacy security. We further optimized the writing process of HCEFT by designing a data writing optimization method called ECWSS (Erasure Code data Writing method based on Steiner tree and SDN) to achieve a better trade-off between the data writing time and traffic consumption. Finally, Comprehensive comparison and extensive experiments show that our scheme can achieve better data robustness, availability and security. Moreover, the writing optimization method can reduce 13%-67% data write time and 20%-62% network traffic consumption while providing better network load balance performance.

**INDEX TERMS** Data writing, edge storage systems, erasure coding, fault-tolerant storage, SDN.

## I. INTRODUCTION

With the prosperity of the Industrial Internet of Things [1] and the commercialization of 5G [2], an exponential growth of data has been generated by edge devices, such as smart IoT devices, autonomous vehicles, and VR devices [3], etc. IDC has predicted that the data volume of edge IoT devices will reach 79.5ZB by 2025 [4], and 75% of the data is generated at the edges [5]. Within this context, the traditional cloud storage paradigm struggles to fulfil the low-latency requirements of numerous applications, due to the large distance between the clouds and the end devices. To tackle this challenge, Edge Storage System (ESS) has been proposed as an emerging solution [6]. ESS pushes the computing power

The associate editor coordinating the review of this manuscript and approving it for publication was Mehrdad Saif.

and data storage from the cloud to the edge, which are located near the end devices. By storing popular data on edge servers, leading app vendors such as Tiktok and Meta can implement a low latency data access for their users [7].

While providing unique benefits, the availability and reliability of ESS is widely concerned. In typical ESSs, edge nodes are distributed in different locations for data collection and storage [8]. These edge servers are usually owned to individuals or small institutions with limited security protection, making the edge data vulnerable to unknown risk [9], such as hardware failures, natural disasters, hacker attacks, or power outages [10]. Thus, incorporating fault-tolerant techniques to ensure the robustness and availability of edge storage systems is essential when server failures occur. Currently, two representatives of fault-tolerant mechanisms are used in storage systems: On the one hand, many ESS [11], [12], [13] deploy multiple replicas of each file, distributed across different edge servers. When data fails from an edge server, usable data can still be retrieved from the remaining replicas. Certainly, the implementation of multiple replicas inevitably triggers a significant increase in storage capacity to guarantee data reliability. On the other hands, several systems [14], [15], [16], [17] utilize erasure coding to provide the same level of fault tolerance as replication while utilizing fewer storage resources. Specifically, By encoding the data, erasure coding allows for the creation of redundant parity chunks that can be used to reconstruct the original information in the event of a node failure. The cost-efficiency of erasure coding made it widely used in today's data center [18].

Despite the aforementioned progress, significant challenges still persist in developing fault-tolerant storage scheme in edge scenarios. **1)** Edge servers have limited storage capacity, and impractical fault tolerance approaches will inevitably result in considerable storage overhead, particularly as data volumes continue to grow rapidly. **2)** Due to the inadequate protection measure in edge node, the current fault-tolerant strategies, including both multi-replication and erasure coding, are vulnerable to the risk of original data leakage to untrusted third-party nodes, thereby compromising data privacy. **3)** The network resources in edge scenarios are more heterogeneous and constrained, which significantly restricts systems performance, making it difficult to improve the data writing efficiency of the system.

In summary, how to improve the fault-tolerance ability of ESSs, with guarantee the privacy of edge user data and improve the data writing efficiency has still an urgent problem to be solved. Motivated by this, we proposed a Hierarchical cloud-edge collaborative fault-tolerant storage architecture based on Software Defined Network (SDN) and erasure codes and its data writing optimization method. This paper presents a novel approach to studying both the secure fault-tolerant architecture and its performance optimization for edge storage systems. Our major contributions include the following:

- We proposed the HCEFT Model based on a new segment based cloud-edge privacy-preserving code (CEPPC, present in section III) and SDN. Compare to the prior scheme, HCEFT permit that a) we use CEPPC avoid distributing the local data chunks to other edge servers, which strengthens the protection of local original data privacy, b) CEPPC transfer the parity chunks to cloud center, which can reduce the edge server storage overhead and enhance the original data availability for user, and c) HCEFT leverages SDN to realize efficient network management and performance optimization in edge scenario.
- To improve the data writing efficiency of HCEFT, we established a mathematical model that aims to minimize the write time and traffic consumption. Then, we design a data writing optimization method for HCEFT, which includes two phases: a) we construct the write topology based on SDN and Steiner tree to make a better trade-off between data writing latency and traffic. b) We transfer the write topology to a directed acyclic graph (DAG), which identify relationship between each node and the accurate write flow direction in the writing topology, then supports the efficient deployment of the write method.
- We conducted a comprehensive analysis of edge fault-tolerant schemes related to HCEFT model, to evaluate its reliability, availability, privacy security, and storage overhead. Then, we perform extensive experiments by using containernet [19], the results show our method can substantially reduce data writing time and traffic consumption while providing better load balancing performance.

The remainder of this paper is organized as follows: The remainder of this paper is organized as follows: Section II given the background and related work of ESS and its fault-tolerant strategy. Section III present the design of proposed CEPPC code and HCEFT model. Section IV formulates the data writing problem and present a detail description of its optimization method. Section V shows the comparison and evaluation results. Section VI concludes our study and suggests future research prospects.

## II. BACKGROUND AND RELATED WORK

In this section, we provide the background and related work in terms of Fault-tolerance strategy in ESSs and the basis of Erasure Coding.

### A. OVERVIEW

Taking Figure 1 as example, nine adjacent edge servers within a specific geographical location are connected by high-speed links to form an edge server network that constitutes an edge storage system. However, the wide geographical distribution and convenient access efficiency of ESS paradoxically expose it to greater security risks [20]. Thus, many efforts

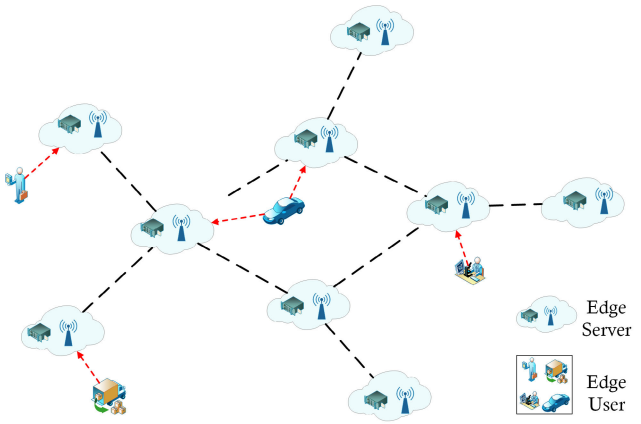


**FIGURE 1.** Example of edge storage system.

have tried to improve the robustness of ESS through fault-tolerance strategies.

### B. FAULT-TOLERANT STRATEGY IN EDGE STORAGE SYSTEM

#### 1) REPLICA-BASED STRATEGY

A straightforward solution is used multi-replicas distributed in the edge scenario to enhance the reliability of ESS [11], [12], [13], [21], [22], [23]. Gao et al. [11] design a replica-based collaborative edge storage model, which optimizes the number and location of replicas in dynamic network topology to improve the storage reliability and efficiency of mobile devices. Aral et al. proposed a decentralized dynamic replica management scheme D-ReP [12] in ESS. D-ReP evaluates the expenses associated with storing replicas, as well as estimates the potential improvement in latency, in order to determine whether to migrate or duplicate data to one of its neighboring nodes. In [13], Linaje et al. design a fault-tolerant scheme for wireless sensor networks FSD that is based on data distribution and replication storage. FSD addresses the issue of unbalanced storage by taking into account the heterogeneity of sensor nodes, allowing storage tasks to be allocated based on node capabilities and load conditions, ultimately resulting in a more equitable distribution of storage. References [21] and [22] optimizes the placement and the number of replicas in fog computing environment to improve the system fault-tolerant performance. In [23], Ke et al. proposed a priority-based multicast flow scheduling technique MDGA for replica-based edge data centers to enhance the backup efficiency and reliability in edge networks.

The aforementioned replica-based strategy [11], [12], [13], [21], [22], [23] can indeed improve system reliability while maintaining low access latency and high data availability. However, distributing replicas in such systems can incur significant storage overhead. Additionally, if any of the replica servers are hacked, it may result in the leakage of the entire original data, which undermines the protection of sensitive information at the edge. While it is true that some of the confidential data may be pre-encrypted, the additional computational burden and the potential risk of the encryption key being cracked cannot be ignored. Consequently, it is essential to prevent malicious actors from accessing the entire data or the ciphertext comprehensively.

#### 2) ERASURE CODE-BASED STRATEGY

Another solution is to enhance the fault-tolerance capability of ESS by utilizing erasure coding techniques. Taking the commonly used RS (Reed-Solomon) codes [24] as an example, RS $(k+r, r)$ encodes $k$ data blocks $(D_1, D_2, \ldots, D_k)$ into $r$ parity blocks $(P_1, P_2, \ldots, P_r)$ using Formula (1), where $C_{i,j}$ represents the coefficients for encoding $D_j$ to $P_i$, $1 \leq i \leq r$,$1 \leq j \leq k$. These $k+r$ chunks are then distributed across various nodes to form an erasure code stripe $S$. As long as any $k$ chunks in the $S$ are alive, the entire stripe can be reconstructed.

$$P_i = \sum_{j=1}^{k} C_{i,j} * D_j \tag{1}$$

In comparison with multi-replica schemes, erasure coding can not only provide high level of fault tolerance but also significantly reduce storage overhead [25]. Thus, emerging research efforts have been tried on exploring the application of erasure coding for fault tolerance in edge scenarios. Liang et al. [14] conducted extensive performance testing of erasure coding applications in edge scenarios. By leveraging multi-core CPUs and accelerating the process with OpenMP, they successfully improved the performance of erasure coding in 5G and WiFi6 use cases. Kim et al. [26] have presented a coding framework utilizing error-correcting data encoding and computation decoding to enhance edge data reliability. Wu et al. [15] designed a hybrid fault-tolerant strategy called MobileRE for mobile distributed system, which balances data fault-tolerant capability and storage cost by dynamically adjusting erasure coding and replica fault-tolerant modes based on network status. Reference [16] proposed a security and trust-oriented edge storage mode (TDOA). TDOA designed a new variant of Local Reconstruction Codes (LRC) called TLRC to enhance the data robustness and adaptability of ESS in IoT environment. Jin et al. [17] considered the efficient data layout strategy of erasure codes in ESS and modeled the problem as an integer programming problem to minimize the cost of fault-tolerant strategy. Lin and Tzeng [27] proposed a secure erasure code storage model by combining the threshold public key encryption scheme, which improves the storage security by setting storage servers and key servers. Similarly to the previously mentioned erasure code schemes, this scheme may harm the availability of the original file since it still needs to split the original file into data chunks and distribute them among other nodes.

Moreover, the efficiency of data writing is a core issue in fault-tolerant storage and a critical factor in the deployment of ESS. Based on Formula (1), we can observe that

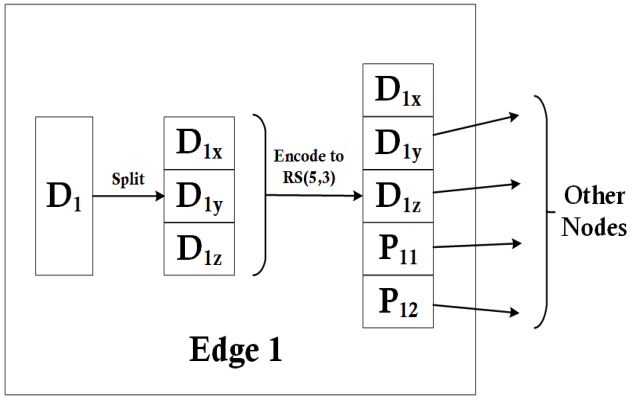


**FIGURE 2.** RS and LRC class code distribute the chunks to other nodes, where D represents the data chunk and P represents the parity chunk.

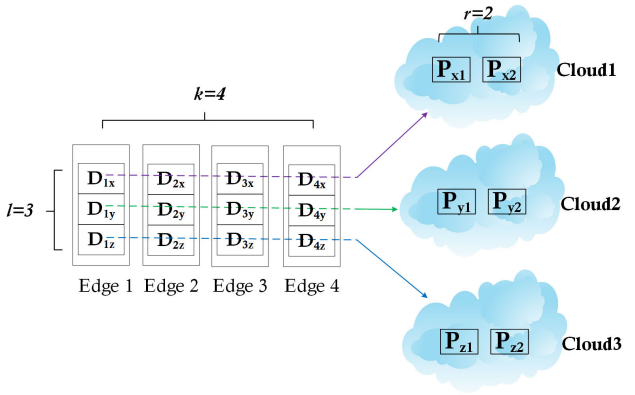


**FIGURE 3.** The CEPPC(4,3,2) code reserve the data chunks D in own edge nodes and place the generated parity chunks P in the cloud.

unlike replica fault-tolerance mode, erasure coding involves more encoding computation and network transmission during data writing. If the writing process takes too long and data loss occurs, the stripe may face the situation where data cannot be recovered. The current mainstream data writing methods and problem analysis were introduced in Section IV.

Compared with the replica-based strategy, existing erasure code-based strategy [14], [15], [16], [17], [24], [25], [26], [27] can greatly reduce storage overhead and avoid complete data leakage, since each edge node only retains a small fragment of the entire file. However, as slices of the original data are distributed across multiple nodes, users are required to download a certain amount of data fragments from other nodes to access the complete data, which compromises the data availability. Furthermore, the additional encoding overhead and network traffic caused by erasure coding operations may significantly impair its writing efficiency and system load balancing. Therefore, it is essential to propose corresponding data writing optimization schemes to enhance the deployment efficiency of erasure coding-based fault-tolerant solutions in edge storage systems.

## III. DESIGN OF HIERARCHICAL CLOUD-EDGE COLLABORATIVE FAULT-TOLERANT STORAGE MODEL

In this section, we first propose a new variant of erasure code, called CEPPC. We show that even without encryption, CEPPC can effectively enhance the system's fault-tolerance capability and ensure the privacy of edge data, while using low storage resources consumption. Then, we present a secure cloud-edge collaborative storage model named HCEFT based on CEPPC. We describe the specific function of each layer in HCEFT and how they work together to ensure the secure and efficient storage of edge data.

### A. SEGMENT BASED CLOUD-EDGE PRIVACY-PRESERVING CODE

As shown in Figure 2, in both RS or LRC class coding schemes, regardless of their encoding algorithms, the data blocks obtained from the original data segmentation and the parity blocks generated during the encoding process must be distributed to different nodes to ensure high fault tolerance. However, these typical encoding schemes have several drawbacks. First, they harm the availability of the original file. Since the original file is split into data chunks and distributed among other nodes, accessing the original file requires downloading the distributed data chunks from many nodes. Second, distributing these data and parity chunks increases distribution traffic and imposes additional storage burdens on other edge servers. Third, distributing data and parity chunks to other servers still poses a privacy and security risk, as the original data can be easily recovered if any $k$ servers containing an RS $(k+r, r)$ stripe are dishonest. To address these challenges, we propose a new erasure code construction called CEPPC that achieves data robustness in ESS with higher privacy and security, improved availability of the original file, and lower storage overhead for edge servers.

CEPPC uses cross-node segments to encode the original file from different nodes. Specifically, for encoding CEPPC $(k, l, r)$, we need to select $k$ original file of the same size from each of the $k$ different nodes. Each original file is sliced in to $l$ data chunks, and then the $k \cdot l$ data chunks are linearly encoded to generate $l$ strips. These $l$ stripes form a group, each stripe has $r$ parity chunks. A group has $l \cdot r$ parity chunks. Compared to traditional RS and LRC class coding methods, CEPPC does not need to distribute the data chunks of the local original file to other servers, and also does not need to store the other server data chunks. This balances the storage load while reducing the delay of data chunks distribution and improving the security of the local original file. CEPPC $(k, l, r)$ has a total of $m=k \cdot l$ data chunks and $l \cdot r$ parity chunks. The total number of chunks in CEPPC $(k, l, r)$ is $n=k \cdot l + l \cdot r$ . Therefore, the fault tolerant storage overhead is $\frac{n}{k \cdot l} = \frac{k \cdot l + l \cdot r}{k \cdot l}$ . In our example, CEPPC (4,3,2) with storage overhead of $\frac{4 \cdot 3 + 3 \cdot 2}{4 \cdot 3} = 1.5$x, as illustrated in Figure 3.

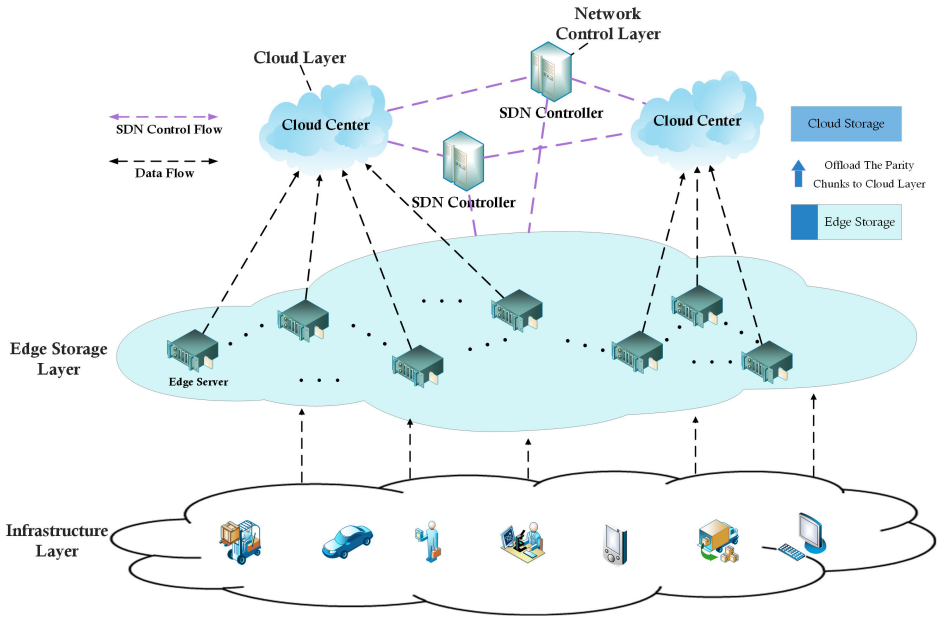


**FIGURE 4.** The architecture of HCEFT.

In summary, the CEPPC scheme ensures privacy and security of the original data by dividing data into chunks and only allowing each server to store the parity segments, without access to other servers' raw data chunks. The placement of parity chunks in the cloud center reduces storage overhead in edge nodes, making CEPPC highly efficient in storage space utilization. In the event of a complete loss of an edge node, data can still be recovered through all sub-stripes, ensuring data fault tolerance. Furthermore, CEPPC has the same maximum distance separable property [28], [29] as RS codes, making it able to tolerate the failure of any $r$ nodes.

### B. THE ARCHITECTURE OF HCEFT

In this section, we propose a hierarchical cloud-edge collaborative fault-tolerant scheme based on CEPPC called HCEFT. Firstly, we use CEPPC codes to achieve fault-tolerant storage of edge data blocks, while enhancing the availability and security of edge servers' sensitive data. Additionally, the generated parity blocks are stored in the cloud data center to alleviate the storage burden on the edge servers. We also leverage SDN technology to efficiently manage and flexibly configure the cloud-edge storage network, providing support for optimized data writing

As shown in Figure 4, HCEFT consists of 4 layers: the cloud center layer, the network control layer, the edge computing layer, and the infrastructure layer. Specifically:

- The cloud layer refers to a remote large-scale cluster of cloud servers with powerful computing and storage resources.
- The network control layer is composed of SDN controllers that centrally manage and flexibly configure the cloud-edge collaborative network to reduce data distribution delay and provide support for optimizing network performance.
- The edge storage layer mainly consists of edge network devices such as routers, switches, base stations, and edge servers with certain storage and computing capabilities. Edge servers form a distributed storage system to enhance edge computing storage capacity and security.
- The infrastructure layer includes edge data acquisition devices, industrial IoT devices, intelligent lathes, and other edge facilities.

In particular, HCEFT alleviates the edge storage overhead by utilizing a hierarchical multi-level edge storage model. The storage capacity and network stability of the entire hierarchical storage model gradually increase from bottom to top. Thus, the local edge nodes focus on real-time feedback and temporary data storage, while the cloud data center focuses on high performance and permanent storage. This hierarchical storage architecture not only meets the requirements of real-time data, but also reduces the loss of important data by solving the problem of limited storage space at the lower levels.

Furthermore, the data stored in the edge storage server nodes are likely to be accessed by users efficiently, requiring high data availability. Therefore,we adopt the CEPPC encoding method, which can provide high data availability while ensuring data fault tolerance and privacy security. Finally, HCEFT leverages SDN technology to effectively manage and flexibly configure the cloud-edge storage network, offering optimized support for data writing during HCEFT deployment. Since the SDN control plane enables centralized management and configuration of the network, as well as collecting node and network status information in real-time,

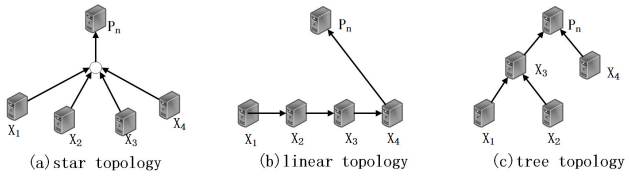


**FIGURE 5.** Example of different types structure of write topology.

providing essential data support for path planning, topology management, and flow table distribution.

## IV. DATA WRITING OPTIMIZATION

In this section, we present a mathematical model that minimizes data writing latency and traffic. we propose a data write topology construction method based on SDN load awareness and Steiner trees. Then, we present WriteDAG transfer the write topology to a directed acyclic graph supports the efficient deployment of the write method. These methods enhance system deployment efficiency and ensure its performance and security.

### A. PROBLEM ANALYSIS

The efficiency of data writing is a core issue in fault-tolerant storage and a critical factor in the deployment of HCEFT, as it directly impacts the system's performance and security. Particularly for fault-tolerant systems that use erasure coding strategies, data writing involves more coding calculations and path selection than replica-based fault tolerance, making it difficult to improve efficiency. According to Section III, due to the characteristic of CEPPC encoding, data blocks are retained in the local node, reducing the distribution latency. Therefore, the entire writing process can be considered as edge nodes encoding and writing data to the cloud center. However, in the edge storage scenario, network resources are dynamic and scarce, constructing an efficient encoding topology is critical to improving data writing efficiency [30].

There are currently three typical writing topologies for erasure coding: centralized star, linear topology, and tree topology, as illustrated in Figure 5(a), 5(b), and 5(c), respectively. Compared to the star [31], [32]and linear [33] topology, organizing the write topology in a tree-structure not only effectively disperses the computational pressure of the nodes but also makes full use of the network's bandwidth resources, reducing the time and traffic consumption during the write process [34]. Here, we focus on the construction problem of the tree-type write topology.

In the cloud-edge collaborative fault-tolerant storage system shown in Figure 6, it contains 14 edge nodes and a cloud center, the number next to the edge indicates the bottleneck bandwidth of the edge, the unit of bandwidth is MBps. Assume that there are 6 data chunks in CEPPC code denotes A,B,C,D,E,F, and the parity chunk P=A⊕B⊕C⊕D⊕E⊕F located in the cloud center. According to literature [35], 90% of the time for data writing and repairing time of erasure coding is transmission time, thus, We focus on

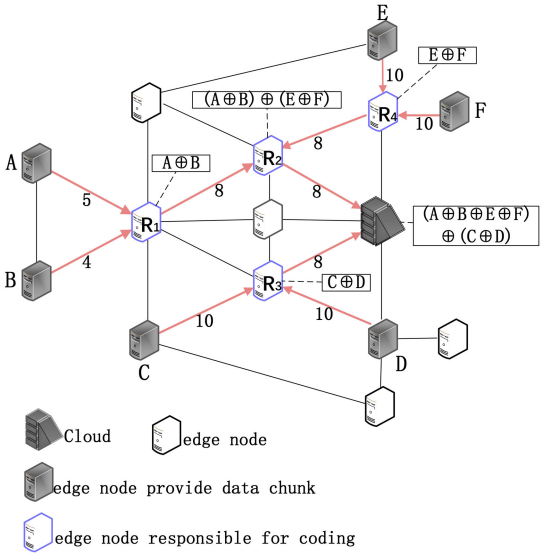


**FIGURE 6.** A cloud-edge fault-tolerant storage system consists of 14 edge nodes and a cloud center, where A,B,C,D,E and F are the edge nodes that provide data chunk. $R_1$,$R_2$,$R_3$, and $R_4$ are the edge nodes responsible for coding. The bottleneck bandwidth of a link is denoted by the number next to the tree edge.

improving data writing efficiency by optimizing the data writing topology in the edge network environment. In this context, based on the encoding and forwarding principle of erasure codes, each encoding node can encode only after receiving all the corresponding source data chunks. In the example of Figure 6, the data writing time of encode A,B,C,D,E,F to P, is determined by the largest cumulative bottleneck bandwidths from the leaf node to the root node. Specifically, The encoding completion time for A⊕B is determined by the latest arrival time of A and B at computing node $R_1$. Then, the encoding completion time for (A⊕B)⊕(E⊕F) is determined by the latest arrival time of (A⊕B) and (E⊕F) at computing node $R_2$. Finally, the encoding completion time for (A⊕B⊕C⊕D⊕E⊕F) is determined by the latest arrival time of ((A⊕B)⊕(E⊕F)) and (C⊕D) at the cloud center. Assuming the data chunk size is 10MB and using the write topology depict in Figure 6, the write finish time of P can calculate by 10MB/4MBps+10MB/8MBps+10MB/8MBps=5s, and the write traffic consumption is 10MB*10= 100MB for the write topology occupy 10 links in the network graph.

According to the description above, the time of data writing is determined by the largest delay from leaf node to root node, and the consumption of data writing traffic is determined by the number of tree edges. Therefore, it is essential to construct a data writing topology that minimizes the data write time and traffic consumption to optimize the data writing efficiency in fault tolerant storage system.

### B. MATHEMATICAL MODEL

The network topology of an ESS system can be modeled by an undirected graph $G(V_G, E_G)$, in which $V_G$ denotes

**TABLE 1. Symbols in the mathematical model.**

| Symbol | Description |
|---|---|
| $G$ | network topology |
| $V_G$ | vertices of $G$ |
| $E_G$ | edge of $G$ |
| $T$ | tree-based write topology |
| $V_T$ | vertices of $T$ |
| $E_T$ | edge of $T$ |
| $M$ | terminals for construct steiner tree |
| $P_{i,j}$ | The path between $i, j$ in the graph $G$ |
| $\beta$ | size of single data chunk |
| $e$ | a path |
| $B_e$ | bottleneck bandwidth of path $e$ |
| $T_R$ | the root node of tree $T$ |
| $T_L$ | the leaf node of tree $T$ |
| $X_{i,j}$ | binary variable for link selection |
| $\lambda_{i,j}$ | binary variable for path selection |
| $(a, b)$ | a link between two edge server $a$ and $b$ |

the set of edge servers and $E_G$ indicates the set of links between adjacent edge servers. According to the analysis in section IV-A, constructing a data writing topology for a parity chunk can be transformed into finding the Steiner tree [36], [37] $T(V_T, E_T)$ that contains all data chunks provider nodes $D$ and cloud center $C$, which is an NP-hard problem. Furthermore, we need to ensure that the write topology has the lowest write latency and traffic consumption to improve the data writing efficiency and network load balancing performance. For easy reference, the symbols used in this section are listed in Table 1.

### 1) THE WRITE TRAFFIC CONSUMPTION

Firstly, the write traffic is proportional to the number of the network links occupied by the write topology and proportional to the size of the write traffic on each link. Therefore, the objective function that minimizes the sum of write traffic of encoding blocks, can be expressed as Formula 2:

$$\min f_{traffic} = \beta \sum_{i,j \in V_T} x_{i,j} \tag{2a}$$

s.t.

$$x_{i,j} = \{0, 1\}, \quad \forall \{i, j\} \in V_G, \quad i \neq j \tag{2b}$$

In Constraint (2b), $x_{i,j}$ denotes a binary variable: it is 1 if edge $(i, j)$ add to $T$, and 0 other wise.

### 2) THE WRITE TIME CONSUMPTION

Furthermore, the transmission delay of writing data is proportional to the write data size and inversely proportional to the bottleneck bandwidth of the write path. The network traffic on each link is represented by the size of the data chunk $\beta$. According to section IV-A, the longest scheduling and forwarding delay from the leaf node to the root node is the write delay. The objective function for minimizing the writing delay of coding blocks can be expressed as Formula 3:

$$\min \quad f_{time} = \max_{,} \sum \frac{\beta}{b_e} \Big| e \in P_{T_L, T_R} \tag{3a}$$

s.t.

$$M = T_L \cup T_R \tag{3b}$$

$$b_e = \min_{(i,j) \in e} b_{(i,j)}, \forall i \neq j \tag{3c}$$

where $P_{T_L, T_R}$ represents the path from leaf node $T_L$ to root node $T_R$, $P$ consists of multiple edges $e$ in the tree. Edge $e$ is defined as the path between two adjacent endpoints in the tree, which is composed of multiple links $(i, j)$. $b_e$ is the bandwidth of edge $e$. Constraints (3c) indicates that the bandwidth of paths $e$ is determined by the bottleneck bandwidth of the link $(i, j)$ contained in $e$.

### 3) OBJECTIVE FUNCTION

Our goal is to achieve low write time and traffic consumption as much as possible. The overall optimization goal can be defined as Formula 4:

$$\min \quad f = [f_{traffic}, f_{time}] \tag{4}$$

However, due to the inherent trade-offs and interdependence between the $f_{traffic}$ and $f_{time}$, it is difficult to simultaneously achieve optimality for both. To deal with this problem, we can use the component weight to transform the objective Formula 4 into the optimization problem in Formula 5, where $c_{traffic}$ and $c_{time}$ represent the weight coefficients of traffic and time respectively.

$$\min f = c_{traffic} \left( \beta \sum_{i,j \in V_T} x_{i,j} \right) + c_{time} \left( \max_{,} \sum \frac{\beta}{b_e} \Big| e \in P_{T_L, T_R} \right) \tag{5a}$$

s.t.

$$c_{traffic} + c_{time} = 1 \tag{5b}$$

$$b_e = \min_{(i,j) \in e} b_{(i,j)}, \forall i \neq j \tag{5c}$$

$$M = T_L \cup T_R, M \in V_G \tag{5d}$$

$$\prod_{k,l \in M} \lambda_{k,l} \neq 0, \ \forall k \neq l \tag{5e}$$

$$\sum_{i,j \in V_T} x_{i,j} = |V_T| - 1 \ , \forall V_T \in V_G, \ V_T \neq \phi, \ i \neq j \tag{5f}$$

$$x_{i,j} = \{0, 1\}, \quad \forall \{i, j\} \in V_G, \quad i \neq j \tag{5g}$$

$$\lambda_{k,l} = \{0, 1\}, \quad \forall \{k, l\} \in V_G, k \neq l \tag{5h}$$

where the constraint (5d) and (5e) ensures that the nodes in the tree are connected to all terminal nodes in $M$, thereby guaranteeing that the generated Steiner tree is a connected graph. Constraint (5f) ensures that the constructed write tree topology $T$ is acyclic. In Constraint (5f) and (5g), $x_{i,j}$ denotes

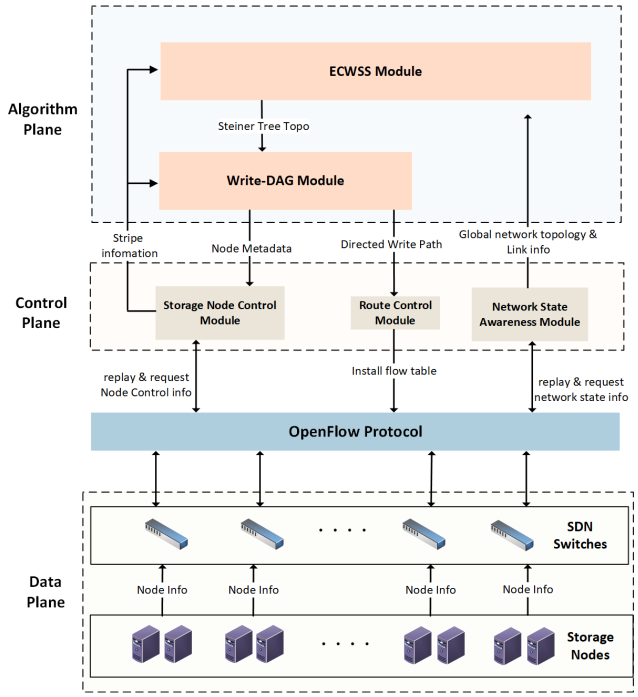


**FIGURE 7.** The architecture of data writing optimization method, which consists of a data plane, a control plane, and an algorithm plane.

a binary variable: it is 1 if edge $(i, j)$ is added to $T$, and 0 other wise. $\lambda_{i,j}$ denotes a binary variable: it is 1 if path $P_{k,l}$ is added to $T$, and 0 other wise.

### C. METHOD DESIGN

Based on the mathematical model we have described, we propose a new erasure code data writing method, named ECWSS-BA (Erasure Code data Writing method based on Steiner tree and SDN Bandwidth Aware). ECWSS-BA utilizes SDN to acquire the global network state, and consider both network bandwidth and hop to construct a minimum Steiner tree-based writing topology. The goal of ECWSS-BA is to minimize both write traffic and write latency.

Then, we proposed a node function recognition and write path partitioning method called Write-DAG to convert the Steiner tree-based write topology produced by ECWSS-BA into a directed acyclic graph. Write-DAG identifies the specific functions of each node involved in data writing, as well as the accurate data flow path, thereby facilitating policy deployment in the control layer of HCEFT.

Figure 7 illustrates the overall architecture of the data writing optimization method. It mainly consists of a data plane, a control plane, and an algorithm plane.

- Data plane

The data plane mainly consists of storage and network devices such as edge storage nodes, cloud center nodes, and SDN switches. The storage nodes and cloud center nodes are responsible for data reception, transmission, and encoding/decoding, while the SDN switches act as forwarding units, forwarding data according to the SDN flow tables issued by the controller.

- Control plane

The control plane serves as a bridge between the algorithm plane and the data plane, providing the algorithm layer with global network status information and node information, and delivering control policies to the data layer. It consists of a network state awareness module, a flow table deployment module, and a storage node control module.

The network state awareness module adopts the SDN-based network state measurement method proposed in the previous works [38], [39] to update and maintain real-time global network topology information and network link state information, which provides fundamental support for constructing the data writing topology.

The routing control module obtains the directed transmission path of data writing between nodes from the algorithm layer and deploys the SDN flow table according to this information. The storage node control module transfers the erasure-coded storage stripe information to the algorithm layer and instructs each storage node to prepare for data transmission, reception, or encoding/decoding based on the node metadata solved by the algorithm layer. These metadata include the functional partition of nodes during data writing, the next hop position, and source node information.

- Algorithm plane

The algorithm plane comprises the ECWSS-BA module and the Write-DAG module. Specifically, the ECWSS-BA module is utilized to construct a write topology that maximizes the bottleneck bandwidth while minimizing traffic consumption. On the other hand, the Write-DAG module transforms the generated tree topology into a Directed Acyclic Graph (DAG) [40]. It identifies the specific functions and accurate data flow directions of each node during data writing based on their degree, position, and category, thereby providing support for the deployment of control strategies in the control layer.

#### 1) GENERATE THE STEINER TREE-BASED WRITE TOPOLOGY

Firstly, we consider a network topology denoted as $G(V, E)$ and a terminal node set $M$, where $M$ comprises the data node set $D$ and the cloud center set $C$. To connect the terminal node set $M$, we employ the ECWSS-BA algorithm, which constructs a Steiner tree with larger bottleneck bandwidth and lower network traffic consumption. ECWSS-BA ensures efficient and optimal connectivity among the nodes in the network topology, thus enhancing the overall data writing performance of the system.

**Step 1**. Initialize the parameters, including the set of nodes to be connected: *Wait*; the set of edges to be added to the tree topology: *TreeEdge*; the set of nodes already added to the topology: *TreeNode*; Adding a new weight attribute $bw\_weight = 1/edge.bw$ to the links based on the current network state. $bw\_weight$ is proportional to the number

**Algorithm 1** ECWSS-BA

**Input:**
global network topology of ESS $G(V, E)$;
data node set $D$;
cloud center set $C$;
**Output:**
undirecte steiner tree-based write topo *TreeGraph*

1: **Initialization**: *Wait* $\leftarrow< D + C >$, *TreeNode* $\leftarrow< D + C >$, *TreeEdge* $\leftarrow< \varnothing >$. *edge.bw_weight* $\leftarrow 1/edge.bw$
2: **while** *Wait* != $< \varnothing >$ **do**
3: $t_1 \leftarrow$ get_Farest_Node($G, C$, *Wait*)
4: $t_2 \leftarrow$ get_Nearest_Node($G, t_1, C$, *bw_weight*)
5: *edge* $\leftarrow$ get_Shortest_Path($G, t_1, t_2$, *bw_weight*)
6: *TreeEdge*.append(*edge*)
7: add the *TreeEdge.Node* to *TreeNode*
8: *Wait*.Remove($t_1, t_2$)
9: **end while**
10: *component_list* $\leftarrow$ get connected components from *TreeEdge*
11: **if** component_list is not connected **then**
12: connect_path $\leftarrow$ getMinCombinePath(*component_list*, *bw_weight*)
13: *TreeEdge*.append(*connect_path*)
14: **end if**
15: *TreeGraph*=CreateGraph(*TreeEdge*)/*construct steiner tree topology
16: **return** *TreeGraph*

of hops and inversely proportional to the residual network bandwidth: (Refer to line 1)

**Step 2**. Constructing edges to form the *TreeGraph* for writing data until all nodes in the *Wait* set are added to the *TreeEdge*.

Initially, the algorithm selects a node $t_1$ from the Wait set that has the maximum hop count distance from the cloud center $C$ (Refer to line 3). Then, the algorithm selects a node $t_2$ from the *TreeNode* set that has the smallest accumulated *bw_weight* value with $t_1$. If there are multiple nodes that have the same accumulated *bw_weight* value, the node that is closer to the cloud center is chosen as $t_2$ (Refer to line 4). The algorithm constructs an *Edge* by connecting the minimum *bw_weight* path between $t_1$ and $t_2$. If there are multiple paths that have the same *bw_weight* between $t_1$ and $t_2$, we select the path with the minimum *bw_weight* closer to the cloud center as *Edge*(Refer to lines 5). Then added The *Edge* to the *TreeEdge* set (Refer to lines 6).

Next, the algorithm adds all nodes in the *TreeEdge* to the *TreeNode* set, in preparation for constructing a new *Edge* in the next iteration(Refer to lines 7). Finally, the algorithm removes $t_1$ and $t_2$ from the *Wait* set and proceeds to the next iteration. (Refer to line 8) The algorithm repeats this process until the *Wait* is empty, indicating that all nodes in the *Wait* set have been added to the *TreeEdge*.

**Step 3**. Build *TreeGraph* based on *TreeEdge*.

First, obtain a list of connected components formed by *TreeEdge* according to the connectivity of each edge in *TreeEdge*, and get $k$-1 *connect_path* through a greedy approach to connect $k$ connected components between all connected components. Then, add *connect_path* to *TreeEdge* (Refer to lines 10-13).

**Step 4**. Build Steiner Tree Topology *TreeGraph* based on all *TreeEdges* (Refer to line 15).

**Step 5**. Return Steiner Tree Topology *TreeGraph* $T(V, E)$ (Refer to line 16).

Since the cumulative *bw_weight* of an edge is positively correlated with the hop count and negatively correlated with the link bandwidth, ECWSS-BA takes *bw_weight*=1/*edge.bw* as the main indicator for constructing the tree-like topology. Through a greedy approach to construct *TreeEdge* and connect each component based on the smallest cumulative *bw_weight*, ECWSS-BA ensures that the path with the smallest hop count and the largest bandwidth is selected during topology construction, avoiding selected links with low bottleneck bandwidth when constructing the topology. ECWSS-BA achieves a good balance between write flow and write latency. Moreover, the computational complexity of ECWSS-BA is $O(|D| + |C|)^2$.

### 2) TRANSFER THE WRITE TOPOLOGY TO DIRECTED ACYCLIC GRAPH

Algorithm 2 utilizes the *TreeGrpah*'s attribution, which includes the degree, location, and type of each node, to identify the specific function of each node during data writing and determine the data flow path. Then, generate the *NodeMetaData*, and transforms the *TreeGraph* into a Directed Acyclic Graph (DAG). The control layer then utilizes *NodeMetaData* and the DAG to ensure data encoding at the designated node and transmission along the designated path.

**Step 1**: Initialize the parameters, including some node role lists: *onlyDataNode* and *encodeNode*, the set of node metadata *NodeMetaData*, and the flow table delivery path list *DirectedFlowPath* (Refer to line 1-2)

**Step 2**: Based on the degree and type of each node in the input *TreeGraph* $T(V, E)$, we divide the nodes into different roles. There are three cases: A) If a node is not the cloud center node and has a degree of 1, it is only a data node and should be added to the *onlyDataNode* list. (Refer to lines 4-5); B) If a node is in the data node set $D$ and has a degree of 2, it should be added to the *encodeNode* list(If it has a degree of 2 but is not a data node, it is just a passing node and does not participate in encoding.) (Refer to lines 6-7); C) If a node has a degree $\geq 3$ and node $\notin C$, it is an encoding node and should be added to the *encodeNode* list. (Refer to lines 8-9)

**Step 3**: Determine the set of *FunctionNode* that participate in the data writing process, including *onlyDataNode*, *encodeNode*, and the *Cloud*. *FunctionNode* are closely involved in the data writing process, while other nodes are non-functional nodes. (Refer to line 12)

**Algorithm 2** Write-DAG

**Input:**
global network topology of ESS $G(V, E)$;
write topology $T(V, E)$;
data node set $D$;
cloud center set $C$;
**Output:**
*NodeMetaData*, *DirectedFlowPath*;

1: **Initialization**:
2: *onlyDataNode* $\leftarrow <\varnothing>$, *encodeNode* $\leftarrow <\varnothing>$, *NodeMetaData* $\leftarrow <\varnothing>$, *DirectedFlowPath* $\leftarrow <\varnothing>$.
3: **for** *node* in $T$.*nodes* **do**
4: **if** $T$.degree(*node*) = 1 and *node* $\notin C$) **then**
5: *onlyDataNode*.append(*node*)
6: **else if** $T$.degree(*node*) = 2 and *node* $\in D$) **then**
7: *onlyDataNode*.append(*node*)
8: **else if** $T$.degree(*node*) $\geq 3$ and *node* $\notin C$) **then**
9: *encodeNode*.append(*node*)
10: **end if**
11: **end for**
12: *FunctionNode* $\leftarrow$ *onlyDataNode* $\cup$ *encodeNode* $\cup$ *C*
13: **for** *node* $\in$ *FunctionNode* **do**
14: **if** *node* $\in$ C **then**
15: *node.src* $\leftarrow$ Get_Connect_functionNode(*node*)
16: **else if** *node* $\in$ *onlyDataNode* **then**
17: *node.dst* $\leftarrow$ Get_Connect_functionNode(*node*)
18: **else if** *node* $\in$ *encodeNode* **then**
19: *tempList* $\leftarrow$ Get_Connect_functionNode(*node*)
20: *node.dst* $\leftarrow$ Get_Nearst_Cloud(*tempList*)
21: *node.src* $\leftarrow$ *tempList* - *node.dst*
22: **end if**
23: *NodeMetaData*.append(*node*)
24: **end for**
25: **for** *node* in *NodeMetaData* **do**
26: **if** *node.dst*! $=<\varnothing>$ **then**
27: path=nx.shortest_path($T$, *node*, *node.dst*)
28: *DirectedFlowPath*.append(*path*)
29: **end if**
30: **end for**
31: **return** *NodeMetaData*, *DirectedFlowPath*

**Step 4**: Determine the source and destination nodes of each *FunctionNode* to construct *NodeMetaData*. There are also three cases: A) If a node $\in$ $C$, it has no *dst* and only has source node *src*, and its *src* are the *Functionnode*s which is directly connected to it. (Refer to lines 14-15); B) If a node is an *onlyDataNode*, it has only one directly connected *FunctionNode*, which is also its destination node *dst*. (Refer to lines 16-17); C) If a node is an *encodeNode*, its destination node *dst* is the nearest *FunctionNode* connected to it, and its source node *src* is all directly connected functional nodes except for the destination node. (Refer to lines 18-21)

**Step 5**: Build the *NodeMetaData* (Refer to lines 23)

**Step 6**: Based on the *src* and *dst* relationships of *FunctionNode* in the *NodeMetaData*, we can determine the data writing flow direction and form a directed acyclic graph (DAG). Then add the directed data transmission paths in DAG to the *DirectedFlowPath*. The *DirectedFlowPath* is the basis for SDN controller to install flow tables. (Refer to lines 25-30)

**Step 7**: Return *NodeMetaData* and *DirectedFlowPath*. (Refer to lines 31) The time complexity of the Write-DAG is O(|D|+|C|). Finally, the *NodeMetaData* and *DirectedFlowPath* are forwarded to the control layer by the algorithm layer.

## V. SYSTEM ANALYSIS AND PERFORMANCE EVALUATION

In this section, we first analyze the robustness and security of the proposed HCEFT and then evaluate the performance of the proposed writing optimization methods ECWSS-BA.

### A. SYSTEM ANALYSIS

The robustness and security of HCEFT are compared with those of related scheme, as shown in Table 2.

#### 1) FAULT-TOLERANT

The schemes mentioned above [11], [12], [13] are all replica-based fault-tolerant storage schemes, while [14], [16], [17], and HCEFT are erasure-coded-based fault-tolerant storage schemes. Reference [15] is a hybrid fault-tolerant storage scheme based on both replica and erasure codes. All these schemes provide high storage reliability and support data fault tolerance. Additionally, the fault-tolerant capability of these schemes depends on the number of their replicas and encoding parameters.

#### 2) STORAGE OVERHEAD

Replica-based schemes [11], [12], [13] typically require several times the storage cost of the original data to ensure fault-tolerant capabilities, resulting in the highest overhead. MobileRE [15] adopts a hybrid approach of erasure coding and replication to reduce some of the overhead and achieve a comparatively cost-efficient solution. Other coded-based schemes including HCEFT can provide lower storage space consumption while ensuring the same fault tolerance capabilities with replica-based schemes, but they all face serious challenges in terms of data write load and write traffic amplification, requiring optimization.

#### 3) DATA AVAILABILITY

Replica-based schemes [11], [12], [13] ensure high data availability as users can directly read replica data without downloading it from other edge nodes. MobileRE [15] adopts a hybrid fault-tolerant scheme using replicas and erasure codes, which also achieves high data availability.

However, the other encoding-based schemes [14], [16], [17] require downloading data blocks or parity blocks from third-party nodes to generate the required original data for reading, resulting in lower data availability. In particular, our HCEFT scheme employs CEPPC code to achieve data

**TABLE 2.** Comparison between the proposed scheme and other related schemes.

| | HCEFT | Replica-based [11] [12] [13] | RS and LRC-based [14] [17] | RoSES [16] | MobileRE [15] |
|---|---|---|---|---|---|
| Fault-Tolerant | yes | yes | yes | yes | yes |
| Low Storage Overhead | yes | no | yes | yes | no |
| High Data Availability | yes | yes | no | no | yes |
| Privacy Security | yes | no | no | yes | no |

fault-tolerant storage without the need to distribute the original data blocks to other nodes, thereby we can ensure the privacy and security of the original data blocks while also providing high data availability.

#### 4) PRIVACY SECURITY

The Replica-based schemes [11], [12], [13] require distributing the entire original data to other edge nodes to ensure system fault tolerance, which poses a serious privacy risk. Similarly, MobileRE [15] may also have replica data placed on non-local nodes, making privacy protection challenging. Other erasure coding schemes such as [14] and [17] also require distributing local data blocks and generated parity blocks to other nodes, leading to increased risk of leakage of original data blocks, and difficulty in ensuring that malicious recovery of the original data does not occur if the number of lost coding blocks exceeds the threshold. Therefore, they also face privacy challenges.

In particular, RoSES [16] protects edge data privacy through a novel edge access control method and generates non-global parity blocks. Moreover, our HCEFT adopts CEPPC code to provide fault tolerance for edge data without the need to distribute original data blocks to other nodes, thereby we can ensure the privacy of original data blocks while offering high data availability.

### B. DATA WRITING PERFORMANCE EVALUATION

To evaluate the data writing performance of proposed the ECWSS-BA in edge scenario, we deployed a prototype system based on Containernet [19], [41], which provided a more realistic environment for distributed storage systems using Docker containers instead of the single file system limitation of Mininet hosts. We utilized the Ryu [42] SDN controller to implement network measurement, Steiner tree topology generation, and Write-DAG modules. Additionally, we added coding and decoding modules for erasure coding to the dataNode in Containernet, enabling the system to perform actual coding and decoding operations.

The methods we compared include a centralized write approach Cent_Star [31], [32], a distributed star-based write method Dis_Star, and a linear pipeline write approach IncEncoding [33]. ECWSS-NX is a simplified version of ECWSS-BA that constructs a Steiner tree topology using the approximate method based on the Minimum Spanning Tree from the NetworkX [43]. Its time complexity is $O(|D| + |C|)^2$. Additionally, we compared these methods with four other erasure coding writing methods. The core

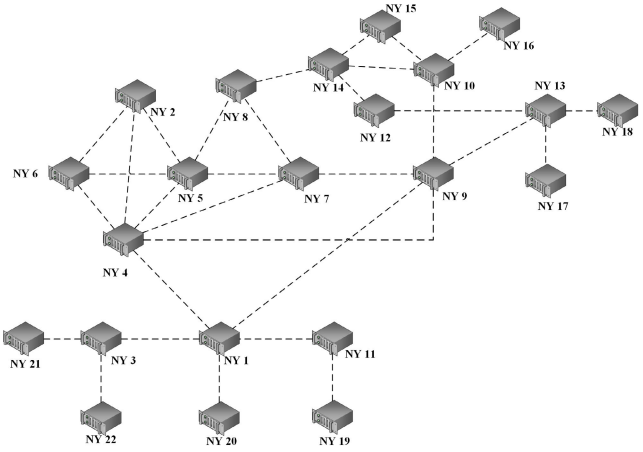


**FIGURE 8.** The experiment edge storage system's topology.

source code of our data writing method can be accessed via https://github.com/cookiecookiechen/ECWSS

#### 1) EXPERIMENTAL SETUP

The experimental prototype system based on Containernet was built on the Dawning A840r-G server, which has a 64-core * 2.1GHz processor, 128GB of memory, 2TB of HDD, and runs Ubuntu 18.04 as the operating system. The ESS network topology is based on the Equinix edge data center topology located in New York City [44], which has been extended to include 22 nodes as shown in Figure 8. The nodes in the topology were implemented using Docker in Containernet. During the data writing process, a random node was selected as the cloud center node, and the remaining nodes were used as edge nodes. The experimental setup was conducted under limited link resources, with each link bandwidth set to 300Mbps.

To validate the effectiveness of our data writing method in various network load scenarios, we simulated different network loads using the background traffic measured in our previous work [39] which was measured on a real-world distributed storage system. Specifically, we simulated multiple network loads by adjusting the number of background flows of different types. This approach allowed us to evaluate our proposed data writing method's performance in challenging network environments where network resources are more heterogeneous and scarce. We set three different scenarios: a late-night-time scenario with little network load, a middle network load scenario, and a heavy network load scenario.

- **Free Load Scenario (FL)**: 10 heart beating flows, 0 user data flows, and 0 migration flows

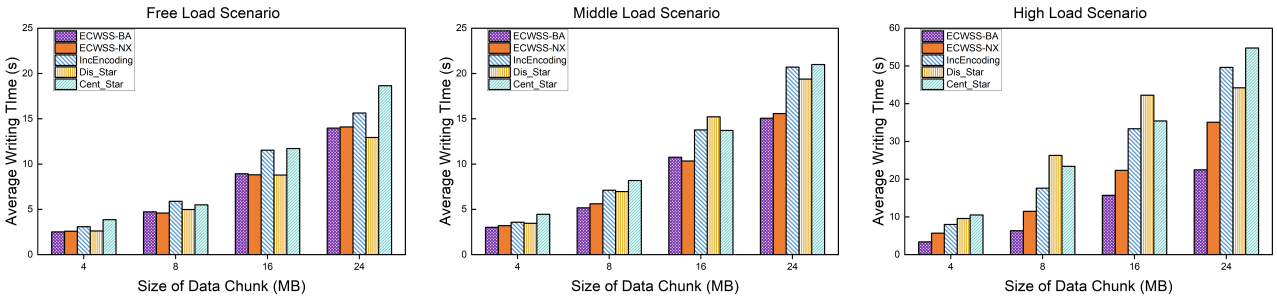


**FIGURE 9.** Influence of different encoding parameters on write completion time under different loads.

**TABLE 3.** Statistical Characteristics of Different Type Flows in Distributed Storage Systems.

| Traffic type | Speed (Mbps) | Duration (s) | Packet number | Avg. packet size (bytes) |
|---|---|---|---|---|
| Heartbeat traffics | 92.87 | 0.006554 | 54 | 1477.42 |
| User data traffics | 12.93 | 39.31 | 67073 | 1508.65 |
| Migration traffics | 4.36 | 654.12 | 340354 | 1505.56 |

- **Middle Load Scenario (ML)**: 20 heart beating flows, 20 user data flows, and 20 migration flows
- **High Load Scenario (HL)**: 40 heart beating flows, 40 user data flows, and 40 migration flows

For both user flows and migration flows, the traffic reproduction rate is consistent with the values presented in Table 3. For heartbeat data streams, the replay rate is set to 1 Mbps to mitigate the impact of packet loss in the experimental environment, according to [45]. All background flow is set to exist throughout the experiment to ensure that the network load during data writing remains stable across the different scenarios.

We compared the performance metrics of different writing methods under different loads and parameters during the data writing process. The performance metrics included the standard deviation of network link bandwidth utilization during data writing, network resource consumption, and the average write completion time. Each experiment was conducted 10 times, and the average value of the experimental results was taken. The experimental parameters are shown in Table 4.

### 2) COMPARISON OF AVERAGE WRITE TIME UNDER DIFFERENT LOADS

Figure 9 presents the write time of each data writing method as the size of the data chunk increase under varying load scenarios. As the total number of chunks increases, there is a corresponding increase in the amount of data that needs to be encoded and transmitted. Therefore, the data writing time also increases.

**TABLE 4.** Parameters in Experiments.

| Parameter | Ranges | Default |
|---|---|---|
| Number of Chunks (Data Chunks , Parity Chunks) | (4,2), (6,3), (8,4), (10,5) | (10,5) |
| Size of Data Chunks (MB) | 2,4,8,16,24 | 16 |
| Load Condition | Free Load, Middle Load, High Load | - |

Figure 9 also illustrates that ECWSS-BA and ECWSS-NX write topologies based on Steiner tree write topology construction takes less time to complete compared to Dis_Star, Cent_Star, and IncEncoding writing schemes. This is because ECWSS-BA and ECWSS-NX built write topology with a larger bottleneck bandwidth, avoiding traditional star and pipeline approaches that tend to select congested links and lead to a significant increase in write time.

Moreover, under high load scenarios depicted in Figure 9(c), ECWSS-BA reduced the completion time of data writing by 37%-67% compared to other schemes. Because ECWSS-BA considers link bandwidth a crucial parameter when constructing the data writing topology. As a result, it becomes easier to avoid congested links in high load scenarios, resulting in higher data writing efficiency.

Figure 10 depicts the impact of varying the number of data chunks and parity chunks on write time consumption under different load scenarios. As the total number of chunks increases, there is a corresponding increase in the amount of data that needs to be encoded and transmitted, resulting in longer data writing time. Notably, similar to Figure 9, ECWSS-BA and ECWSS-NX exhibit lower writing time consumption than other schemes, and ECWSS-BA reduced the completion time of data writing by 13%-62% compared to other schemes, when the number of chunks changes.

### 3) COMPARISON OF AVERAGE NETWORK RESOURCE CONSUMPTION UNDER DIFFERENT LOADS

Figure 11 presents the network traffic consumption of each data writing method as the size of data chunk increase under varying load scenarios.

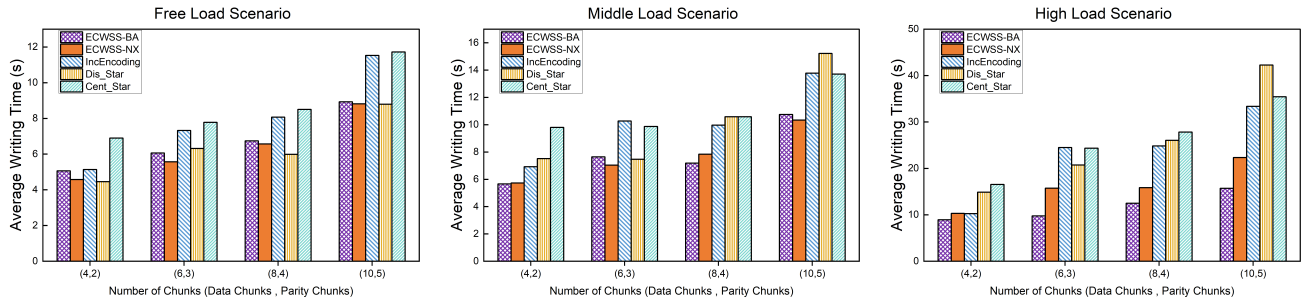


**FIGURE 10.** Influence of different chunk sizes on write completion time under different loads.

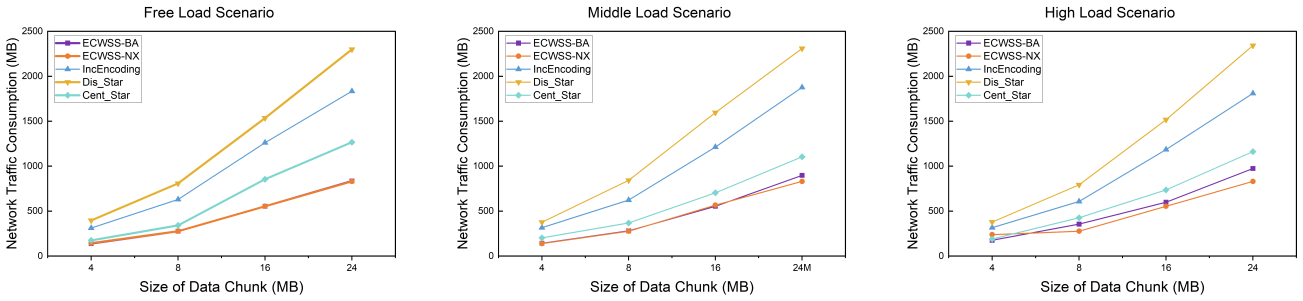


**FIGURE 11.** Influence of different block size on network flow consumption under different loads.

We can found that in Figure 11, in contrast to the star-shaped and linear writing schemes, ECWSS-BA and ECWSS-NX based on Steiner tree writing topology construction exhibit a reduction in writing traffic consumption by 28%-64%. Additionally, it is worth noting that Cent_Star, Dis_Star, IncEncoding, and ECWSS-NX exhibit writing traffic sizes that are independent of and insensitive to network load. This is due to these schemes' failure to consider link bandwidth during link construction, making them incapable of sensing changes in network load. Consequently, the writing traffic of these schemes remains relatively constant across different loads.

Moreover, it is worth noting that the traffic consumption lines of ECWSS-NX and ECWSS-BA coincide in the free load scenario depicted in Figure 11(a). This is because in free load scenario, the network bandwidth's impact on writing topology construction is limited, resulting in ECWSS-BA achieving similar traffic consumption to ECWSS-NX. However, in high-load scenarios depicted in Figure 11(c), ECWSS-BA tends to select idle links to improve the network load balancing performance, thereby avoiding links with less hops but heavier network loads during topology construction. Consequently, ECWSS-BA exhibits slightly higher network traffic consumption than ECWSS-NX under high load scenarios.

Figure 12 illustrates the impact of varying the number of data chunks and parity chunks on network traffic consumption under different load scenarios. As the total number of

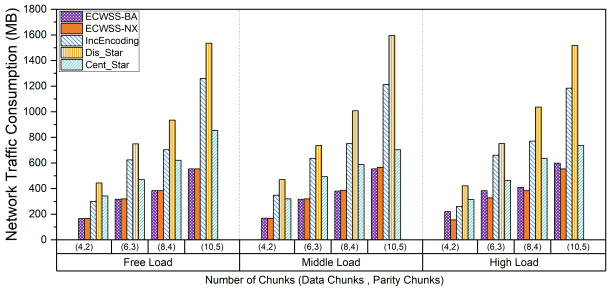


**FIGURE 12.** Influence of different encoding parameters on network flow consumption under different loads.

chunks increases, there is a corresponding increase in the amount of data that needs to be encoded and transmitted, leading to higher network consumption. Notably, in contrast to other schemes, ECWSS-BA and ECWSS-NX exhibit a reduction in writing traffic consumption by 35%-62%. Additionally, except for ECWSS-BA, other schemes exhibit insensitivity to load, with their traffic consumption remaining relatively stable across the three load scenarios due to their inability to sense load changes.

Furthermore, under high load scenarios illustrated in Figure 12, ECWSS-BA exhibits slightly higher network traffic consumption than ECWSS-NX. This can be attributed to ECWSS-BA's preference for avoiding links with fewer hops but higher network loads to improve the system's load balancing performance. Consequently, this leads to a slightly increased traffic consumption during the data writing process.

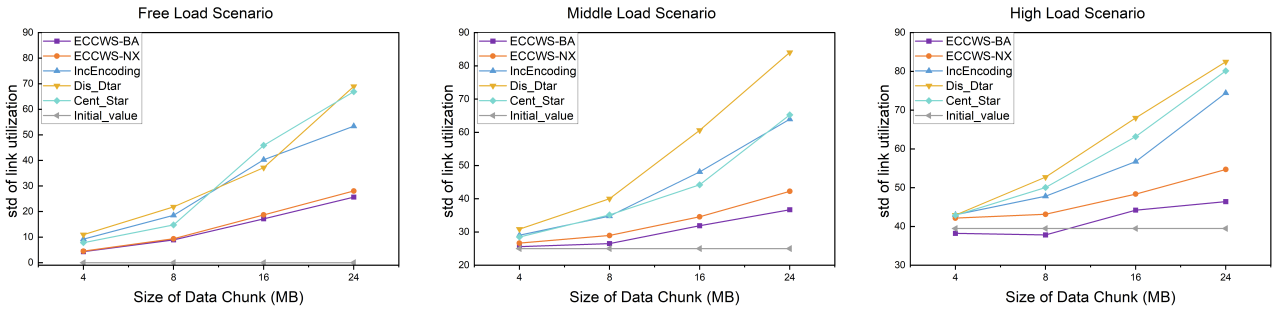


**FIGURE 13.** Influence of different block size on standard deviation of link utilization under different loads.

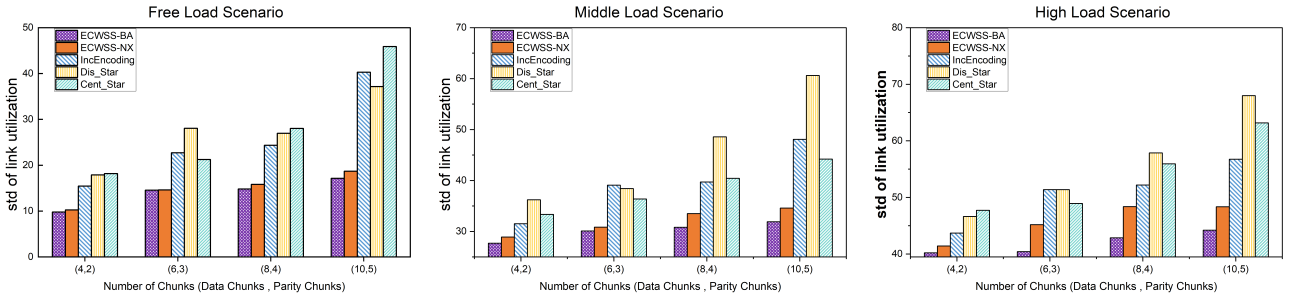


**FIGURE 14.** Influence of different encoding parameters on standard deviation of link utilization under different loads.

#### 4) COMPARISON OF SYSTEM BALANCING PERFORMANCE UNDER DIFFERENT LOAD

In Figure 13, the impact of different data writing methods on the standard deviation of network link utilization is presented under varying load scenarios as the data chunk size increases. Larger data chunks correspond to increased amounts of written data, which in turn leads to higher standard deviation of network link utilization and poorer load balancing performance of the system.

Upon comparing the results presented in Figure 13, we can found that Cent_Star, Dis_Star, and IncEncoding data writing methods have a substantial impact on the on the load balancing performance of the system as the data chunk size increases. In contrast, ECWSS-BA and ECWSS-NX exhibit minimal impact on network load balancing due to their ability to achieve lower traffic consumption and avoid traditional star and pipeline approaches that tend to select congested links and lead to unbalanced network load.

Additionally, as presented in Figure 13(b) and 13(c), ECWSS-BA exhibits superior load balancing performance compared to ECWSS-NX. This can be attributed to the fact that ECWSS-BA takes into account both the topology hop count and network bandwidth as essential factors when constructing the Steiner tree-based topology.

Figure 14 presents the impact of different number of chunks (data chunks, parity chunks) on the standard deviation of network link utilization in different load scenarios. As the total number of chunks increases, there is a corresponding increase in the amount of data that needs to be encoded and transmitted. Therefore, the standard deviation of network link utilization also increases, which means that the load balance of the system becomes worse.

Figure 14 also illustrates that ECWSS-BA and ECWSS-NX solutions, which utilize Steiner tree write topology configuration, exhibit superior load balancing performance compared to traditional star and pipelined approaches that often lead to unbalanced network load due to the selection of congested links. ECWSS constructs the write topology with larger write bandwidth and lower traffic consumption, thereby avoiding congestion. Moreover, similar to the results analyzed in Figure 13, when the data chunks ECWSS-BA also achieves better load balancing performance than ECWSS-NX, when the number of chunks changes.

## VI. CONCLUSION AND FUTURE WORK

In this paper, we discussed the secure fault-tolerant and data writing problems in ESS. The motivation of this work is to provide a secure fault-tolerant storage scheme and improve the data writing efficiency for ESS. Specifically, we proposed a novel and secure cloud-edge collaborative fault-tolerant storage scheme, called HCEFT, which utilizes our designed CEPPC code. It can provide high fault-tolerant capability

while maintaining the edge node data privacy security and high availability. Then, the data writing optimization method for HCEFT was future proposed, which can reduce writing time and minimizes writing traffic. Comprehensive comparative analysis and extensive experimental results show the effectiveness of the HCEFT scheme and the improvement of the data writing optimization method.

In the future, our research will focus on addressing data repair and data update problems in dynamic network environments for ESS. Additionally, researchers investigating general erasure coding data writing problems for ESS can draw on the core ideas presented in this study for optimization, and adapt them to meet the specific requirements of their scenarios. Moreover, with the scale of the edge storage network continuing to expand, it is crucial to study the collaborative management of multiple SDN controllers in ESS, including cooperative communication, control domain partitioning, and other related strategies. These research all hold the potential to enhance the scalability, efficiency, and reliability of the edge storage systems.

## ACKNOWLEDGMENT

The authors would like to thank the associate editor and the anonymous reviewers for their valuable comments.

## REFERENCES

[1] T. Wu, G. Jourjon, K. Thilakarathna, and P. L. Yeoh, "MapChain-D: A distributed blockchain for IIoT data storage and communications," *IEEE Trans. Ind. Informat.*, early access, Jan. 6, 2023, doi: 10.1109/TII.2023.3234631.

[2] S. H. A. Kazmi, F. Qamar, R. Hassan, and K. Nisar, "Routing-based interference mitigation in SDN enabled beyond 5G communication networks: A comprehensive survey," *IEEE Access*, vol. 11, pp. 4023–4041, 2023.

[3] B. W. Nyamtiga, A. A. Hermawan, Y. F. Luckyarno, T. Kim, D. Jung, J. S. Kwak, and J. Yun, "Edge-computing-assisted virtual reality computation offloading: An empirical study," *IEEE Access*, vol. 10, pp. 95892–95907, 2022.

[4] M. Carrie and R. David, *The Growth in Connected IoT Devices is Expected to Generate 79.4ZB of Data in 2025, According to a New IDC Forecast*. Accessed: Apr. 8, 2023. [Online]. Available: https://www.businesswire.com/news/home/20190618005012/en/The-Growth-in-Connected-IoT-Devices-is-Expected-to-Generate-79.4ZB-of-Data-in-2025-According-to-a-New-IDC-Forecast

[5] R. van der Meulen. *What Edge Computing Means for Infrastructure and Operations Leaders*. Accessed: Apr. 8, 2023. [Online]. Available: https://www.gartner.com/smarterwithgartner/what-edge-computing-means-for-infrastructure-and-operations-leaders

[6] L. A. Haibeh, M. C. E. Yagoub, and A. Jarray, "A survey on mobile edge computing infrastructure: Design, resource management, and optimization approaches," *IEEE Access*, vol. 10, pp. 27591–27610, 2022.

[7] H. Zhang, Y. Yang, X. Huang, C. Fang, and P. Zhang, "Ultra-low latency multi-task offloading in mobile edge computing," *IEEE Access*, vol. 9, pp. 32569–32581, 2021.

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

[9] L. Yuan, Q. He, F. Chen, J. Zhang, L. Qi, X. Xu, Y. Xiang, and Y. Yang, "CSEdge: Enabling collaborative edge storage for multi-access edge computing based on blockchain," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 8, pp. 1873–1887, Aug. 2022.

[10] S. Kisseleff, S. Chatzinotas, and B. Ottersten, "Reconfigurable intelligent surfaces in challenging environments: Underwater, underground, industrial and disaster," *IEEE Access*, vol. 9, pp. 150214–150233, 2021.

[11] X. Gao, W. Bao, X. Zhu, G. Wu, and L. Liu, "An edge storage acceleration service for collaborative mobile devices," *IEEE Trans. Services Comput.*, vol. 15, no. 4, pp. 1993–2006, Jul. 2022.

[12] A. Aral and T. Ovatman, "A decentralized replica placement algorithm for edge computing," *IEEE Trans. Netw. Service Manage.*, vol. 15, no. 2, pp. 516–529, Jun. 2018.

[13] M. Linaje, J. Berrocal, and A. Galan-Benitez, "Mist and edge storage: Fair storage distribution in sensor networks," *IEEE Access*, vol. 7, pp. 123860–123876, 2019.

[14] L. Liang, H. He, J. Zhao, C. Liu, Q. Luo, and X. Chu, "An erasure-coded storage system for edge computing," *IEEE Access*, vol. 8, pp. 96271–96283, 2020.

[15] Y. Wu, D. Liu, X. Chen, J. Ren, R. Liu, Y. Tan, and Z. Zhang, "MobileRE: A replicas prioritized hybrid fault tolerance strategy for mobile distributed system," *J. Syst. Archit.*, vol. 118, Sep. 2021, Art. no. 102217.

[16] J. Xia, G. Cheng, S. Gu, and D. Guo, "Secure and trust-oriented edge storage for Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4049–4060, May 2020.

[17] 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.*, vol. 16, no. 2, pp. 1039–1050, Mar. 2023.

[18] A. Datta and F. Oggier, "Concurrency control and consistency over erasure coded data," *IEEE Access*, vol. 10, pp. 118617–118638, 2022.

[19] *Containernet*. Accessed: Apr. 8, 2023. [Online]. Available: https://containernet.github.io/

[20] B. Ali, M. A. Gregory, and S. Li, "Multi-access edge computing architecture, data security and privacy: A review," *IEEE Access*, vol. 9, pp. 18706–18721, 2021.

[21] H. Sun, H. Yu, G. Fan, and L. Chen, "QoS-aware task placement with fault-tolerance in the edge-cloud," *IEEE Access*, vol. 8, pp. 77987–78003, 2020.

[22] M. I. Naas, L. Lemarchand, P. Raipin, and J. Boukhobza, "IoT data replication and consistency management in fog computing," *J. Grid Comput.*, vol. 19, no. 3, pp. 1–25, Sep. 2021.

[23] W. Ke, Y. Wang, M. Ye, and J. Chen, "A priority-based multicast flow scheduling method for a collaborative edge storage datacenter network," *IEEE Access*, vol. 9, pp. 79793–79805, 2021.

[24] P. Liu, Z. Pan, and J. Lei, "Parameter identification of reed-solomon codes based on probability statistics and Galois field Fourier transform," *IEEE Access*, vol. 7, pp. 33619–33630, 2019.

[25] K. Liu, J. Peng, J. Wang, Z. Huang, and J. Pan, "Adaptive and scalable caching with erasure codes in distributed cloud-edge storage systems," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 1840–1853, Apr./Jun. 2022.

[26] K. Taik Kim, C. Joe-Wong, and M. Chiang, "Coded edge computing," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Jul. 2020, pp. 237–246.

[27] 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.

[28] H. Q. Dinh, B. T. Nguyen, A. K. Singh, and W. Yamaka, "MDS constacyclic codes and MDS symbol-pair constacyclic codes," *IEEE Access*, vol. 9, pp. 137970–137990, 2021.

[29] A. Zhang and K. Feng, "A unified approach to construct MDS self-dual codes via reed-solomon codes," *IEEE Trans. Inf. Theory*, vol. 66, no. 6, pp. 3650–3656, Jun. 2020.

[30] H. Bao, Y. Wang, and F. Xu, "A cross-datacenter erasure code writing method based on generator matrix transformation," *J. Comput. Res. Develop.*, vol. 57, no. 2, pp. 291–305, 2020.

[31] B. Calder, "Windows azure storage: A highly available cloud storage service with strong consistency," in *Proc. 23rd ACM Symp. Operating Syst. Princ.*, 2011, pp. 143–157.

[32] C. Lai, S. Jiang, L. Yang, S. Lin, G. Sun, Z. Hou, C. Cui, and J. Cong, "Atlas: Baidu's key-value storage system for cloud data," in *Proc. 31st Symp. Mass Storage Syst. Technol. (MSST)*, May 2015, pp. 1–14.

[33] F. Xu, Y. Wang, and X. Ma, "Incremental encoding for erasure-coded cross-datacenters cloud storage," *Future Gener. Comput. Syst.*, vol. 87, pp. 527–537, Oct. 2018.

[34] M. Ye, H. Qiu, Y. Wang, Z. Zhou, F. Zheng, and T. Ma, "A method of repairing single node failure in the distributed storage system based on the regenerating-code and a hybrid genetic algorithm," *Neurocomputing*, vol. 458, pp. 566–578, Oct. 2021.

[35] H. Zhou, D. Feng, and Y. Hu, "Bandwidth-aware scheduling repair techniques in erasure-coded clusters: Design and analysis," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 3333–3348, Dec. 2022.

[36] H. Tang, G. Liu, X. Chen, and N. Xiong, "A survey on Steiner tree construction and global routing for VLSI design," *IEEE Access*, vol. 8, pp. 68593–68622, 2020.
[37] L. Martins, D. Santos, T. Gomes, and R. Girão-Silva, "Determining the minimum cost Steiner tree for delay constrained problems," *IEEE Access*, vol. 9, pp. 144927–144939, 2021.
[38] Y. Wang, M. Ye, Q. He, Y. Huan, and W. Kang, "A new node selecting approach in Ceph storage system based on software defined network and multi-attributes decision-making model," *Chin. J. Comput.*, vol. 42, no. 2, pp. 93–108, 2019.
[39] W. Ke, Y. Wang, and M. Ye, "GRSA: Service-aware flow scheduling for cloud storage datacenter networks," *China Commun.*, vol. 17, no. 6, pp. 164–179, Jun. 2020.
[40] J. Chen, Y. Yang, C. Wang, H. Zhang, C. Qiu, and X. Wang, "Multitask offloading strategy optimization based on directed acyclic graphs for edge computing," *IEEE Internet Things J.*, vol. 9, no. 12, pp. 9367–9378, Jun. 2022.
[41] M. Peuster, H. Karl, and S. van Rossem, "MeDICINE: Rapid prototyping of production-ready network services in multi-PoP environments," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2016, pp. 148–153.
[42] Y. Li, X. Guo, X. Pang, B. Peng, X. Li, and P. Zhang, "Performance analysis of floodlight and ryu SDN controllers under mininet simulator," in *Proc. IEEE/CIC Int. Conf. Commun. China (ICCC Workshops)*, Aug. 2020, pp. 85–90.
[43] A. A. Hagberg, D. A. Schult, and P. J. Swart. *The NetworkX Documentation*. Accessed: Apr. 8, 2023. [Online]. Available: https://networkx.org/documentation/stable
[44] *New York Metro IBX Data Center Data Sheet*. Accessed: Apr. 8, 2023. [Online]. Available: https://www.equinix.com/resources/data-sheets/nyc-metro-data-sheet
[45] K. Wenlong, W. Yong, Y. Miao, and C. Junqi, "Priority differentiated multicast flow scheduling method in Ceph cloud storage network," *J. Commun.*, vol. 41, no. 11, pp. 40–51, 2020.

**JUNQI CHEN** received the B.Eng. degree in networking engineering from the Changshu Institute of Technology, Suzhou, China, in 2019. He is currently pursuing the Ph.D. degree with the School of Computer Science and Information Security, Guilin University of Electronic Technology, Guilin, China. His current research interests include edge storage systems, erasure coding, and software-defined networks.

**YONG WANG** received the Ph.D. degree from the East China University of Science and Technology, Shanghai, China, in 2005. He is currently a Full Professor and a Ph.D. Supervisor with the Guilin University of Electronic Technology. His current research interests include cloud/edge computing, distributed storage systems, software-defined networks, and information security.

**MIAO YE** received the B.S. degree in theory physics from Beijing Normal University, in 2000, and the Ph.D. degree from the School of Computer Science and Technology, Xidian University, in 2016. He is currently a Full Professor and a Ph.D. Supervisor with the Guilin University of Electronic Technology. His current research interests include software-defined networks, edge computing and edge storage, wireless sensor networks, and deep learning.

**QIUXIANG JIANG** received the master's degree from the Guilin University of Technology, Guilin, China, in 2005. She is currently a Senior Engineer with the Guilin University of Electronic Technology. Her current research interests include software-defined networks, wireless sensor networks, and edge computing.