future internet

MDPI

Article

Decentralized Storage with Access Control and Data Persistence for e-Book Stores

Keigo Ogata and Satoshi Fujita *

Graduate School of Advanced Science and Engineering, Hiroshima University, Higashi-Hiroshima, Hiroshima 739-0046, Japan; m223332@hiroshima-u.ac.jp
* Correspondence: satoshi.fujita.g@gmail.com

Abstract: The e-book services we use today have a serious drawback in that we will no longer be able to read the books we have purchased when the service is terminated. One way to solve this problem is to build a decentralized system that does not depend on a specific company or organization by combining smart contracts running on the Ethereum blockchain and distributed storage such as an IPFS. However, a simple combination of existing technologies does not make the stored e-book data persistent, so the risk of purchased e-books becoming unreadable remains. In this paper, we propose a decentralized distributed storage called d-book-repository, which has both access management function and data durability for purchased e-books. This system uses NFTs as access rights to realize strict access control by preventing clients who do not have NFTs from downloading e-book data. In addition, e-book data stored on storage nodes in the distributed storage is divided into shards using Reed–Solomon codes, and each storage node stores only a single shard, thereby preventing the creation of nodes that can restore the entire content from locally stored data. The storage of each shard is not handled by a single node but by a group of nodes, and the shard is propagated to all nodes in the group using the gossip protocol, where erasure codes are utilized to increase the resilience against node departure. Furthermore, an incentive mechanism to encourage participation as a storage node is implemented using smart contracts. We built a prototype of the proposed system on AWS and evaluated its performance. The results showed that both downloading and uploading 100 MB of e-book data (equivalent to one comic book) were completed within 10 s using an instance type of m5.xlarge. This value is only 1.3 s longer for downloading and 2.2 s longer for uploading than the time required for a simple download/upload without access control, confirming that the overhead associated with the proposed method is sufficiently small.

Keywords: blockchains; decentralized applications; decentralized storage system; e-book store; smart contracts

check for updates

Citation: Ogata, K.; Fujita, S. Decentralized Storage with Access Control and Data Persistence for e-Book Stores. *Future Internet* **2023**, *15*, 406. https://doi.org/10.3390/fi15120406

Academic Editor: Qiang Qu

Received: 16 November 2023
Revised: 8 December 2023
Accepted: 14 December 2023
Published: 18 December 2023

1. Introduction

The electronic publishing market is projected to grow at a CAGR of 6.4%, reaching USD 37,940.7 million by 2028 from USD 24,650.5 million in 2021 [1]. Furthermore, the sales of e-books accounted for over 22% of total book sales in 2018. This trend regarding e-books is expected to continue for the next few decades. However, current e-books have the drawback that purchased books become unreadable when the service is terminated. This issue was exemplified when an e-book service provided by Microsoft became inaccessible in July 2019 [2]. To address this problem, one potential solution is to deploy a decentralized architecture that does not rely on any specific company or organization.

Recently, distributed technologies such as blockchain [3–5] and Merkle tree [6] have garnered significant attention as a method to decentralize server-based distributed systems. Many appealing services, including Audius and Bluesky, have been developed using these technologies and have gained popularity among users (according to the official Audius website, Audius has over 6 million monthly active users). Bluesky is a decentralized social networking service that offers personal data server (PDS) functionality, allowing users to store their private data, including chat messages. This approach greatly reduces the risks of censorship and tampering. Audius is a music distribution service that utilizes an IPFS [7] as the distributed storage, enabling direct payments from listeners to music creators using tokens issued by Ethereum.

In the realm of e-book systems, a distributed e-book store called Publica [8] was launched in 2017. Publica effectively addresses issues associated with server-based systems, such as single points of failure and censorship, by storing e-book data on IPFS-based distributed storage. However, it still faces the problem of being unable to download purchased content due to the lack of data persistence in the existing distributed storage. For example, the IPFS has a property that makes it difficult for unpopular files to be cached, leading to situations where a file may only be cached by a very small number of nodes. Consequently, if these nodes leave the network or selectively delete the file to increase available storage capacity, it becomes impossible to download the file from an IPFS (see Section 2 for a detailed discussion on this point).

In the context of e-book stores, distributed storage requires not only data persistence but also robust access control mechanisms. This is because e-book data should only be accessible to users who have purchased the book, and preventing non-purchasers from downloading e-book data is crucial. It is important to note that such non-purchaser users include those who provide distributed storage for e-book data. The challenge in decentralizing e-book systems lies in achieving both data persistence and robust access control, as described above. However, to the best of the authors' knowledge, there is currently no existing distributed storage solution that meets these requirements simultaneously.

This paper presents a decentralized storage system for e-books, referred to as "d-book-repository", that effectively addresses the aforementioned requirements. The proposed system employs a smart contract to manage access rights, ensuring that data are only transferred to authorized clients and preventing unauthorized downloads. To bolster data security, we divided the nodes in the system into multiple groups, each responsible for storing a distinct data shard. This approach prevents any single storage-providing node from accessing the entirety of the stored content, enhancing data security. To improve data persistence, we utilized Reed–Solomon codes [9] to generate shards and implement group-based replication. Furthermore, we introduce an economic incentive mechanism that regularly audits storage content, promoting reliable storage practices and supporting the goal of data integrity. In summary, our designed decentralized storage system, d-book-repository, simultaneously fulfills the requirements of robust access control and data persistence, providing enhanced security and encouraging reliable data storage practices.

We constructed a prototype of the proposed system on AWS and conducted an evaluation of its basic performance. The results demonstrated that both downloading and uploading 100 MB of e-book data (equivalent to one comic book) were accomplished within 10 s. Notably, when using the m5.xlarge instance type, the time required for the aforementioned tasks was only 1.2 s longer for downloading and 2.2 s longer for uploading than a simple download/upload process without access control. This finding confirms that the overhead associated with the proposed method is negligible, indicating the efficiency of our approach.

We also conducted simulations on the increase and decrease in node numbers for various group settings and evaluated the data retention capability of the proposed system. Furthermore, we carried out simulations under the same conditions for Arweave's architecture and compared the results. As a result, it was shown that when the total amount of stored data is large (more than 10 TB), even with the same number of nodes, the proposed system has a lower risk of data loss.

The remainder of this paper is organized as follows. Section 2 provides an overview of blockchain and smart contracts. In Section 3, we review the decentralized distributed storage systems that have been built so far and point out that all of them are insufficient for our purpose. Section 4 describes the details of our proposed method. The details of our prototype system and the results of experiments conducted on it are presented in Section 5. Section 6 discusses several important considerations for the practical application of the d-book-repository. Finally, Section 7 concludes the paper with concluding remarks and future work.

2. Distributed Ledger Technology

This section presents an overview of distributed ledger technology, which is a mechanism for maintaining a consistent ledger across a distributed network without the need for centralized authority. The subsequent subsections outline the fundamental framework of distributed ledger technology, using Bitcoin as an illustrative example (Section 2.1), and then delve into the significance of smart contracts in controlling user access rights within the proposed system (Section 2.2).

2.1. Blockchain as a Distributed Ledger

Blockchain is a distributed technology originally proposed for a cryptocurrency called Bitcoin [10]. Its fundamental concept revolves around enabling each user to verify the balance of funds held in their account by sharing the transaction history between accounts across all nodes in the network. In blockchain technology, a fixed number of transactions are combined into a single block, and each transaction undergoes verification to ensure its consistency with previous transactions. Consequently, each block is designed to be devoid of inconsistent transactions, such as double expenditures or an insufficient balance. As there is no centralized entity managing the entire system, every node within the network can independently create blocks from transactions issued by each user. Each generated block includes a hash value of the preceding block, creating a collection of blocks linked together by their respective hash values. It is important to note that tampering with the transactions contained in a block can easily be detected, as the subsequent block contains the hash of the affected block, thus preserving data integrity within the chain.

However, this process is not enough for our purpose, as each node can generate its own chain of blocks. To address this, the blockchain protocol enforces a rule known as the "longest chain rule", where the longest chain of blocks is considered valid. Bitcoin, in particular, employs a mechanism called proof of work (PoW), which requires nodes to expend a certain amount of time and computational effort to create a block [11]. Attempting to tamper with a transaction within an approved block, following the longest chain rule, necessitates modifying the hash values in all subsequent blocks at a faster pace than the blockchain grows through contributions from other nodes. Nonetheless, under PoW, such tampering is practically infeasible, establishing a crucial property of the blockchain: once consensus is achieved, it becomes practically immutable and cannot be overturned [12].

2.2. Ethereum as a Decentralized Platform

Ethereum [13] is another cryptocurrency with the notion of blockchain, but it is widely recognized as a decentralized platform designed for the execution of self-executing programs known as smart contracts [14,15]. The description language for these smart contracts is considered Turing-complete, allowing for the support of various types of computations. This subsection provides an overview of Ethereum as a decentralized platform and its capabilities.

The concept of accounts is crucial in the context of smart contracts. Accounts are categorized into two types: externally owned accounts (EOAs) and contract accounts, each assigned a unique address as an identifier. EOAs are accounts managed by regular users using their private keys. Conversely, contract accounts are specialized accounts responsible for executing smart contracts, housing internal code and data storage (note that in Ethereum, once a contract account is created, its internal code is fixed and cannot be altered).

In the Ethereum network, each node is equipped with a virtual machine called an Ethereum virtual machine (EVM), which executes the code embedded in a contract account on each node. The execution of the internal code within a contract account is triggered by transactions initiated by an EOA. In these transactions, the contract account serves as the counterparty of the EOA, and the desired functions and parameters are included in the transaction as supplementary data. To prevent infinite execution of loops and reduce the burden on the network, each instruction in the code is assigned a predetermined cost (e.g., 3 for ADD, 5 for MUL, etc.), and the execution of an instruction consumes a specific amount of gas, which corresponds to the transaction fee. This gas mechanism ensures that resource usage is controlled during code execution. However, it is important to note that this mechanism can increase the gas costs associated with writing data, making Ethereum less feasible as a data repository for large files, such as e-books, due to potential high costs. On the other hand, simple data fetching without state transitions or money transfers can be performed without creating a transaction, leading to no gas consumption in such cases.

One important application of smart contracts is non-fungible tokens (NFTs) [16]. In the NFT standard, ERC721 [17], each token is identified by a uint256 token ID, and the owner of each token is recorded on the blockchain through mappings on the contract. The owner of an NFT has the right to transfer their NFT to another address so that the NFT can be used as a digital collector's item that can be bought and sold, such as games or art. The most notable feature of NFTs is that ownership is managed on the blockchain, so information about the item and its owner is never lost.

3. Existing Decentralized Storage Systems

This section offers an overview of existing decentralized storage systems, with a particular focus on access management and data persistence aspects. We explore the distinctive characteristics of each storage type and identify the challenges associated with using these systems for e-book storage. Table 1 compares the characteristics of the existing decentralized storage systems reviewed in this section with those of the proposed system.

Table 1. Feature-based comparison of decentralized storage systems.

Storage Systems	DLT	Crypto Incentives	Data Persistence	Access Control	Contract Duration
IPFS	N/A	No	Depends	No	No
ACL-IPFS	Ethereum (smart contract)	No	Depends	Yes	No
Filecoin	Filecoin blockchain	Yes	Yes	No	Yes
Arweave	Arweave (blockweave)	Yes	Yes	No	No
Proposed System	Ethereum (smart contract)	Yes	Yes	Yes	No

3.1. InterPlanetary File System (IPFS)

The InterPlanetary File System (IPFS) is an open-source, peer-to-peer decentralized file system primarily developed by Protocol Labs in 2014 [7]. Since the architecture of the IPFS does not depend on a central server or super node, the IPFS has no single point of failure and is expected to operate with zero downtime. Each peer within the IPFS is recognized by a PeerID, obtained by hashing its public key. Peers can be located efficiently using a distributed hash table (DHT) based on S/Kademlia and Coral. Specifically, content uploaded to the IPFS is given a unique content identifier (CID), generated by hashing the content, and is split into numerous chunks and stored using a data structure known as a Merkle DAG. The vertices within this DAG are data types called IPFS objects, which comprise chunks and links to other IPFS objects. Each IPFS object possesses the CID of the original content as one of its attributes. A content-based addressing scheme leverages this CID, facilitating decentralized content storage with an expectation of high availability. Moreover, a DHT is used to create and index a mapping between the CID and its owning PeerID for rapid content retrieval.

In a distributed application called IPFS Cluster, file persistence and availability are enhanced by combining the replication of stored files to multiple peers and pinning them to the local storage of the peers. Specifically, a peer cluster formed by the IPFS Cluster manages the pins of a specific dataset to prevent important files from being lost from the network and evenly allocates files to peers in the cluster to distribute the load of uploaders and increase the tolerance to peer failures and departures.

Despite its advantages, an IPFS lacks access control functions, allowing anyone to freely obtain uploaded content [18]. Furthermore, the absence of a mechanism for persisting uploaded content poses challenges, as nodes holding chunks can leave the network, and required chunks may become unavailable due to factors like priority deletion when a node nears its storage capacity [19]. Consequently, an IPFS may not be suitable as a repository for e-book services that require exclusive access to e-book data by the purchasing user on a permanent basis.

3.2. Acl-IPFS

Acl-IPFS [20], proposed by Steichen et al. in 2018, is an extension of an IPFS that incorporates access control features. It leverages Ethereum smart contracts to control access rights, allowing nodes to send chunks of data after referring to and validating these smart contracts. The risk of a node accessing the entire content without proper access rights can be mitigated by dividing the content into multiple shards. Different nodes are then assigned to store these shards, with each node granted specific access rights.

One of the key challenges for acl-IPFS is that nodes cannot freely replicate chunks unless access rights are granted, which poses an obstacle to ensuring data persistence. To address this limitation, one possible approach is to maintain the number of chunk replicas above a certain threshold to enhance resilience against node departures and chunk deletions. However, achieving this requires a mechanism for autonomously generating chunk replicas when a decrease in the number of replicas is detected by a node. Unfortunately, under the above restriction, nodes can only generate replicas if they are granted access rights, leading to a natural decrease in the number of granted nodes and creating the need to grant new access rights to other nodes, which conflicts with our goal of strict access control.

3.3. Filecoin

Filecoin [21] serves as an incentive layer of an IPFS, introduced in 2014 with its Mainnet launched in 2020, addressing the critical issue of data persistence. To store data in Filecoin, users first engage in contracts with storage providers, referred to as miners. These contracts define the data's storage volume, duration, and the corresponding price for the storage service. Upon entering into an agreement, users reward miners with native tokens (FIL), and the transaction history of FIL is recorded on the Filecoin blockchain. Filecoin implements two incentive mechanisms, namely proof of replication (PoRep) and proof of spacetime (PoSt), to bolster data persistence within the IPFS. PoRep operates as a challenge-response protocol that verifies the presence of a file in the designated storage. Once the network successfully verifies the proof, the result is recorded on the blockchain. PoSt, on the other hand, enforces miners to periodically submit proof and validate the correctness of their stored data. Miners failing to submit or verify their PoSt will incur penalties, leading to a partial loss of their deposit. Additionally, Filecoin leverages a combination of erasure coding and replication strategies to heighten data availability.

Access control for stored data in Filecoin can be implemented in the same manner as in acl-IPFS. However, Filecoin is not suitable for our e-book repository, which requires indefinite data retention. This is because the content storage contract in Filecoin is established by specifying a completion time for the contract. The requirement to set a contract completion time is rooted in the design concept of paying for data storage with native tokens and the inherent challenge of accurately predicting the long-term price fluctuations of the cryptocurrency used for rewards. As a result, distributed storage systems such as Storj [22] and Sia [23], which offer contract-based persistence, are also unsuitable for our intended purpose.

3.4. Arweave

Arweave [24] is a project primarily aimed at persistent data storage and was introduced as "Archain" in 2017. In contrast to Filecoin, it allows indefinite data storage for a one-time fee. Arweave employs a blockchain-like data structure known as Blockweave as the content storage, and nodes in Arweave can verify transactions without storing all blocks, which can be realized by synchronizing the block hash list with the wallet list. The data storage incentive structure comprises two elements: proof of access (PoA) and Wildfire. In PoA, older blocks are randomly selected and required for block generation, while in Wildfire, block and transaction dissemination priority is based on the speed of response to requests. These aspects encourage peers to retain as much data as possible.

The capability of indefinite data storage in Arweave seems well-suited for our use case. However, the lack of an access control function presents a challenge, and implementing such a function is also difficult. Specifically, the system makes it problematic to permit data storage exclusively on nodes that have been granted access rights. This is due to Arweave's requirement that all nodes maintain a complete copy of the transactions referenced in a block within their local transaction pool to validate them. If access rights are granted to only a subset of nodes, it would hinder the ability of other nodes to validate the block.

3.5. Summary

In this section, we have surveyed several existing decentralized storage solutions. Each of these is designed and implemented to fulfill specific objectives, and adapting them for our purposes presents difficulties. In the following section, we will introduce a novel decentralized storage mechanism that can efficiently offer both access control and data persistence.

4. Proposed System

As described in the previous sections, the basic characteristics required of the e-book store envisioned in the proposed methodology are the following two points:

- Persistent distributed storage is used so that data are never lost once uploaded.
- Only the user who has the access right to the e-book can retrieve the e-book data. (Without access rights, the node providing storage cannot retrieve the entire e-book data.)

The proposed system realizes these functions using a decentralized distributed storage called d-book-repository and a smart contract called Access Control Contract (ACC) to manage access rights. In the following, we show a concrete procedure for uploading/-downloading content in the proposed system, followed by an explanation of the supporting mechanisms in the order of access control and data persistence.

4.1. Two Specific Usage Scenarios

To facilitate a clear understanding of the roles of individual components in the proposed system, we outline two specific usage scenarios. The first scenario involves the upload of e-book data. See Figure 1 for an illustration. The author A of an e-book uploads their e-book data to the proposed system. The system employs Reed–Solomon codes to divide the data into k shards. The value k corresponds to the number of node groups, which will be described later. The `register` function of the ACC is called to record metadata, such as the title and price of the e-book, in the blockchain. Next, each of the k shards is uploaded to the d-book-repository. The system automatically propagates each shard to multiple storage nodes that constitute the d-book-repository, enabling efficient distribution without requiring A (the author) to wait for the entire process to be completed. Upon receiving a shard from A, each storage node verifies the integrity of the data to ensure they has not been tampered with; the specific tamper detection method will be detailed later.

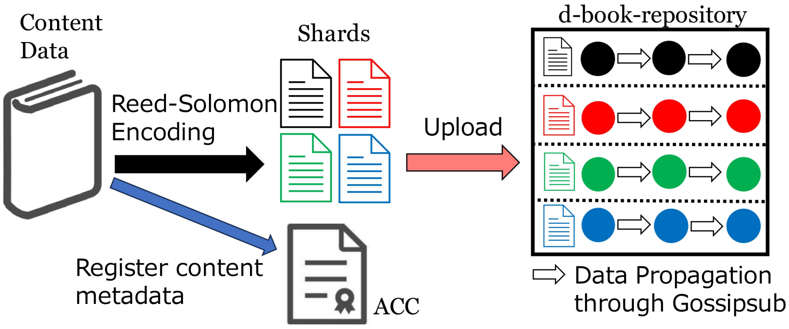

Figure 1. The flow of uploading e-book data.

On the other hand, when reader B purchases a book and seeks to obtain its e-book data, the following steps are executed (see Figure 2 for illustration). Initially, B acquires the non-fungible token (NFT) of the desired e-book by invoking the `mint` function of the ACC. Subsequently, B establishes a connection with the d-book-repository to identify the storage nodes holding the necessary shards for reconstructing the e-book data. Afterwards, B sends a request for a shard to each storage node, where the request contains the "book title and the signature of reader B", where the signature is generated by the private key of B's Ethereum account with the PeerID of the receiver as the message. Upon receipt of the request, each storage node verifies the Ethereum address of sender B from the signature and consults the ACC to confirm the access rights to the requested book of B's account. Upon confirmation, the requested shard is sent back to B. Subsequently, B restores the e-book data by decoding the shard using Reed–Solomon code after verifying the integrity of the received shards.

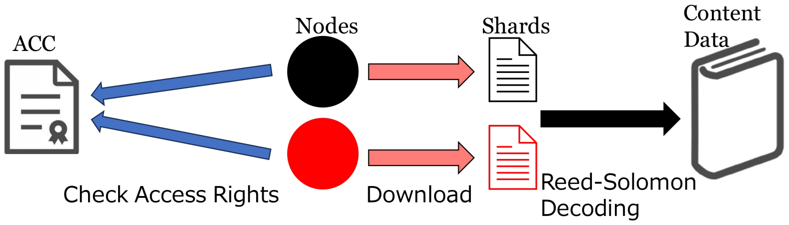

Figure 2. The flow of downloading e-book data.

4.2. Decentralized Access Control to the Stored Data

In distributed storage such as an IPFS, the provision of storage functions is open to anyone, which introduces complexity to access management for the data stored within the distributed storage. This complexity arises from the need to apply access control not only to external users but also to the storage nodes themselves that constitute the distributed storage. To address this challenge, the proposed system employs an Access Control Contract (ACC), a smart contract responsible for managing access rights as NFTs, enabling decentralized control of access rights. Additionally, to effectively manage access rights for storage nodes, the proposed method involves dividing e-book data into shards and distributing them among node groups. In this subsection, we provide a detailed step-by-step explanation of the access management mechanism implemented in the proposed system.

4.2.1. Access Control Contract (ACC)

An ACC provides three fundamental functions used for access management: `register`, which registers books, `mint`, which issues NFTs corresponding to access rights, and `hasAccessRight`, which checks whether a user has access rights to a book. The relationship between ACCs, authors, and readers is shown in Figure 3.

First, the authors register e-book metadata using the `register` function, but this registration must be performed before the e-book data are uploaded to the d-book-repository. The `register` function is given the book title, price, and Merkle root of the e-book data as arguments, and the price, Merkle root, and author's Ethereum address are recorded on the blockchain as mappings keyed to the book title. After registration is complete, readers can use the `mint` function to publish NFTs associated with the book. The `mint` function issues new NFTs by paying a cryptographic asset equal to the price set by the author, and the paid cryptographic asset is transferred directly to the author. At that time, the token ID of the issued NFT is associated with the purchased book, this mapping is also recorded in the blockchain. Only books registered with the `register` function can be associated with NFTs, but there is no upper limit to the number of NFTs that can be issued per book. The `hasAccessRight` function is used to check whether a given account has access rights to a given book. This function is called by a storage node when the node receives a request to upload a shard to a reader, and the requested shard will not be uploaded if it finds that the user does not hold access rights. Note that since this function only refers to data on an ACC, no gas cost is required to execute it.

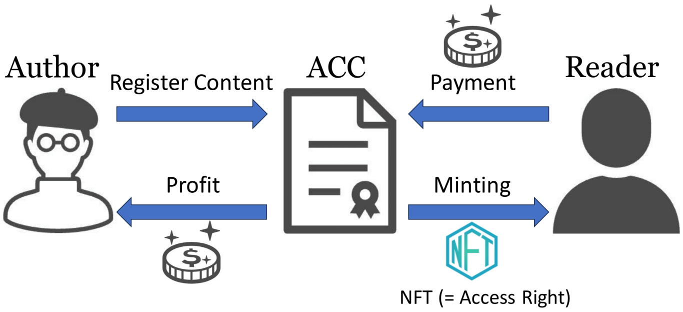

Figure 3. The relationship between an ACC, an author, and a reader.

Although it is possible to implement a similar system on a smaller scale by simply mapping books to their purchasers without utilizing NFTs, the adoption of NFT standards is believed to facilitate more active utilization through third-party services such as OpenSea.

The code for the `register` Listing 1, `mint` Listing 2, and `hasAccessRight` Listing 3 functions is as follows:

Listing 1. Solidity code for the register function.

```
function register(uint price, string memory title, string memory merkleRoot)
    ↪ public {
    require(bytes(_contentMerkleRoots[title]).length == 0);
    _authors[title] = msg.sender;
    _prices[title] = price;
    _contentMerkleRoots[title] = merkleRoot;
    _content_title_list.push(title);
}
```

Listing 2. Solidity code for the mint function.

```
function mint(string memory title, address to) payable external {
    require(_prices[title] == msg.value);
    payable(_authors[title]).transfer(msg.value);
    _contents[nextTokenId] = title;
    _safeMint(to, nextTokenId, '');
    nextTokenId++;
}

// This function is called when minting.
function _beforeTokenTransfer(address from, address to, uint tokenId)
    ↪ internal override {
    if(from != address(0)) _accessRights[_contents[tokenId]][from]--;
    _accessRights[_contents[tokenId]][to]++;
}
```

Listing 3. Solidity code for the hasAccessRight function.

```
function hasAccessRight(address account, string memory title) public view
    ↪ returns(bool) {
    return _accessRights[title][account] != 0 || _authors[title] == account;
}
```

4.2.2. Distributed Storage of Shards in Node Groups

Storage nodes participating in the d-book-repository are assigned unique PeerIDs generated from their secret keys, similar to existing distributed storage systems like the IPFS. However, in the proposed system, these storage nodes are organized into k groups based on the remainder of their PeerID when divided by k. The number of groups, k, is specified at the time of ACC deployment and cannot be changed thereafter. The number of groups can be determined by taking into account the persistence strength, as described in Section 5.3. Upon uploading, the e-book data provided by the author are divided into k shards using Reed–Solomon codes, with the ith shard being stored on all storage nodes within the ith group. Utilizing Reed–Solomon codes for shard generation ensures that the original data can be restored from only a certain number of shards without requiring all types of shards. This property enables the restoration of the e-book data even if some groups encounter issues or entirely disappear (similar techniques are also observed in RAID5 and RAID6).

In the mentioned distributed storage method, a storage node belonging to the ith group stores only the ith shard and no other shards. Consequently, if a storage node B does not own the access rights to the e-book, it cannot restore the complete e-book data solely from locally stored shards. Although it may attempt to recover e-book data by acquiring shards one by one from $k-1$ other storage nodes, the ACC prevents unauthorized acquisition requests if storage node B lacks access rights to the e-book.

4.3. Incentive Mechanisms for Permanence Enhancement

In the proposed method, each shard generated by the author is stored in all storage nodes in a node group, and such a configuration is achieved by a combination of two processes. The first is a push process in which storage nodes participating in the d-book-repository are subscribed to a topic for the node group to which they belong in advance, and shards published to that topic by the author are propagated to the entire group using gossipsub [25]. Note that gossipsub can spread data only to the nodes that have not yet received them by sending gossip messages before spreading the data themselves, so shard propagation can be performed more efficiently than normal floodsub.

The second is a pull process, in which each node goes for its own missing shard, usually after the shard has been distributed to the entire group by the gossibsub. From the discussion described above, each storage node can identify its own missing shards in the following steps: (1) Each storage node can obtain a list of e-books from the book information registered on the ACC, and if it stores no shard of an e-book in the list, it recognizes that a shard of the e-book is missing. (2) Each node can determine which of the k shards it should store from its own PeerID, and if shards generated from the e-book have been pushed by the author, someone in the group must have stored the shard (even if it does not have it itself).

In order to enhance the persistency of stored data, the proposed system provides an incentive for each storage node to “repeat the pull process within a node group until the storage node that finds its missing shard runs out of missing shards”. Specifically, it verifies the absence of missing shards by using a sampling quiz, where smart contracts are used to answer quizzes and receive rewards. (In the current implementation, such a quiz function is included on the ACC to save on gas costs, but it is possible to implement it as a completely separate smart contract).

The sampling quiz is issued every time N blocks are added to the blockchain used by the ACC. Note that it is issued at approximately regular time intervals without any trigger by the central authority since blocks in a blockchain are autonomously created at regular time intervals. In the sampling quiz, each storage node is requested to (1) extract segments specified by random values from the shards in its own storage and (2) concatenate them to form the answer (i.e., a proof), where the hash value of the latest (i.e., the Nth) block is used as the random number. See Figure 4 for an illustration. The **answer period** for a quiz is until the $(N + X)$th block is generated after issuing the quiz. Each storage node creates a new common key by the end of the answer period and records the encrypted proof in the ACC. The period between the end of the answer period and the generation of X more blocks is called the **disclosure period**, during which the common key is passed to the ACC to decrypt the encrypted proof. Next to the disclosure period is the **claim period**, in which the most common answer recorded in the answer period is considered the correct answer, and storage nodes that correctly answer the quiz receive a reward in proportion to the amount of the deposit, where the reward of the quiz is paid in the original token called DBookToken (DBT), which is issued by the ACC. On the other hand, incorrect answers to the quiz will forfeit the deposit.

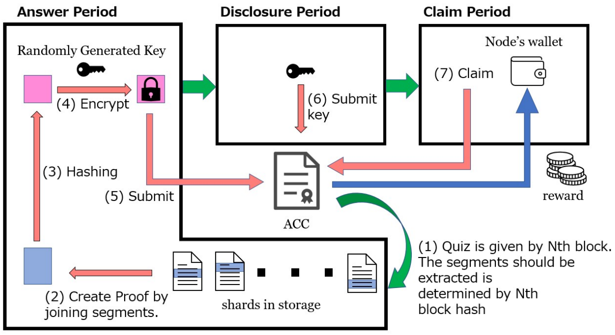

Figure 4. The flow of sampling quiz.

The code for the `vote` Listing 4, `disclosure` Listing 5, and `claim` Listing 6 functions is as follows:

Listing 4. Solidity code for the vote function.

```
function vote(bytes32 encrypted_answer) payable public {
    require(block.number % event_length < event_length/3, "Out of voting
        ↪ period.");
    uint event_id = block.number / event_length;
    uint group = _groups[_account_peer_id_map[msg.sender]];
    require(group > 0, "You aren’t registered.");
    require(!encrypted_answer_exists[event_id][group][encrypted_answer], "The
        ↪  same answer already submitted.");
    encrypted_answer_exists[event_id][group][encrypted_answer] = true;
    _deposits[event_id][group][msg.sender] += msg.value;
}
```

Listing 5. Solidity code for the disclosure function.

```
function disclosure(bytes32 answer, bytes32 key) public {
    require(block.number % event_length >= event_length/3 && block.number %
        ↪ event_length < event_length*2/3, "Out of disclosure period.");
    uint event_id = block.number / event_length;
    uint group = _groups[_account_peer_id_map[msg.sender]];
    require(encrypted_answer_exists[event_id][group][keccak256(abi.encode(
        ↪ answer, key))], "invalid answer and key");
    require(_answers[event_id][group][msg.sender] == bytes32(0), "Answer
        ↪ already disclosed.");
    _answers[event_id][group][msg.sender] = answer;
    if (++_answer_counts[event_id][group][answer] == 1) _answer_lists[
        ↪ event_id][group].push(answer);
}
```

Listing 6. Solidity code for the claim function.

```
function claim() public {
    require(block.number % event_length >= event_length*2/3, "Out of
        ↪ disclosure period.");
    uint event_id = block.number / event_length;
    uint group = _groups[_account_peer_id_map[msg.sender]];
    bytes32 ans = _answers[event_id][group][msg.sender];
    uint ans_cnt = _answer_counts[event_id][group][ans];
    for (uint i=0; i < _answer_lists[event_id][group].length; i++) {
        bytes32 ans_i = _answer_lists[event_id][group][i];
        require(ans_i == ans || _answer_counts[event_id][group][ans_i] <
            ↪ ans_cnt, "Your answer is wrong");
    }
    uint deposit = _deposits[event_id][group][msg.sender];
    _deposits[event_id][group][msg.sender] = 0;
    payable(msg.sender).transfer(deposit);
    dbt.mint(msg.sender, deposit * 10); // reward
}
```

4.4. Autonomous Management of Node Groups

In the proposed method, the number of replicas of a shard is equal to the size of the node group corresponding to the shard, so a group with a small number of nodes is susceptible to node departures and chunk deletions. In order to overcome this problem, the d-book-repository introduces a smart contract that assigns a newly joined node to the group with the smallest number of nodes at that time, whereas before, the current implementation realizes this smart contract as part of the ACC. The specific flow for the registration of a storage node proceeds as follows: First, a new storage node identifies a group with the smallest number of nodes by referring to the `next_group` function of the ACC. Then, the node creates a PeerID for that group by executing `registerNode` and pays a registration fee to join the ACC. At the end of the registration, the number of nodes in the group is increased by one on the ACC. A similar procedure is used for the unregistration of a node. Specifically, it calls `leaveNode` for the registration and canceling of the registration, the number of nodes in the group recorded in the ACC is decreased by one, and the registration fee is refunded when the process of unregistration is completed. Note that the PeerID of the caller can be calculated from msg.sender.

The code for the `registerNode` Listing 7, `next_group` Listing 8, and `leaveNode` Listing 9 functions is as follows:

Listing 7. Solidity code for the registerNode function.

```
function registerNode(string memory peer_id) payable public {
    require(_groups[peer_id] == 0, "This peer id is already used.");
    require(msg.value == registration_fee, "Registration fee is required.");
    uint group = next_group();
    _groupNodeCounter[group]++;
    _groups[peer_id] = group;
    _account_peer_id_map[msg.sender] = peer_id;
}
```

Listing 8. Solidity code for the next_group function.

```
function next_group() public view returns (uint) {
    uint group = 1;
    uint mi = _groupNodeCounter[1];
    for (uint i=2; i<=node_number; i++) {
        if (mi > _groupNodeCounter[i]) {
            mi = _groupNodeCounter[i];
            group = i;
        }
    }
    return group;
}
```

Listing 9. Solidity code for the leaveNode function.

```
function leaveNode() public {
    string memory peer_id = _account_peer_id_map[msg.sender];
    uint group = _groups[peer_id];
    require(group != 0, "This peer is not registered.");
    _groups[peer_id] = 0;
    _groupNodeCounter[group]--;
    _account_peer_id_map[msg.sender] = "";
    payable(msg.sender).transfer(registration_fee);
}
```

Note that charging a participation fee for node registration is intended to prevent Sybil attacks. In addition, the fact that a node cannot specify the group it will join increases the difficulty of a 50% attack on the incentive system and makes it difficult to create multiple accounts to join the system as storage nodes to restore arbitrary content.

4.5. Lightweight Tamper Detection Using Merkle Tree

In order to provide tamper resistance to stored data, the d-book-repository provides a verification function using a Merkle tree. The specific procedure is as follows: First, the author constructs a Merkle tree with the hash values of the shards as the leaves and submits the Merkle root to the ACC when the e-book is registered. See Figure 5 for an illustration. The Merkle proof is then attached when the shard is uploaded to the d-book-repository. In this way, a client receiving a shard from a storage node can verify whether the shard has been tampered with or not by referring to the Merkle proof attached to the shard and the Merkle root stored in the ACC. The main reason for using the Merkle tree is to save on gas costs. In addition, this method can significantly reduce the storage cost since only the root, not the hash value of each shard, needs to be stored.

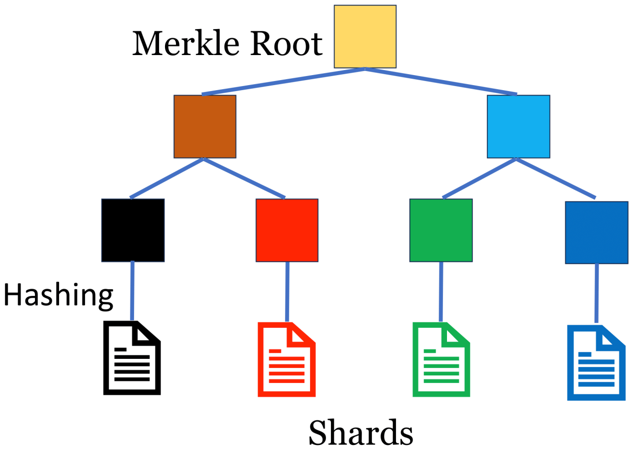

Figure 5. Overview of Merkle tree built from content shards.

5. Evaluation

In this section, we assess the performance of the proposed system by combining empirical data obtained through experiments conducted with a cloud-based prototype system and predictive values derived from event-driven simulations. We begin by describing the implementation of the core functionalities of the prototype system in Section 5.1, followed by a presentation of experimental outcomes obtained from the prototype in Section 5.2. The primary objective of these experiments is to verify the practical speed of the proposed system's operation. Subsequently, in Section 5.3 and subsequent sections, we proceed to evaluate various aspects, including persistence, fault tolerance, and the cost of the functionalities.

5.1. Prototype System

As described in the previous section, the proposed system comprises two key components: the d-book-repository and the ACC. In the present implementation, the ACC is written in Solidity, a JavaScript-like programming language for specifying smart contracts, while we selected Rust for writing the d-book-repository because it is a memory-safe and fast language. The implementation of d-book-repository leverages various pre-existing technologies. The management of PeerIDs, the propagation of e-book data through the gossipsub protocol, and message exchange between peers are all realized through the utilization of the libp2p library [26], a technology previously employed in the development of the IPFS. Additionally, the discovery of a peer with a designated PeerID is achieved by utilizing the Kademlia DHT [27] provided by the libp2p library. In addition, sha256 was adopted for the hash function used to create the Merkle tree and verify the received data.

In the prototype system, we employed an EC2 instance of Amazon Web Services (AWSs) as the host of the storage nodes for the d-book-repository. In the experiments described below, the number of node groups is fixed at 40, and Reed–Solomon codes are set so that the original e-book data could be restored by collecting 20 of the 40 shards. Each group is pre-registered with one storage node that never leaves the group, so that each group has at least one storage node. In addition to those forty storage nodes, the d-book-repository has one node for issuing upload/download requests. For the ACC, the Hardhat Network was run on another EC2 instance and deployed there, where the Hardhat Network is a test Ethereum client embedded in the smart contract development tool called Hardhat. In summary, the number of EC2 instances used in the experiment is 42, including 1 for the Hardhat Network, which is used as a mock-up of the Ethereum network, and 41 for running the d-book-repository node. Experiments were conducted using two instance types: m5.xlarge (vCPU: 4, RAM: 16 GiB) and t2.micro (vCPU: 1, RAM: 1 GiB). All instances were hosted in the Asia Pacific (Tokyo, Japan) region, and communication between nodes was achieved with sufficiently low latency.

5.2. Time Required to Download/Upload e-Book Data

The central functionalities of the proposed system encompass enabling e-book authors to upload their e-book data to the d-book-repository and facilitating e-book readers, who have made purchases, to access and download the e-book data from the repository. The practical viability of the proposed system hinges significantly on the overhead associated with managing access rights and ensuring the persistence of e-book data. Consequently, we conducted measurements to gauge the time required for both uploading and downloading e-book data on the prototype system. For this purpose, we utilized three distinct sizes of e-book data: 1 MB, 10 MB, and 100 MB. These sizes were chosen because they encompass the typical range of text-based e-book data, which varies from 1 MB to 100 MB, while also accounting for the size of a comic book, which typically falls at around 100 MB in size.

5.2.1. Downloading

Table 2 summarizes the time taken to download e-book data from d-book-repository for different types of EC2 instances and data sizes. This table indicates that, in the cloud environment where the prototype system is deployed, 100 MB of e-book data could be retrieved within 10 s, indicating that there is no problem in practical use. In fact, in the case of Amazon Kindle, it takes about one minute to retrieve 100 MB of e-book data from the cloud server and depends on the network environment.

Table 2. Total time for downloading e-book data.

	1 MB	**10 MB**	**100 MB**
t2.micro	1033 ms	1381 ms	7692 ms
m5.xlarge	517 ms	922 ms	4815 ms

In the prototype system, the downloader of the e-book data performs the following four steps in this order: (1) Discover 20 storage nodes belonging to different groups; (2) Request and receive shards from the discovered storage nodes; (3) Retrieve the Merkle root of the e-book data from the ACC and verify the authenticity of shards using the Merkle proof attached to the downloaded shard; (4) Apply the Reed–Solomon decoding to restore the original e-book data. Since a simple file transfer from storage nodes corresponds roughly to the second step, we measured the time required for each of the above four steps. Figure 6 summarizes the results. The figure shows that, as the size of e-book data increases, the percentage of overhead other than file transfer decreases, but when the data size is 1 MB, the percentage of time used for peer discovery is the largest among all steps, confirming that the overhead is not negligible. However, such a high percentage of overhead causes no practical issues since downloading 1 MB of e-book data itself only takes about 1 s.

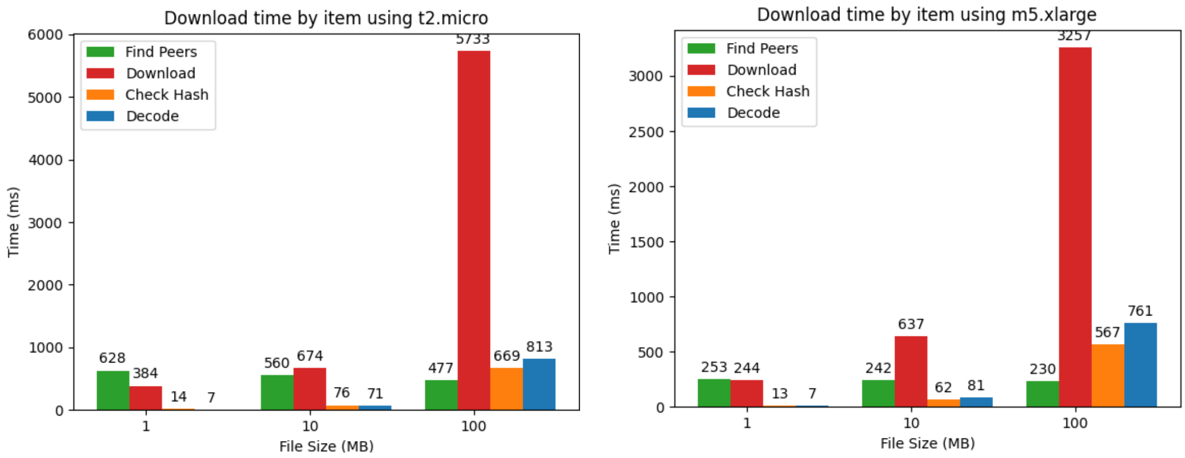

Figure 6. Breakdown of the download time.

5.2.2. Uploading

Table 3 summarizes the time required for uploading e-book data for different EC2 instance types and data sizes. As shown in the table, the upload of 100 MB of data using t2.micro failed because the uploader ran out of memory at the time of sending data to 40 storage nodes, causing a forced termination. This problem can be solved by preventing the uploader from using more memory than the upper limit, for example, by sending shards one at a time. In this case, however, the trade-off with the total upload time would become another issue. In all other cases, the upload of e-book data is completed within 10 s, as in the case of downloading.

Table 3. Total time for uploading e-book data.

	1 MB	**10 MB**	**100 MB**
t2.micro	1101 ms	2582 ms	failed
m5.xlarge	458 ms	1367 ms	9442 ms

Uploading e-book data to the d-book-repository consists of the following four steps, where we assume that the metadata of e-books has been registered with the ACC in advance and will not be included in the measurements: (1) Divide the e-book data into 40 shards by Reed–Solomon encoding; (2) Construct a Merkle tree of the shards using sha256 and record the Merkle root in the ACC; (3) Discover 1 storage node from each of the 40 node groups; (4) Send the corresponding shard and Merkle proof to the discovered nodes. It should be noted that the uploader is only involved in sending a shard to the first node in each group in the fourth step, and it is not involved in propagating the shard within the group.

Figure 7 shows a breakdown of the time required for each step. In the d-book-repository, the process of uploading is similar to downloading in the sense that it requires the discovery of storage nodes. However, the upload requires 40 nodes to be discovered, whereas the download requires only 20 nodes, so it requires extra time for the fourth step of uploading. In fact, although the ratio of data transmission to the total time increases as the data size increases, the increase in the ratio is more significant because the number of destinations increases from 20 to 40. For the same reason, the ratio of node discovery time becomes larger when the data size is 10 MB. Specifically, it is 86% for uploads on t2.micro, whereas it is 61% for downloads.

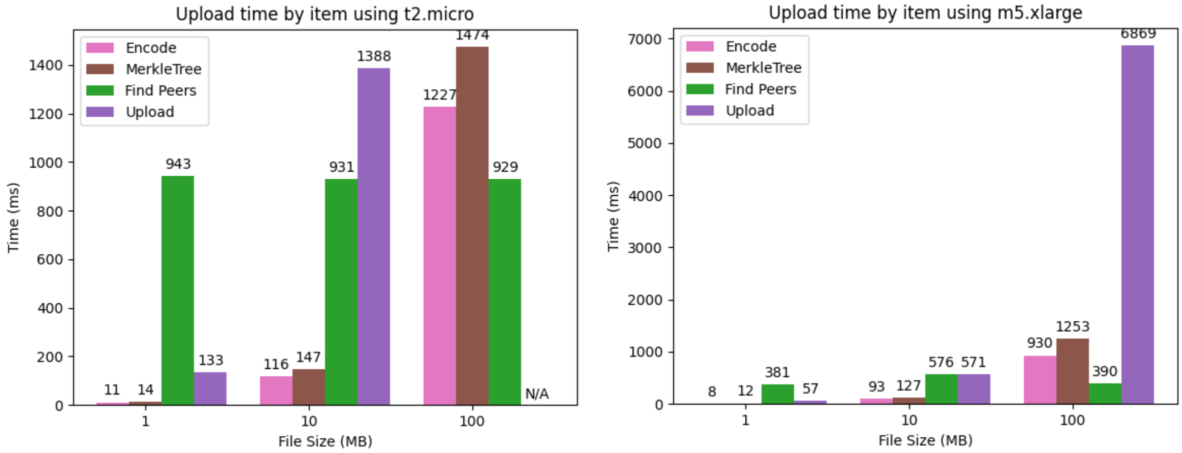

Figure 7. Breakdown of the upload time.

5.3. Evaluation of Persistency of Stored Data

Next, we evaluate the persistence of data stored in the d-book-repository. In the experiments, we evaluate the degree to which the continuity of storage service is disrupted by the departure of storage nodes. We assume that the join and leave of the nodes occur asynchronously and follow a random walk in which a new node joins with probability p and an existing node leaves with probability $1 - p$. We say that the system fails to persist if an e-book's data cannot be downloaded even once in the middle of an event sequence of length L generated, assuming $p = 1/2$. Note that $L = 10{,}000$ corresponds to about 270 years when the change in the node number occurs once every 10 days on average.

We assess the persistence of the proposed system in comparison to Arweave, using the following specified parameters: In the proposed system, the initial state consists of X nodes allocated within each group. When a new node joins, it is assigned to the group with the smallest number of nodes at that moment. Conversely, departing nodes are randomly selected from the entire set of nodes. As previously indicated, persistence is deemed unsuccessful if, at any point within an event sequence of length L, a group containing zero nodes emerges. In a scenario where the number of groups is set to 20, with each node offering 1 TB of storage capacity, the entire system can accommodate 20 TB of e-book data. This corresponds to 200,000 e-books of 100 MB each, excluding the size of the Merkle proof. In contrast, Arweave employs a different approach. When a storage node capable of accommodating 1 TB of e-book data joins, it receives 10,000 e-books of 100 MB each, randomly selected without duplication. In the simulation, the initial state consists of Y nodes, each holding 1 TB worth of e-books, and the criterion for persistence failure is the presence of at least one e-book with zero replicas at any point within an event sequence. For the experiment, we generated 1000 event sequences, each with a length of $L = 10{,}000$, and recorded the count of event sequences that successfully persisted.

Table 4 summarizes the experimental results for the proposed method. As can be seen from the table, the number of nodes required to maintain the same level of persistence increases as the number of groups k increases. Specifically, if we want to keep the persistence failure rate below 1%, 15, 11, and 9 (initial) nodes are required per group when the number of groups is 20, 30, and 40, respectively, which corresponds to a total of 300, 330, and 360 storage nodes, respectively. This indicates that, for the same number of nodes, a smaller k is better in terms of data persistence. However, it should be noted that, in reality, the storage service can continue even if several groups disappear up to the number of parities due to the nature of the Reed–Solomon codes. In addition, in actual environments, persistence can also be improved by monitoring the number of nodes for each group and adding storage nodes when it falls below the required success rate.

Table 4. The number of event sequences that did not fail to persist in the proposed system (out of 1000).

X	5	6	7	8	9	10	11	12	13	14	15
$k = 20$	399	536	648	763	795	868	910	942	964	980	990
$k = 30$	555	719	841	902	942	979	992	999	999	1000	1000
$k = 40$	678	856	927	971	992	999	999	1000	1000	1000	1000

Table 5 summarizes the results for Arweave. Comparing with Table 4, it can be seen that the proposed system achieves higher data persistence for the same number of storage nodes. For example, when storing 200,000 books (of 100 MB each) with 400 storage nodes, the number of successes of the proposed system is higher than that of Arweave regardless of the number of groups $k(\geq 20)$. On the other hand, Arweave has an advantage in that the data persistence increases as the number of stored books decreases, due to an increase in the number of replications per e-book. In fact, Table 5 shows that the number of successes is much higher at 100,000 books than at 200,000 books, while the proposed system still shows superior results even with 100,000 books.

Table 5. The number of event sequences that did not fail to persist in Arweave (out of 1000).

Y	100	200	300	400	500	600
200,000 books (=20 TB)	0	0	56	585	943	998
100,000 books (=10 TB)	0	454	900	993	1000	1000

5.4. Evaluation of Incentive Mechanism

In the proposed system, the persistence of stored data is maintained by actively reducing the number of chunks that each storage node is missing, in addition to devising methods for replicating and storing chunks. Specifically, the proposed system encourages each storage node to acquire all of the corresponding chunks by using a quiz as an incentive mechanism that increases the percentage of correct answers (and the amount of reward obtained) for the absence of missing chunks. In this subsection, we evaluate the cost of the incentive mechanism. The frequency of quiz questions affects the strength of persistence, and the evaluation results described below confirm that a quiz frequency of about once a day has little or no effect on the system performance.

As mentioned in the previous section, to answer the quiz, each storage node generates a proof of holding chunks specified by the contestant. Specifically, in the current implementation, a proof is generated according to the following procedure:

1. Query the ACC for the block hash;
2. Obtain the bit indicating the location of the chunk calculated from the block hash for all stored shards, and add the obtained bits to Vec<bool> sequentially;
3. Hash Vec<bool> with keccak256 to complete the proof.

In the experiments, we measure the time to generate the proof by changing the number of files as 10,000, 100,000, and 200,000 on an EC2 instance hosting the storage node, where, in order to focus on the computational cost of storage nodes, we exclude the query time for the block hash in the first step from the measurement time. Note that, since the generation of a proof does not require reading the entire file, there is no direct relationship between the e-book size and the generation time.

Table 6 summarizes the generation time of a proof for different numbers of files and for two different EC2 instances, t2.micro and m5.xlarge. The most time-consuming case is the handling of 200,000 files on t2.micro, but even in this case, the time required is only about one and a half minutes. Therefore, we can conclude that this is well within the practical range as long as the quiz is issued at a frequency of about once a day. In addition, m5.xlarge is faster than t2.micro, and the degree of speedup is more significant when the number of files is smaller. Behind this phenomenon may be the influence of the cache and hardware characteristics of the EC2 instance.

Table 6. Time taken for generating an answer to the quiz.

# of e-Books	10,000	100,000	200,000
t2.micro	4128.06 ms	39,196.68 ms	90,958.94 ms
m5.xlarge	74.59 ms	1804.42 ms	24,736.23 ms

5.5. Execution Cost of ACC Functions and Performance Comparison by Blockchain

As of 2023, numerous public blockchains have emerged as viable options for implementing decentralized applications like ACCs. Given that the smart contracts of the proposed system are deployable on any blockchain that supports the Ethereum virtual machine (EVM), the choice of blockchain for app development becomes a crucial issue. In this subsection, we focus on Polygon [28], Arbitrum [29,30], and Solana [31,32] as notable alternatives to Ethereum and evaluate these blockchains in terms of max recorded TPS, real-time TPS, block time, and average transaction fee, where TPS stands for the number of transactions processed per second and block time is the average time for generating a block. Table 7 summarizes the results. The table shows that Ethereum exhibits the lowest TPS, and its long block time and high transaction fee make it less user-friendly. In fact, the best choice in terms of TPS and transaction fee is Solana, especially in TPS, which is far ahead of other blockchains, and Arbitrum is the best choice in terms of block time. It should be noted, however, that there is a tradeoff between speed and security [33], and this consideration arises because public blockchains necessitate security-dependent transaction fees, which are ultimately borne by the app's users.

Table 7. Comparison of performance metrics across major blockchains.

Blockchain	Max Recorded TPS [34]	Current TPS [34]	Block Time [34]	Transaction Fee (Avg) [35]
Ethereum	57.91 tx/s	10.98 tx/s	12.24 s	1.30 USD
Polygon	273 tx/s	57.05 tx/s	2.44 s	0.00061 USD
Arbitrum	175 tx/s	15.56 tx/s	0.25 s	0.0046 USD
Solana	2712 tx/s	447 tx/s	0.42 s	0.00032 USD

Finally, we evaluate the cost of running ACC functions on Ethereum. Table 8 summarizes the cost of each function measured using the hardhat-gas-reporter. Although the costs of functions vary widely depending on Ethereum price fluctuations and network conditions, the gas price and ETH price were 26 Gwei/gas and 2151.72 USD/ETH at the time of evaluation. The table shows that the most expensive ACC function is `registerNode` at USD 10.21, while the `mint` function, which is considered the most frequently used by users, costs USD 8.49. Considering that the price of a Kindle book is around USD 4 to 15, this result means that deploying the service on Ethereum will be a big burden for e-book users. Since the execution cost on other blockchains is proportional to the transaction fee (avg) shown in Table 7, the execution cost on those blockchains is not a big issue.

Table 8. Estimated gas fees for ACC contract functions at 26 Gwei/gas, 2151.72 USD/ETH.

Function	Execution Cost (USD)
claim	5.09
disclosure	6.03
leaveNode	2.62
mint	8.49
register	7.63
registerNode	10.21
vote	4.45

6. Discussion

This section discusses several points to be noted in the practical application of the d-book-repository.

The first issue concerns the storage capacity limit. The d-book-repository is designed to store shards generated from contents on all storage nodes in a uniform manner. Therefore, as long as the storage capacity provided by each node is fixed, the total capacity does not increase with the number of nodes, as in many other distributed storage systems. For example, assuming that each node provides 1 TB of storage and that each content item is encoded in 20 shards, the system can store up to 20 TB of content no matter how many nodes are added. This amount is equivalent to 200,000 e-books of 100 MB, which seems to be sufficiently large at the present time. However, since the amount of stored content grows monotonically, it is likely to eventually be used up over time.

The easiest way to solve this problem is to keep replacing the storage devices of each node in the system with larger capacity ones as the underlying device technology advances, but this imposes the burden of updating on the storage node provider, thus compromising sustainability. One solution to overcome this issue is to increase the capacity of all nodes' contributions after reaching an agreement among all storage nodes. If less than half of the storage nodes do not comply with the agreement, they will not be able to answer the incentive quiz correctly and will not receive a reward, thus the result of the agreement will be binding.

The second issue is how to prevent unauthorized downloads. There may be rogue nodes on the network that fail to verify the downloader's access rights, and users of the proposed system can restore the entire content regardless of the existence of access rights if they find the required number of rogue nodes belonging to different groups. Potential solutions to this issue encompass penalizing nodes that fail to verify access rights or utilizing encryption in conjunction with content protection measures.

The third issue is countermeasures against uploading illegal contents and unauthorized registration by non-copyright holders [36]. Once illegal content is uploaded to the network, it is difficult to completely remove it, and it is also difficult to verify with certainty whether the Ethereum address (EOA) used for content registration belongs to the copyright holder. Similar problems have been noted with other distributed storage systems. For example, Arweave claims to employ a blacklist to prevent the spread of illegal content, but the effectiveness of this approach is questionable. Presently, the most efficacious preemptive measure against such threats appears to involve pre-upload screening by a trusted third party.

The fourth issue is the protection of user privacy. If a public blockchain such as Ethereum is used to implement the proposed method, the history of books purchased by users through their Ethereum accounts can be read by anyone. A possible approach to addressing this problem is to utilize self-sovereign identity (SSI), a system that allows individuals to manage their own identity information and share it securely as needed. This could allow users to have strict control over their own identity information, purchase history, and other data [37].

7. Concluding Remarks

In this paper, we propose and evaluate a decentralized architecture for realizing persistent e-book services. The proposed system consists of a smart contract called ACC to control access rights and a d-book-repository to store e-book data in a distributed and persistent manner. Similar to existing decentralized storage systems, the d-book-repository applies Reed–Solomon codes to e-book data to generate k redundant shards, but unlike existing systems, the set of storage nodes is explicitly partitioned into k groups, and each shard is uniformly copied to all nodes in one node group. The persistence of the stored e-book data is achieved by the fact that nodes within a node group continue to maintain copies of all shards corresponding to that group, and access rights control is achieved by having each storage node verify the access rights of the downloaders. We built a prototype of the proposed system on AWS and evaluated its performance in various settings. As a result, we confirmed that uploading and downloading a 100 MB file can be completed within 10 s in an environment with low network latency and that the costs required for access control and persistence of stored data are sufficiently small.

The proposed method can solve the problem of the inaccessibility of purchased e-books due to reasons such as service termination or intentional data removal by the publisher. However, several issues remain to be addressed, including the enlargement of storage capacity, enhancement of access control functions, and countermeasures against illegal content and unauthorized uploads.

Author Contributions: Conceptualization, K.O. and S.F.; methodology, K.O. and S.F.; software, K.O.; validation, K.O.; writing—original draft preparation, K.O.; writing—review and editing, S.F. and K.O. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. The Insight Partners. Digital Publishing Market Forecast to 2028—COVID-19 Impact and Global Analysis by Content Type (Text, Video, and Audio) and End User (Individual, and Enterprises). Available online: https://www.marketresearch.com/TIP-Knowledge-Services-v4095/Digital-Publishing-Forecast-COVID-Impact-30120207/ (accessed on 18 November 2023).
2. Lee, D. Microsoft's eBook Store: When This Closes, Your Books Disappear too. *BBC News*, 4 April 2019.
3. Nofer, M.; Gomber, P.; Hinz, O.; Schiereck, D. Blockchain. *Bus. Inf. Syst. Eng.* **2017**, *59*, 183–187. [CrossRef]
4. Yaga, D.; Mell, P.; Roby, N.; Scarfone, K. Blockchain technology overview. *arXiv* **2019**, arXiv:1906.11078 .
5. Crosby, M.; Pattanayak, P.; Verma, S.; Kalyanaraman, V. Blockchain technology: Beyond bitcoin. *Appl. Innov.* **2016**, *2*, 71.
6. Merkle, R.C. Protocols for public key cryptosystems. In Proceedings of the 1980 Symposium on Security and Privacy, Oakland, CA, USA, 14–16 April 1980; IEEE Computer Society: Washington, DC, USA, pp. 122–133.
7. Benet, J. IPFS—Content Addressed, Versioned, P2P File System. *arXiv* **2014**, arXiv:1407.3561.
8. Publica. Publica.com, White Paper. 2017. Available online: https://www.bbc.com/news/technology-47810367 (accessed on 18 November 2023).
9. Plank, J.S. A tutorial on Reed–Solomon coding for fault-tolerance in RAID-like systems. *Softw. Pract. Exp.* **1997**, *27*, 995–1012. [CrossRef]
10. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. Bitcoin.org, White Paper, October 2008. Available online: http://www.bitcoin.org/bitcoin.pdf (accessed on 18 November 2023).
11. Bonneau, J.; Miller, A.; Clark, J.; Narayanan, A.; Kroll, J.A.; Felten, E.W. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In Proceedings of the 2015 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 17–21 May 2015; pp. 104–0121.
12. Gervais, A.; Karame, G.O.; Wüst, K.; Glykantzis, V.; Ritzdorf, H.; Capkun, S. On the security and performance of proof of work blockchains. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016; pp. 3–16.
13. Buterin, V. Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform. Ethereum Foundation, White Paper, January 2014. Available online: https://blockchainlab.com/pdf/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf (accessed on 18 November 2023).

14. Zheng, Z.; Xie, S.; Dai, H.-N.; Chen, W.; Chen, X.; Weng, J.; Imran, M. An overview on smart contracts: Challenges, advances and platforms. *Future Gener. Comput. Syst.* **2020**, *105*, 475–491. [CrossRef]
15. Wu, K.; Ma, Y.; Huang, G.; Liu, X. A first look at blockchain-based decentralized applications. *Softw. Pract. Exp.* **2021**, *51*, 2033–2050. [CrossRef]
16. Wang, Q.; Li, R.; Wang, Q.; Chen, S. Non-Fungible Token (NFT): Overview, Evaluation, Opportunities and Challenges. *arXiv* **2021**, arXiv:2105.07447.
17. Entriken, W.; Shirley, D.; Evans, J.; Sachs, N. ERC-721: Non-Fungible Token Standard. Ethereum Improvement Proposals, Created 2018-01-24. Available online: Https://eips.ethereum.org/EIPS/eip-721 (accessed on 18 November 2023).
18. Benisi, N.Z.; Aminian, M.; Javadi, B. Blockchain-based decentralized storage networks: A survey. *J. Netw. Comput. Appl.* **2020**, *162*, 102656. [CrossRef]
19. Daniel, E.; Tschorsch, F. IPFS and friends: A qualitative comparison of next generation peer-to-peer data networks. *IEEE Commun. Surv. Tutorials* **2022**, *24*, 31–52. [CrossRef]
20. Steichen, M.; Fiz, B.; Norvill, R.; Shbair, W.; State, R. Blockchain-Based, Decentralized Access Control for IPFS. In Proceedings of the 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Halifax, NS, Canada, 30 July–3 August 2018; pp. 1499–1506. [CrossRef]
21. Protocol Labs. Filecoin: A Decentralized Storage Network. Protocol Labs, White Paper, July 2017. Availbale online: https://filecoin.io/filecoin.pdf (accessed on 18 November 2023).
22. Wilkinson, S.; Lowry, J. *Metadisk: Blockchain-Based Decentralized File Storage Application*; White Paper; Storj Labs Inc.: Atlanta, GA, USA, 2014.
23. Vorick, D.; Champine, L. *Sia: Simple Decentralized Storage*; White Paper; Nebulous Inc.: Boston, MA, USA, 2014.
24. Williams, S.; Diordiiev, V.; Berman, L.; Uemlianin, I. Arweave: A Protocol for Economically Sustainable Information Permanence. arweave.org, Tech. Rep., 2019. Available online: https://arweave.org/yellow-paper.pdf (accessed on 18 November 2023).
25. Vyzovitis, D.; Psaras, Y. GossipSub: A Secure PubSub Protocol for Unstructured, Decentralised P2P Overlays. In *Proceedings of the Protocol Labs TechRep, PL-TechRep-gossipsub-v0.1-Dec19*; Protocol Labs: San Francisco, CA, USA, 2019 ; 8p.
26. Protocol Labs. libp2p: A Modular Network Stac. Protocol Labs, Software Repository, 2023. Available online: https://github.com/libp2p (accessed on 18 November 2023).
27. Maymounkov, P.; Mazieres, D. Kademlia: A peer-to-peer information system based on the xor metric. In Proceedings of the International Workshop on Peer-to-Peer Systems, Cambridge, MA, USA, 7–8 March 2002; Springer: Berlin/Heidelberg, Germany, 2002; pp. 53–65.
28. Kanani, J.; Nailwal, S.; Bhardwaj, A. *Polygon: A Scalable Framework for Building Ethereum-Compatible Blockchain Networks*; White Paper; Polygon Technology: Bengaluru, India, 2021.
29. Zhou, Q.; Huang, H.; Zheng, Z.; Bian, J. Solutions to scalability of blockchain: A survey. *IEEE Access* **2020**, *8*, 16440–16455. [CrossRef]
30. Kalodner, H.A.; Goldfeder, S.; Chen, X.; Weinberg, S.M.; Felten, E.W. Arbitrum: Scalable, private smart contracts. In Proceedings of the USENIX Security Symposium, Baltimore, MD, USA, 15–17 August 2018.
31. Neon Labs. Neon EVM set to go live on mainnet: Welcome to a new era of Ethereum scalability on Solana. Medium, Article, November 2022. Available online: https://medium.com/neon-labs/neon-evm-set-to-go-live-on-mainnet-welcome-to-a-new-era-of-ethereum-scalability-on-solana-63b25bcc77a3 (accessed on 18 November 2023).
32. Yakovenko, A. *Solana: A New Architecture for a High Performance Blockchain*; White Paper; Solana Labs: San Francisco, CA, USA, 2017.
33. Kiayias, A.; Panagiotakos, G. Speed-Security Tradeoffs in Blockchain Protocols. In *Cryptology ePrint Archive*; Paper 2015/1019; 2015. Available online: https://eprint.iacr.org/2015/1019 (accessed on 18 November 2023).
34. ChainSpect. Blockchain Transaction Per Second (TPS) Metrics. ChainSpect, Dashboard, December 2023. Available online: Https://chainspect.app/dashboard/tps (accessed on 18 November 2023).
35. CoinTool. Cryptocurrency Analytics and Tools. CoinTool, Dashboard, December 2023. Available online: https://cointool.app/dashboard (accessed on 18 November 2023).
36. Gottsegen, W. NFT Forgeries Aren't Going Away. *Coindesk*, December 2021. Available online: https://www.coindesk.com/layer2/2021/12/20/nft-forgeries-arent-going-away/ (accessed on 18 November 2023).
37. Ferdous, M.S.; Chowdhury, F.; Alassafi, M.O. In search of self-sovereign identity leveraging blockchain technology. *IEEE Access* **2019**, *7*, 103059–103079. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.