EasyChair Preprint

№ 15199

Erasure Coding Based Optimization in Decentralized Distributed Storage Systems

Yiwei Gan, Zhijie Huang, Yulong Shi, Xiao Zhang and Nannan Zhao

EasyChair preprints are intended for rapid dissemination of research results and are integrated with the rest of EasyChair.

October 8, 2024

Erasure Coding Based Optimization in Decentralized Distributed Storage Systems

Yiwei Gan*, Zhijie Huang†, Yulong Shi‡, Xiao Zhang§, Nannan Zhao¶
School of Computer Science, Northwestern Polytechnical University, China
*‡{ganyw,yulongshi}@mail.nwpu.edu.cn,
†§¶{jayzy.huang,zhangxiao,nananzhao}@nwpu.edu.cn
†corresponding author

***Abstract*—Node failures in decentralized distributed storage systems are common. To ensure data availability, these systems employ data redundancy mechanisms, typically relying on replicas. This paper proposes a decentralized data redundancy scheme based on erasure coding, with Reed-Solomon and IPFS as examples. Compared with replication, the proposed scheme reduces storage space and enhances fault tolerance. Files are sharded and encoded across multiple nodes, avoiding the high redundancy of replicas. Users can retrieve any K shards from N nodes to reconstruct the original files. This erasure coding optimization combines efficient data exchange among decentralized nodes with erasure coding technology, significantly reducing storage space compared with the replica mechanism. The implementation involves truncating and sharding the blocks within the Merkle DAG generated by files, enabling flexible adjustments to the code rate of erasure codes and the allocation of storage nodes based on user needs and available resources. This method achieves a balance between storage efficiency and data availability.**

***Index Terms*—Decentralized Distributed Storage System, Erasure Coding, Reed-Solomon, IPFS, Merkle DAG**

I. Introduction

With the rapid growth of data, traditional RAID storage system, acting as a single-point storage solution, can no longer meet the demands for massive data storage and fault tolerance. Distributed storage systems, leveraging physical disks and network technology, provide a solution by dispersing data across different nodes in a cluster. This scheme not only enhances data reliability and scalability but also facilitates the decoupling of storage and computation.

Distributed storage systems can be categorized into centralized and decentralized. Centralized distributed storage systems, such as Google File System (GFS) [1] and Hadoop Distributed File System (HDFS) [2], are well-known for their large-scale storage capacity and robust data fault tolerance. However, these systems heavily rely on the metadata management and scheduling capabilities of a master node, which introduces a significant risk of a single point of failure.

Decentralized distributed storage, which does not rely on a central node, addresses the limitations of centralized systems. Prominent decentralized systems such as Sia [3], Storj [4], and IPFS [5] provide enhanced scalability and fault tolerance. IPFS, for example, utilizes content addressing and P2P networking to enable decentralized data storage and transmission. While IPFS aims to create a global, decentralized network for file storage and sharing, it currently lacks support for erasure codes.

This paper focuses on enhancing the fault tolerance and storage efficiency of IPFS by adopting erasure coding instead of replication-based fault tolerance in decentralized distributed storage. This scheme leverage P2P networks, enabling nodes to interact directly without the need for central coordination, which reduces the burden on central nodes and enhances cluster scalability. Furthermore, the equal status of nodes ensures that the system remains operational even in the event of nodes failures.

The main contributions of this paper can be summarized as follows:

1) A decentralized storage optimization scheme that utilizes erasure coding, which has been experimentally validated for reliability.
2) A shard allocation algorithm designed for decentralized storage systems.
3) Implementation of Merkle DAG-based erasure encoding and decoding techniques.
4) A method to recover complete files from a decentralized storage system, even in the event of node failures and some shards being corrupt. Which is based on content-addressing and unique file identification, ensuring the integrity and availability of files in the system.

II. Background and Motivation

A. Fault Tolerance for Distributed Storage Systems

Currently, the primary data fault tolerance mechanisms in distributed storage systems can be divided into two types: replication and erasure coding.

- Replication: replication involves maintaining multiple copies of data across various nodes to ensure data availability. For example, the Google File System (GFS) employs replication by duplicating data at the chunk level. While this approach provides high data availability, it has several drawbacks, including increased storage costs and high write amplification. Furthermore, to maintain consistency across these replicas, a robust synchronization mechanism or consensus algorithm is necessary to ensure the cluster's status remains consistent.
- Erasure Coding: erasure coding is a technique that divides original data into data shards and generates parity shards,

which are then distributed across various nodes. This method allows the original data to be reconstructed from the remaining shards in the event of shards corruption or loss. Compared to replication, erasure coding significantly reduces the need for data synchronization between nodes, which decrease the load on both the storage system and network. This method enhances storage efficiency and enables the flexible adjustment of redundancy to balance storage resources with fault tolerance. However, erasure coding requires computational resources during encoding and decoding. Moreover, even though it's less than replication, erasure coding is also susceptible to write amplification during the encoding phase. For example, HDFS+ [6] employs Reed-Solomon codes for erasure coding, dividing data into shards and distributing them across nodes. This system also implements both online and offline encoding strategies to accommodate various data access patterns. Additionally, Local Reconstruction Codes (LRC) [7] can be used to accelerate data recovery by utilizing additional redundant space.

B. Erasure Coding in Decentralized Distributed Storage

Decentralized storage systems, which leverage global devices for peer-to-peer (P2P) data storage, operate without a central control node. This approach enhances scalability and bolsters fault tolerance. Erasure coding is frequently used in decentralized systems due to the relative instability of nodes. For instance, Storj, a decentralized content storage network, employs Reed-Solomon (RS) codes with a (40,20) configuration, meaning that data is split into 40 shards and 20 parity shards are generated for recovery. Storj connects users to storage nodes via "satellites", which coordinate metadata storage and facilitate data transfer. While erasure coding significantly enhances data reliability and availability, Storj's dependency on satellites and the Ethereum network for conducting transactions and coordination somewhat limits its level of decentralization compared to IPFS.

Regarding erasure coding in IPFS, Shin et al. [8] shown that incorporating erasure coding into IPFS Cluster can reduce storage costs compared to the original replication mechanisms. Their study focused on the evaluation of the storage overhead for availability of erasure coding in 22 nodes IPFS Cluster but did not delve into the specifics of implementation. While Ling et al. [9] proposed a decentralized Alpha Entanglement (AE) codes scheme for IPFS. This approach enables users to achieve a reliability level comparable to replication while consuming less storage and incurring only a minor bandwidth overhead. However, this is merely a proof-of-concept implementation aimed at demonstrating how AE codes contributes to IPFS and has not been tested in a decentralized environment. Our scheme implement a comprehensive erasure coding solution within the IPFS Cluster and conduct extensive tests to evaluate its effectiveness and reliability in a small decentralized environment. Our testing process will cover various scenarios and conditions to simulate real-world usage patterns, allowing for a comparative analysis with the replication fault tolerance mechanism.

C. the Challenges of Decentralized Erasure Coding

Erasure coding in decentralized systems like IPFS presents unique challenges due to the lack of a central authority and the dynamic nature of the network.

1) Decentralized Network: In decentralized systems, all nodes can communicate and exchange data with each other without any hierarchical roles. IPFS allows nodes to join and leave freely, using the Bitswap mechanism to facilitate data exchange. When a IPFS node requires data blocks, it first searches its local storage. If the block is not found, it sends a request to connected peers or the Kademlia DHT [10], as illustrated in Fig. 1. The Bitswap mechanism manages these interactions, and each node maintain multiple connections to enhance data retrieval reliability and speed. For erasure coding in IPFS, the challenge is to allocate shards effectively in this unstable environment, which will be addressed in III-E on the hash-based shard allocation algorithm.

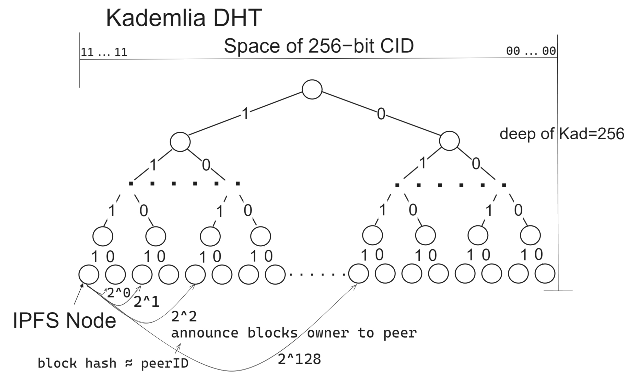

Fig. 1. IPFS Kademlia DHT network

2) Decentralized File Structure: Decentralized storage systems often use Merkle Directed Acyclic Graphs (Merkle DAGs) [11] to organize data. In IPFS, files are segmented into 256 KB blocks, which are then constructed into a Merkle DAG. This structure is pivotal for enabling efficient data verification and retrieval. The Merkle DAG architecture brings several benefits:

- Efficient Data Verification: Files are segmented into 256 KB blocks and verified through their unique CIDs, reducing the need for extensive data transfer and ensuring correctness with minimal overhead.
- Content Addressing: IPFS uses cryptographic hash functions to uniquely identify data blocks, enabling efficient retrieval and minimizing metadata overhead.
- Tamper-Proofing: Each node in the Merkle DAG represents a data block, and changes to the tree alter the CID (Content Identifier), ensuring data integrity and version control. This feature is the basic of Merkle-CRDT [12].
- Deduplication: Identical data blocks share the same CID, allowing the system to automatically identify and avoid

duplicates, which optimizes storage usage across the network.

Integrating erasure coding into IPFS requires maintaining the Merkle DAG structure. Which involves adapting the encoding and decoding processes to preserve the integrity of DAG nodes.

III. Erasure Coding for Decentralized Storage

A. Definitions and Symbols

To clarify, the following basic concepts commonly used in this paper are described:

- **Erasure Coding**: In storage systems, erasure coding is a method used to enhance data availability and ensure fault tolerance beyond simple replication. It involves splitting data into K shards and encoding them to generate $N-K$ parity shards. The total of N shards is then distributed and stored across different nodes. For data recovery, retrieving any K shards is sufficient to decode and reconstruct the complete data, thus achieving fault tolerance.
- **MDS (Maximum Distance Separable)**: This property ensures that raw data can be decoded from any K out of N shards, satisfying the Singleton bound [13] of the encoding method. This achieves theoretically optimal storage utilization by maximizing data recovery efficiency.
- **Stripe**: An instance of erasure coding consisting of N individual shards distributed across different nodes. Stripes are independent of each other.
- **Code Rate**: The ratio of the number of data shards K to the total number of shards N in a stripe. A code rate greater than 0.5 is considered high, otherwise, it is considered low.
- **IPFS (InterPlanetary File System)** [14]: A decentralized distributed storage system and modular file transfer protocol designed for content addressing and P2P networking. It enables decentralized storage and transfer of data, aiming to create a global, decentralized network for storing and sharing files.
- **IPLD (InterPlanetary Linked Data)**: Designed for interoperability across various data formats and storage models, IPLD is the standard implementation in IPFS for creating addressable and linked decentralized data structures. IPLD allows the processing of different data sources using a unified interface, facilitating the creation of complex formats across multiple data sources.
- **Pin**: In IPFS, the term "pin" indicates marking a data block or Merkle DAG to prevent it from being garbage collected. In IPFS Cluster, pinning describes files fixed within the cluster and includes various types such as DataType, MetaType, ClusterDAGType, and ShardType. The processing of ShardType is particularly relevant to sharding and erasure coding implementations.

B. Principle of Erasure Coding

This paper employs Reed Solomon codes (RS codes), which were proposed by Reed and Solomon in 1960 [15] and are now widely utilized in the field of distributed storage systems. By performing polynomial operations over a Galois field, RS codes, which are MDS codes, can adapt to any number of data and redundancy disks. This flexibility in adjusting the code rate makes them highly suitable for decentralized storage [16].

The principle involves multiplying the data shards with a specific generating matrix to generate parity shards, which are then distributed across the decentralized network. In case an error occurs during data transmission or storage, the original data can be retrieved by using the inverse matrix of the generating matrix to decode the data from the remaining $N-K$ parity shards, accommodating the unreliable nature of nodes.

RS encoding and decoding operations in a Galois field support a variety of matrix, with common examples being Vandermonde and Cauchy matrices. These matrices support transversal coding, with polynomial operations performed in the Galois fields $GF(2^w)$ [17].

When using Vandermonde matrix for RS encoding, the vector representation of the data shards is denoted as $\mathbf{M}^T = (m_1, m_2, \ldots, m_k)$. By specifying the parameters (i.e., K data shards and $N-K$ parity shards), the Vandermonde encoding matrix for the Galois field can be transferred as $\mathbf{G}^T = [g_{j,i}]_{(0<i\leq n, 0<j\leq k)}$. The encoded stripe, including data shards and parity shards, is given by:

$$\mathbf{W}^T = \mathbf{GM} = \begin{bmatrix} g_{1,1} & \cdots & g_{1,k} \\ \vdots & \ddots & \vdots \\ g_{n,1} & \cdots & g_{n,k} \end{bmatrix} \begin{bmatrix} m_1 \\ \vdots \\ m_k \end{bmatrix} = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} \tag{1}$$

Since RS codes satisfy the MDS property and the coding matrix is invertible, it is only necessary to acquire any K rows of $\mathbf{W}$ to recover the full amount of data by the inverse operation of the matrix:

$$\mathbf{M} = \mathbf{G}_K^{-1}\mathbf{W}_K^T \tag{2}$$

Where $\mathbf{G}_K$ is the submatrix of $\mathbf{G}$ containing K rows, and $\mathbf{W}_K$ is the corresponding K rows of the encoded data.

C. Erasure Coding in IPFS Cluster

IPFS Cluster, as a cluster management tool for IPFS, does not change the main mechanism of IPFS. Its primary role is IPFS orchestration and pinset management. IPFS Cluster is capable of tracking and recording the status information of each IPFS node as well as IPFS Cluster node. When a user pins a file to the cluster, the IPFS Cluster node can communicate with all IPFS in the cluster via P2P network and stream the data of blocks to specific IPFS nodes. Additionally, each IPFS Cluster node maintains the same pinset via the Raft or CRDT consensus protocol. The pinset includes pins' structure containing the Peer IDs of storage nodes, organization, and file types.

Based on the cluster orchestration and metadata management features provided by IPFS Cluster, we further implement decentralized erasure coding fault tolerance. The default fault

tolerance mechanism of IPFS Cluster is replication, which copies files to all IPFS peers in the cluster. The number of peers where files are stored can be changed by setting `replication-min` and `replication-max`. If a file is added using erasure coding, the `replication-min` and `replication-max` are automatically set to 1. Then each data shard and parity shard will be stored in only one IPFS peer respectively, and only one replica of shards will be maintained in the cluster. The remaining peers only need to store the metadata associated with this pin.

Files are segmented into 256 KB blocks in RAW format before erasure coding. These blocks are then interconnected and structured into a Merkle DAG by InterPlanetary Linked Data (IPLD). Only the leaf nodes of the Merkle DAG are required to store the RAW data, while the non-leaf nodes are Protobuf nodes that store the CIDs of the child nodes. After successfully constructing the Merkle DAG, the data blocks of the Merkle DAG are sent to the DAG Service module in order for further processing. Since the construction adheres the depth-first-search Merkle DAG construction strategy, the last Protobuf node serves as the root of this file.

When the DAG Service receives data blocks, it transmits blocks to a specific IPFS peer over the P2P network. Concurrently, the data blocks are also directly dispatched to the erasure coding module. After obtaining data blocks, the erasure coding module will incorporate the data blocks into shards in accordance with a specific shard size. In cases where a data block does not fully occupy a shard, the remaining space is appropriately padded with zeros to guarantee a complete shard. After accumulating K data shards, the erasure coding module will encode the data shards and form a stripe along with the parity shards. Utilizing the hash-based allocation algorithm, IPFS Cluster will find the suitable IPFS peer for each shard within the stripe and dispatch it.

Upon the completion of the shards transfer, the IPFS Cluster adds this pin to the pinset, within which there are included the CID of the file, ClusterPin, data shards, parity shards, and the Peer IDs of the stored file.

And after that, the consensus layer further maintains the file as well as the pinset state. The process of file sharding and encoding is shown in Fig. 2, and the corresponding pseudocode is presented in Algorithm 1.

Algorithm 1 Sharding and Erasure Encoding

```
1:  fileStream ← open(file)
2:  blocks ← fileStream.Read(256KB) ▷ IPFS blocks size
3:  DAG, CID ← layoutMerkleDAG(blocks)
4:  shardSize ← getShardSize(fSize, K, N)
5:  dataShards ← mergeBlock(blocks, shardSize)
6:  parityShards ← RSEncode(dataShards, K, N)
7:  shards ← dataShards + parityShards
8:  peers ← consensus module
9:  shard2peer ← shardAllocate(file.Meta, shards, peers, K, N)
10: IPFS ← sendShards(shard2peer, shards)
11: shardPins ← createPin(dataShards, shard2peer)
12: pins ← createPin(parityShards, shard2peer)
13: clusterDAGPin ← createPin(pins, shardPins, metadata)
14: pin ← createPin(DAG.Root, CID, clusterDAGPin)
```

D. Shard Size Determination

When adding files to a decentralized distributed file system, shard size is critical to the efficiency of adding files and the performance of RS encoding and decoding. IPFS Cluster can be considered a P2P storage system, as characterized by Lin et al. [18]. The optimal selection of shard size is ideally influenced by several factors: the reliability of the nodes μ, the code rate of the erasure coding K/N, and the desired reliability A of the target files. Consequently, the chosen shard size results in a variable number of file blocks. The relationship between the number of file blocks b and these factors is outlined below:

$$A_b(b) = \sum_{i=b}^{\frac{Nb}{N-K}} \binom{\frac{Nb}{K}}{i} \mu^i (1-\mu)^{\frac{Nb}{N-K}-i} \quad (3)$$

When considering the reliability of erasure coding, $\frac{K}{N-K} \cdot \mu > 1$ indicates a higher reliability compared to replication. While increasing the number of shards can enhance availability, we need to consider that typical behaviour of IPFS cluster peers neither join nor leave the cluster at extreme frequencies. Furthermore, the performance limitations of the IPFS nodes need to be taken into account, particularly their capacity to handle the reading and processing of a single stripe during encoding and decoding tasks.

Given these factors, it is prudent to set a reasonable threshold for shard size in relation to the total stripe size. A practical ceiling for the stripe size might be set below 1 GB to ensure that the processing demands on IPFS nodes remain manageable. Moreover, to leverage the benefits of erasure coding, files should be divided into at least K shards as much as possible. Within this framework, it is beneficial to maximize the shard size as much as possible, balancing the trade-off between reliability and performance to optimize the overall efficiency of the IPFS Cluster.However, it is usually impossible to control the actual total size of the IPFS storage file blocks during the file addition process because additional metadata such as non-leaf nodes of the Merkle DAG, nodes of the folder, etc., need to be maintained when converting the file into a Merkle DAG.

When determining the shard size, we can first calculate the file size, then estimate the size of the metadata, and finally divide the total size according to the number of peers to get the shard size. This method effectively reduces metadata and maximizes computing resources utilization. It minimizes the number of RS encoding operations while leverage the benefits of erasure coding, ensuring that the actual shard size is closely approximate $\frac{N \cdot FileSize}{K}$. The pseudocode is shown in Algorithm 2.

E. Hash-based Shard Allocation Algorithm

After determining the shard size, it's necessary to decide where each shard will be stored. The default storage scheme for IPFS Cluster is replication. Nevertheless, as we use RS codes for different IPFS peers to store distinct data and parity shards for fault tolerance, we need to assign each shard to a

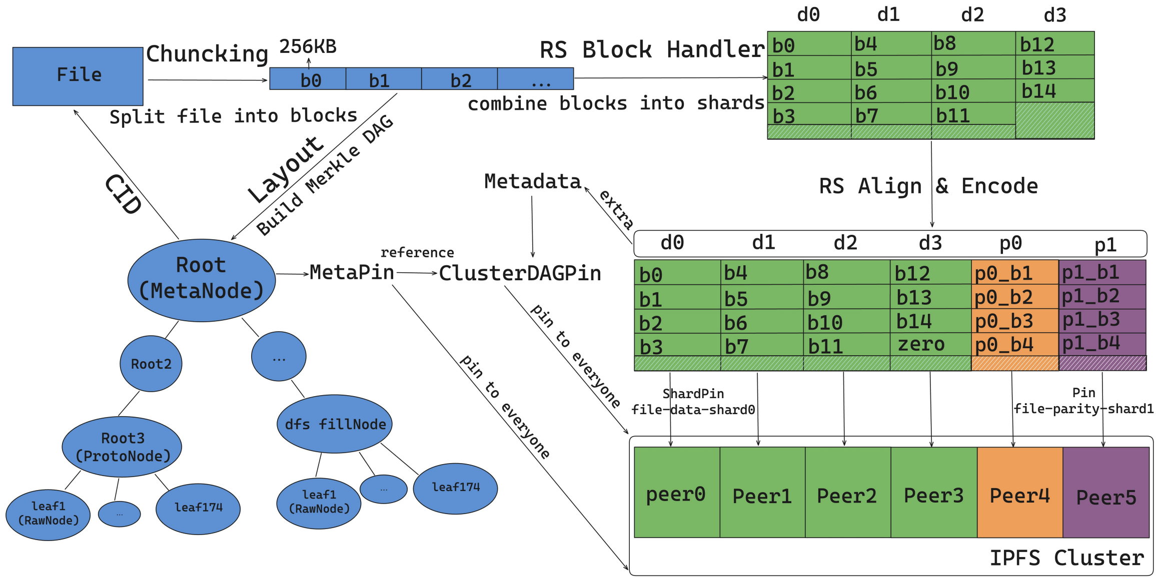

Fig. 2. IPFS erasure encoding process for RS(6,4)

Algorithm 2 Shard Size Determination

1: **procedure** getShardSize(fSize, K, N)
2: BlockSize ← 256 KB ▷ *IPFS blocks size*
3: Threshold ← 1GB ▷ *threshold of stripe*
4: DAGMetaSize ← fSize/5KB ▷ *metadata cost of Merkle DAG*
5: ShardSize ← (fSize+DAGMetaSize)/N
6: **while** ShardSize × N > Threshold **do**
7: ShardSize ← ShardSize/2
8: **end while**
9: ShardSize ← ShardSize + 0.5 × BlockSize
10: **return** ShardSize
11: **end procedure**

specific IPFS peer. The shard allocation algorithm is shown in Fig. 3.

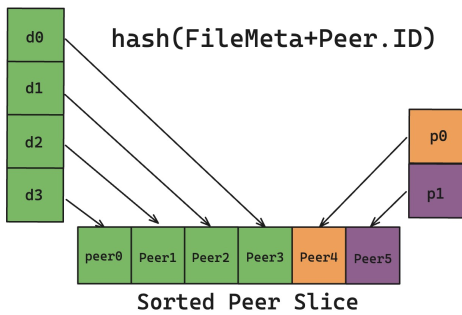

Fig. 3. shard allocation

The goal is to evenly distribute shards among IPFS peers, allowing each peer to store both data and parity shards. Since user access data shards more frequently than parity shards, which can make the IO throughput of the IPFS peers more balanced. The implementation steps are as follows:

1) Get Peer IDs of all IPFS peers within the cluster through the consensus module.
2) Hash the combination of the file metadata and Peer IDs, then sort the hash result as the key to ensure the consistent order of IPFS peers across different shards of the same file. As the hash value changes, the order of IPFS peers assigned to each file may vary, thus distributing the storage load of data and parity shards evenly among the IPFS peers.
3) Allocate the shards to the specified IPFS peers. The first half of the nodes are assigned to store data shards, while the second half is reserved for parity shards. Employing Round-Robin Scheduling, the storage of both type of shards begins with the first IPFS peer in the ordered list. This sharding allocation algorithm ensures optimal fault tolerance when adding files.

Once the destination for each shard is determined, the data blocks corresponding to the shard are then sent from the IPFS Cluster node to different IPFS peers through the P2P network, and the shard is pinned at the corresponding IPFS peer.

What's more, in scenarios where the number of nodes is insufficient to evenly distribute the shards, such as having five nodes available to store 14 shards (comprising 10 data shards and 4 parity shards), our approach is to designate 4 nodes as data nodes and 1 node as parity nodes, then use Round-Robin Scheduling to evenly distribute 10 data shards to 4 data nodes and 4 parity shards to 1 parity node. The corresponding pseudocode is presented in Algorithm 3.

F. Data Recovery in IPFS Cluster

In decentralized distributed storage, the instability of nodes may lead to data loss, especially when replication fault tolerance is utilized, if the last replica goes down, the corresponding data may be lost permanently. Employing fault tolerance with erasure coding, where each node stores merely one shard of the file, reduces the storage pressure and enables the

Algorithm 3 Hash-based Shard Allocation

```
1: procedure shardAllocate(file.Meta,shards,peers,K,N)
2:     hashFunc ← sha256
3:     peersWithMeta ← merge(peers,hashFunc(file.Meta))
4:     peers ← sort(peers,orderFunc(peersWithMeta))
5:     dNum ← ⌈K × len(peers)/N⌉
6:     if dNum = len(peers) then
7:         dNum ← dNum − 1
8:     end if
9:     dPeers ← peers[:dNum]
10:    pPeers ← peers[dNum:]
11:    shard2peer ← new map[Shard][Peer]
12:    for shard,i in shards do
13:        if shard is dataShard then
14:            shard2peer[shard] ← dPeers[i%len(dPeers)]
15:        else
16:            i ← i-K
17:            shard2peer[shard] ← pPeers[i%len(pPeers)]
18:        end if
19:    end for
20:    return shard2peer
21: end procedure
```

Algorithm 4 Data Recovery

```
1: metaPin ← CID
2: clusterDAGPin ← metaPin.Reference
3: shard2peer ← extract(clusterDAGPin.Metadata)
4: peers ← consensus module
5: piple ← make Shard piple          ▷ used by shards
   transmission from Retrieve to Decode
6: createThread(Retrieve(piple,shard2peer,peers))
7: createThread(Decode(piple,K,N))
8: procedure Retrieve(piple,shard2peer,peers)
9:     shards ← new []Shard
10:    for each shard,peer in shard2peer do
11:        shard ← getShard(shard,peer)    ▷ fetch shard
   from IPFS peer concurrently
12:        piple ← shard
13:        shards ← shard
14:        if len(shards) == K then
15:            return shards
16:        end if
17:    end for
18: end procedure
19: procedure Decode(piple,K,N)
20:    shards ← new []Shard
21:    while len(shards) < K do
22:        shards<-piple
23:    end while
24:    shards ← RSDecode(shards,K,N)
25:    IPFS ← repin(shards,shard2peer,peers)
26:    return shards
27: end procedure
```

storage of more data. Additionally, we can introduce cronjob to periodically check whether a certain shard exists in each node. When the file is at the risk of being lost, it will be recovered promptly and stored in a shard on a new node. When some nodes go down, the process of file recovery is as follows:

1) Get file metadata by CID. Obtain the MetaPin associated with the CID to acquire ClusterDAGPin for shard retrieval.
2) Retrieve shards rely on ClusterDAGPin. We can obtain the CIDs of the data blocks within the shard by reading the metadata of ClusterDAGPin, then use the Bitswap protocol to download the block contents and reassemble the shards. While parity shard can be regarded as an individual Merkle DAG, which can be retrieved directly through the DAG Reader.
3) Parallel retrieve mechanisms. Let's focus on the scenario of retrieving one stripe. The primary process fetches all shards of a stripe from the IPFS network, while a secondary process simultaneously begins Reed-Solomon decoding with the first k shards to reconstruct any missing data. Both processes operate in parallel, and either can be halted once successful retrieval is achieved.
4) Repin the recovered shards to ensure their availability on the IPFS network. For data shards, divide them into blocks according to the metadata, stream these blocks, and pin the ShardPin. For parity shards, pin them directly as files using the original parameters. This method circumvents the need to repin the entire file, streamlining the recovery process.

The pseudocode for the primary part of the data recovery process is presented in Algorithm 4.

IV. Experiments and Analyses

The previous section describes how to implement RS codes in IPFS Cluster, this section will conduct performance tests to prove its reliability. The system is deployed on 37 Ali Cloud servers spanning several regions in China. Each node is configured with a dual-core CPU, 4 GB of RAM, and 20 GB of SSD storage, with a public broadband limit of 100 Mbps.

A. *Storage Efficiency*

In the experiments, we employ both erasure coding and replication to add files of different sizes and compare their performance with the same fault tolerance level. The experiments use RS(14,10), which split files into 10 data shards and generate 4 parity shards, permitting any 4 nodes to fail. Hence, 5-Replica fault tolerance is chosen as a compare experiment. The experiment shows the variation of the time required to add files using different fault tolerance methods by adding 21 files of different sizes multiple times, recording the time required for each addition and calculating the average.

As shown in Fig. 4, when the file size is less than 16 MB, the time to add a file using erasure coding is slightly longer than replication. However, when the file size is greater than or equal to 16 MB, the time taken to add a file using erasure coding is gradually less than that required for 5-Replica replication. Especially for large files like 1 GB, the time to add a file using erasure coding is even less than half of that using replication. This is because 5-Replica addition requires replicating the file to five nodes, resulting in more storage and transmission overhead. Whereas erasure coding merely distributes data and parity shards to each node. When the file is small, the advantage of erasure coding is not obvious because it requires encoding and distribution of shards, but as the file size increases, the time required for encoding and distributing shards in erasure coding increases at a much slower rate compared to the 5-Replica, and thus the fault-tolerant scheme of erasure coding demonstrates a higher degree of efficiency.

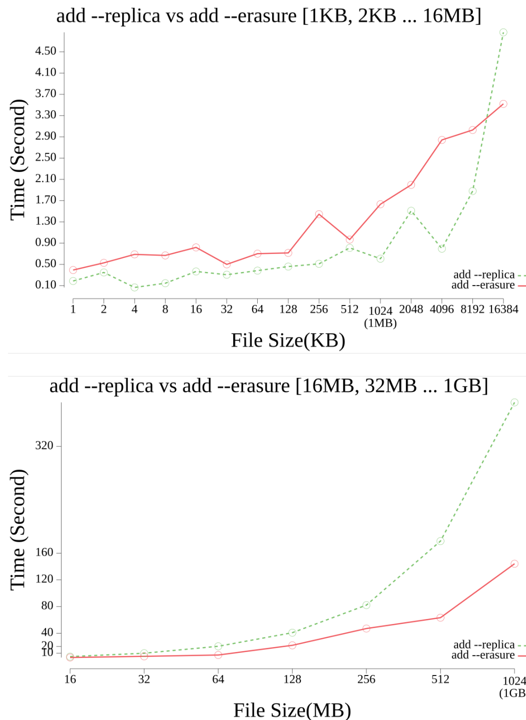

Fig. 4. response time of adding files

Although there are some fluctuations in the time needed to add files in both ways, through multiple tests, for small files (less than 16 MB), replication is slightly faster than erasure code. For larger files (greater than 16 MB), erasure coding gradually outperforms replication in terms of efficiency of addition. Especially for large files (greater than 256 MB), the addition time of erasure coding is significantly reduced and demonstrates higher efficiency. This indicates that file size is an important factor when choosing a storage strategy. For larger files, the use of erasure coding can significantly reduce storage and transmission overheads. Based on the test results, we can observe that the time required to add a file using erasure coding is always less than replication when the file is large. The average throughput from these tests is summarized in TABLE I.

TABLE I
Average Throughput of addition

Method	Average Throughput
RS(14,10)	7.57 MB/s
5-Replica	2.71 MB/s

The table shows that the average throughput of adding a file using RS(14,10) is always greater than that of the 5-Replica method. Additionally, we calculate the total size of the data blocks actually pinned in IPFS and divide the portion larger than the original file size by the original file size to obtain the write amplification factor. We measured the write amplification factor for files of varying sizes, added using erasure coding, and compared it with the 5-Replica method. The results are depicted in Fig. 5.

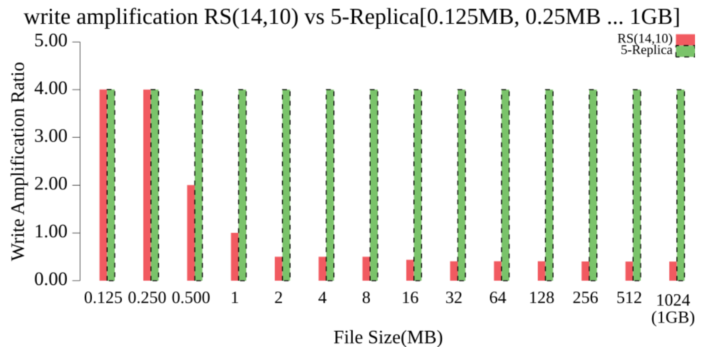

Fig. 5. write amplification

When the file size is less than 0.5 MB, the write amplification factor for two methods of adding files is 4. In other words, when the file is small, the actual content written to disk when adding the file is nearly five times the size of the file. This occurs because the default block size of IPFS is 256 KB, RS encoding generates 4 parity shards based on the data shards of one block. As a result, the write amplification factor is 4 when the file size is less than or equal to 256 KB. When the file size is larger than 256 KB, the shard size grows larger, the number of blocks that constituting the Merkle DAG increases, and the write amplification factor decreases, eventually approaching the desired write amplification factor of 0.4 for RS(14,10), which is in line with the evaluation of [8]. On the other hand, the write amplification factor of replication is basically kept to 4, as it requires replicating all the blocks of the Merkle DAGs to 5 different IPFS peers.

B. Recovery Efficiency

In this section, the primary objective is to evaluate the data recovery efficacy following the loss of 4 data storage nodes. In this experiment, we maintain the same parameters as before but randomly shut down 4 nodes that stored the file, then recover this file and repin the shards to the IPFS Cluster. Specifically, we measured the time taken to retrieve the shards from the remaining nodes, decode them back into the original data, and repin the recovered shards to the IPFS Cluster. The results of this process are illustrated in Figure 6, and we also calculated the average throughput for recovery, summarized in TABLE II.

It can be inferred that when dealing with small files, the retrieval time may fluctuate. This is due to the fact that during shard allocation, if the storage node of a shard coincides with the retrieval node, or if the storage node and retrieval node are located in the same LAN, it will speed up the retrieval of the shard, and the logic for repinning the shard remains the same.

It's important to note that the main bottleneck in file recovery lies in the processes of retrieving a sufficient number of shards and repinning the file shards, while the throughput of RS decoding is much larger than that of retrieving and repinning the shards.

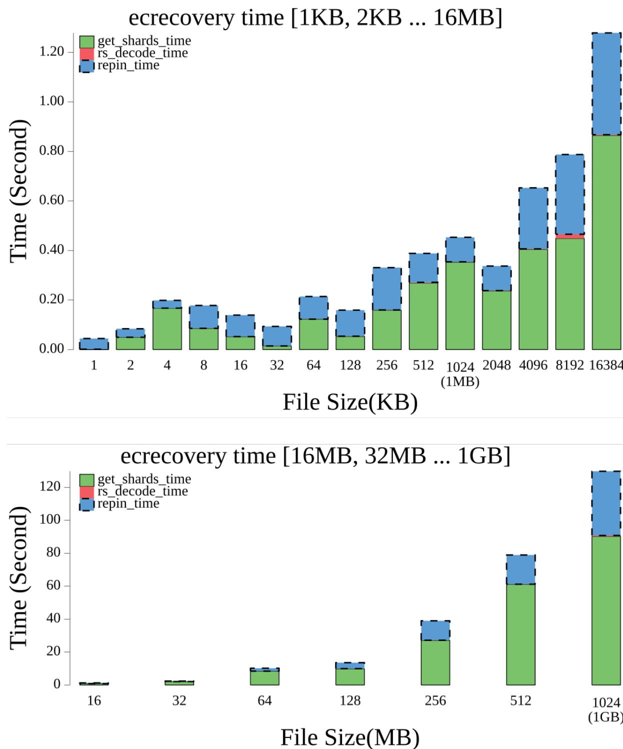

Fig. 6. recovery time

TABLE II
Average Throughput of recovery

Method	Average Throughput
ecrecovery all step	10.33 MB/s
ecrecovery retrieve shards	10.16 MB/s
RS decode	2254.40 MB/s
ecrecovery repin	10.79 MB/s

V. Conclusion

We proposed an erasure coding based optimization scheme for decentralized storage systems, which dramatically improves the storage efficiency and reduces the write amplification in these systems. Furthermore, the average throughput of adding files using erasure coding is 7.57 MB/s with a broadband limitation of 100Mbps, which is almost triple that of the 5-Replica replication. When it comes to recovery efficiency, the average throughput is 10.33 MB/s, and the bottleneck mainly resides in the network transmission of retrieving and repinning the shards.

Our future work encompasses improving the erasure encoding and decoding of Merkle DAGs by modifying their structure to accommodate the calculation of erasure coding and exploring the most appropriate erasure codes to facilitate highly efficient encoding and decoding. Additionally, we will also try to explore diverse code rates for erasure codes and design better shard allocation algorithms to make it suitable for dynamic scenarios of decentralized distributed systems.

Acknowledgment

This work is supported in part by the National Natural Science Foundation of China (No. 62272394), the National Key Research and Development Program (No.2022YFB2702101), and the Fundamental Research Funds for the Central Universities.

References

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*. Bolton Landing NY USA: ACM, Oct. 2003, pp. 29–43. [Online]. Available: https://dl.acm.org/doi/10.1145/945445.945450

[2] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. Incline Village, NV, USA: IEEE, May 2010, pp. 1–10. [Online]. Available: http://ieeexplore.ieee.org/document/5496972/

[3] T. S. Foundation, "GitHub - SiaFoundation/siad: The Sia daemon." [Online]. Available: https://github.com/SiaFoundation/siad

[4] Storj Labs, "GitHub - storj/storj: Ongoing Storj v3 development. Decentralized cloud object storage that is affordable, easy to use, private, and secure." [Online]. Available: https://github.com/storj/storj

[5] J. Benet, "IPFS - Content Addressed, Versioned, P2P File System," *arXiv preprint arXiv:1407.3561*, 2014.

[6] F. RomanusIshengoma, "HDFS+: Erasure Coding Based Hadoop Distributed File System," *International Journal of Scientific & Technology Research*, vol. 2, no. 9, 2013.

[7] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "XORing Elephants: Novel Erasure Codes for Big Data," Jan. 2013, arXiv:1301.3791 [cs, math]. [Online]. Available: http://arxiv.org/abs/1301.3791

[8] H. Shin, M. Lee, and S. Kim, "Space and cost-efficient reed-solomon code based distributed storage mechanism for ipfs*," in *2023 14th International Conference on Information and Communication Technology Convergence (ICTC)*, 2023, pp. 1165–1169.

[9] Q. Liang and Y. Yang, "Making the interplanetary file system (ipfs) more reliable," in *EPFL*, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:259305472

[10] I. Baumgart and S. Mies, "S/Kademlia: A practicable approach towards secure key-based routing," in *2007 International Conference on Parallel and Distributed Systems*. Hsinchu, Taiwan: IEEE, 2007, pp. 1–8. [Online]. Available: http://ieeexplore.ieee.org/document/4447808/

[11] M. Szydlo, "Merkle tree traversal in log space and time," in *Advances in Cryptology - EUROCRYPT 2004*, C. Cachin and J. L. Camenisch, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 541–554.

[12] H. Sanjuan, S. Poyhtari, P. Teixeira, and I. Psaras, "Merkle-CRDTs: Merkle-DAGs meet CRDTs," Apr. 2020, arXiv:2004.00107 [cs]. [Online]. Available: http://arxiv.org/abs/2004.00107

[13] F. J. MacWilliams and N. J. A. Sloane, *The theory of error-correcting codes*. Elsevier, 1977, vol. 16.

[14] D. Trautwein, A. Raman, G. Tyson, I. Castro, W. Scott, M. Schubotz, B. Gipp, and Y. Psaras, "Design and evaluation of IPFS: a storage layer for the decentralized web," in *Proceedings of the ACM SIGCOMM 2022 Conference*. Amsterdam Netherlands: ACM, Aug. 2022, pp. 739–752. [Online]. Available: https://dl.acm.org/doi/10.1145/3544216.3544232

[15] I. S. Reed and G. Solomon, "Polynomial Codes Over Certain Finite Fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, Jun. 1960. [Online]. Available: http://epubs.siam.org/doi/10.1137/0108018

[16] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Decentralized Erasure Codes for Distributed Networked Storage," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2809–2816, Jun. 2006, arXiv:cs/0606049. [Online]. Available: http://arxiv.org/abs/cs/0606049

[17] L. Xianghong and S. Jiwu, "Summary of research for erasure code in storage system," *Journal of Computer Research and Development*, vol. 49, no. 1, pp. 1–11, 2012. [Online]. Available: https://crad.ict.ac.cn/cn/article/id/2654

[18] W. Lin, D. Chiu, and Y. Lee, "Erasure code replication revisited," in *Proceedings. Fourth International Conference on Peer-to-Peer Computing, 2004. Proceedings.* Zurich, Switzerland: IEEE, 2004, pp. 90–97. [Online]. Available: http://ieeexplore.ieee.org/document/1334935/