Space and Cost-Efficient Reed-Solomon Code based Distributed Storage Mechanism for IPFS*

Heekyung Shin*†, Myungcheol Lee*, Seongmin Kim†
Electronics and Telecommunications Research Institute(ETRI), Daejeon, Korea*,
Sungshin Women's University, Seoul, Korea†,
Email: *{mclee}@etri.re.kr, †{220224009, sm.kim}@sungshin.ac.kr

***Abstract*—The peer-to-peer (P2P) based distributed file system known as IPFS (InterPlanetary File System) has recently garnered significant attention as a decentralized storage solution in the Web3 landscape. However, IPFS operates on a voluntary participation basis, which can limit data availability. To enhance automated data availability on IPFS, an extended solution known as IPFS Cluster is being developed. This solution leverages a data replication approach. This paper proposes a Reed-Solomon based distributed content storage mechanism for IPFS, enhancing data availability and significantly improving storage efficiency compared to traditional replication methods. This technology is anticipated to serve as a fundamental building block for efficiently storing, managing, and utilizing large-scale, high-capacity digital content in various Web3 environments.**

***Index Terms*—Decentralized Storage, Web3, P2P, IPFS, Reed-Solomon Erasure Code**

I. Introduction

The emergence of large-scale data platforms supporting the easy creation, management, and distribution of content in the Web 2.0 era led to accelerated data generation as users directly participated in content creation. It leads to a rapid increase in the generation of massive and unstructured data [1] coupled with next-generation information and communication technologies (ICTs), such as artificial intelligence and the Internet of Things (IoT). Typically, centralized enterprise systems, such as cloud storage [2], would have been a rational choice to store and manage the data in practice. However, this introduces potential risks raised by the delegation and ownership transfer, as the companies control and monopolize user data [3]. Such concerns prompted the need for a Web3 environment to ensure user data privacy and ownership [4].

To address the challenge of managing large amounts of generated data while ensuring privacy, utilizing decentralized storage in the Web3 context to place the ownership on the client side has become essential. A leading technology in shaping Web3's file storage system is the InterPlanetary File System (IPFS) [5], a peer-to-peer-based distributed file storage system. IPFS provides a distributed data store that ensures data integrity while allowing efficient access [6] by utilizing Merkle Directed Acyclic Graph (DAG) for content integrity and Distributed Hash Table (DHT) for content discovery. IPFS also addresses a single point of failure as it operates in a decentralized manner, unlike traditional HTTP protocols reliant on central servers.

* Corresponding author: Myungcheol Lee. Heekyung Shin conducted this research as a student researcher at ETRI.

However, IPFS suffers from data availability because it relies on voluntary participation by design. Due to the decentralized nature of IPFS, nodes in the IPFS network have limited storage space, so there is no obligation to store data permanently. Such a design could result in accessibility problems if nodes with specific data fail or intentionally disconnect from the network. Moreover, malicious users could erase data from all nodes leading to permanent data loss, since anyone with the content's identifier (CID) can delete data [7]. Therefore, IPFS requires complementary technologies to ensure data persistence [8]. In response, Filecoin [9] emerged as an incentive layer for IPFS, ensuring data permanence by rewarding Blockchain-based cryptocurrency. Users pay for utilizing Filecoin storage, and storage providers receive rewards for securely storing users' data during requested periods. But still, institutions dealing with critical and privacy-sensitive data, such as corporations and public organizations, worry about the data openness and hesitate to adopt it as Filecoin operates on the public IPFS network.

A recently proposed IPFS Cluster [10], serving as a clustering tool for IPFS, has garnered attention as an alternative for cryptocurrency-based schemes. Rather than relying on a public network that reveals data information, it guarantees data availability and reliability by leveraging replication and pinning mechanisms within the private IPFS network to coordinate data in a privacy-preserving manner. Nevertheless, the replica-based approach that duplicates the entire original data might lead to poor storage efficiency. In fact, NFT.storage, the largest IPFS Cluster node in operation, suffers from the inflated storage space. Because each pin is replicated three times by default, employing a replication mechanism results in a total storage consumption of 855 TiB, where NFT.storage utilizes 285 TiB and fixes 80 million pins in 2022 [10]. However, existing research on IPFS Cluster has predominantly focused on enhancing data availability, neglecting studies addressing storage efficiency [11].

To this end, this paper explores the practical implications of leveraging Reed-Solomon erasure code with IPFS Cluster to improve storage efficiency. We propose a novel IPFS Cluster architecture by extending the chunk and shard components of IPFS to use Reed-Solomon encoding and build a storage interface between IPFS daemon and Cluster to cope with file management, including store API and retrieve API. To systemically integrate into IPFS and IPFS Cluster, we develop the encoding and decoding methodology compatible with the

existing pinning mechanism and DAG service. Our simulation result shows that storage overhead reduction of up to 6x compared to existing IPFS Cluster architectures.

II. Background

A. IPFS

IPFS, proposed by Protocol Labs, is a decentralized file system that operates on a peer-to-peer (P2P) network rather than relying on centralized servers to store and distribute files [6]. IPFS utilizes a distributed hash table, allowing all participating nodes within the network to perform the task of mapping files to their corresponding node addresses. As a result, this approach enables efficient file retrieval and storage. Consequently, IPFS exhibits the capability to conduct content searches within $O(logN)$, even without knowledge of the precise file location [12].

When adding files to IPFS, it efficiently divides them into suitable-sized chunks and stores them in the file system. Each chunk is assigned a CID based on the content, which is used to identify the file. Identical content results in the same CID, preventing redundant storage of files and reducing space wastage. These chunks are structured and stored using a Merkle DAG. The DAG maintains the relationships and order of consecutive chunks through links. The lowest leaf nodes of the Merkle DAG store CIDs and each higher node up to the root node consists of the hashed values of their child nodes. Consequently, any alteration in a single node's hash value has to affect its parent node's hash values. Thus, IPFS ensures data integrity and verifies unintended modification to data.

B. Reed-Solomon Erasure Code

The Reed-Solomon erasure code is a method of original data and its encoded parity to enable data recovery in the case of corruption or loss [13]. Parity fragments are generated and stored as linear combinations of original data fragments. When data recovery is necessary, the damaged data fragments are identified, and by combining the remaining data fragments and parity fragments, the original data can be recovered.

The notation for the Reed-Solomon erasure code is typically denoted as $RS(k, m)$, where k represents the number of original data fragments and m represents the number of parity fragments, respectively. The encoding process of Reed-Solomon codes involves a simple linear algebra operation [14]. The $RS(k, m)$ encoding is achieved by multiplying $k * 1$ data words with a matrix composed of Generator matrices in Vandermonde form, and this multiplication is performed against a parity codeword of dimensions $m * 1$ [13], [14]:

$$Gk = m \tag{1}$$

where G is the transpose of Generator Matrix. Note that Reed-Solomon erasure codes can be effectively utilized as a key technique to optimize storage systems to optimize storage space efficiency while preventing data loss and enabling recovery [15], [16].

III. Approach

A. Requirements for data availability in an IPFS Cluster

IPFS Cluster is a distributed application for IPFS nodes that extends the functionalities of IPFS to enhance data availability and management. The approach through which IPFS Cluster provides data management functionalities for IPFS is as follows [8]. IPFS Cluster takes on the role of replicating and pinning the original data across multiple IPFS nodes, effectively managing the data storage within IPFS itself. Within the Cluster, information about the IPFS nodes where data is stored is composed into a pinset. This pinset is continuously monitored and tracked, allowing the Cluster to maintain the data. The replication and pinning mechanism of the IPFS Cluster ensures data availability and stability even in the event of a failure or offline status of a single node in the Cluster. If a node within the Cluster fails or is down, the data can still be stored and accessed from other replicated nodes, ensuring data availability and reliability.

IPFS Cluster ensures that users can access their stored data from anywhere at any time by synchronizing the node allocation information (pinset) for the data among Cluster nodes through consensus algorithms. When nodes are added or removed, the changes are shared with all nodes, maintaining the consistency of data allocation information. This ensures that the reliability of data storage and access within IPFS Cluster, as a distributed file system, is upheld.

B. Reed-Solomon Code based IPFS Cluster

By utilizing IPFS Cluster to replicate and store data across multiple IPFS nodes, the likelihood of ensuring data availability increases. However, the efficiency of storage space diminishes as the same data is duplicated across numerous nodes. In essence, a trade-off between data availability and storage space efficiency arises from a system perspective. In this paper, we propose a solution to address this challenge by applying the Reed-Solomon encoding technique to the IPFS Cluster Chunk and Shard mechanisms.

Architecture overview. The architecture resulting from the application of the proposed Reed-Solomon encoding technique to IPFS and IPFS Cluster is composed of six main components as depicted in Figure 1. The proposed mechanism extends the state-of-the-art IPFS Cluster design, and the core functionalities of each component are as follows.

- **RSChunker**: Chunking Reed-Solomon encoded shards for the original data.
- **RSDagBuilder**: Building a ClusterDAG from the sharded chunks, which are later pinned to allocated IPFS daemons.
- **RSAllocator**: Be responsible for identifying and reserving IPFS nodes capable of storing the sharded chunks.
- **RSTracker**: Monitoring the health of IPFS nodes and tracks whether there are sufficient Shard copies for recovery in case of node failures.
- **RSDecoder**: Gathering the shard and decoding the original file, when a user requests to retrieve a file and a node is in failure.

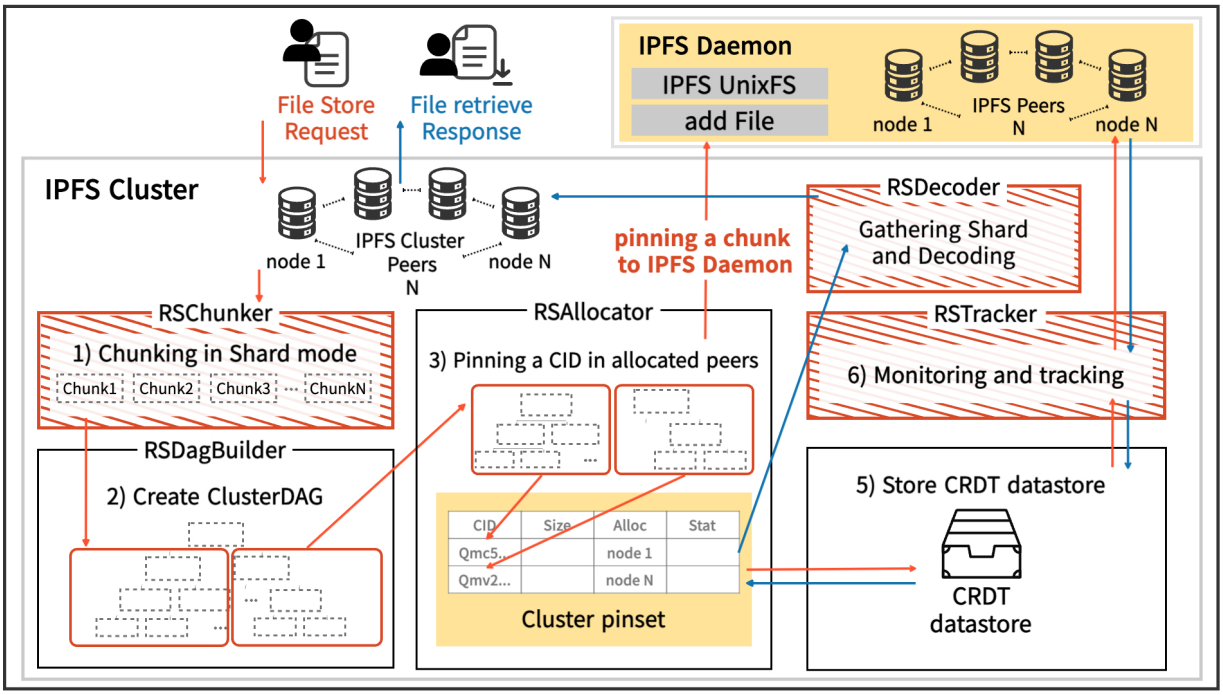

Fig. 1: IPFS/IPFS Cluster internal flow using Reed-Solomon component (The red dotted box indicates modified or newly added components from the original IPFS/IPFS Cluster. IPFS daemon and IPFS Cluster operate as a pair within a single host, and Cluster nodes communicate with the IPFS daemon through the HTTP API.)

- **CRDT Datastore**: A Datastore for synchronizing pinset information across multiple IPFS Cluster nodes.

We note that RSChunker, RSTracker, and RSDecoder are modified or newly added components from the existing IPFS Cluster, while other components remain unchanged.

Extending IPFS Cluster to leverage Reed Solomon. To utilize Reed-Solomon erasure code, we modify the chunking process of the IPFS Cluster and add a decoding module. First, the proposed architecture extends the chunker module with the shard mode of the IPFS Cluster to incorporate the Reed-Solomon encoding technique, which we call RSChunker. In the original IPFS Cluster, the Chunker function divides a file into smaller segments when it is added to IPFS. Each segment is hashed, and a CID is generated for each chunk. Additionally, to accommodate large files, a Shard mode allows the original file to be split and stored. The Larger shards can be further divided into smaller chunks for better filesystem efficiency, with each chunk stored as multiple blocks in UnixFS format. The RSChunker chunks the data into k RSShards and m RSParityShards, as opposed to the chunking mechanism of the original IPFS Cluster, which divided shards and m replicas for those shards. In summary, the differences lie in two aspects: 1) how the chunks are divided and 2) how the architecture provides data availability.

Then, IPFS Cluster generates ClusterDAG based on the divided shard. To illustrate the distinctions between the original IPFS Cluster and the proposed architecture, we depict a ClusterDAG drawn with RSDagBuilder (Figure 2) and that of the original IPFS Cluster created with the DAG service (Figure 3). In both figures, there are common elements: both ClusterDAGs feature a MetaNode and a ClusterDAG Root Node. The MetaNode contains the root CID of the ClusterDAG along with metadata about its child nodes, and it is stored across all IPFS Cluster nodes. The ClusterDAG Root Node holds information about all shards connecting them. All IPFS nodes that have a divided Shard also store the Root Node. In our proposed architecture, there is a notable distinction due to the inclusion of encoded RSShards. The RSMetaNode now includes metadata pertaining to the Reed-Solomon encoding. When users request data retrieval, the RSShards can be recovered and decrypted using the Root Node information, facilitating the retrieval of the original data. At the end of the chunking process, the way of distributing the shards to IPFS daemons is also different. In the original IPFS Cluster, shards are distributed across multiple IPFS daemon, as depicted in Figure 3. In contrast, the proposed architecture stores the original and parity shards individually on an IPFS node, as shown in Figure 2. For this, we design RSTracker by extending the existing pin tracker of the IPFS cluster. As previously mentioned, RSShards differ in how they are stored individually on an IPFS node. As a result, RSTracker's responsibility is to monitor the status of shards stored on these IPFS nodes. In contrast, in the original IPFS Cluster, a tracker should maintain awareness of the status of all shards and replicas distributed across IPFS Daemons. This extensive oversight is necessary due to concerns about the potential failure of any single piece

of data.

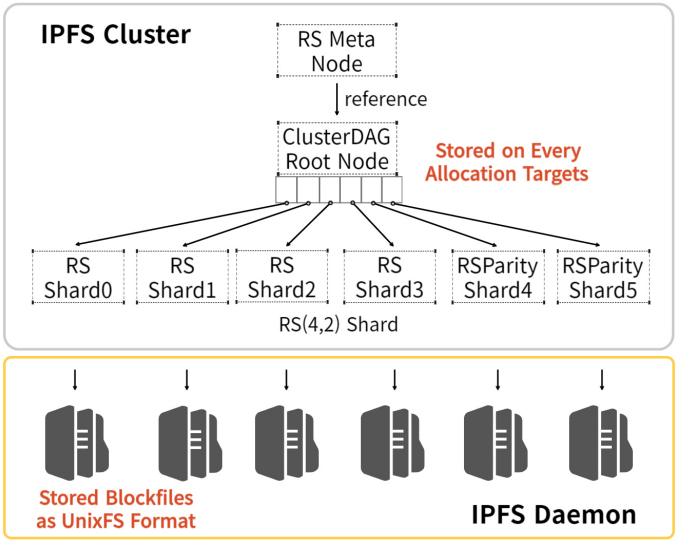

Fig. 2: An example ClusterDAG from RS(4, 2) encoding

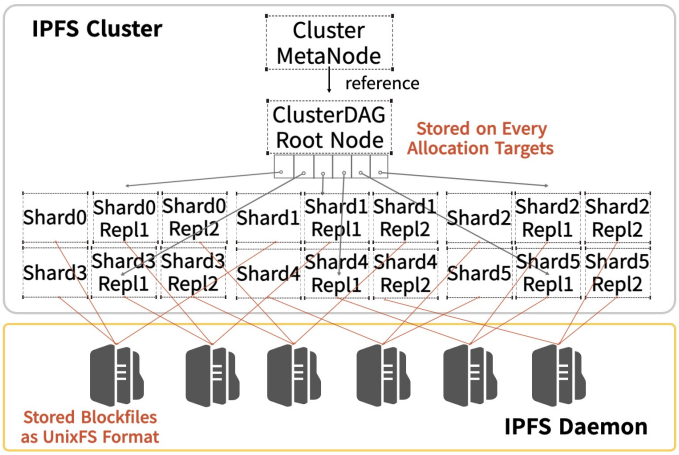

Fig. 3: An example ClusterDAG from original 3 replica method

Finally, RSDecoder is newly added compared to the original IPFS Cluster architecture. In the conventional setup, the shards are basically not encoded, eliminating the need for decryption. Within the proposed architecture, the presence of encoded RSShards introduces a decryption requirement.

Architecture workflow. Figure 1 illustrates how these components work together to create an efficient system that integrates Reed-Solomon encoding into IPFS and IPFS Cluster. When a user requests to store a file, an IPFS Cluster node forwards the user's data storage request to the RSChunker. The RSChunker divides the original file into k shards and generates m parity shards. Each divided shard is transformed into an RSShard node through the RSDagBuilder, creating a ClusterDAG as illustrated in Figure 2.

RSAllocator pins the divided shards to multiple known IPFS nodes. It then constructs a Cluster pinset containing the CID hash values of shards and information about the storing IPFS nodes. This is stored in the CRDT datastore, shared among all Cluster nodes to manage IPFS node allocation information. Lastly, RSTracker periodically monitors and tracks the status of IPFS nodes while data is stored. In the event of node failure, RSDecoder recovers the entire data including the shard, which was stored on the failed node, by collecting and decrypting the shards from other nodes. Once the original data is successfully recovered, the IPFS Cluster starts the entire process of re-chunking and re-allocating shards, starting with RSChunker, except for the failed IPFS node.

When a user requests to retrieve a stored file, a similar process follows. The request is forwarded to the IPFS Cluster nodes. The IPFS Cluster nodes identify the IPFS nodes' information stored in the CRDT datastore. It then gathers the shards from the IPFS daemon, goes through the decryption process with RSDecoder, and delivers the original file to the user.

IV. Evaluation

To evaluate the storage efficiency of our architecture, we conduct a preliminary simulation of storage usage for the prototype Reed-Solomon code based IPFS Cluster and the original one. We are considering a scenario where 22 nodes within the IPFS network can be organized into shards, and aim to assess the impact on space efficiency by varying the combinations of shard numbers. It is worth noting that the largest IPFS Cluster node in NFT.storage, which is the largest IPFS Cluster network, currently consists of 24 nodes which serve as a reference point for configuring the number of nodes [10]. The number of shards $k + m$ that can be created is 22, which guarantees the same degree of availability for the m parity shards and replicas created by each architecture. The procedure for evaluating the storage overhead is as follows. In the traditional IPFS Cluster replica-based approach, a single shard is replicated. Consequently, if you create m replicas of the original data of size 1, the storage overhead increases by $m + 1$. On the other hand, in the proposed architecture, the original data of size 1 is chunked into k shards, and an additional m parity shards of the same size are generated.

As shown in the Figure 4, when $m = 2$, the replica approach has an additional storage usage of 2 for 2 replicas, so the storage overhead is 3. According to the proposed Reed-Solomon approach, the original data is divided into 20 shards, and 2 parity shards of size $1/20$ are generated. The additional storage usage of the original data is 0.1, so the overhead of the storage is 1.1.

When flexibly adjusted the values of k and m, the storage overhead of the Reed-Solomon approach increased more as the value of k became smaller, but the rate of increase was moderate. In particular, observing the largest impact in $k = 14, m = 8$, a storage overhead reduction of about 6x compared to the replication approach.

In contrast to the replica-based availability approach, the proposed encoding technique using combinations of data and parity shards ensures data availability while reducing storage

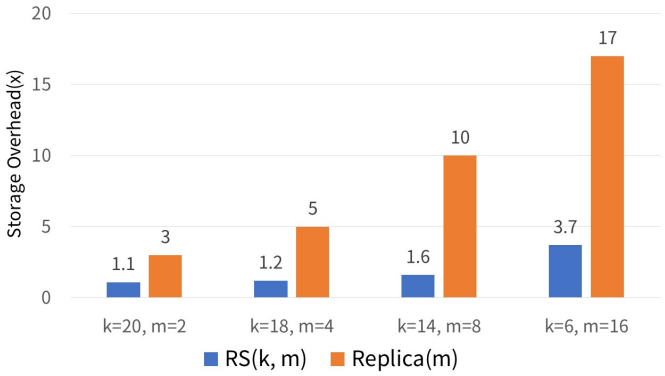

Fig. 4: Storage Overhead for Availability (Compared to Original Data)

space. Furthermore, considering the importance of data within the IPFS file storage system and user requirements for data availability and storage usage, adjusting the k and m values of shards flexibly allows optimization of storage space efficiency. For enhanced resilience to data failure, we can configure it to generate more parity fragments. Thus, the application of Reed-Solomon encoding reduces space burden and offers a robust architecture against data loss.

While various IPFS-based decentralized storage services like Filebase and Pinata have been introduced, the replica-based storage model often incurs substantial costs for users. The proposal in this paper, through the utilization of Reed-Solomon erasure codes within the IPFS Cluster, is expected to maximize storage space efficiency in IPFS-based decentralized storage, enabling users to access services at a lower cost.

V. Conclusion

IPFS, developed to ensure individual content ownership beyond the traditional HTTP and Web 2.0 environments, is widely utilized as decentralized storage for large-scale data in numerous blockchain-related services. In this study, we applied the Reed-Solomon erasure code within IPFS Cluster to address the storage capacity issue and enhance data availability and stability when storing and accessing large-scale data. Furthermore, the proposed space-efficient management technique for large-scale data is expected to play a crucial role in various interconnected metaverses in a Web3-based multiverse environment. It would serve as a foundational technology for storing, managing, and utilizing digital avatars and content. Subsequent research will implement the Reed-Solomon erasure code into the IPFS and IPFS Cluster architecture to validate the efficiency, stability, and storage/retrieval performance of the proposed method.

Acknowledgment

This work was supported by Institute for Information& communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2021-0-00136, Development of Big Blockchain Data Highly Scalable Distributed Storage Technology for Increased Applications in Various Industries).

References

[1] D. Delen and H. Demirkan, "Data, information and analytics as services," pp. 359–363, 2013.
[2] J. Wu, L. Ping, X. Ge, Y. Wang, and J. Fu, "Cloud storage as the infrastructure of cloud computing," in *2010 International Conference on Intelligent Computing and Cognitive Informatics*, 2010, pp. 380–383.
[3] P. Yang, N. Xiong, and J. Ren, "Data security and privacy protection for cloud storage: A survey," *IEEE Access*, vol. 8, pp. 131 723–131 740, 2020.
[4] J. Park and S. Choi, "Web 3.0 Reboot: Issues and Prospects," *Electronics and Telecommunications Trends*, vol. 37, no. 2, pp. 73–82, 2022.
[5] J. Benet, "Ipfs-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.
[6] Protocol Labs, "IPFS powers the Distributed Web," https://ipfs.tech/ [Online].
[7] H. Huang, J. Lin, B. Zheng, Z. Zheng, and J. Bian, "When blockchain meets distributed file systems: An overview, challenges, and open issues," *IEEE Access*, vol. 8, pp. 50 574–50 586, 2020.
[8] Protocol Labs, "Persistence, permanence, and pinning," https://docs.ipfs.tech/concepts/persistence/ [Online].
[9] Y. Psaras and D. Dias, "The interplanetary file system and the filecoin network," in *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*. IEEE, 2020, pp. 80–80.
[10] S. J. Hector, "IPFS Cluster: scaling IPFS data storage," https://blog.ipfs.tech/2022-07-01-ipfs-cluster/ [Online].
[11] S. Muralidharan and H. Ko, "An interplanetary file system (ipfs) based iot framework," in *2019 IEEE International Conference on Consumer Electronics (ICCE)*, 2019, pp. 1–2.
[12] Protocol Labs, "Distributed Hash Tables (DHTs)," https://docs.ipfs.tech/concepts/dht/ [Online].
[13] J. S. Plank, "A tutorial on reed–solomon coding for fault-tolerance in raid-like systems," *Software: Practice and Experience*, vol. 27, no. 9, pp. 995–1012, 1997.
[14] J. S. Plank, J. Luo, C. D. Schuman, L. Xu, Z. Wilcox-O'Hearn *et al.*, "A performance evaluation and examination of open-source erasure coding libraries for storage." in *Fast*, vol. 9, 2009, pp. 253–265.
[15] B. Choi, C. Kim, and M. Lee, "Research trends on distributed storage technology for blockchain transaction data," *Electronics and Telecommunications Trends*, vol. 37, no. 3, pp. 85–96, 2022.
[16] S. Park, B. Choi, C. Kim, M. Lee, and I. Lee, "Erasure code-based distributed storage system for storage-efficient hyperledger fabric blockchain," *ISSN: 2383-8302*, pp. 555–556, 2022.