Decentralized Storage Network based on Blockchain Technology: A Survey

Qi Xiang Zhang
Institute of Information Management,
National Yang Ming Chiao Tung University, Taiwan

***Abstract*—When we explore the application of blockchain technology in decentralized storage networks (DSN), we will examine three papers from the IEEE Transactions journal in sequence. Each of these papers proposes their own solutions for DSN. DSN is favored because it offers many advantages over centralized storage networks. One significant advantage is the ability to avoid the issue of data inaccessibility caused by a single point of failure. Moreover, the three main characteristics of blockchain—decentralization, immutability, and high transparency—help address the issues that have arisen in centralized storage networks. Additionally, the application of smart contracts on blockchain simplifies the management of distributed nodes and can automatically reward honest storage nodes while eliminating and penalizing dishonest ones. In these three papers, we will see the importance of game theory in blockchain-based decentralized services, and the authors will present their respective solutions to the previously cumbersome storage verification mechanisms (PoS). Let us delve deeper into this topic in this survey paper!**

***Index Terms*—Distributed Systems, Blockchain, Smart Contract, InterPlanetary File System (IPFS), Game Theory, Decentralized Storage Network (DSN), Proof of Storage (PoS).**

I. Introduction

In many past research papers, we have seen the limitations of centralized storage systems, such as their vulnerability to single points of failure, the misuse of customer data by a few centralized storage service providers, and the monopolistic control over pricing and regulations by a handful of central storage service companies. As a possible solution, decentralized storage networks (DSNs) have been proposed. These networks allow small-scale storage service providers to participate in the market and use blockchain technology to store file information. They also employ smart contracts deployed on the blockchain to automatically move files and assets according to predefined rules.

Currently, experts in the field of decentralized storage networks are exploring various methods to disrupt the existing cloud storage service market through DSNs. First, these decentralized storage networks are likely to operate on the basis of a free market open to public participation. Thus, anyone can join the DSN without relying on a single node or storage service provider, with data replicated across multiple nodes. Moreover, public key encryption is a natural feature integrated with blockchain. Servers typically encrypt data before storage, ensuring that only the legitimate private key holders and their approved clients can decrypt the data. This process makes decentralized storage services more resistant to censorship and manipulation. Even in the event of a data breach, attackers would be unable to exploit the leaked data without the private key.

Using decentralized networks to store data as an alternative to traditional centralized storage networks, peer-to-peer storage is emerging as a disruptive force. Below are some advantages of storage systems that adopt decentralized storage networks [4]:

1) **High Reliability**: Decentralized networks use multiple servers to transmit and store data. Redundant copies of data are stored, eliminating the impact of single points of failure. In the event of hardware failures or file loss, backup copies of files will be accessible. Additionally, each piece of shared data is assigned a unique hash value. This added layer of protection makes the data more secure.
2) **Low Deployment Cost**: Decentralized data storage systems significantly reduce hardware and storage costs. In a decentralized environment, the performance requirements for storage nodes are lower, reducing the need for expensive investments in high-performance hardware and software. Moreover, there may be millions of nodes in a decentralized network storing data, which greatly increases available storage space. Decentralized storage systems can continually utilize all idle storage space, reducing waste and avoiding the need to invest in new storage equipment. Overall storage costs are significantly lower compared to traditional centralized cloud storage.
3) **Increased File Access Speed**: Unlike traditional centralized storage, decentralized storage systems rely on peer-to-peer technology. During peak traffic times, data transfer is not conducted through a single central server. Multiple copies of data are stored at different node locations, allowing clients to choose nodes with data copies, thereby speeding up overall file download times.
4) **Load Balancing**: Blockchain-based decentralized storage systems follow load balancing principles. Servers can locally cache data to avoid repeatedly accessing the server. This not only reduces the burden on servers but also alleviates network traffic pressure. In addition to transferring and optimizing data, non-centralized servers help eliminate bottlenecks created by centralized storage systems.

5) **Fair Market Pricing**: With millions of nodes, decentralized storage systems form a fully competitive market where micro-storage service providers can compete with larger companies. A single node cannot charge higher prices in a fully competitive storage service market, resulting in uniformly distributed prices across all nodes with minimal price discrepancies. Most importantly, this market ensures that only high-quality nodes can survive and compete.
6) **Enhanced Security and Privacy**: The high security offered by decentralized data storage systems is their most notable advantage. Shared data is encrypted using hash values or public-private keys and broken down into smaller blocks, which are then shared among nodes in the distributed network. This process ensures that data is protected from malicious attackers. Additionally, none of the stored data contains information about the original data owners, which is different from centralized storage systems.

Despite the numerous advantages of decentralized storage networks (DSNs) highlighted above, these networks are also susceptible to various malicious attacks that can impact any decentralized storage system. Below are some possible malicious attacks on DSNs and their corresponding solutions [4]:

1) **Spartacus Attack**: On Kademlia (a distributed hash table (DHT) technology used for storing and retrieving data in distributed networks), Spartacus attacks, also known as identity theft, can occur. Any node can impersonate another node by replicating its node ID and receive certain information intended for that node. To mitigate this attack, all messages for nodes and data can be required to be signed, with node IDs implemented as hashed ECDSA public keys. This prevents malicious actors from successfully signing information or participating in decentralized storage systems.
2) **Sybil Attack**: The Sybil attack involves setting up numerous nodes to disrupt the entire network by dropping or stealing messages. Conducting such attacks on Kademlia is challenging due to its reliance on redundant messaging and specific distance metrics. Most messages are sent to at least three neighboring nodes in the network, which are selected based on their node IDs. When attackers control 50
3) **Google Attack**: This hypothetical attack is conducted by a well-resourced entity, similar to the Sybil attack. It's challenging to defend against Google attacks due to the unpredictability of Google's actions. The only defense is to create a network with resources comparable to those of the malicious attacker. Establishing such a network aims to match the adversary's capabilities, but it requires significant resources, which may not be sustainable.
4) **Honest GEPPETTO Attack**: A variant of the Google attack, Honest GEPPETTO targets storage devices. Malicious actors operate numerous puppet nodes in the network over time to accumulate trust and valid contracts. Once a threshold is reached, attackers remove or manipulate these puppet nodes from the network to execute data withholding attacks. Similar to previous attacks, a large-scale network renders such attacks ineffective. Prior to this, relevance analysis of nodes can partially address this issue. When downtime, latency, and other attributes are applied to Bayesian inference, data owners should distribute data across as many unrelated nodes as possible.
5) **Eclipse Attack**: Eclipse attacks isolate one or more nodes in the network by ensuring all outbound traffic connects to malicious nodes. Eclipse attacks may induce malicious nodes to operate normally while only obscuring specific critical messages. Attackers need to generate key pairs until finding three hashes that are closer to the target ID than the IDs of neighboring honest nodes and protect this position from attacks by nodes with closer IDs. In practice, there are typically many nodes in the network, making such attacks increasingly difficult as nodes increase, scaling with the difficulty of the proof-of-work problem. Therefore, to defend against Eclipse attacks, the total number of nodes in the network should be increased.
6) **Data Segment Withholding Attack**: Attacks specific to storage, like data segment withholding attacks, involve malicious storage service providers refusing to transmit data segments or segment parts to extort additional payments from data owners. Data owners can protect themselves from data segment withholding attacks by storing redundant segments among multiple nodes. If clients maintain their erasure coding secret, malicious storage service provider nodes cannot determine the last byte. Practical applications of this attack are mostly resolved through redundant storage. However, redundant storage is not a complete solution. Multiple malicious nodes must collaborate to break the defense of redundant storage, which is challenging in practice.
7) **Owner Fraud Attack**: Data owners may refuse to verify the authenticity of audits to avoid paying storage fees to storage service providers. Storage service provider nodes may then discard data segments of the data owner. Due to such attacks, any future decentralized reputation system will struggle to verify its reputation. Currently, there are no publicly verifiable storage proofs or independently verifiable processes to confirm whether planned private verifiable audits are sent or responded to. Therefore, any reputation system still faces the issue of fraudulent clients.

In the following paragraph, I will first briefly explain some important Related Work that will be mentioned in the three IEEE Transactions journal papers. Then, I will divide the content into three paragraphs to respectively introduce the important content summaries of these three journal papers, as well as some insights I gained after reading them. Finally, I will provide a brief conclusion regarding the research topic of this study.

II. Related Work

In this paragraph, I will provide a summary of the key content in these three IEEE Transactions journal papers. The

focus will primarily be on blockchain, smart contracts, and the InterPlanetary File System (IPFS). Other related technologies mentioned in these papers will not be introduced here. For those interested in technical details, please refer to the six referenced papers in the References section.

- **Blockchain [5]**: Blockchain technology is based on P2P network. Blockchain is a decentralized ledger recording transactions, hence, no central supervisory authority exists. Each node in the blockchain network regularly synchronizes the distributed ledger. Cryptography is utilized in blockchain to enhance its security. Blockchain employs cryptographic hash mechanism, making data stored in blocks immutable. If data in one block is altered, then data in each subsequent block should be recalculated with new hash value, which is practically unfeasible. Immutability is one of the fundamental applications of blockchain. Blockchain supports transparent and immutable transactions, which are easily monitored. Additionally, blockchain can be categorized into several types, including public, private, or consortium chains. Bitcoin is a prominent example of a public blockchain where anyone can mine, transact, and join the network. In a private blockchain, one organization is responsible for governing transactions and determining who can join the network. When multiple organizations collaborate to implement blockchain, the responsibility of maintaining the blockchain is shared among them, known as a consortium blockchain.
- **Smart Contract [6]**: Ethereum is one of the prominent branches of blockchain, and a significant component of Ethereum is smart contracts. Smart contracts have become increasingly popular, especially with the recent boom in non-fungible tokens (NFTs), which are issued through smart contracts. They were first proposed by computer scientist and cryptographer Nick Szabo in the early 1990s as a form of digital transaction protocol. Smart contracts explain how users can input data or value and perform restricted operations on machines (for example, a vending machine). In simple terms, smart contracts are typically user programs, algorithms, or protocols that can be used to verify, validate, or execute irreversible transactions. This represents a clear paradigm shift, where dishonest organizations seek to gain trust in their agreements to ensure the proper functioning of computer systems. A well-executed smart contract can enable crowdfunding without the need for third-party intervention. Third-party involvement often centralizes the system, concentrating all trust in one organization. Since smart contracts deployed on the blockchain are immutable, transparent, and authoritative, disrupting their execution is nearly impossible. The largest blockchain branch, Ethereum, has been supporting specially designed smart contracts since 2014. Solidity is one of the programming languages specifically popular for developing smart contracts on Ethereum. In recent years, transboundary insecure programming crimes have resulted in significant economic losses and social deterioration, posing challenges for the proper development, validation, and execution of smart contracts.
- **InterPlanetary File System (IPFS) [5]**: IPFS (InterPlanetary File System) is a file storage system distributed across multiple computers or servers. It addresses security, reliability, and scalability issues present in existing file storage systems. IPFS provides high throughput for accessing stored files. In the mining process, the previous hash value is added to the new block, and the miner who first calculates the block hash shares it with other blockchain nodes. Subsequently, other nodes verify the authenticity of the block mined by that miner. In practice, image files are stored in the IPFS distributed file system using the IPFS API. The SHA-256 algorithm is used to generate the hash value, and a salting procedure is utilized to convert the message digest into hexadecimal format. The message digest (MD) and hash value are combined to form a hash of the content address, which is then stored in the blockchain network. IPFS generates a 46-byte long hash value, resulting in less storage space consumption compared to traditional file storage systems. In this proposed system, miners collect transactions, mine valid ones, and store them in the IPFS distributed network, then generate new blocks in the blockchain network. IPFS provides a unique hash value for stored transactions, allowing other nodes to access the transactions using this unique hash value.

III. An Incentive-Compatible Mechanism for Decentralized Storage Network [1]

A. Introduction

- **Concerns with the centralized architecture of storage service providers**:
  - Data centers are more susceptible to single points of failure.
  - Companies may misuse customers' personal data for greater profits.
  - Prices and regulations are dominated by a few large companies, leading to monopolies.
- **Decentralized Storage Network (DSN) - Advantages**:
  - Small-scale storage service providers can also participate in the storage service market.
  - Information is stored across a network of computers (blockchain technology).
  - Various incentive mechanisms can be developed using blockchain smart contracts, which automatically move digital assets according to predefined rules.
  - Customers can outsource their data storage by paying fees to the network.
  - Storage service providers share their storage resources with the network and receive additional rewards in return.
  - To ensure data confidentiality, customers' data is encrypted end-to-end on the client side, preventing storage service providers from accessing the decryption keys (eliminating centralized control of data).

  - Small-scale storage service providers can rent out their excess storage resources, increasing network throughput and reducing maintenance costs for data centers.
- **Decentralized Storage Network (DSN) - Challenges**:
  - How to verify the correct storage of data by storage service providers?
  - A DSN platform should be equipped with a Proof of Storage (PoS) method to monitor the honesty of storage service providers (PoS should be performed periodically during the storage service contract).
  - However, continuous PoS verification is costly and vulnerable to attacks by malicious storage service providers who submit PoS to DSN nodes while refusing to serve customers.
- **Decentralized Storage Network (DSN) - Improvements**:
  - In this paper, the authors propose and analyze a new decentralized storage method that does not require constant storage verification, but only a one-time verification when challenged by customers (challenge request).
  - The authors also designed a new incentive-compatible mechanism, enabling participants to achieve optimal results by truthfully choosing their actions.
  - More specifically, the authors designed a non-cooperative repeated dynamic game where the only subgame perfect equilibrium is for storage service providers to honestly share the stored data.
  - The authors enforce the rules of the proposed storage game using blockchain smart contracts and an oracle network. Eliminating continuous storage verification significantly improves DSN performance and protects customers from denial-of-service attacks.

B. Overview

In summary, the decentralized storage network (DSN) is equipped with two main components: payment settlement and storage verification:

- In the payment settlement module, the DSN charges customers for storage services and pays the storage service providers. If a storage service provider fails to deliver the storage services, the DSN can penalize them and compensate the customers accordingly.
- In the verification module, the DSN verifies whether the storage service providers accurately provide the storage services. To achieve this, storage service providers are required to submit proof of storage to the DSN, which then verifies the accuracy of such proof.

DSN utilizes public blockchain technology to execute storage contracts. Blockchain technology provides an acceptable platform for parties to make payments without the need for a single trusted third party. Therefore, in a blockchain-enabled DSN scheme, no single party can control any storage contract. In DSN storage services, customers may encrypt their data before submitting it to the storage service provider to protect the confidentiality of the data. Additionally, DSN can provide redundancy, high availability, and failover capabilities by storing data across multiple nodes in the network. The authors assume that storage service providers manage their storage resources, including redundancy, server location, backup services, network bandwidth, etc., to maximize their returns according to the Service Level Agreement (SLA). The overall approach of a blockchain-based DSN is illustrated in Figure 1, where clients request data storage services from storage service providers. Subsequently, related storage payments and service verifications can be performed through smart contracts deployed on the blockchain.

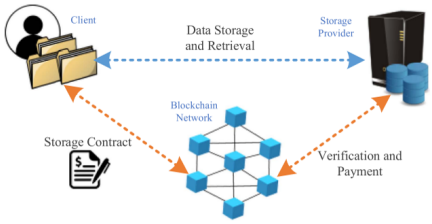


Fig. 1. The overall scheme of a Blockchain-based Decentralized Storage Network.

Fig. 1: Image Source from [1]

Mechanism Design Objectives:

- **Blockchain Platform Independence**: Propose a compatible storage solution that can be integrated into existing general-purpose blockchain platforms with smart contract execution compatibility.
- **Prevention of Denial of Service Attacks**: Protect the DSN network from such fraudulent storage service providers.
- **On-Chain Efficiency**: The proposed solution should minimize on-chain storage and computation costs without compromising security expectations.
- **Computation Requests**: Many storage services expect to calculate the number of customer requests to dynamically compute the cost of storage services.

C. Proposed Incentive-Compatible Mechanism

The authors' mechanism design aims to establish a set of rules for storage services in DSN to meet the previously mentioned requirements. A mechanism can be specified by a game $g : M \rightarrow X$, where M is the set of possible input messages and X is the set of possible outputs of the mechanism. In the storage system model, the players are the storage service providers and the storage-demanding customers. The authors assume that players are rational and self-interested, with the goal of maximizing their profits. A rational player will choose a specific strategy to increase their utility. The utility of storage customer players is based on the cost of paying for storage services and the cost or benefit of data accessibility after outsourcing storage to storage service providers.

When designing mechanisms for decentralized storage networks, the following questions need to be addressed:

- How does the mechanism verify the honesty of storage service providers in sharing data?
- What is the payment channel for storage data services?
- How should the mechanism charge customers?
- How should the mechanism penalize storage service providers for data loss or poor service quality?

A straightforward model involves using a Trusted Third Party (TTP) to mediate the relationship between customers and storage service providers. In this scenario, the customer requests data from the TTP, which then receives the data from the storage service provider. The TTP can check the data integrity using the hash value of the stored data and verify it upon receiving it from the storage service provider. However, this method is inefficient, costly, and non-scalable due to the necessity of the TTP acting as an intermediary for each request and response.

To address this issue, Decentralized Storage Networks (DSNs) rely on blockchain platforms as a trusted third party, eliminating the need for a single central entity to manage the system. Additionally, DSNs minimize the involvement of the trusted third party in the data retrieval process, allowing customers to retrieve data directly from the storage service providers. However, decentralized storage networks continuously verify the storage proofs provided by storage service providers to ensure that they are genuinely storing the data:

- For the network, continuous verification of storage is expensive.
- Dishonest storage service providers can successfully submit storage proofs to decentralized storage networks while refusing to provide services to customers.

To address these issues, in the model proposed by the authors, decentralized storage networks do not continuously verify storage services. Instead, verification occurs when a client submits a challenge request. (When dishonest storage service providers refuse to provide data or send incorrect data, clients can send challenge requests to a trusted third party.) In the mechanism proposed by the authors, the interaction between clients and storage service providers can be modeled as a non-cooperative repeated dynamic game:

- Once the storage service contract begins, in the first stage of the game, the storage service provider can choose a strategy of sharing data or not sharing data.
- Clients can choose to challenge or not challenge the storage service provider. Challenging means the client submits a challenge request to the DSN, indicating that the storage service provider has not shared the data.
- Upon receiving the challenge request, the DSN performs storage verification. Once the storage service provider is challenged, their strategy set is to provide storage proof or not provide storage proof.

The goal of the authors' mechanism design is to ensure that the subgame perfect equilibrium of this non-cooperative repeated dynamic game is achieved (where the storage service provider is willing to share data, and the client does not challenge the storage service provider's storage proof). The storage contract between the client and the storage service provider can be represented as the non-cooperative repeated dynamic game depicted in the diagram below. In this game, the storage service provider can choose whether to share data, and the client can choose whether to challenge the storage service provider. The nodes S and C at the terminal of the tree represent the payoff obtained by the storage service provider and the client from the game, respectively, and this payoff is calculated based on the utility function according to the strategies adopted by both parties.

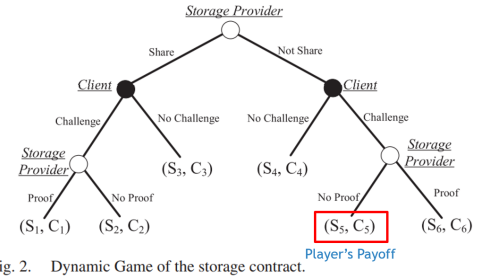


Fig. 2. Dynamic Game of the storage contract.

Fig. 2: Image Source from [1]

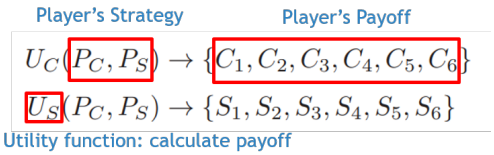


Fig. 3: Calculate the player's utility

Therefore, the authors aim for the storage service provider to choose *Sharing* when data is not shared, and the client should choose Challenging when data is not shared. To achieve this goal, the mechanism should incentivize the client to choose to *challenge* when data is not shared; however, challenging requests are costly. To cover the cost of challenging, the proposed mechanism is designed such that the proof strategy mandates sending a copy of the data to the client to enhance the payoff of the challenging strategy when data is not shared. Let P represent the probability that the storage service provider cannot provide storage service. Let V represent the value of the client accessing the data. Then, the expected utility of the client choosing to challenge can be represented by the equation below.

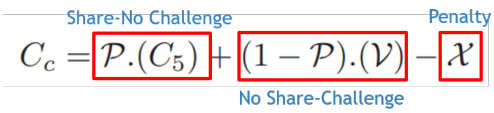


Fig. 4: Client's challenge cost

The following figure illustrates the payoffs obtained by the two participants in the non-cooperative dynamic game under each other's strategy choices. The equilibrium point of the game is at "share-nochallenge," where the total payoff is maximized.

The authors utilize smart contracts as a trusted third party to manage the agreements between clients and storage service providers. Smart contracts are powered by blockchain

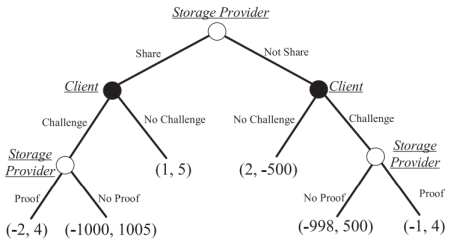


Fig. 3. Dynamic Game of the example storage contract.

Fig. 5: Image source from [1]

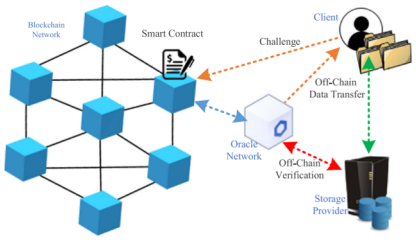


Fig. 4. Interaction of different components in the proposed challenge based DSN storage contract.

Fig. 6: Image source from [1]

technology, which is managed by a peer-to-peer network governing a digital ledger. In the proposed model, clients and storage service providers first reach a storage agreement. This agreement includes information such as the contract duration, the Merkle root of the data, the storage service fee, data delivery time, and compensation. The Merkle root of the data is stored on the blockchain and will be used for data verification. Storing the entire data on the blockchain is costly, so a Merkle tree is used to minimize the cost of the storage verification process.

One thing to note is that the blockchain platform operates as an isolated network and cannot extract or push data from external systems. This issue is known as the Oracle problem. To address this challenge, the concept of an Oracle network is introduced. The Oracle network provides a trusted source for the blockchain to access data outside of its network. The interaction between different components in the method proposed by the authors is illustrated in the diagram below. It can be observed that data transmission occurs off-chain. The only on-chain operation is the challenge request. It is noteworthy that, in this approach, the storage service provider should send the challenged data to the Oracle network, which then forwards the data to the client. This design serves two primary purposes. Firstly, it prevents the service denial attacks explained earlier. This is because if a storage service provider refuses to serve a client, the client will receive a copy of the data in the challenge request if the storage service provider can provide proof. Therefore, the storage service provider cannot refuse service to the client while proving storage to the DSN. Secondly, when the storage service provider has not yet shared the data, the mechanism should incentivize the client to submit challenges. By forwarding the data, the authors increase the value of the payoff for clients who choose the challenging strategy, achieving the expected subgame perfect equilibrium discussed earlier.

Data transmission is completed off-chain. The only on-chain operation is the challenge request. It's worth noting that in this approach, the storage service provider should send the challenged data to the Oracle network, which then forwards the data to the client. This design serves two primary purposes:

- Preventing service denial attacks: The storage service provider cannot refuse to provide service to the client while proving storage to the DSN.
- When the storage service provider has not yet shared the data, the mechanism should provide incentives for the client to submit challenges.

To enhance the efficiency of the method, the authors considered different levels of challenge requests. They propose that when the client and storage service provider agree to divide the data into a specific number of segments, during the challenge phase, the client can submit a challenge for a group of segments. Rather than requiring verification of the completeness of the entire data when the client perceives incomplete data, they can selectively verify specific data segments, thereby improving the overall efficiency of the DSN.

Given the security parameter λ, the PoS method consists of a set of four probabilistic polynomial-time (PPT) algorithms, including Setup, Challenge, Prove, and Verify, as presented below:

- (d, h) $\leftarrow$ Setup(1^λ, D, sz). This algorithm takes the security parameter λ, outsourced data D, and segment size sz as inputs. It outputs the data digest d for verification and the maximum number of data segments h.
- c $\leftarrow$ Challenge(h). This algorithm takes the maximum number of data segments h as input and outputs the challenge number c.
- π $\leftarrow$ Prove(D_c, c). This algorithm takes the corresponding data segment D_c and challenge number c as inputs. It outputs the proof π for the data segment stored corresponding to the challenge number c.
- 0/1 $\leftarrow$ Verify(d, h, c, π). This algorithm takes the data digest d, Merkle Tree height h, challenge number c, and proof π as inputs. If the proof π is valid, it outputs 1; otherwise, it outputs 0.

The first property that the authors require from the PoS method is completeness, which means that the probability that the proof output by the algorithm for a valid statement is correctly verified is negligibly close to one. This can be expressed by the following equation, and if a PoS method is complete, it will satisfy the following condition:

$$\Pr\left[\begin{array}{c}(d,h)\leftarrow \text{Setup}(1^\lambda,\mathcal{D},sz)\\ c\leftarrow \text{Challenge}(h)\\ \pi\leftarrow \text{Prove}(\mathcal{D}_c,c):\\ 1\leftarrow \text{Verify}(d,h,c,\pi)\end{array}\right] > 1-\text{negl}(\lambda).$$

Fig. 7: Image source from [1]

The second property is challenge statement, which captures that a malicious challenger cannot generate a challenge c even if the prover holds the entire document. It can be expressed by the following equation, and if a PoS method is challenge statement, it will satisfy the following condition:

$$\Pr\left[\begin{array}{c}(d,h)\leftarrow \text{Setup}(1^{\lambda},\mathcal{D},sz)\\ c'\leftarrow \text{Challenge}(h)\\ \pi\leftarrow \text{Prove}(\mathcal{D},c'):\\ 0\leftarrow \text{Verify}(d,h,c,\pi)\end{array}\right]<\text{negl}(\lambda).$$

Fig. 8: Image source from [1]

The third property is proof statement, which captures that even with all other document segments, a malicious storage prover cannot generate a valid proof if it does not hold the challenged document segment. It can be expressed by the following equation, and if a PoS method is proof statement, it will satisfy the following condition:

$$\Pr\left[\begin{array}{c}(d,h)\leftarrow \text{Setup}(1^{\lambda},\mathcal{D},sz)\\ c\leftarrow \text{Challenge}(h)\\ \pi\leftarrow \text{Prove}(\hat{\mathcal{D}}_c,c):\\ 1\leftarrow \text{Verify}(d,h,c,\pi)\end{array}\right]<\text{negl}(\lambda).$$

Fig. 9: Image source from [1]

Cryptographic Building Blocks: The author utilizes an encryption-secure hash function $H \leftarrow Hash(X)$ with collision resistance and difficult-to-reverse computation. Digital Signatures: The author employs a standard EU-CMA secure digital signature function, consisting of three functions: one for generating public and private keys, one for signing, and one for verifying the validity of the signature.

- *(pk, sk)* $\leftarrow$ *KeyGen*(1^{λ})
- $\sigma \leftarrow Sign(sk, m)$
- *1/0* $\leftarrow$ *Verify(pk, m,* σ*)*

Integrity comes directly from the protocol. The assertiveness of challenges arises from the maximum number of challenges verified in the setting function. The assertiveness of proofs comes from the hash value being a hash function resistant to collisions and having recovery difficulty. Obviously, if there is an adversary undermining the assertiveness of proofs, authors would use this adversary to undermine the collision resistance of the hash value, contradicting the assumption. The following figure is a simple example of a Merkle tree:

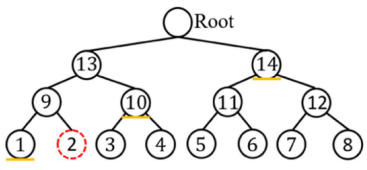


Fig. 5. Merkle tree example.

Fig. 10: Image source from [1]

There are two main reasons why the authors chose Merkle trees to construct their proof-of-stake (PoS) based system:

- In a decentralized setting, the authors aimed for trustless setups and fewer common parameters, which limits the use of RSA and bilinear pairing-based solutions.
- Costs of both proving and verifying were considered in the incentive layer. The primary concern of the proposed method was the size of the digest data, as this data needs to be stored on-chain.
- At this point, Merkle trees emerged as a viable solution, as authors only need to store the Merkle root on-chain, which is a hash value.

However, due to the potential for malicious storage service providers to refuse data delivery, clients should send a signature confirmation message after successful data delivery. On the other hand, malicious clients may also refuse to send signature confirmation messages:

- To address this issue, a straightforward approach is to segment the data into smaller segments and send the next portion of data after receiving confirmation of the previous message.

At the application layer, this method incurs high costs because each confirmation message should be signed and verified. Additionally, there is no guarantee that the client will send confirmation of the last message.

- To solve this problem, a simple approach is employed. In the proposed method, authors follow the proposed non-cooperative repeated dynamic game theory approach to challenge storage service providers. In the proposed model, clients only send a signed request message.
- For keeping track of the number of requests, storage service providers only need to submit the last signed request message. It's worth noting that the request message includes a counter, recording the number of requests the client has submitted so far.

D. Implementation

The authors implemented the proposed smart contract using the Solidity programming language (version 0.8.7). For the blockchain solution, they chose Kovan, which is Ethereum's test network, used for smart contract development. The authors utilized the Remix IDE for the development, deployment, and management of smart contracts. During implementation, they employed Chainlink's Oracle, the most renowned Oracle network currently dominating most of the Oracle market, and they have shared their implementation on GitHub. Considering gas fees and the expensive on-chain computation costs, the authors only recorded basic information on-chain, including smart contract premiums, contract terms, compensation rates, and the Merkle root value of files.

The smart contract has two primary functions:

- Recording storage tasks: After reaching a commitment between the client and storage service provider, the contract records the signed basic information of both parties on-chain, based on the designed data structure.
- Resolving challenge requests: Once a client submits an on-chain request challenging a specific file segment, the authors need to establish a connection with off-chain data.

A basic Oracle request model comprises four components: Chainlink client, Oracle contract, off-chain Oracle nodes, and external adapters.

- **Chainlink client**: It's an upper-level contract enabling the smart contract to be constructed and issue requests. A complete request requires the Oracle address, job ID, and callback function. Through the job ID, Oracle knows which tasks to execute; upon completing tasks, Oracle sends responses to the callback function.
- **Oracle contract**: Responsible for handling on-chain requests and emitting events containing request messages. Off-chain Oracle nodes monitor these events, and once the Oracle contract receives the job result, it returns the result to the Chainlink client using the callback function.
- **Off-chain Oracle nodes**: Run concurrently with the Oracle contract. They listen to events emitted by the on-chain Oracle contract and execute tasks using the dispatched data. Here, the node sends GET requests to external adapters to obtain the result of whether the storage service provider passed the challenge. It then submits the boolean result through a transaction back to the Oracle contract.
- **External adapter**: Sends requests to storage service providers, with parameters being the file ID and the specific data segment number chosen by the client. The storage service provider returns the challenged data segment along with the computed Merkle path to the external adapter. The external adapter processes the response using the Merkle path and the hash value of the data segment to calculate the Merkle root and compares it with the value stored on-chain. Subsequently, it submits the challenge result to the Oracle contract.

The external adapter is written in JavaScript and operates as an HTTP server in Node.js. For storage service providers, the authors simulate them as HTTP servers in Node.js. It provides an external API to calculate the Merkle root, access the original file, and generate the Merkle path. In subsequent experiments, the authors conducted performance analysis of the proposed method on a macOS (version 12.0.1) laptop equipped with an Apple M1 Pro CPU and 32GB of memory. They evaluated files of various sizes, including 10 MB, 50 MB, 100 MB, 500 MB, and 1 GB. The computation time was calculated based on different segment sizes of the file. Specifically, the authors compared four dimensions: file read time, Merkle root calculation time, Merkle path generation time, and Merkle path verification time.

The following figure illustrates the computational costs of file read time, Merkle root calculation, Merkle path generation, and Merkle path verification, respectively, for files of sizes 10 MB, 50 MB, and 100 MB.

The following figure depicts files of sizes 100 MB, 500 MB, and 1 GB with the same setup as Figure 6. It can be observed that as the segment size increases, the computational costs of file read time, Merkle root calculation, and Merkle path generation exhibit a decreasing trend. Merkle path verification is fast, with its time decreasing as the segment size increases. Conversely, when files of the same segment size increase in

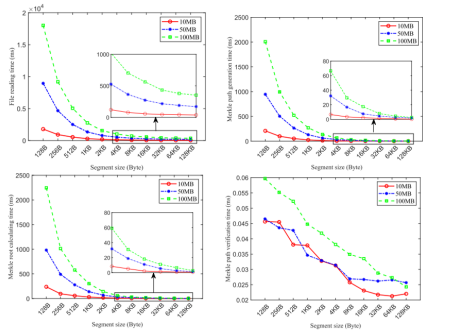


Fig. 6. Computation cost of File reading, Merkle path generation, merkle root calculation, and Merkle path verification for files of 10 MB, 50 MB, and 100 MB with varying segment size.

Fig. 11: Image source from [1]

size, the file read time, Merkle root calculation, and Merkle path generation times increase synchronously. On the contrary, Merkle path verification time remains almost constant and very small.

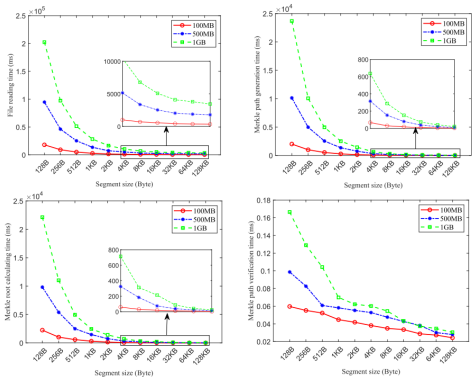


Fig. 7. Computation cost of File reading, Merkle path generation, merkle root calculation, and Merkle path verification for files of 100 MB, 500 MB, and 1 GB with varying segment size.

Fig. 12: Image source from [1]

The method proposed in the paper is divided into two layers: the incentive layer and the PoS layer:

- In the incentive layer, players are incentivized to participate honestly in storage games. To achieve this, the authors use smart contracts to enforce game rules, making it incentive-compatible.
- In the PoS layer, the system verifies storage services after a client submits challenge requests. This verification module is implemented using the oracle network within the smart contract, so the security of the proposed method relies on the security of both the smart contract and the oracle network.

The authors utilized the Chainlink oracle network, which is currently the most widely used oracle network. The requirement for an oracle network in the design can be seen as a limitation of the proposed method. This is mainly because the security of the proposed method depends on the security of the adopted oracle network.

For storage service providers and storage verifiers, the computational cost of PoS is low and negligible. The primary advantage of the proposed method is that the network does not continuously verify storage services but only performs one-time verification when storage service clients submit challenge

requests. It is expected that the system will not need to execute PoS during the contract's duration unless storage service providers fail to provide storage services. It should be noted that for each update request (data change), the Merkle root stored on-chain should also be updated. In this case, the smart contract should verify the signatures of both the client and the storage service provider to ensure agreement on the update. Therefore, the proposed method is the most suitable for storing archival data and is inefficient for data storage with frequent change requests. Improving the method to handle update requests more efficiently can be considered a future research direction.

E. Conclusion

In this paper, the authors introduce a novel game theory mechanism for decentralized storage networks, allowing clients to challenge storage service providers. This enables the authors to eliminate the requirement for continuous verification of storage service providers, thus improving the performance of the DSN. In addition, clients are protected against service denial attacks, where a dishonest storage service provider submits storage proofs to the network while refusing to serve clients. The model proposed in this paper can be integrated into any blockchain platform with smart contract execution capabilities. The authors utilize smart contracts and oracle networks to manage the rules of storage contracts and implement the proposed solution using the Solidity language and the Chainlink oracle network. The experimental results demonstrate the applicability of the proposed method.

F. Discussion

The core of this paper lies in integrating game theory with the design of decentralized networks. However, the game design proposed in the paper may be overly idealized. In reality, malicious storage service providers may stand to gain more benefits than the portion they are penalized by smart contracts, potentially leading to a deviation from the ideal game equilibrium described in the paper. Moreover, considering the actual deployment costs on the blockchain, this paper was implemented on the Ethereum test network. Therefore, the metrics that can be evaluated during the experimental process are very limited and may not fully reflect real-world conditions.

IV. Blockchain-Based File Replication for Data Availability of IPFS Consumers

A. Introduction

Although blockchain technology offers security and transparency, it faces limitations such as on-chain storage capacity. To address the storage limitations of blockchain, the InterPlanetary File System (IPFS) is often used as a storage layer. However, IPFS lacks inherent mechanisms to ensure data availability, making it susceptible to node maintenance or failures. Ensuring long-term data availability is crucial for various applications, such as non-fungible tokens (NFTs) that require long-term storage value. Replicating data to other nodes is a practical and simple method to protect data from hardware failures and improve data availability. Keeping backup copies of data not only enhances data availability but also increases retrieval efficiency.

Previously used replication methods in peer-to-peer (P2P) networks can also be employed to enhance data availability in the IPFS network. However, previous replication methods have some limitations. These replication methods can be categorized into two types: clusters and replication contracts. Clusters, for example, divide the replication system into clusters where each cluster is responsible for local data storage and maintenance, optimizing storage use. However, clusters lack flexibility and cannot adapt to uneven distribution of peer availability or storage capacity, nor to dynamic changes in P2P network systems. In replication contracts, for example, peer nodes find replication partners within the system to establish replication contracts and mutually replicate data. Replication contracts are vulnerable to selfish behaviors of other peer nodes, leading to increased data availability for highly available peer nodes and decreased data availability for less available peer nodes.

To optimize overall data availability while ensuring flexibility, this paper introduces a blockchain-based file replication mechanism. By utilizing blockchain to record peer node information used during the file replication process, the proposed mechanism ensures authenticity and reliability. Unlike previous methods, the mechanism proposed in this paper adopts a file replication algorithm inspired by Arweave. This algorithm autonomously replicates files from other peer nodes according to predefined rules, prioritizing files with lower availability. By applying system-wide rules to restrict selfish behavior of peer nodes, data availability for all peer nodes is balanced and optimized. Additionally, the file replication mechanism includes a smart contract for detecting and excluding dishonest peer nodes, promoting honest cooperation among peer nodes. The smart contract allows peer nodes to autonomously detect and exclude dishonest peer nodes without involving any third party, enhancing the decentralization of the replication system and the autonomy among peer nodes, while reducing operational costs.

B. Overview

The mechanism proposed in this paper employs a replication algorithm that prioritizes files with lower availability in the system. Replication continues until the availability of all files is optimized. Consequently, the proposed mechanism can effectively optimize overall data availability in the replication system. This mechanism uses blockchain to store peer node information used by the file replication algorithm. The immutability of the blockchain ensures the authenticity and reliability of this information. This information includes the availability of peer nodes, content identifiers (CIDs) and sizes of critical files, the amount of shared storage space, and the IDs of IPFS peer nodes. For each file, there is a pool of replicators on the blockchain, with the file owner being the first peer node added to this pool.

The proposed mechanism in this paper requires peer nodes to consume storage resources proportionate to their contri-

butions. If the ratio of the shared storage space size to the total size of its critical files does not reach the predetermined value *P*, the smart contract on the blockchain will reject the input. (For a peer node, the value of *P* is inversely proportional to its availability; the greater the file availability, the smaller *P* becomes, indicating that the file no longer needs to be replicated.) However, if a peer node secretly deletes copies and is subsequently detected, it will be accused of fraud. Therefore, the system should have a secure monitoring protocol to monitor the availability of peer nodes, enabling peer nodes to check the availability of other peer nodes. A peer node accused of fraud will be judged by the smart contract, and if found guilty, will be excluded from the file replication system by the smart contract.

The figure below illustrates the basic concept of the proposed mechanism. The file replication system consists of nodes or peer nodes, representing independent IPFS users. Peer nodes allocate storage space for their critical files and provide additional storage space for sharing with other peer nodes. Each peer node aims to increase the availability of its critical files. Unlike previously proposed methods, the mechanism proposed by the authors adopts a replication algorithm that prioritizes files with lower availability in the file replication system. File replication continues until the availability of all files is optimized. Therefore, the authors' mechanism can effectively achieve overall data availability optimization within the file replication system. The authors' mechanism utilizes blockchain to store peer node information used by the file replication algorithm.

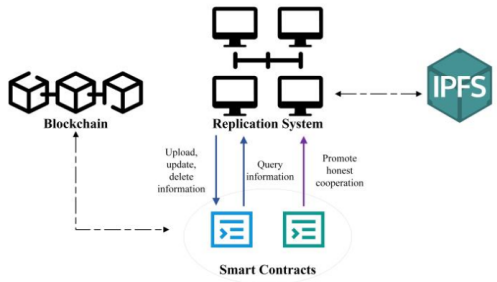


Fig. 1. The file replication mechanism based on the blockchain.

Fig. 13: Image source from [2]

C. Arweave-Inspired File Replication

The file replication algorithm proposed in this paper is inspired by Arweave's game-theoretic design for permanent storage. Arweave is a stable, mature, and widely applied economically sustainable protocol for permanent information storage. Arweave achieves this grand objective through the clever combination of blockweave and Succinct Proofs of Random Access (SPoRA). In the blockweave, each block is linked to two previous blocks: the preceding block in the blockchain and a block from earlier in the blockchain history, known as the "callback block." For a miner to successfully mine a new block and receive the corresponding token reward, they must first verify the callback block. The only way to do this is by storing it. SPoRA incentivizes miners to store relatively "rare" blocks rather than easily reproducible blocks since fewer miners can compete for the same level of reward when choosing rare blocks.

In the file replication algorithm, the authors define the availability $A(f)$ of a file f as the probability that at least one of its replicators (including the owner) is available. $A(f)$ can be calculated based on the availability of its replicators $\{r_1, r_2, ..., r_j\}$ using the mathematical expression below. When the replicas of a peer node are fewer than its shared storage capacity, it periodically replicates the file with the lowest availability. This replication increases the file's availability. Clearly, the more a file is replicated, the higher its availability. However, this also increases storage overhead costs. Considering the storage capacity constraints among peer nodes and the demand for file availability, the proposed algorithm sets a satisfactory availability threshold T_A. If the file availability $A(f) > T_A$, then f will not be included in any peer node's replication list and will not be replicated by any peer node.

Below is the pseudo-code for the file replication algorithm. When p is large, it indicates that the file availability is insufficient and other peer nodes need to help replicate the file. The value of p is inversely proportional to file availability; the greater the file availability, the smaller p becomes, indicating more peer nodes have a copy of the file. In the loop below, when the sequence of files to be replicated is empty, it selects files from neighboring peer nodes to add to the sequence. Next, when the file replication sequence is not empty, it first sorts the files in the sequence by availability. If the availability $A(f)$ of a specific file is less than the set threshold T_A, the peer nodes in the system will begin to assist in creating replicas of the file. Once the file replica is successfully replicated, it is added to the replica pool.

Algorithm 1 Pesudocode for the File Replication Algorithm

```
p.replicate ( ){
//periodically executed by a peer
while p has less replicas than its shared storage capacity do
    while the list is null do
        Select neighbors' files to add to the list
    end while
    while the list is not null do
        //replicates the file
        updates the list
        orders files in the list in ascending order of A(f)
        for each file from the top of the list do
            if the file is not replicated A(f) < T_A then
                replicates the file
                if the replication is successful then
                    joins the file's pool of replicators
                    break
                end if
            end if
        end for
    end while
end while
}
```

Fig. 14: Image source from [2]

D. Dishonest Peer Judgement

In the file replication system, it is possible to encounter dishonest peer nodes. These peer nodes might engage in dishonest behaviors, such as secretly discarding file replicas they hold (as storing these files consumes their resources) or providing false information and intentionally lowering file availability. In this paper, the authors first use game theory to analyze this situation and then introduce the proposed mechanism for effectively identifying and excluding dishonest peer nodes using smart contracts.

Game Theory Analysis: Each peer node in the file replication system has two choices: a peer node can choose to cooperate (C), such as by storing files from other peer nodes to help them increase data availability, or defect (D), such as by secretly discarding files from other peer nodes to save their own resources.

The small diagram below represents the payoff matrix in the game between two peer nodes. We can see that in the short-term equilibrium, the outcome falls at (D, D), where both peer nodes have a payoff of 1. However, in the long-term equilibrium, the outcome shifts to (C, C), where both peer nodes have a payoff of 2.

短期均衡和長期均衡

	C	D
C	2, 2	0, 3
D	3, 0	1, 1

Fig. 2. The game for the case of two peers.

Fig. 15: Image source from [2]

Game Theory Analysis: Overall, engaging in dishonest behavior might yield short-term gains. If peer node a finds that one of its replica peer nodes b is behaving dishonestly, such as discarding a's file, a will not only discard b's file (if a is storing b's file) but also notify other peer nodes about b's dishonesty, potentially leading to b's exclusion from the file replication system.

Detection and Exclusion of Dishonest Peer Nodes: Although rational peer nodes should cooperate honestly, it is still necessary for peer nodes to periodically check the fidelity of their replicas. A peer node can reasonably check its replicas every few hours or even daily. IPFS provides the functionality to find file providers based on the file's CID using a Kademlia-based DHT. This mechanism can be used as a means to verify file storage status. Therefore, distinguishing between peer nodes attempting to cheat and those that are merely offline or experiencing temporary failures is crucial. Peer nodes should not be marked as dishonest due to temporary disconnections or failures.

The authors suggest that if the time between two failed checks exceeds a predetermined time T, the peer node can be considered dishonest. To maintain accountability and transparency, the results of each check should be recorded on the blockchain. The proposed mechanism primarily uses smart contracts to effectively identify and exclude dishonest peer nodes, avoiding the involvement of any third-party entities. It is important to note that only potential victims have the right to initiate a vote to determine a peer node's honesty, and only other potential victims of its dishonest behavior can participate in the voting process.

The following Algorithm 2 is designed for peer node eligibility screening. The algorithm uses the peer node's ID and address, processes them through a hash function to identify potential victim peer nodes, and determines who is eligible to initiate a distrust vote against a specific peer node, as well as which peer nodes have the right to vote in the distrust voting process.

Algorithm 2 Peer Authentication

Input: IPFS Peer ID of the peer ID_p, Address of the peer Add_p, Dishonesty of the accused peer D_A
Output: Hash value of the peer's information h_p

1: $h_p == generateHash(ID_p, Add_p)$
2: **if** $h_p == Hashmatch(h_p, D_A)$ **then**
3: return true
4: **else**
5: return false
6: **end if**

Fig. 16: Image source from [2]

Since some peer nodes in the system may be eligible to vote but do not participate in the voting process, the smart contract does not require all peer nodes to vote. Instead, if more than half of the voters vote "dishonest" or if half of the voters vote "honest," the final result can be determined. This improves the efficiency of the voting process. The entire process of identifying and excluding dishonest peer nodes is illustrated in the diagram below.

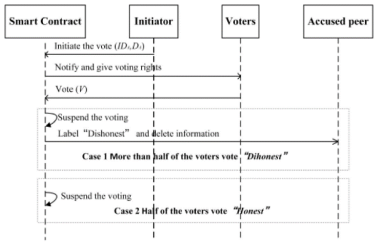


Fig. 3. The dishonest peer judgment and exclusion sequence diagram.

Fig. 17: Image source from [2]

Below is the pseudo-code for initiating a distrust vote and the exclusion process on the smart contract for a specific node. The entire process can be divided into three steps:

1) The initiator of the vote sends the IPFS Peer ID of the node accused of dishonesty and the nature of its dishonesty D_A to the smart contract. D_A can have three possible values: "DR" indicating secret deletion of file copies, "IA" indicating providing inaccurate availability or deliberately lowering availability, and "IS" indicating providing inaccurate file size. Upon completion of the

authentication procedure, the smart contract will initiate the vote, grant voting rights to the voters, and create notifications to inform the voters. Importantly, the vote initiation process cannot be duplicated. If the initiation fails, a notification will be created to inform the initiator. This ensures transparency and feedback to the user when an initiation attempt fails.

2) After the authentication procedure is completed, voters cast their votes (”honest” or ”dishonest”). Each voter is allowed only one vote, and multiple voting is prohibited. Once a voter casts their vote, their voting rights are revoked to prevent multiple voting attempts. If a voter tries to vote multiple times, their request will be denied. This ensures the integrity of the voting process.
3) If more than half of the voters vote ”dishonest,” the smart contract will suspend the voting, mark the accused node as a dishonest peer node, delete its information, and broadcast its dishonest behavior to all nodes in the system. This effectively excludes the node from the file replication system. If half of the voters vote ”honest,” the smart contract will also suspend the voting, clearing the accused node's name.

Algorithm 3 Pseudo-Code of the Smart Contract for Dishonest Peer Judgment and Exclusion

Input: IPFS Peer ID of the accused peer ID_A, Dishonesty of the accused peer D_A, Set of votes V, Count of votes for "*Honest*"C_H, Count of votes for "*Dishonest*"C_D
Output: Honesty of the accused peer H_A (The default value is "*Honest*".)

the vote initiator sends ID_A and D_A to the smart contract
if the vote was not initiated **then**
  Authentication = Peer Authentication (ID_p, Add_p, D_A)
  if (D_A == "*DR*" || D_A == "*IA*") **Authentication=** true **then**
    initiates the vote
    $C_H = 0, C_D = 0$
    gives voting rights to owners of the accused peer's replicas
    $++C_D$
    creates notifications to notify the voters that the vote has been initiated
  else if D_A == "*IS*" **Authentication=** true **then**
    initiates the vote
    $C_H = 0, C_D = 0$
    gives voting rights to replicators of the accused peer's files
    $++C_D$
    creates notifications to notify the voters that the vote has been initiated
  else
    creates a notification to notify the vote initiator that the initiation has failed
  end if
else
  creates a notification to notify the vote initiator that the voting is in progress
end if
voters cast their votes, $V = v_1, v_2, \ldots, v_i$
Authentication = Peer Authentication (ID_p, Add_p, D_A)
if voting is in progress the voter has the right to vote **Authentication=** true **then**
  if $\mathbf{v_i}$=="**Dishonest**" **then**
    $\mathbf{++C_D}$
    deprives the voter of the right to vote
    else if v_i == "*Honest*" **then**
    $\mathbf{++C_H}$
    deprives the voter of the right to vote
  end if
end if
if $\mathbf{C_D > i/2}$ **then**
  suspends the voting
  H_A = "*Dishonest*"
  deletes the information of the accused peer
  creates notifications to notify the peers that the accused peer is dishonest
  else if $C_H \geq i/2$ **then**
  suspends the voting
end if

Fig. 18: Image source from [2]

E. Performance Evaluation

The file replication algorithms used for comparison in the experiment:

- Replication Contracts: Optimistic Query, Practical Query
- Replication Contracts & Clustering: Explicit Click

To comprehensively evaluate the proposed algorithm, the authors implemented these algorithms on the cycle-driven P2P simulator PeerSim. The aim was to assess the algorithm's performance under different levels of churn in a large-scale P2P network. The authors began the simulation of the P2P file replication system with a basic assumption: the departure, arrival, disconnection, and reconnection of all peer nodes are completely independent. The P2P network comprised 10,000 peer nodes. For simplicity, each peer node had a single important file to replicate, and all these files were of equal size. Each experiment lasted for 9,000 cycles.

In the experiment, each peer node was associated with a specific configuration profile. Subsequently, Gaussian noise with σ=0.1 was added to each node's availability. Finally, file availability was restricted to the range of 3% to 97%. The table below shows the distribution of peer nodes in different states, and nodes in different states had different availabilities.

TABLE I
PEER PROFILES

Profile	Proportion	Availability
Durable	10%	95%
Stable	25%	87%
Unstable	30%	75%
Erratic	35%	33%

Fig. 19: Image source from [2]

Based on the distribution of peer node states in the previous table, we can represent the distribution of generated peer nodes using the histogram below. The availability of each peer node remains constant throughout the simulation process.

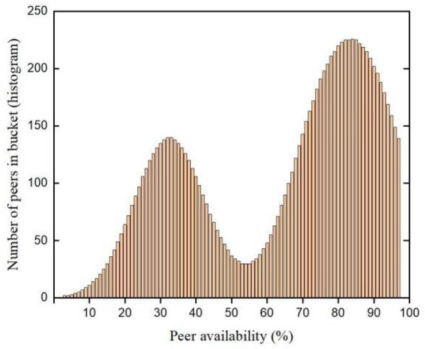
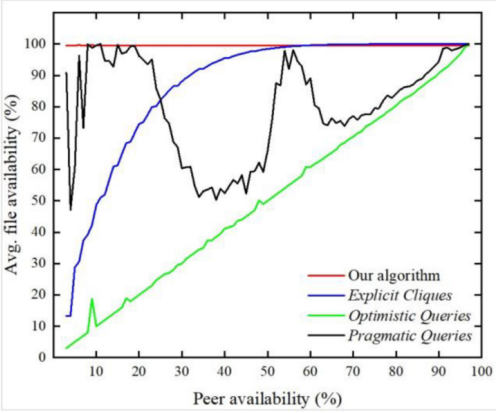


Fig. 4. Histogram shows the number of peers in each availability bucket.

Fig. 20: Image source from [2]

The figure below, part (a), shows the impact of different peer node availabilities on the average file availability. Clearly, Explicit Pinning generates a diverse spectrum of file availabilities: files of highly available peer nodes have an availability above 99%, while files of peer nodes with availability below approximately 50% have an availability below

99%. As the peer node availability decreases, file availability also decreases, and the least available peer nodes are almost unaffected by replication, resulting in file availabilities below 15%. In Optimistic Querying, only the files of the most available peer nodes have an availability above 99%, while the file availabilities of other peer nodes remain almost unchanged due to the difficulty in finding partners to help replicate files. The results of Practical Querying show that all peer nodes have a file availability above 45%, and some peer nodes, including both highly available and less available nodes, have a file availability above 99%. Except for the authors' algorithm, none of the methods can ensure optimization of overall data availability. In contrast, the authors' algorithm ensures that the file availability of all peer nodes is above 99%. Experimental results demonstrate that the authors' algorithm can achieve optimization of overall data availability.



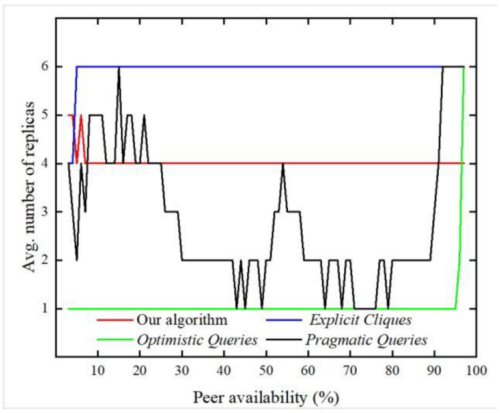
(a) Average file availability

Fig. 21: Image source from [2]

Part (b) of the figure describes the relationship between the average number of replicas for all methods and peer node availability. We can observe that, in Optimistic Querying, there are essentially no replicas stored on peer nodes other than the most available ones. Practical Querying exhibits a trend similar to part (a) of the figure, as highly available peer nodes have more replication partners, leading to higher file availabilities. Additionally, we can observe that both Explicit Pinning and the authors' algorithm demonstrate stability. In the authors' algorithm, even without any storage capacity limitations on peer nodes, the average number of replicas for most peer nodes remains around 4. In contrast, in Explicit Pinning, the average number of replicas for most peer nodes remains around 6. These results confirm that the authors' algorithm produces fewer file replicas and incurs lower storage costs.

The figure below shows the effectiveness of alternative versions of the authors' proposed algorithm under different thresholds T_A values (7, 8, and 9). In all cases, all 10,000 files successfully achieved satisfactory availability. To simplify the presentation of experimental results, the authors chose to display the results for 100 files here. The results also indicate that these alternative versions of the proposed algorithm can optimize the overall data availability within the system.

The figure below illustrates the relationship between the average availability of all files in the system and the join/leave



(b) Average number of replicas

Fig. 22: Image source from [2]

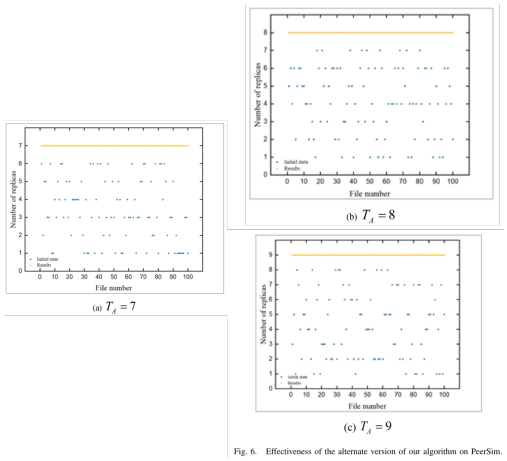


Fig. 23: Image source from [2]

rate of peer nodes. Except for the algorithm proposed by the authors, the performance of other algorithms drops sharply with higher node churn rates, as departing nodes might possess file replicas. Due to the optimal progress advantage of the authors' proposed algorithm, which maximizes overall data availability at each step of the file replication process, its performance is less affected by node churn. The experimental results also demonstrate the strong robustness of the proposed algorithm against node churn.

F. Conclusion

This paper proposes a blockchain-based file replication mechanism where IPFS users can mutually assist in preventing hardware failures and improving data availability. The proposed mechanism includes an Arweave-inspired file replication algorithm and smart contracts for information collection and updates, as well as for detecting and excluding dishonest peers. This mechanism uses blockchain to record peer information used by the file replication algorithm to ensure authenticity and credibility. The mechanism prioritizes the replication of files with lower availability until all files in the replication system, composed of IPFS users, achieve satisfactory availability. Furthermore, the mechanism can dynamically adapt to changes in the replication system by

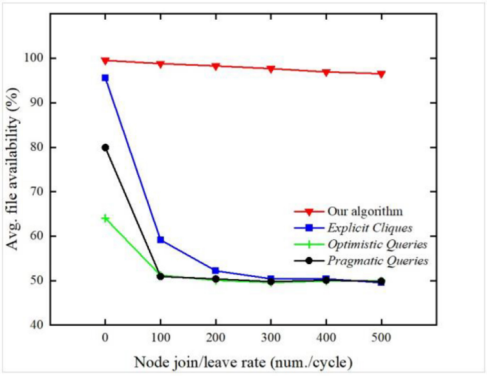


Fig. 7. Performance of the algorithms with churn in Peersim.

Fig. 24: Image source from [2]

utilizing blockchain information and the independence of file replication by each peer. The smart contract used to detect and exclude dishonest peers allows peers to fairly judge accused dishonest peers and effectively exclude them from the file replication system, promoting honest cooperation among peers without involving any third party.

The proposed mechanism in this paper achieves optimization of overall data availability and provides flexibility. Experimental results demonstrate the superiority of this mechanism compared to other methods. It achieves significant improvements in data availability across the entire replication system, reduces storage overhead, and shows excellent robustness. Its low overhead and superior performance make it particularly suitable for deployment in P2P data networks like IPFS. Additionally, the smart contracts are compiled and deployed using the Truffle framework. The feasibility of the smart contracts has been thoroughly tested and verified.

G. Discussion

In practical applications, as the number of peer nodes in the network increases, the scalability of the proposed mechanism in this paper may face challenges. Utilizing blockchain for information recording and smart contracts can lead to scalability issues, especially in large networks. The increase in the number of transactions and the size of the blockchain may affect the overall performance and efficiency of the system. Implementing smart contracts on the blockchain requires computational resources. Depending on the chosen blockchain platform (e.g., Ethereum), the cost of executing smart contracts and storing data on the blockchain can be high. This could pose challenges in resource-constrained environments and potentially limit the feasibility of the proposed mechanism.

V. Towards Practical Auditing of Dynamic Data in Decentralized Storage

A. Introduction

The openness requirement of Distributed Storage (DS) demands convincing technical means to ensure the integrity of users' remotely stored data. To date, nearly every relevant application of distributed storage has an inherent data auditing mechanism built into its design core, where an auditor periodically verifies random portions (chunks) of outsourced data from storage nodes and rewards or penalizes nodes based on the audit results. However, delegating the critical function of data storage auditing to any single entity violates the principle of decentralization. Most applications related to distributed storage rely on a common blockchain as a fair public auditor. On the blockchain, it can verify storage proofs submitted by nodes and fairly resolve potential payment disputes between parties. This architecture, known as on-chain auditing, requires storage proofs and auditor states to occupy as little storage resources as possible. Classic storage proofs (PoS) schemes offer a promising solution as they can generate highly compact proofs, and recording auditor states requires minimal storage space. However, this approach only guarantees storage proofs for static data.

In academia, a series of dynamic storage proof methods have been proposed in the past, applicable to traditional cloud storage. However, some of them are inherently unsuitable for the on-chain auditing framework due to security reasons. For the context of distributed storage, potential adaptive dynamic storage proof schemes can be classified into two main categories:

- The first category designs combine homomorphic tags and Authentication Data Structures (ADS) to protect outsourced data while allowing efficient updates at sublinear costs. Their main drawback is the larger storage space required for storage proofs.
- The second category can generate shorter proofs compared to classic static storage proof methods. However, they require auditors to maintain dynamic states with high update costs, which is too expensive for on-chain auditing.

In summary, the actual auditing of dynamic data in the context of distributed storage remains largely unexplored. In this paper, the authors explore the design space of dynamic storage proofs and observe that technical details revolve around the handling of index information. Through appropriate index transformation mechanisms, static storage proofs can be converted into dynamic storage proofs while retaining the compact storage proof of the former. This mechanism introduces an unusual index state to all parties in the auditing protocol, with the key technical challenge being how to manage this index state so that it can be practically maintained on a resource-sensitive public blockchain, where every byte is crucial.

B. Overview

A distributed storage system is constructed from multiple layers. Architecturally, the data auditing layer sits between the underlying storage layer and the upper incentive layer, gluing them together. It interacts with the underlying storage system and provides necessary support, namely, furnishing evidence for the integrity of outsourced (uploaded) data and ensuring the correct operation of blockchain incentive mechanisms. In the authors' design, storage nodes are audited individually, meaning the owners of original data will sign separate contracts with different storage nodes. This is reasonable since nodes in

decentralized networks typically do not work in a coordinated manner, and different nodes vary in capacity and service quality commitments. The authors utilize a public blockchain as the decentralized infrastructure and do not consider designing consensus protocols based on storage proofs, similar to some existing proposed methods.

The on-chain auditing functionality of the blockchain is realized through a smart contract called the auditing contract. It encompasses functionalities for storage contract setup, data auditing, updates, and more. The auditing contract should define interfaces to handle disputes between off-chain parties and interact with the incentive mechanisms of distributed storage, which can be implemented in other smart contracts. In the proposed architecture, the owner of original data sets up the auditing protocol locally. This private setup is implicitly considered in the literature related to storage proofs. The authors emphasize this distinction as it leads to different security implications, especially when one seeks integrity guarantees for multiple replicas of the same data. In a public setup, storage nodes can deceive the owner of original data by storing less data than required. In contrast, in a private setup, the owner of original data can encrypt data replicas with different keys, making them independent of each other, to address the aforementioned issue.

The system model proposed by the authors comprises three entities: the owner of original data for local data storage, storage nodes assisting in creating file storage replicas, and the on-chain auditing smart contract overseeing both parties. The data auditing layer is built on top of the underlying storage layer, serving the upper incentive layer in the distributed storage framework. The arrows in the image below denote the primary flow of data, while the dashed lines represent the normal data retrieval process.

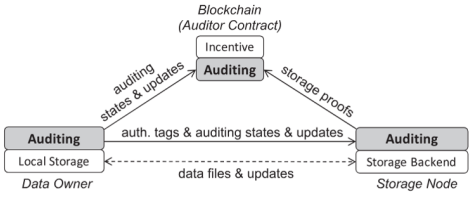


Fig. 1. Our system model with three parties. The data auditing layer builds on the underlying storage layer and serves for the upper incentive layer in the DS architecture. The arrows indicate main data flows. The dashed line indicates normal data retrieval process.

Fig. 25: Image source from [3]

The authors assume that the blockchain platform is trustworthy and that the smart contracts used for auditing are correctly implemented. They do not consider attacks against the blockchain itself. However, they assume mutual distrust between data owners and storage nodes. To comply with the principles of the open economy in distributed storage, the authors also consider malicious data owners. They may falsely accuse storage nodes to potentially obtain compensation enforced by smart contracts. Finally, the authors assume that the communication channels between parties are authenticated. Previous research has shown that in the context of public storage auditing, third-party auditors can recover original blocks by repeatedly challenging storage proofs. Whether this poses a serious problem in the case of distributed storage is debatable, as the stored data is almost always encrypted by default. In the proposed design, the authors can conveniently instantiate existing structures to make the underlying storage proofs zero-knowledge, thereby preventing malicious recovery of on-chain data.

C. On-Chain Auditing of Dynamic Data

The authors aim to address the issue of data dynamism while maintaining the efficiency of storage proof methods. Their key idea is to bind chunks with alternative index information that is insensitive to data dynamism. Ideally, an update to a chunk should only affect its own tag. They refer to this index as a pseudo-index, denoted by p_i, corresponding to chunk f_i. For their proposed design to work correctly, there must be a one-to-one mapping between logical indices and pseudo-indices. This mapping should be stored in some data structure, introducing an unconventional state that must be maintained by all parties involved in the storage proof method. It allows each party to access necessary index information for tag calculation, proof generation and verification, and update operations. This approach can transform any static storage proof method into a dynamic one. As long as all pseudo-indices used at any given time are different, and the (public) auditors always maintain the correct state, the security guarantees for data integrity or retrievability remain unchanged.

The index mapping is stored in an explicit form (index i and pseudo-index p_i), and regardless of the data structure used, the time complexity for insert and delete operations is always $O(n)$. The authors found that index mapping can be elegantly encoded through linked lists. Assuming each node of the linked list stores a data chunk in the correct order, a chunk's logical index is its position in the linked list, and the pseudo-index can be set to the address of its node. During updates, the index mapping is automatically maintained by the linked list. Since the linked list does not need to store actual chunks, we can use a regular array that only stores pointers (and other small metadata) to implement it. The authors refer to this result as the Pseudo-Index Linked List (PIL), a concise data structure.

Over time, the same pseudo-index can be assigned to different chunks, creating the possibility of a relay attack, where storage nodes use outdated, invalid chunks matching the index to forge proofs. To ensure freshness, the authors associate the latest update time of each chunk and store it as part of the state, denoted by IS. It's noteworthy that auditors can directly sample chunks using their pseudo-indices (or equivalently using array indices of IS). Storage nodes can map pseudo-indices in the challenge back to logical indices and generate proofs as usual. Since auditors no longer need to perform index conversion, they only need to maintain a timestamp array, A_ac, consistent with IS from the other two parties. Auditors can easily update A_ac using pseudo (array) indices and timestamps sent from the client.

In On-Chain Auditing (OAD), the authors manage the index state using a single PIL. Now, they attempt to add a

separate PIL for each segment. A segment manages an index mapping for a series of contiguous logical chunk indices, which are naturally ordered. A global logical chunk index can be uniquely represented in the form of segment indices and local logical chunk indices within the segment. This design produces a two-tier structure: one PIL for segment indices and another set of PILs for chunk indices. Segments lead to two independent states: one containing IS+ for chunk index mapping and another for managing segment information, denoted as IS-. Thus, the authors propose a second on-chain auditing protocol, OAD+. The main difference between OAD and OAD+ lies in how they handle index information. Here, the Audit Contract (AC) relies on verification chunk index information sent from the Storage Node (SN) to verify the validity of storage proofs.

D. Data Abstraction for Auditing

Frequent data relocations can lead to significant fragmentation issues, resulting in many underutilized chunks and performance impacts. An ideal design should limit changes to chunks within themselves and maintain a balanced and highly utilized state for all chunks. Despite these complex factors, the actual content required for auditing is very straightforward: data should be viewed as a series of ordered chunks. How data is organized and stored by the underlying storage system is irrelevant to the auditing protocol. By providing an appropriate data abstraction layer, the authors can decouple auditing from data storage, allowing the proposed auditing protocol to be deployed on any storage system. Another benefit is that the authors found in practice, this small data abstraction layer can exist entirely in main memory. This makes it easier to accurately analyze performance optimizations for both the auditing protocol and the storage system.

From the diagram below, it can be seen that through the Data Abstraction Layer proposed by the authors, the two auditing protocols presented in this paper can be applied to various different data storage devices or systems.

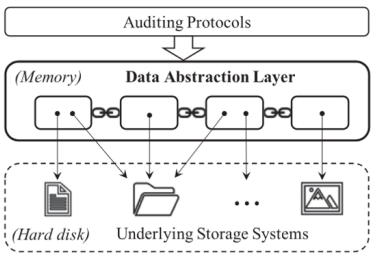


Fig. 9. Illustration of the data abstraction layer that hides the low-level details of data storages from auditing protocols.

Fig. 26: Image source from [3]

The core of this data abstraction design is the Lightweight Audit Chunk (LAC) proposed by the authors. LACs store virtual data segments. The capacity of an LAC is the maximum virtual data it can accommodate, which equals the sum of the lengths of all the triplets it currently stores. An LAC is not necessarily full, meaning its size can be smaller than or equal to its capacity. An object can span multiple LACs and is always allocated to the minimum necessary number of contiguous LACs. This effectively reduces data fragmentation and management overhead since the total number of triplets is minimized. LACs can be freely modified, inserted, or deleted to meet the requirements of the auditing protocol. Updating an LAC involves operating on metadata triplets but not actual data, making it highly efficient.

E. Evaluation

The authors implemented their protocols in C++ for evaluation. They relied on the mcl library to implement the main cryptographic operations. For 128-bit security, they used the BN12-381 curve to implement the storage proof method. The same curve was also used to implement the BLS signature protocol OAD+. They also constructed a data abstraction layer and integrated it into the native Linux file system to understand its impact on performance. The authors used a test platform equipped with an Intel E-2174 CPU (3.8GHz, 8 cores) and 64GB RAM. It had a capacity of 4TB and a 7200 RPM HDD. All experiments were conducted on a single core. Unless otherwise specified, each data point in the charts represents the average of 100 independent runs. For I/O-related experiments, the authors always cleared the page cache of the Linux system to ensure that the operating system did not cache files in memory, thus presenting realistic experimental results.

The table below shows the specific storage proof sizes under the same 128-bit security and for a 1TB file. The storage proof size of ICPOR is equal to the size of one chunk plus one tag. The storage proof size of DPDP is close to the size of one chunk plus c tags and c verification data structures (ADS) of length log(n). From the table, it can be seen that while ICPOR may be suitable for on-chain auditing with small chunks, using DPDP (and general ADS-based designs) is impractical due to their consistently large storage proof sizes. In contrast, the storage proofs generated by OAD and OAD+ are much smaller and can support real-world blockchain platforms well.

TABLE 1
Proof Size (in KB) for 1TB Data

Chunk (KB)	2	4	8	16	32	64	128
OAD				0.13			
OAD$^+$				1.2			
ICPOR	2.1	4.1	8.1	16.1	32.1	64.1	128.1
DPDP	166	164	164	168	180	208	268

Fig. 27: Image source from [3]

The graph below illustrates the size of audit states corresponding to file sizes ranging from 1GB to 1TB. From the results, it can be seen that the proposed auditing protocols can identify an optimal chunk size that minimizes the state size for a given file size. For a 1TB file with a chunk size of 256KB, OAD+ produces a state size of 0.78MB with m = 100; for a chunk size of 64KB, it produces a state size of 0.25MB, with m = 1000. It is noteworthy that such small state sizes can even rival the proof sizes of ADS-based methods. This represents a significant improvement over designs requiring O(n) state sizes. (Here, the parameter m in OAD+ is used to adjust the size of the IS- index state.)

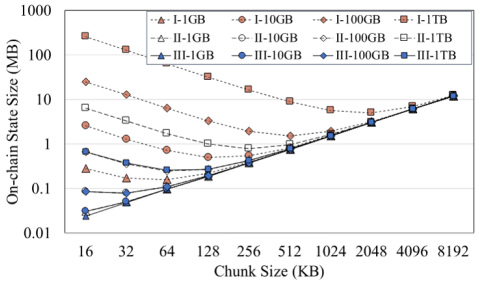


Fig. 10. Overall on-chain state size versus varying file size and chunk size. In the legends, 'I' = OAD, 'II' = OAD$^+$ ($m = 100$), 'III' = OAD$^+$ ($m = 1000$).

Fig. 28: Image source from [3]

The graph below illustrates the generation time of storage proofs for 1TB of data during a single audit process. Compared to static auditing protocols, dynamic auditing protocols require linear scanning of the state to find the correct index, while OAD+ also requires aggregation of the signatures of the index bytes. As observed in the graph, the cost is evident due to the substantial linear scanning work required for small chunk sizes. However, as the chunk size increases, this cost diminishes rapidly. Starting from a chunk size of 64KB, the performance of dynamic auditing protocols approaches that of static auditing protocols.

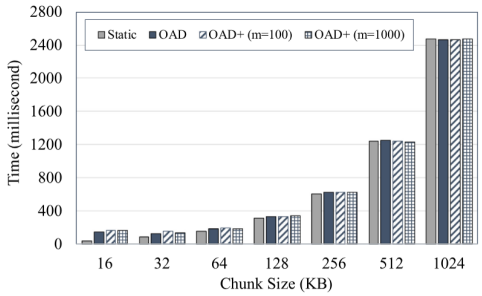


Fig. 11. Proof generation time at the storage node on 1TB data. "Static" indicates the underlying PoS (Fig. 8) used to instantiate our protocols.

Fig. 29: Image source from [3]

The graph below illustrates the local data update cost for data owners. Here, the authors only consider insertion operations as they are the most expensive among the three update operations. In addition to computing the new tag, the auditing protocol also requires finding the corresponding PIL of the index state and locating the correct position for insertion. From the results of the implementation evaluation, it can be seen that the process of finding the corresponding PIL of the index state dominates the local data update cost in OAD, but it is not the primary reason for the high local data update cost in OAD+.

The graph below illustrates the setup time of the auditing protocol for data owners. The cost of OAD should essentially be the same as that of the static auditing protocol structure because its index state initialization can be negligible. OAD+, on the other hand, requires additional costs to sign the index bytes, but its overall setup cost is still dominated by the underlying storage proof.

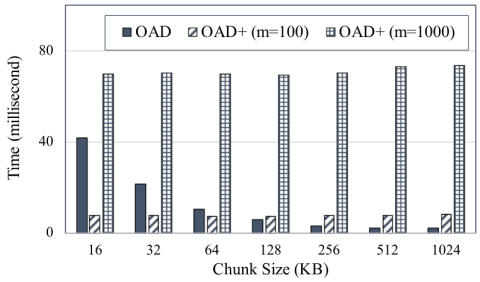


Fig. 12. Update cost at the data owner for inserting one chunk for a 1TB dataset. The position of insertion is $n/2$, representing the average case cost.

Fig. 30: Image source from [3]

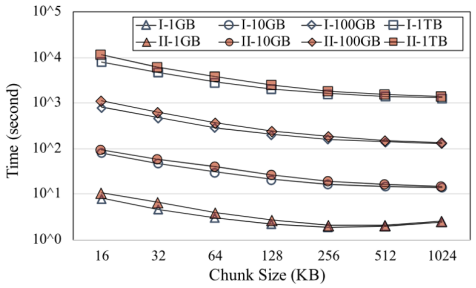


Fig. 13. The setup cost for our auditing protocols at the data owner. 'I' = the underlying PoS (Fig. 8) and 'II' = OAD$^+$ ($m = 100$, the value of m is insignificant).

Fig. 31: Image source from [3]

The table below provides an overview of the datasets used in the authors' experiments. Four of them are generated using FileBench, following a gamma distribution, with average file sizes of 1MB, 2MB, 5MB, and 10MB. The last one is obtained from a GitHub clone of the Linux codebase, with the .git folder removed.

TABLE 2
Summary of File System Datasets

Dataset	FB-1	FB-10	FB-100	FB-1TB	Linux
Size	1GB	10GB	100GB	1TB	1.02GB
Num. files	968	5531	21426	103356	66458
Num. dirs	27	63	270	591	4,390
Dir. depth	2	4	7	10	9

Fig. 32: Image source from [3]

In the figure below, the authors evaluate the construction time of the data abstraction layer for different chunk sizes. The results indicate that even for the smallest chunk size that would result in a large number of chunks, setting up a 1TB dataset only takes 10 seconds. Such low cost is attributed to the fact that building the data abstraction layer only requires reading the meta data of the files, not their actual contents.

In the figure below, the authors illustrate the loading time of random chunks. In theory, the I/O cost should be proportional to the chunk size, which is generally evident in the graph. However, for smaller chunks, the results on two larger datasets show the opposite direction. This is attributed to the sequential scanning of chunks to access them. We can enhance lookup

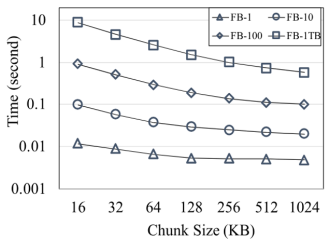


Fig. 14. Construction time for the data abstraction layer with varying chunk sizes over four datasets.

Fig. 33: Image source from [3]

performance by storing pointers to chunks in data structures that support fast searches.

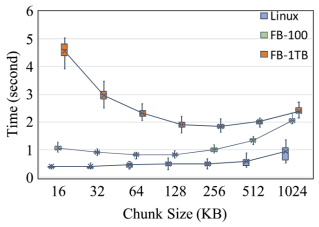


Fig. 15. Time to load $c = 128$ challenged chunks into memory for proof generation in an audit.

Fig. 34: Image source from [3]

Additionally, the authors measured the cost of writing updated files on storage nodes, which includes updating the relevant triplets within chunks. The estimated cost corresponds to the time it takes to write the file to the disk only. They considered file sizes ranging from 1KB to 1GB, and for each size, they randomly sampled files from the FB-1TB dataset. The results in the figure below indicate that this cost is negligible for all file sizes and chunk sizes, as the overhead is less than 1% in all cases.

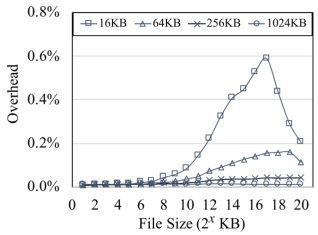


Fig. 16. Worst-case overhead of random file writes on the FB-1TB dataset at the storage node, with four chunk sizes.

Fig. 35: Image source from [3]

F. Conclusion

The emerging decentralized storage paradigm demonstrates tremendous potential. To incentivize cautious users, such systems must offer robust data integrity guarantees while supporting full data dynamism to enhance availability, an area that has been relatively underexplored. This paper delves deep into the shortcomings of existing data auditing solutions in meeting these new paradigms. To address the gaps in the research field, the authors have developed a series of on-chain auditing protocols optimized for dynamic data and demonstrated their practical efficiency. With the trend towards decentralized storage with enforceable auditability continuing to gain momentum, the authors hope their research findings will inspire further exploration into the availability aspects of this enticing technology.

G. Discussion

This paper's primary drawback lies in the challenge of managing index states in resource-sensitive public blockchains, where every byte is crucial. The proposed mechanisms face challenges in achieving efficiency in off-chain updates between data owners (OW) and storage nodes (SN). Additionally, the proposed methods require auditors to maintain dynamic states with complexity O(n) and high update costs, which is too expensive for on-chain auditing. This limits their application in real decentralized storage networks with growing user numbers and data volumes. Furthermore, the use of authenticated data structures (ADS) in dynamic storage proof schemes leads to large proof sizes, posing a significant challenge as the proof size becomes impractical for on-chain auditing due to its size. These drawbacks underscore the need for more efficient and practical solutions for auditing dynamic data in decentralized storage networks.

VI. Limitations of Decentralized Storage Networks (DSNs) [4]

While decentralized storage network systems hold promise, they still come with some drawbacks. As the technology is still in its nascent stages, researchers are exploring ways to address its challenges. Here are some of the challenges faced by blockchain-based decentralized data storage networks:

- **Lack of Trust**: Utilizing P2P technology, data is stored in a decentralized manner, bypassing centralized regulations. However, the lack of relevant accountability mechanisms in case of data loss or transaction mishaps may make it challenging for enterprises and consumers to trust decentralized networks. Due to the lack of trust in decentralized networks, developers are striving to enhance the highest levels of security, and new technologies may need time to gain the trust of businesses.
- **Complexity of Development**: Developing blockchain-based decentralized storage networks introduces additional complexity to consensus mechanisms. Proof-of-Storage (PoS), based on consensus mechanisms, ensures the authenticity of documents by verifying the integrity of remote files. To illustrate, each node in the system must prove that its submitted data qualifies it to add new records. Otherwise, users might perceive the consensus mechanism of the blockchain network as flawed. While the designed consensus mechanism is relatively complex, as any developer would tell users, this is an application worth paying attention to.
- **Security Concerns**: Despite the strong security of decentralized storage systems, malicious nodes may still potentially disrupt and even potentially destroy entire

decentralized storage network systems through launching centralized attacks. And blockchain-based decentralized storage systems are still in development to prevent these malicious attacks.

- **Time to Enter the Mainstream**: Decentralized storage undeniably addresses common issues with centralized storage. Compared to traditional mainstream storage systems, decentralized storage offers many advantages. To be widely adopted, however, decentralized storage network systems must provide services superior to the existing market. But currently, this technology is still in its early stages, and until it sees broader use in enterprises, it will remain niche in the market.

VII. Research Challenges and Future Directions

Despite the potential of blockchain-based decentralized storage network (DSN) systems, there are still several shortcomings that need to be addressed. In this paragraph, we will discuss the challenges faced by individuals and organizations when using services related to blockchain-based decentralized storage networks.

- **Security**: While blockchain networks offer higher security compared to traditional centralized systems, it's essential to note that blockchain may not provide complete security. Due to the decentralized nature of the network, security issues are less frequent but still possible. For instance, each time data needs editing or sharing with a third party, decrypting and re-encrypting the encrypted files could pose security risks. Additionally, while data is secure during storage, it may not be as secure during network transmission. Some attacks could severely damage the blockchain itself and its applications. For example, a 51% attack on a Proof-of-Work (PoW) consensus algorithm blockchain is possible, where nodes with high computational power could control the blockchain, leading to attacks like selfish mining and double spending. Increasing the number of nodes in the system can effectively prevent such attacks to ensure that no single entity can control the entire blockchain.
- **Lack of Decision-Making Data**: In many companies and organizations, collected data is considered a valuable resource for analysis and decision-making. However, as all data is encrypted before storage, blockchain-based storage systems cannot facilitate this process. Companies can authorize certified agents to store data in blockchain-based storage systems like Block House. Hence, company agents can retrieve and analyze all information as needed. Furthermore, private blockchains with certified members do not require data encryption. Through blockchain, data can be stored securely, reliably, and traceably on the chain.
- **Legal Limitations**: Smart contracts deployed on the blockchain hold both parties accountable for important information and conditions written down. However, in case of fraud, deception, or other unforeseen issues, there is a lack of practical legal support or court systems to rely on.
- **Scalability Issues**: Anyone wishing to join a blockchain network can do so by becoming a voluntary node. Maintaining network efficiency and security becomes challenging as the network grows. Scalability issues of blockchain networks can lead to delays and other problems. The scalability issue of a blockchain network may result in delays and other problems. As new nodes join the network, the startup time for new nodes includes the time needed for downloading and analyzing the network's history, which can be costly and time-consuming for older and larger blockchains like Bitcoin.
- **Access Control**: Blockchain always contains records of previous transactions, and it can be expected that a large amount of data will be constantly replicated among all nodes. However, this does not mean that the blockchain itself is a database. Large files stored on the blockchain may inflate according to both specifications. The blockchain storage network cannot share files between users. Solutions based on smart contracts have been proposed to overcome this issue, but they are only applicable to IPFS.

VIII. Conclusion

This paper explores the application of blockchain technology in decentralized storage networks (DSNs). The paper primarily introduces three different solutions proposed in IEEE Transactions journal articles. Firstly, paper [1] proposes an incentive-compatible mechanism that allows clients to initiate challenge requests to storage service providers without continuous verification of storage services. The mechanism utilizes blockchain smart contracts to execute storage contract rules and uses Oracle networks for storage verification, effectively preventing denial-of-service attacks. Secondly, paper [2] introduces a blockchain-based file replication mechanism that optimizes the data availability of the entire system using a decentralized storage algorithm inspired by Arweave. The mechanism employs smart contracts to monitor and eliminate dishonest nodes, promoting honest cooperation among nodes. Finally, paper [3] discusses how to audit dynamic data on resource-constrained public chains and proposes two audit protocols, OAD and OAD+, based on pseudo-indexes. These protocols transform static storage verification schemes into dynamic storage verification schemes through appropriate index transformation mechanisms while maintaining compact verification overhead. Overall, these three papers explore the application of blockchain in decentralized storage systems from different perspectives, offering valuable solutions to improve data availability, prevent malicious behavior, and support dynamic data auditing. These research findings provide important references and insights for the future development of decentralized storage networks.

Acknowledgments

This document was originally written in Traditional Chinese and subsequently translated into English using ChatGPT, with grammar corrections made through Grammarly. The writing

tool primarily used for this document was Overleaf, supplemented by the Writefull extension, which proved to be a useful AI tool for writing academic papers. Special thanks are due to the research seminar course for introducing me to the Writefull tool for writing papers. The content of this document is based primarily on the author's extracurricular research on topics learned in the Distributed Systems course offered by the Institute of Information Management at National Yang Ming Chiao Tung University. The document includes the author's personal interpretation and assimilation of knowledge gained from three IEEE Transaction journal papers; therefore, the accuracy of the content cannot be guaranteed. Additionally, I extend special thanks to the professor who taught this course, as well as to all the individuals who have made significant research contributions to the field of distributed systems. It is because of your contributions that I have been able to dive deeper into this fascinating field.

REFERENCES

[1] I. Vakilinia, W. Wang, and J. Xin, "An incentive-compatible mechanism for decentralized storage network," *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 4, pp. 2294–2306, 2023.

[2] F. Yang, Z. Ding, L. Jia, Y. Sun, and Q. Zhu, "Blockchain-based file replication for data availability of ipfs consumers," *IEEE Transactions on Consumer Electronics*, vol. 70, no. 1, pp. 1191–1204, 2024.

[3] H. Duan, Y. Du, L. Zheng, C. Wang, M. H. Au, and Q. Wang, "Towards practical auditing of dynamic data in decentralized storage," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 1, pp. 708–723, 2023.

[4] M. I. Khalid, I. Ehsan, A. K. Al-Ani, J. Iqbal, S. Hussain, S. S. Ullah, and Nayab, "A comprehensive survey on blockchain-based decentralized storage networks," *IEEE Access*, vol. 11, pp. 10 995–11 015, 2023.

[5] R. Banoth and M. B. Dave, "A survey on decentralized application based on blockchain platform," in *2022 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS)*, 2022, pp. 1171–1174.

[6] R. Sujeetha and C. A. S. Deiva Preetha, "A literature survey on smart contract testing and analysis for smart contract based blockchain application development," in *2021 2nd International Conference on Smart Electronics and Communication (ICOSEC)*, 2021, pp. 378–385.

[1] [2] [3] [4] [5] [6]

Qi Xiang Zhang received the BBA degree from the Department of Information Management, National Central University of Taiwan (NCU), in 2023. He is currently pursuing an MS degree with the Institute of Information Management, National Yang Ming Chiao Tung University of Taiwan, NYCU. His main research interests include data mining, machine learning and privacy, and network security.