FinTech

MDPI

Article

A Low-Storage Blockchain Framework Based on Incentive Pricing Strategies

Po-Han Ko [1,2], Yu-Ling Hsueh [1,2,*] and Chih-Wen Hsueh [3]

[1] Department of Computer Science & Information Engineering, National Chung Cheng University, Chiayi 621301, Taiwan
[2] Advanced Institute of Manufacturing with High-Tech Innovations (AIM-HI), National Chung Cheng University, Chiayi 621301, Taiwan
[3] Department of Computer Science and Information Engineering, National Taiwan University, Taipei 10617, Taiwan
* Correspondence: hsueh@cs.ccu.edu.tw; Tel.: +886-5-272-0411

Abstract: Nowadays, blockchain bloat is an endangering issue caused by inefficient transaction storage mechanisms. Based on the Distributed File System (DFS), the blockchain network can reduce the local storage to solve the blockchain bloat problem. However, storing all blocks on DFS is not durable or scalable. Hence, classifying blocks into hot and cold was adopted in previous works. The blockchain nodes can reduce the time consumption and storage consumption by storing hot blocks locally. However, the previous works are not able to periodically check block integrity and do not provide a reward mechanism to encourage DFS system nodes to store blocks. We extend previous works based on the InterPlanetary File System (IPFS) and design an innovative scheme to incentivize IPFS nodes. The IPFS nodes are regulated with smart contracts and behave under the pricing strategy controls to increase profit. By adopting proof of retrievability, we guarantee the integrity of the blocks. Further, the redundant scheme extends our pricing strategy to improve the durability of our proposed framework. A load-balancing pricing strategy and a general pricing strategy are provided in the framework to reward the DFS nodes. Extensive experiments are presented to demonstrate that the latency and throughputs of our model are competitive, while still maintaining data integrity in the system. The additional increased throughput takes only 0.167% of that produced by the original Bitcoin and the upload latency takes only 6.67% of the mining time of the Bitcoin Mainnet. Furthermore, our load-balancing pricing strategy achieves the effectiveness to ensure the redundancy of blocks and reduces the overall storage consumption up to 97% using the load-balancing pricing strategy, compared to the non-load-balancing pricing strategy.

Keywords: blockchain; blockchain bloat; smart contract; load balancing; proof of retrievability

check for updates

Citation: Ko, P.-H.; Hsueh, Y.-L.; Hsueh, C.-W. A Low-Storage Blockchain Framework Based on Incentive Pricing Strategies. *FinTech* **2022**, *1*, 250–275. https://doi.org/10.3390/fintech1030020

Academic Editor: Shyan-Ming Yuan

Received: 18 July 2022
Accepted: 27 August 2022
Published: 6 September 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Blockchain [1], which maintains a decentralized distributed digital ledger with a protocol requiring a block architecture, was first announced by Satoshi Nakamoto in the Bitcoin white paper of 2008. Various researchers have worked with blockchain and have provided different chains, such as Bitcoin, Ethereum [2], and Hyperledger [3]. The blockchain mechanism guarantees that the system can not be controlled by a specific organization or government. Blockchain technology is applied in many different fields, including e-voting [4,5], green energy [6,7], the banking and financial services industry [8,9], the Internet-of-Things (IoT) [10,11] and supply chain management [12–14]. Blockchain has increasingly been studied in recent years. Change et al. [15] analyzed a serious of literature reviews on blockchain technologies and their applications in various fields. In [16], Change et al. designed a blockchain Newsvendor model with the optimal adoption considering both profit optimization and adoption cost. Furthermore, a modeling framework was provided to serve as a guideline for various application domains. Similarly, in [17], the

impact of blockchain technology on supply chain management was investigated to study the information flow. In particular, the proposed model is built for industry to maximize the profit and a generic framework is provided for blockchain technology design.

Blockchain transactions, especially Bitcoin, are formed into blocks and are stored in local storage. For example, Bitcoin nodes store all transaction data into local LevelDB. However, there is an endangering issue in the blockchain storage mechanism. For instance, each Bitcoin node has to store whole all transaction blocks in its local storage, requiring over 300 GB [18], which is shown in Figure 1. It is hard to promote a new node to join the chain if it does not have enough local storage. Some researchers have provided solutions. In [19,20], their models adopt Byzantine Fault Tolerance (BFT) and Reed–Solomon coding to reduce the overall storage consumption in the network. Data blocks are encoded through Reed–Solomon coding into chunks and are distributed to nodes. Each node does not need to store an entire block in its storage but can decode the block by retrieving the chunks from the nodes. Through the BFT mechanism, the encoding messages are broadcast to all nodes. The adoption of encoding messages ensures the lower bound of the number of storing service nodes. Their model reduces the overall storage consumption from $O(n)$ to $O(1)$. However, they constructed their model based on the permissioned blockchain, which is not suitable for public blockchain.

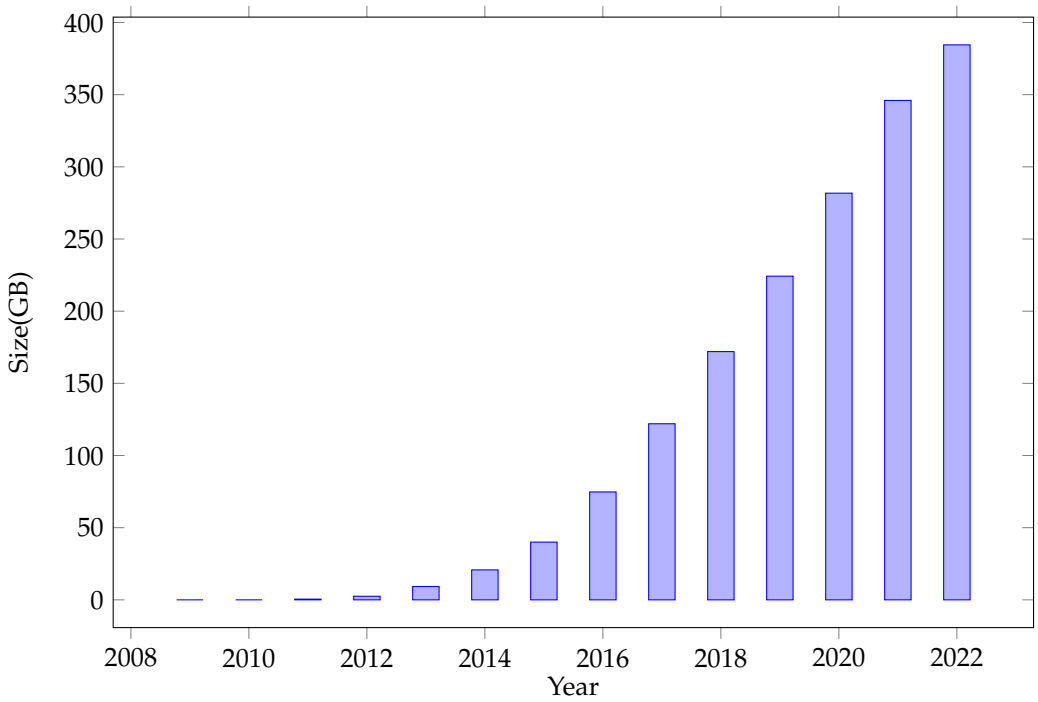

Figure 1. The size of the Bitcoin blockchain from 2009 to 2022.

Other researchers have investigated different solutions to the previous problem by applying Distributed File Systems (DFS). Chou et al. [21] proposed a scheme that classifies the hot data and cold data in Bitcoin. Their scheme stores cold data on the InterPlanetary File System (IPFS) and hot data on the local storage system. Zheng et al. [22] proposed an IPFS-based blockchain data storage model in which all transactions are stored in IPFS. The miners in the network verify the received transactions and save the verified transactions into IPFS. The transaction hash is packed into a new block. Subsequently, the new block is broadcast and verified by other miners. The miners append the new block to the blockchain after verifying it. The research mentioned above only focuses on the solution to reduce the transaction data on the local storage. However, for the DFS solutions, there are three challenges that the previous works do not solve. First of all, these solutions do not guarantee the integrity of data that is stored on DFS for a certain period. These

phases are not formulated to execute sequentially, which may incur a synchronous issue. However, our framework becomes more stable after being performed for a period of time. Secondly, it is unsafe to trust the storage providers without a vetting process. Lastly, these solutions do not provide a solution to the load-balancing problem. As a result, the storage servers prefer to store much more popular blocks and make multiple replications in the network. This may cause some unpopular blocks to be lost and there may be consensus failure in the network. In this work, a novel mechanism is proposed to distribute blocks to the IPFS network with auditing and reward for a certain period of time to overcome three challenges. The mechanism integrates the reputation table to reduce the risk of tampering with the blocks or malicious offline behavior. Furthermore, we propose our pricing policy and provide a way to reward the nodes and regulate the replications in the network. In summary, the significant contributions of this work are listed as follows:

1. We propose a novel storage reduction mechanism executed periodically to audit the integrity of data stored on the IPFS nodes and reward the IPFS nodes. We integrate a smart contract into our system to control storing blocks and auditing. By adopting our smart contract, IPFS nodes are not likely to manipulate blocks or go offline to prevent failure, which damages their profits during the challenging progress. With the proof of retrievability, we effectively ensure the integrity of blocks in low-time consumption. This mechanism maintains the consensus of the blockchain and provides scalability of the blockchain. Furthermore, we limit the nodes to uploading duplicated blocks to decrease the bandwidth and storage consumption;
2. A redundant scheme is adopted to replicate the blocks and decrease the risk of losing blocks in our pricing strategies. The IPFS nodes are encouraged to involve the replication of blocks so as to improve the tenacity of our framework. This scheme improves the durability of blocks. However, it is possible that nodes that are responsible for storing the same block go offline. Therefore, as a last resort, we select full nodes in the contract to improve the availability of blocks to prevent the failure of the node. The full node receives the most rewards to remain online;
3. We have designed pricing strategies to incentivize the IPFS nodes. This paper provides two pricing strategies: a balancing strategy and a non-balancing strategy. The non-balancing pricing strategy provides the IPFS nodes with more rewards when storing more blocks. The balancing pricing strategy controls the IPFS nodes to provide storage and bandwidth and restricts the number of IPFS nodes that store the same block in their storage. These two strategies provide different levels of durability and consume different storage sizes. We evaluate the access full node frequency of two pricing strategies in our experiments. This framework can be adopted as a general solution and applied in various fields, such as e-voting, IoT, and banking and financial service industries.

The rest of the paper is organized as follows. Section 2 reviews the related work. The assumptions and the tools of our model are described in Section 3. The system framework is proposed in Section 4. In Section 5, various experiments were conducted to show the effeteness of our system. Finally, we conclude the paper in Section 6.

2. Related Work

In this section, the previous works are briefly introduced in three parts: scalable blockchain, data integrity and data redundancy.

2.1. Scalable Blockchains

To achieve scalability of blockchain, Dai et al. [23] proposed a network encoding distributed storage (EN-DS) framework concept to reduce the storage consumption in the blockchain network. Blocks are encoded through NC-DRDS and NC-RLDS, and architecture distributed to all nodes. With their scheme, the storage consumption is reduced to $1/k$, where k represents the number of split packets that can recover the original block, and the bandwidth is reduced to $1/n$, where n represents the number of all split packets.

Janitha et al. [24] proposed a scheme to reduce the data bloating problems caused by the IoT devices in their food supply system. They define three layers in their framework: storage, blockchain and data. The actual data are stored in the distributed database in the distributed File System Networks (DFSNs), which belong to storage layer. The hash of the data in the distributed database will be put into the smart contract on the blockchain layer. Their scheme provides storage recycling to identify expired products and execute storage recycling once a week. The construct of the blocks is similar to the one in [22] and recorded the product information. This scheme provides a way to distribute their data into the distributed file system and to reduce the storage consumption of the network. However, it is only suitable for private chains. It is thus necessary to figure out a public chain solution. Sohan et al. [25] designed a system using IPFS and dual-blockchain methods. IPFS was adopted as a secondary blockchain. The IPFS nodes store the transactions and put the IPFS Content-Identifier (CID) into raw blocks. The IPFS node that generates the raw block does not distribute the raw block to the whole network and generates the CID of the raw block. The IPFS nodes put the raw block's CID into blocks that broadcast in the main blockchain network. IPFS nodes are in charge of storing the blocks in this scheme. The storage consumption of the blockchain nodes decreases significantly. However, the incentive mechanism, which ensures the blocks' availability, is not proposed in their scheme. A summary is shown in Table 1.

Table 1. Literature Comparison of scalable blockchains.

	Pros	**Cons**
Qi et al. [19]	• Low storage consumption • Enhanced storage scalability	• Only for permissioned blockchain
Zheng et al. [22]	• Low storage consumption • IPFS-based framework	• Risk of losing transactions • Large bandwidth occupancy
Chou et al. [21]	• Hot and cold classifiers • IPFS-based framework • Low bandwidth occupancy	• Risk of losing bandwidth
Yin et al. [26]	• Load-balancing • Smart contract support • Local proof of storage	• Inaccessible public data

2.2. Data Integrity

When clients store data on a cloud server or distributed file system, the servers need to guarantee data integrity. Ateniese et al. [27] proposed a PDP scheme that allows clients to verify the file stored on the server without accessing the entire file from the server. The PDP scheme does not require high bandwidth, and clients do not have to store the metadata to verify the file stored by the server. However, their scheme does not prevent the server from losing the file after validating it.

A scheme based on blockchain was proposed to prevent losing files after validating them. In [28], Endolith was proposed to verify the file integrity. Endolith adopts a smart contract to store the audit data and store the file into the Hadoop distributed file system. The data owner uploads, retrieves and verifies his data through Endolith. The hash of the data is held in the smart contract and compared when the data owner retrieves or verifies it. Nevertheless, this model could not prevent the single point failure caused by a centralized service. Li et al. [29] developed a new PDP model based on blockchain with multiple replicas. Multiple replicas are generated by data owners and are uploaded to multiple cloud storage. They redesigned the PDP model based on pseudorandom functions with shorter parameters, enhancing efficiency. Although this model offers an efficient way to execute the PDP model, it does not provide a solution to frequently verify the data stored

on cloud storage. Xu et al. [30] designed a fair auditing scheme to verify data integrity for the user. This scheme contains a smart contract that guarantees data integrity on the network storage service provider through the commutative hash technique at the setup phase. The arbitrable smart contract provides a way to audit the data stored by the network storage service provider. Authority nodes maintain the network's blockchain, and users do not have to keep all transaction blocks. Xu et al. [31] proposed a deduplicate data auditing scheme based on blockchain to solve the overhead produced by uploading many duplicate data. The user will first check for duplicates on the blockchain to prevent duplicate upload data to the server. Once the comparable data have been uploaded, the owners have to prove they are the owners and send the proof to the storage provider. Storage providers verify the proof and add the user into the index table of the data if the verification is passed. This scheme efficiently reduces the overhead and executes without third-party authentication.

Tron et al. [32,33] developed a decentralized file system based on Ethereum. This system splits files into small pieces and spreads the file through a distributed pre-image archive. This system also provides an incentive mechanism for providing network bandwidth and storage. The incentive mechanism of bandwidth promotes the trade services between peers, and the incentive mechanism of storage incentivizes peers, guaranteeing long-term data preservation. Protocol Labs proposed Filecoin [34]. This framework provides a decentralized storage network that allows clients to store files by spending individual coins. The storage miner receives coins when it stores files for a certain period of time. Filecoin presents two schemes: Proof-of-Replication and Proof-of-Spacetime, to reward the storage miner. These schemes against storage miners are opportunistic to lose the file and receive the reward from the user. Yu et al. [35] developed a checking model to check the integrity of large continuous data. This model provides a data time sampling scheme to randomize a set of files to audit and combine the trapdoor delay function into proof of retrievability to reduce the time delay. The user executes the sampling function and computes the verification tag before uploading his files. The verification tag set includes every proof while storing the data. This model reduces the time delay of verifying the proof during storage and guarantees data safety.

To avoid the decreasing storage durability caused by the unbalancing data distribution, Yin et al. [26] proposed the scheme with financial incentives. The nodes that store the file are rewarded according to the number of data they store. Smart contracts were adopted to ensure the integrity of the data. The nodes need to prove the integrity of the data stored before the expiry time defined by the user. They designed an income function for nodes to achieve load-balancing. Different nodes compete to store the file to obtain rewards and are punished when the number of stored data blocks is over the balanced line. A balanced line is dynamically chosen according to the number of nodes and data in the network. Shen et al. [36] proposed a scheme to incentivize all network participants to collaborate on data sharing. The scheme adopts the Shapley value to evaluate the contributions of each participant to the network. They also proposed a reliable model to transfer critical data in the blockchain network. The miners in the network are rewarded after transferring data. This scheme encourages the participants to contribute more data to receive more rewards.

2.3. Data Redundancy

Storj Labs, Inc. presented Storj [37]. This framework presents a data-repair process to repair the data lost by the offline nodes and an auditing process to validate the integrity of the data. Storj adapts the erasure code to achieve necessary redundancy and increase the durability of data through repairing the data. This framework provides a protocol to ensure that users trust the storage server which manages to preserve the outsourcing data, and the storage server trusts that it can be paid by the user for allocating the storage and bandwidth for use. Liang et al. [38] provide a scheme to replicate data in the network 4.0 environments and it involves with the techniques related to industrial internet of things (IIoT), cloud computing, and cloud computing for industry. In their scheme, they design a model to recover the data in few nodes and reduce the tine complexity during the encoding

process. The cloud network recovers the data through the block and checks the hash stored on the blockchain network when one or multiple nodes fail. This scheme improves the durability of data stored on the blockchain-based industrial network.

3. Preliminary

This section introduces the necessary preliminaries and the tools utilized in our network.

3.1. Assumptions and Notations

The following assumptions which are used in our paper and the notations in Table 2 are defined in this section.

1. **IPFS nodes do not manipulate the content while passing the content to other nodes.** The IPFS is combined into our blockchain model in this paper. IPFS constructs a peer-to-peer network and distributes the network's content to provide a distributed database. The content is passed through several nodes while retrieving the content in the IPFS network. We assume that the IPFS nodes do not tamper with the content while passing it;
2. **IPFS nodes benefit themselves as much as they can.** IPFS nodes provide their own space to store the blocks from our chain and receive the reward according to the contributions. IPFS nodes ensure their benefit by storing more blocks in their space until the space is full;
3. **IPFS nodes should not be trusted.** In Filecoin [34], the Storage Miner is punished when it fails to provide the content stored on its local storage. However, the blockchain network could not afford to lose any blocks. When IPFS nodes joined the network to provide the storage service, users in the blockchain network could not justify the trustworthiness of the newly joined IPFS nodes. The reputation system and the pricing strategy based on this assumption are developed to audit IPFS nodes for users to ensure that they would not lose blocks for malicious reasons.

Table 2. Notations.

Symbol	Description
λ	Secret parameters to generate key_{mac}.
key_{mac}	Mac key generated by the user.
PoR	Proof of the retrievability scheme.
F	The file used in the proof of the retrievability scheme.
$\tilde{F}$	The file processed in $PoR.St$.
I	Subset of $[1, n]$ with a randomly selected q chunks index.
n	Number of chunks.
s	Number of sectors.
$name$	Random name of the file $\tilde{F}$.
$chal$	Challenge message of the block produced by user.
$Chal$	The set of cid_{chal}. $Chal$: $\{cid_i\}(i \leq i \leq number_{chal})$.
$interval$	A challenge interval.
Tag	A tag produced via $PoR.St$.
$number_{chal}$	Number of challenge blocks.
$proof$	Proof of the file $\tilde{F}$ generated by IPFS nodes.
blk_{cold}	A cold block.
blk_{latest}	The latest block containing the contract most of the time in the blockchain network.
cid	Identifier of the block in the IFPS network.
o	Number of blocks in the contract.
q	Number of elements picked from subset $[1, n]$.
m	Number of registered IPFS nodes.
w	Punishment factor.

Table 2. *Cont.*

Symbol	Description
l_m	Number of blocks stored in the IPFS node m.
c_i	Number of nodes that storing the block i, $i = 1, 2, 3, \ldots, l_m$.
$R_{nb}(l_m)$	Non-balancing reward of node m.
$R_b(l_m)$	Balancing reward of node m.
α	Number of the single block replica.
θ	Reward of storing blocks.
δ	Threshold of the balance strategy.

3.2. Proof of Retrievability

In our model, proof of retrievability (PoR) [39] is adopted to verify the integrity of the transaction stored in IPFS. keys are generated through the simple message-authentication codes (MAC) scheme, which is much more efficient than generating keys through RSA algorithm [40]. We formulate five five following key functions:

- $PoR.Keygen(\not\geq\not\leq) \rightarrow (key_{mac})$. The Keygen function generates the mac key key_{mac} of user by using the security parameters $\not\geq\not\leq$;
- $PoR.St(key_{mac}, F) \rightarrow Tag, \tilde{F}, name$. This St function takes the file F and mac key key_{mac} as input. The file F is split into n chunks by the size of sector s. The file F consists of each chunks $\{f_{i,j}\}(1 \leq i \leq n, 1 \leq j \leq s)$. Subsequently, we compute the $Tag : \{tag_{i,j}\}(1 \leq i \leq n, 1 \leq j \leq s)$ using the MAC function that takes the random file name $name$ and each chunks as the inputs. The Tag, the processed file $\tilde{F}$ and the random file name $name$ are output in this function;
- $PoR.Chal(key_{mac}, name, time_{limit}) \rightarrow chal$. The challenge function takes a random subset I in the set $[1, n]$ with l elements as $chal$ to challenge the storage provider;
- $PoR.Prove(chal, \tilde{F}, Tag) \rightarrow proof$. The response algorithm takes $chal$ and the file $\tilde{F}$, which the storage provider stores. The storage provider decodes the file $\tilde{F}$ into $\{f_{i,j}\}(1 \leq i \leq n, 1 \leq j \leq s)$ and computes the proof of the $chal$ with file chunks $f_{i,j}(i \in I, 1 \leq j \leq s)$ with Tag. The storage provider returns the $proof$ to the users;
- $PoR.Verify(chal, proof, name) \rightarrow \{0, 1\}$. The verify algorithm takes $chal$, $name$ and the $proof$ as the inputs and the $\{0, 1\}$ as the output. If the storage provider passes verification, where all q chunks are validated, the algorithm outputs 1. Otherwise, the algorithm outputs 0.

3.3. Role Definitions

Before introducing our mechanism, three roles—user, IPFS node and miner—are defined as follows.

- *User* has the requirements of achieving the blocks data and uploads the cold blocks which are not required by the IPFS nodes;
- *IPFS node* owns multiple storage devices and is paid by storing the blocks data and providing users with bandwidth to achieve the blocks. This role has the responsibility to prove the integrity of blocks. Therefore, if the IPFS nodes lose the transaction data pieces, they lose their rewards and are not allowed to store blocks based on the smart contract. IPFS nodes are part of the users;
- *Miner* participates in mining blocks. Miner may also be a user.

3.4. Hot and Cold Classifiers

In [21], Chou et al. proposed a solution to classify cold blocks and hot blocks. They designed two algorithms to classify the hot and cold blocks during the initial phase. According to their discovery, the most nearly genesis blocks and recent blocks are high-frequency blocks. We adapt their approach in our model to classify the hot and cold blocks in the initial phase. Two algorithms were defined in this scope, the front-k algorithm

and the recent-k algorithm. In their definition, F_k represents the maximum index of the blocks which is near the genesis block and R_k represents the minimum index of the most recent blocks. User stores those blocks that block index from 0 to F_k and from R_k to Blk_{total}, representing the highest index of blockchain. The users can define these parameters. The front-k algorithm and recent-k algorithm are shown in Equation (1).

$$blk(i) = \begin{cases} Hot, & \text{if } 0 \leq blk(i) \leq F_k, \\ Hot, & \text{if } R_k \leq blk(i) \leq Blk_{total}, \\ Cold, & otherwise. \end{cases} \tag{1}$$

After the initial block download (IBD) phase, some hot blocks might be cold blocks for users. Therefore, the working-set is adapted and integrated with the classifier. When the number of blocks stored on the user's local storage reaches the maximum of the working-set storage, the low-frequency blocks are seen as cold blocks and are stored on IPFS. The pseudocode of inserting blocks into and reading blocks from the working-set is shown as Algorithms 1 and 2, respectively. Algorithm 1 takes the most recent block blk_{latest}, the blocks set in working-set $Blk_{workingset}$, and the cold block blk_{cold} as inputs. Lines 2 to 5 present the order process with the working-set when a new block is received. The most frequent block is placed at the top of the working-set table. Lines 6 to 9 present the process of popping-out the low-frequency blocks from the woringset. If the working-set table is full, these lines are executed. The last block of the working-set table is seen as the low-frequency block and is prepared to upload to IPFS. The cold block is output and is ready to be uploaded to IPFS.

Algorithm 1: InsertBlockIntoWorkingSet(): Inserts new block into the working-set table.

input : blk_{latest}: the most recent block; $Blk_{workingset}$: the blocks stored in the working-set
output: blk_{cold}: the cold block need to be distributed

1 $blk_{now} = blk_{latest}$;
2 **for** $index = 1$ *to* $Blk_{workingset}.Maximumsize$ **do**
3 $\quad Blk_{workingset}[index] = blk_{now}$;
4 $\quad blk_{now} = Blk_{workingset}[index + 1]$;
5 **end**
6 **if** $Blk_{workingset}.size = Blk_{workingset}.Maximumsize$ **then**
7 $\quad blk_{cold} = Blk_{woringset}.last$;
8 $\quad Blk_{workingset}.last.pop$;
9 $\quad$ ▷ Pop-out the low-frequency block from the working-set table.
10 return blk_{cold};

Algorithm 2 takes the index of the block $Index_{blk}$ as input. If the block does not exist in the working-set, the user needs to obtain the block from the IPFS network, shown in Line 3. Otherwise, the user needs to find the blocks in the working-set table and reorder the working-set table caused by the reading behavior. The reading and reordering blocks procedure is shown in Lines 4 to 10. Then the block is output at the end of this algorithm. With the working-set, users can free their space by distributing cold blocks to the IPFS nodes. Hence, we adopt this classifier in our model.

Algorithm 2: ReadBlockFromWorkingSet(): Read block from the working-set and update the order

input : $Index_{blk}$: the index of the block
output: blk_{index}: the block needed by user

if $Blk_{workingset}.notexist(Index_{blk})$ **then**
  $blk_{index} = GetFromIPFS(Index_{blk})$;
  return blk_{index};
$p = Blk_{workingset}.findIndex(Index_{blk})$;
$blk_{now} = Blk_{workingset}[p]$;
for $index = 1$ *to* p **do**
  $blk_{old} = Blk_{workingset}[index]$;
  $Blk_{workingset}[index] = blk_{now}$;
  $blk_{now} = blk_{old}$;
end
$blk_{index} = Blk_{woringset}[1]$;
return blk_{index};

4. System Framework

This section describes our storage reduction mechanism and the pricing strategy. Our framework is briefly shown in Figure 2. Our framework is based on the public chain which adopts Proof-of-Work (PoW) as a consensus mechanism. When the new block is generated, the user receives the new block from the blockchain network via the Peer-to-peer (P2P) protocol in the blockchain core. The core classifier classifies the hot blocks and cold blocks. The hot blocks are stored at the local storage and maintained through the working set by updating the order of the blocks. The cold blocks need to be distributed to the IPFS nodes. However, as mentioned in Section 3.1, IPFS nodes benefit themselves as much as possible and should not be trusted before auditing. Therefore, smart contracts are adopted to reward and regulate IPFS nodes. Different IPFS nodes would join the smart contract as a coalition. We further describe the smart contract in Section 4.1.

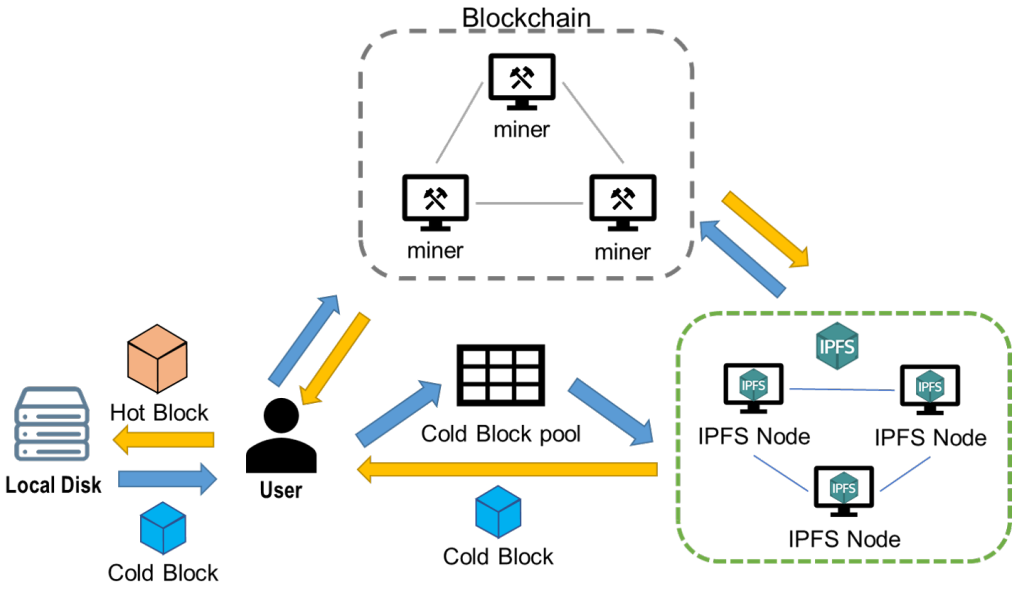

Figure 2. The concept of our framework.

At the end of this section, our pricing strategy and the reputation mechanism described in Section 4.2 are introduced. To encourage the IPFS nodes to store the blocks appropriately, the pricing strategy are applied in our model. We combine the redundancy with a factor

for replicating the blocks to multiple IPFS nodes to decrease the risk of losing the blocks in our pricing strategy. The details of our pricing strategy are described in Section 4.2.1. Furthermore, the misbehavior of IPFS nodes endangers blocks saving. We integrate the reputation system for each user and select full nodes to store all blocks. The reputation system is integrated with the blockchain core to prevent deploying the user's blocks to misbehavior contracts. The details are proposed in Section 4.2.2.

4.1. Smart Contract

The class diagram of our smart contract is shown in Figure 3. There are four phases to deploy and retrieve blocks: the setup phase, upload phase, challenge phase and retrieval phase.These phases are not formulated to execute sequentially, which may incur a synchronous issue. However, our framework becomes more stable after being performed for a period of time. Our contract is inherited from ORC20, which is similar to ERC20 [41] in Ethereum and is implemented in our contract. Six major functions are designed in our contract: $save\_blocks()$, $remove\_blocks()$, $proof\_blocks()$, $user\_sign\_up()$, $user\_offline()$ and $calculate\_reward()$. The function $save\_blocks()$ records the blocks saved by IPFS nodes. In this function, $PoR.Verify()$ is adopted to verify IPFS nodes' proof. Next, $remove\_blocks()$ is used to remove the registration of storing some blocks, which means that IPFS nodes give up the right to store these blocks and receive the reward of these blocks. Then the $proof\_blocks()$ function is used to prove the blocks in the challenge phase. IPFS nodes sign up the contract through the $user\_sign\_up()$ function and sign out the contract through the $user\_offline()$ function when they no longer store the blocks. At last, the $calculate\_reward()$ function is used to calculate the reward of all IPFS nodes. This function implements our pricing strategy in Section 4.2.1. The details of each phase are described as follows.

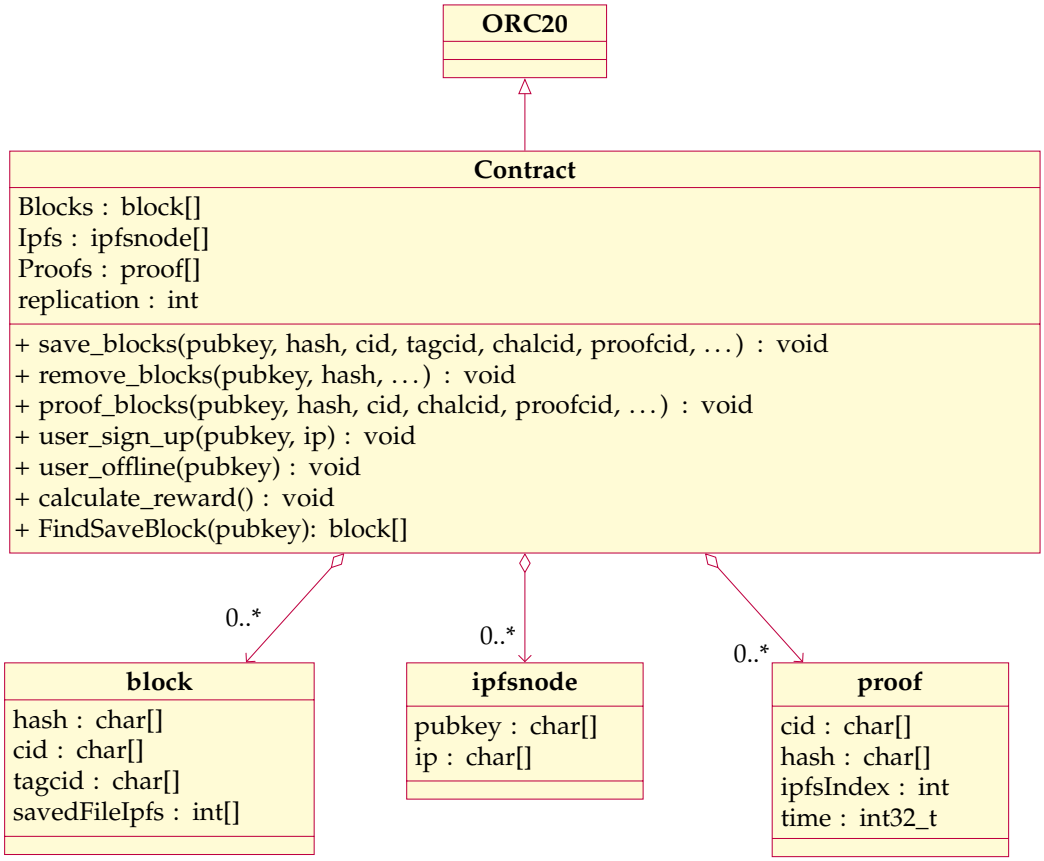

Figure 3. The class diagram of our smart contract.

4.1.1. Setup Phase

In the setup phase, users must know the available IPFS nodes to distribute the blocks. Therefore, IPFS nodes have to sign up to inform their online status to users. There are two ways to sign up: deploy a new contract or join an existing contract. As we mentioned before, the smart contract can be seen as an IPFS nodes coalition. IPFS nodes in the same contract can share the reward equally. For IPFS nodes, the fewer IPFS nodes signed up in the same contracts, the more rewards they can gain. However, we integrate the reputation system to distribute the blocks to those high-reputation coalitions so it is much riskier for IPFS nodes to deploy a new contract. Blockchain users store the IPFS nodes' information in local storage and prepare to distribute their blocks to the IPFS nodes.

Before IPFS nodes start to store transactions for gaining rewards, they submit a transaction to the blockchain to sign up on the network. IPFS nodes need to provide proof of local space, which they claim in the transaction. The setup transaction is broadcast to each blockchain node and is validated. The pseudocode of the setup contract is shown as Algorithm 3. Line 1 presents the strategy of IPFS nodes. If the strategy of the IPFS node is the coalition, the IPFS node finds the best profit contract to join. Otherwise, creating a new contract in Line 4 is another option for IPFS nodes. Once the contract is created or selected, the IPFS node signs up and sends the transaction to the contract, which is presented in Line 5.

Algorithm 3: Setup(): the IPFS node deploys a new contract or signs up to an existing contract.

input : *pk*: public key; *strategy*: the strategy of IPFS nodes: including joining the coalition or creating a new contract; *ipAddress*: the IP address of IPFS nodes

output: *contract*: the contract to send transaction or deploy

```
1 if strategy = StrategyOption.coalition then
2 |   contract = chooseBestProfitContract()
3 else if strategy = StrategyOption.newContract then
4 |   contract = Contract.new()
5 contract.Signup(pk, ipAddress)
6 return contract
```

4.1.2. Upload Phase

After a user receives the blocks from the blockchain network and the cold blocks are classified by a hot-cold classifier, the user then needs to upload the cold blocks to the IPFS network. The pseudocode of users and IPFS nodes are shown in Algorithms 4–6. The sequence diagram of this phase is shown in Figure 4.

Before the user uploads his blocks to the IPFS network, the user checks up on the contracts already uploaded to the blockchain. If there is a contract containing the cold block's hash, other users have already uploaded that cold block. As a result, the users are involved in validating the blocks instead of uploading their blocks. However, if the validation fails, the user has to pick up another contract to store the blocks. Furthermore, the reputation of the failed contracts can be decreased. The reputation factors influence the behavior of users. The details are described in Section 4.2.2. Once the user chooses the contract, the user notifies the IPFS nodes to store the specific block. The user sends the message including the block hash, the *cid* of the blocks, the *cid* of the challenge, and the *cid* of the tag files to the IPFS nodes via the P2P network.

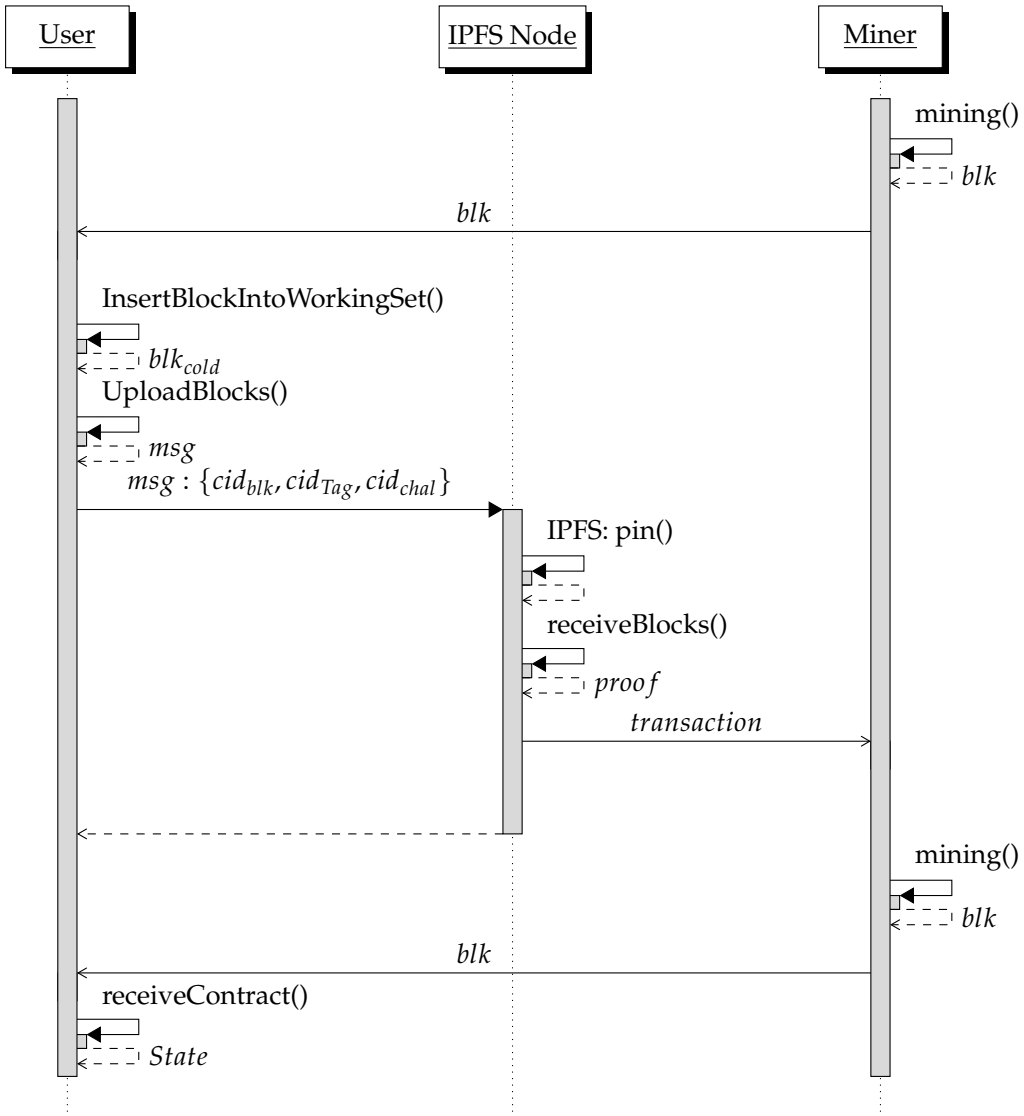

Figure 4. The sequence diagram of the upload phase.

When IPFS nodes receive the user's message to gather the cold blocks from the IPFS network, they generate the proof and push it into the IPFS network. Although uploading the proof into the smart contract is another solution to validate the integrity, the size of the CID is much smaller than the whole proof. After validating, the proofs are no longer necessary for the future and are enabled to forget to reserve more free storage space for storing other blocks. Therefore, the IPFS nodes send the contract transactions with the CID, the hash of the block and the CID of proof to the miner for packing them into new blocks. When other IPFS nodes receive the contract transaction, they try to request the blocks they have not stored before from the IPFS node and store them for rewards. Eventually, this block is stored in duplicate in different IPFS nodes. The details of this process are described in Section 4.2.1. During this process, the user receives the contract transaction from the network and the hash of the block is compared with the hash of the cold blocks at the local storage. If the hashes of two blocks are the same, users execute the proof validation process. The block is to be deleted after the validation is succeeded. Otherwise, the user has to request other contracts to store the block.

Algorithm 4 shows the pseudocode of the user uploading their blocks to IPFS nodes.

Algorithm 4: UploadBlocks(): User uploads the cold blocks

input : Blk_{cold}: the set of cold block to upload; *Parameters*: the parameters which are needed for integrity validation; *Contracts*: the list of contracts; *Reputations*: the list of reputation

output: *msg*: the message to be sent to the IPFS nodes;

```
1  for index = 0 to Contracts.size() do
2      msg = [];
3      if Repuatations.find(Contract[index].hash).value > 0 then
4          foreach blk_cold ∈ Blk_cold do
5              if contract.exists(blk_cold.hash) then
6                  ProduceChallenge();
7                  break;
8              else
9                  Tag, F̃, name = PoR.St(key_mac, blk_cold);
10                 chal = PoR.Chal(key_mac, name, time_limit);
11                 cid_blk = uploadIPFS(blk_cold);
12                 cid_Tag = uploadIPFS(Tag);
13                 cid_chal = uploadIPFS(chal);
14                 msg.append(Contract.nodes, cid_blk, cid_Tag, cid_chal);
15             end
16         end
17     end
18 end
19 return msg;
```

The user selects the contract with a reputation of not less than 0 in Line 3. If the user finds that the block already exists in the contract, the user produces the challenge and goes ahead into the challenge phase in Line 5 and Line 6. Otherwise, the user produces the *Tag* in Line 9 and the *Chal* in Line 11. After uploading the *Tag*, *Chal* and *blk*, the user appends the cid of these files into *msg* in Line 14 and sends it to the IPFS nodes. In Algorithm 5, Line 5 shows that the IPFS nodes gather the block from IPFS. One of the specific features of the IPFS network is that all files are stored single. In other words, only one user stores a unique file. However, IPFS provides some solutions to prevent such a single failure. The IPFS node creates a cache file when providing files. It is unsafe to relate to the cache to prevent the blocks from getting lost in our situation. We use the Pin function to gather the file at the local storage and provide them for the whole IPFS network. According to the user challenge, Line 6 generates the proof of blocks through the *PoR.chal* protocol. Hence, the proof is stored in the IPFS network in Line 7 and is used to prove the block in the contract. In Line 13, the IPFS node dynamically stores the blocks into local storage; the details are described in Section 4.2.1.

When the user receives the new transaction which contains the contract, the user executes Algorithm 6. The user conducts the contract with the contract function *save_blocks*() in Line 3. Then user verifies the proofs in Line 9. If the verification passes, the user finds the block in the set of the already deployed list. If the block is in the set, the user increases or decreases the reputation of the contract in Lines 12 and 14. Otherwise, the users save the blocks on disk in Line 17.

Algorithm 5: ReceiveBlocks(): IPFS node receives the storing request

input : *msgs*: the message of storing data request;*pubkey*: the public key of the IPFS node;
output: *transaction*: the contract transaction

transaction.command = "save_blocks";
transaction.pubkey = *pubkey*;
$i = 0$;
foreach $msg \in msgs$ **do**
  $blk_{cold}, Tag_{blk}, Chal_{blk}$ = gatherFromIPFS($msg.cid_{blk}, msg.cid_{Tag}, msg.cid_{chal}$);
  $proof_{blk} = PoR.prove(blk_{cold}, Tag_{blk}, Chal_{blk})$;
  $cidProof = uploadIPFS(proof_{blk})$;
  $transaction.Blocks[i].cid = cid$;
  $transaction.Blocks[i].proofcid = cidProof$;
  $transaction.Blocks[i].hash = Blk_{cold}.hash$;
  $transaction.Blocks[i].chalcid = msg.cid_{chal}$;
  $transaction.Blocks[i].tagcid = msg.cid_{Tag}$;
  dynamicStoringBlocks();
end
return *transaction*;

Algorithm 6: receiveContract(): User receives the contract transaction.

input : *transaction*: the contract transaction; *contract*: the contract whose address and the contract address of the transaction are the same;
output: $contract_{updated}$: the contract which has already been modified;

$contract_{updated} = contract$;
if $transaction.command == "save\_blocks"$ **then**
  $contract_{updated}$.save_blocks($transaction.pubkey$, $transaction.Blocks[0].hash$, $transaction.Blocks[0].cid$, $transaction.Blocks[0].tagcid$, $transaction.Blocks[0].chalcid$, $transaction.Blocks[0].proofcid$, ..., $transaction.Blocks[i].hash$, $transaction.Blocks[i].cid$, $transaction.Blocks[i].tagcid$, $transaction.Blocks[i].chalcid$, $transaction.Blocks[i].proofcid$);
  $hash_{alreadyDeploy}, cid_{alreadyDeploy}$ = FindAlreadyDeployBlk();
  foreach $block \in transaction.Blocks$ **do**
    $proof$ = GetBackFromIPFS($block.proofcid$);
    $name$ = FindOutBlockName($block.hash$);
    $chal$ = GetBackFromIPFS($block.chalcid$);
    $result = PoR.Verify(chal, proof, name)$;
    if $block.hash \in hash_{alreadyDeploy}$ **then**
      if $result == 1$ **then**
        IncreaseContractReputation($contract.hash$);
      else
        DecreaseContractReputation($contract.hash$);
      end
    else
      SaveToDisk($block$);
    end
  end
end
return $contract_{updated}$;

4.1.3. Challenge Phase

To make the challenge phase clearer, Figure 5 shows the procedure of this phase.

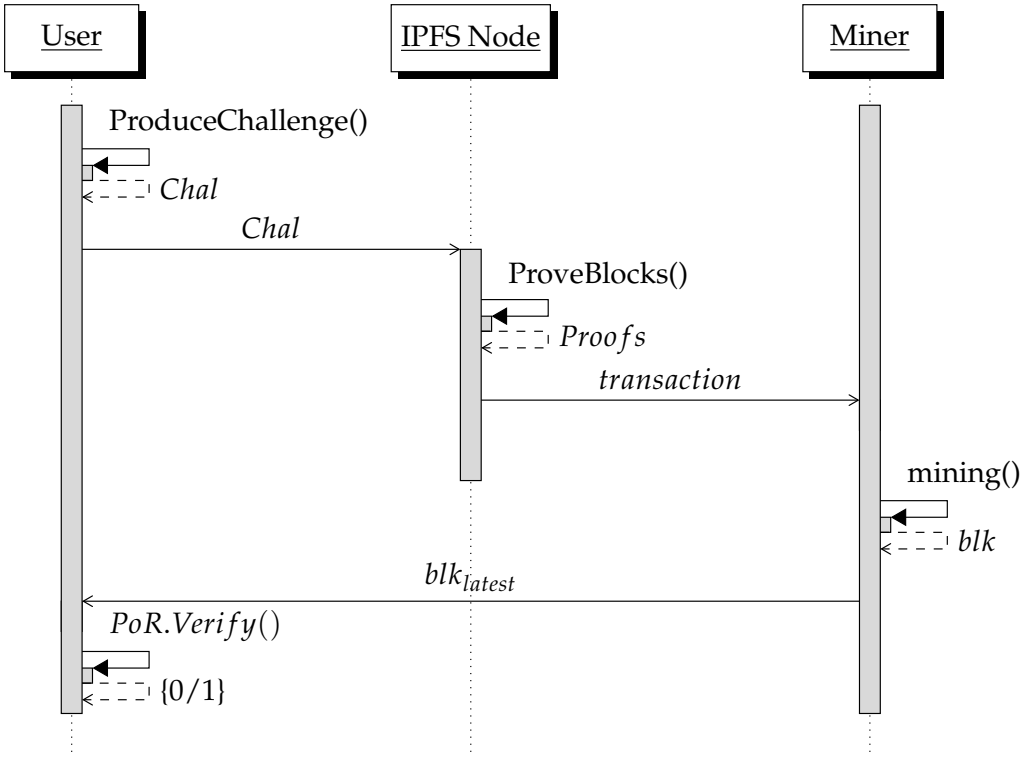

Figure 5. The sequence diagram of the challenge phase.

After the user uploads the blocks to the IPFS node, the IPFS node has to be validated by the user through the *PoR* protocol to commit the block integrity. Therefore, the user has to generate the *chal* to that IPFS node, which already stored the blocks and declared the store action in the smart contract. In this phase, we define *interval*, which represents the challenge interval, and $number_{chal}$, which represents the number of challenging blocks. For example, if *interval* equals 1000 and $number_{chal}$ equals 50, the user randomly selects 50 blocks to validate every 1000 blocks. In each block which needs to be challenged, the user randomly chooses l pieces of the blocks and produces the *Chal*: $\{chal_i\}(1 \leq i \leq number_{chal})$ through the *PoR.Chal* protocol. Once the user generates the *chal*, the user sends the *chal* to the IPFS node via P2P. During the P2P connection, there are two ways to send the challenges and proofs: sending them directly via P2P or uploading them to the IPFS network. In this scope, the second solution is much better. The hash of the IPFS network is much smaller than the size of the challenge or proofs, so the bandwidth of the message is much smaller. Furthermore, other users could also prove the blocks by retrieving the challenge and the proof with the CID hash. After receiving the *chal* from the user, the IPFS node needs to prove the block $\tilde{F}$ corresponding to the *chal* through the *PoR.Prove* protocol and produce the $proof$. The $proof$ is then uploaded to the smart contract and is validated by every blockchain node. If the $proof$ has been proved successfully, the user increases the reputation of the contract to which the IPFS node belongs and may select this contract the next time. Otherwise, the user decreases the reputation of the IPFS nodes according to our reputation strategy.

In Algorithm 7, the user randomly selects $number_{chal}$ blocks in Line 2. Suppose that the rand() function generates a unique random number from a given set. Line 5 finds the *name* of this block from local storage with the hash $hash_{index}$. Then, Line 6 generates the *chal* of this block through *PoR.Chal*. When the IPFS nodes receive the *Chal* from the users, they conduct Algorithm 8. IPFS nodes find the *Tag* and *blk*, which are related to *hash*. Then

$proof$ is computed through $PoR.Prove$ with Tag and blk. Once the $proof$ is received from the IPFS nodes, the user conducts Algorithm 9. The user verifies the $Proofs$ in Line 5 and updates the reputations according to the $result$ in Line 7 or Line 9.

Algorithm 7: ProduceChallenge(): User produces challenge.

input : $node_i$: the node which is the target of the challenge, $1 \leq i \leq m$; $Hash_i$: the hash set of blocks which are stored by the IPFS node i; $number_{chal}$: the number of challenging blocks; key_{mac}: the mac key generated by the user; $State$: the contract state;

output: $Chal$: the challenge set

1 **for** $j = 0, 1, \ldots,$ $number_{chal}$ **do**
2 $index$ = rand(0, $Hash_i$.length);
3 $hash_{index}$ = $Hash_i[index]$;
4 $Hash_i$.pop($index$);
5 $name_j$ = FindOutBlockName($hash_{index}$);
6 $chal_{index} = PoR.Chal(key_{mac}, name_j,$ timeNow());
7 $cid_{chal} = uploadIPFS(chal_{index})$;
8 $Chal$.append($hash_{index}, cid_{chal}$);
9 **end**
10 return $Chal$;

Algorithm 8: ProveBlocks(): IPFS node proves the challenge.

input : $Chal$: the set of $chal$ produced by user in Algorithm 7;

output: $Proofs$: the set of $proof$ generated through $PoR.Prove$

1 **for** $i = 0, 1, \ldots, Chal.length$ **do**
2 $chal_i$ = gatherFromIPFS($Chal[i][1]$);
3 Tag_i = FindTag($Chal[i][0]$);
4 blk_{hash} = FindBlock($Chal[i][0]$);
5 $proof = PoR.Prove(chal_i, blk_{hash}, Tag_i)$;
6 cid_{proof} = uploadIPFS($proof$);
7 $Proofs$.append(cid_{proof});
8 **end**
9 return $Proofs$;

Algorithm 9: ValidateChal(): User validates the challenge.

input : $Chal$: the challenge produced by the user; $Proofs$: proof produced by IPFS node; $add_{contract}$: the contract address;

output: $result$: the result of validating the proof. The result will be $1/0$.

1 **for** $j = 0, 1, \ldots, Proofs.length$ **do**
2 $name_j$ = FindOutBlockName($Chal[i][0]$);
3 $chal_j$ = gatherFromIPFS($Chal[i][1]$);
4 $proof_j$ = gatherFromIPFS($Proofs[j]$);
5 $result = PoR.Verify(chal_j, proof_j, name_j)$;
6 **if** $result == 1$ **then**
7 IncreaseContractReputation($add_{contract}$);
8 **else**
9 DecreaseContractReputation($add_{contract}$);
10 **end**
11 **end**
12 return $result$;

Although the IPFS node may decide to refuse the challenge without the supervision of users, the IPFS node does not lose any reward and still benefits while providing the blocks. To prevent this situation, these misbehaving IPFS nodes have to be removed from the contracts. As we mentioned in Section 3.3, IPFS nodes are the part of users and have the right to challenge other IPFS nodes. Hence, the IPFS node challenges misbehaving IPFS nodes and receives a failure response or no response. It sends the contract transaction to remove that failure node. When half of the IPFS nodes send and receive the remove transaction, that misbehaving IPFS node no longer has permission to store the blocks and can not achieve any reward from the contract.

4.1.4. Retrieval Phase

To connect the block to the previous block and to prevent double-spending, the user needs to read the particular blocks from the disk. There are three places to find out the blocks in our scheme: the working-set table, cold pool, and IPFS network. The higher frequency blocks may be more possibly stored in the working set table to improve the hit ratio. However, the cold blocks may be fulfilled in the working set table and be appended to the cold pool. Therefore, the user finds the cold block in the cold pool after finding it in the working-set table. When the user cannot find it in the local storage, they can retrieve the blocks from IPFS with the CID. With the *PoR* protocol, users can validate the blocks' integrity and push blocks into their working-set table.

4.2. Pricing Strategy, Redundancy and Reputation

In previous sections, we have already introduced all phases of our contract. Although the *PoR* protocol could validate blocks stored in the IPFS nodes, it is not a solution to prevent the IPFS nodes from getting offline or behaving maliciously. We adopt two solutions in our framework, pricing strategy and reputation. On the one hand, the pricing strategy regulate IPFS in an incentive way. The involvement of replicating blocks based on a pricing strategy is incentive for IPFS nodes. When the IPFS node stores more blocks, it can receive more rewards and increase the redundancy of blocks. On the other hand, the reputation provides a way for the users to prevent deploying their blocks to those low credit coalition contracts.

4.2.1. Pricing Strategy and Redundancy

To incentivize IPFS nodes to store blocks, it is necessary to design a reward strategy. In our framework, we provide two pricing strategies: the balancing pricing strategy and the non-balancing pricing strategy. The symbols we defined are shown as Table 2. The pricing strategy is conducted by each node in each interval. Our framework defines the interval as 100 blocks because the mining reward is offered to the miner every 100 blocks. Furthermore, the IPFS nodes dynamically adjust their number of saving blocks to fit each block's pricing strategy to achieve the maximum benefit.

In general, IPFS nodes prefer to store as many blocks as possible to maximize their profit. We first propose a non-balancing strategy to reward the IPFS nodes. The reward of this strategy is shown as Equation (2). The symbol o represents the block number deployed to this contract and m represents the number of IPFS nodes registered in this contract. Let l_m denote the number of the saving blocks stored in node m. The reward is positively related to the number of stored blocks.

$$R_{nb}(l_m) = \begin{cases} \frac{l_m \times \theta}{o \times m}, & (0 < l_m \leq o \times m), \\ 0, & otherwise. \end{cases} \tag{2}$$

However, there might be a situation in which a single node stores numerous blocks, which causes a single failure. To balance the block distribution in the IPFS network, we then designed a balance pricing strategy. We defined a dynamic threshold to rule the maximum storing block number. The threshold is defined as Equation (3), where α represents the replication number of this contract. For example, $\alpha = 3$ means there are at least three

nodes to store the same block i. In other words, there is three times data redundancy in this contract.

$$\delta = \alpha \times \frac{o}{m}. \tag{3}$$

Furthermore, the reward mechanism is given by Equation (4) according to our dynamic threshold in Equation (3). In each block, there are c_i nodes that store the same blocks, where $i = 1, 2, 3, \ldots, l_m$. We let θ denote the rewards. Furthermore, we define a punishment factor w, which limits the reward. We suppose that the IPFS node gains the maximum rewards when it stores δ blocks and gains no pay when it stores $w \times \delta$ blocks. In our reward mechanism, we increase each block's replication factor to encourage the IPFS nodes to retrieve blocks whose numbers are lower than α. If the IPFS node is storing blocks in which c_i is 2, the IPFS node may achieve $1.5 \times l_m \times \frac{\theta}{\delta}$ when α is 3 and $l_m \leq \delta$.

$$R_b(l_m) = \begin{cases} \sum_{i=0}^{l_m} (\frac{\alpha}{c_i} \times \frac{\theta}{\delta}), & (0 \leq l_m \leq \delta), \\ \sum_{i=0}^{l_m} (\frac{\alpha}{c_i}) \times \frac{-\theta}{\delta \times (w-1)} + \frac{\theta}{w-1}, & (\delta < l_m \leq w \times \delta), \\ 0, & otherwise. \end{cases} \tag{4}$$

However, according to [26], given a fixed profit, it is hard to encourage IPFS nodes to retrieve data blocks when storing the same quantity of blocks. Therefore, a threshold is defined to push IPFS nodes to store more blocks. Assume that an IPFS node stores l_m blocks, where $l_m \leq \delta$. Thus, the IPFS node stores $l_m + 1$ blocks after retrieving one more block. Given the storage cost constant factor z, $z \times l_m$ represents the storage cost of an IPFS node. Equation (4) is simplified to Equation (5), as the saving condition in each block is not considered.

$$R_b(l_m) = \begin{cases} \frac{l_m}{\delta}, & (0 \leq l_m \leq \delta), \\ l_m \times \frac{-\theta}{\delta \times (w-1)} + \frac{\theta}{w-1}, & (\delta < l_m \leq w \times \delta), \\ 0, & otherwise. \end{cases} \tag{5}$$

Finally, $l_m + 1$ blocks are stored in the IPFS nodes, where $l_m + 1 > \delta + \frac{1}{m}$. The formulas of the threshold before encouraging IPFS nodes, and the modified threshold are given by Equations (6) and (7), respectively.

$$y = \frac{l_m \times \theta}{\delta} - z \times l_m \tag{6}$$

$$y' = (l_m + 1) \times \frac{-\theta}{(\delta + \frac{1}{m}) \times (w-1)} + \frac{\theta}{w-1} - z \times (l_m + 1). \tag{7}$$

Let y' blocks push to IPFS nodes for storing more blocks, where $y' > y$. We assume the maximum profit, where $l_m = \delta$. To simplify the equation, let $\theta = 1$ and $m \geq 1$. As shown in Equation (8), the mathematical derivation is similar to the details in [26].

$$\begin{aligned} \frac{-(l_m+1)}{(\delta + \frac{1}{m}) \times (w-1)} + \frac{1}{w-1} - z \times (l_m + 1) &> \frac{l_m}{\delta} - z \times l_m \\ \frac{-(\delta+1)}{(\delta + \frac{1}{m}) \times (w-1)} + \frac{1}{w-1} - 1 &> z \\ \frac{-(\delta+1)}{\delta + \frac{1}{m}} + 1 - (w-1) &> z \times (w-1) \\ 1 - \frac{\delta - 1}{\delta + \frac{1}{m}} &> z \times (w-1) + w + 1 \\ \frac{1}{m} - 1 &> (z \times (w-1) + w + 1) \times (\delta + \frac{1}{m}). \end{aligned} \tag{8}$$

In Equation (5), since we define that w must be greater than 0, the value of the equation must also be greater than 0. The threshold is redefined as shown in Equation (9).

$$\delta = \alpha \times (\lfloor \frac{o}{m} \rfloor + 1). \tag{9}$$

In our smart contract, we design a function called dynamicStoringBlocks() to drive the IFPS nodes to dynamically adjust the number of blocks stored in their own local storage. The pseudocode is shown in Algorithm 10. In Line 1, the IPFS node finds out the number of blocks it registered to store in the contract. The IPFS node compares the num with the δ, as shown in Line 2 and Line 13. On the one hand, if the num is smaller than δ, the IPFS node randomly selects the block from the block set that it has not saved as a candidate to store. If the num has not been over the δ, this block is chosen into $Blocks_{save}$. On the other hand, when num is larger than δ, the IPFS node removes those blocks already over the α.

Algorithm 10: dynamicStoringBlocks(): IPFS nodes dynamically adjusts the number of blocks with load-balancing pricing strategy.

input : *contract*: the contract; *pubkey*: the public key of the IPFS node
output: $Blocks_{save}$: the Block set which the IPFS node wants to save; $Blocks_{remove}$: the Block set which S wants to remove

```
1  num = contract.FindSavedBlock(pubkey).length ;
2  if num < α × contract.Blocks.length / contract.Ipfs.length then
3      Blocks_never_saved = NotSavedBlocks(contract,pubkey);
4      saving_num = 0 ;
5      foreach block : contract.Blocks do
6          while num + saving_num <
             contract.replication × contract.Blocks.length/contract.Ipfs.length do
7              randIndex = rand(0,Blocks_never_saved.length);
8              Blocks_save.append(Blocks_never_saved[randIndex]);
9              Blocks_never_saved.pop(randIndex);
10             saving_num = saving_num + 1;
11         end
12     end
13 else if num > α × contract.Blocks.length/contract.Ipfs.length then
14     sortedDESCByIpfsRegisterNumber(contract.Blocks);
15     foreach block : contract.Blocks do
16         if contract.FindSaveBlock(pubkey).has(block) == 1 and num >
             contract.replication × contract.Blocks.length/contract.Ipfs.length then
17             Blocks_remove.append(block);
18             num = num − 1;
19         end
20     end
21 return Blocks_save,Blocks_remove;
```

Although the pricing strategy provides a way to protect the blocks from being lost, blocks are still facing the risk of offline nodes. We design a full node in our contract to store the whole blocks in the contract. The full nodes are selected randomly from those low failure rate nodes. The reward of full nodes is significantly higher than general nodes to encourage these full nodes.

4.2.2. Reputation

Each user has its reputation table, which means their reputation mechanism works independently. The user selects those contracts with a high reputation in the upload phase. As we mentioned in Section 4.1.3, the reputation of the contract increases when the user

verifies the proof provided by the IPFS node. Most of the time, the reputation table is a form of self-insurance and is not involved in the reward mechanism. However, it is still critical to the IPFS node because users provide the blocks to them to achieve the rewards.

5. Experimental Results

The experiments aim to evaluate the performance of our framework. The experiments were conducted by running several nodes in the Bitcoin network in regtest mode. The details of our hardware specification are shown in Table 3. The Bitcoin core version is 0.18.0 and has already been modified to implement smart contracts. These nodes are all running in docker containers with IPFS. The IPFS version is 0.12.2. First of all, we evaluate different time latency of running the six nodes that act in different roles, including the upload, challenge and retrieval phases. Next, we prove our load-balancing pricing strategy by evaluating the number of block in six nodes. Furthermore, the throughput of our framework was evaluate. Finally, the access full node rate of our framework was evaluated.

Table 3. Hardware specification for our experimental environment.

CPU	Intel(R) Core(TM) i9-9900K CPU *1 & i7-8700K CPU *3 & i7-7700K CPU *1 & E5-1620 CPU *1
Ram	32 GB *4 & 24GB *2
SSD	1TB *1 & 512 GB *4 & 250GB *1
OS	Linux Ubuntu 18.04 LTS Distribution

5.1. Latency

In the first experiment, the latency between different phases was discussed. We create one miner, three IPFS nodes, and two users in our blockchain network. We then ran each node on separate computers to make the scenario more real. The latency of different phases in generating 3000 to 4000 blocks was evaluated. Table 4 shows the parameters of this experiment. When the size of the working set is large, the space requirement at the local storage is high. As a result, the hit ratio is very likely to be high as well so that the chance of requesting data blocks from IPFS nodes is low. On the other hand, a larger size of cold pool incurs more transmission time. The interval of challenge determines the level of data security. If the interval is short, more challenges are issued frequently for validation; hence, the system incurs longer latency.

Table 4. Experimental parameters.

Parameters	Values
Size of working set	100 blocks
Size of cold pool	30 blocks
Interval of mining	3 s
Interval of challenge	300 blocks
Blocks to challenge	10 blocks
Total blocks	4000

To generate the coin to deploy and call the contract, we conducted mining of 100 blocks in IPFS nodes computers, which means the experiment starts at a height of 300. Firstly, the results of the upload phase were evaluated. Figure 6 shows these results.

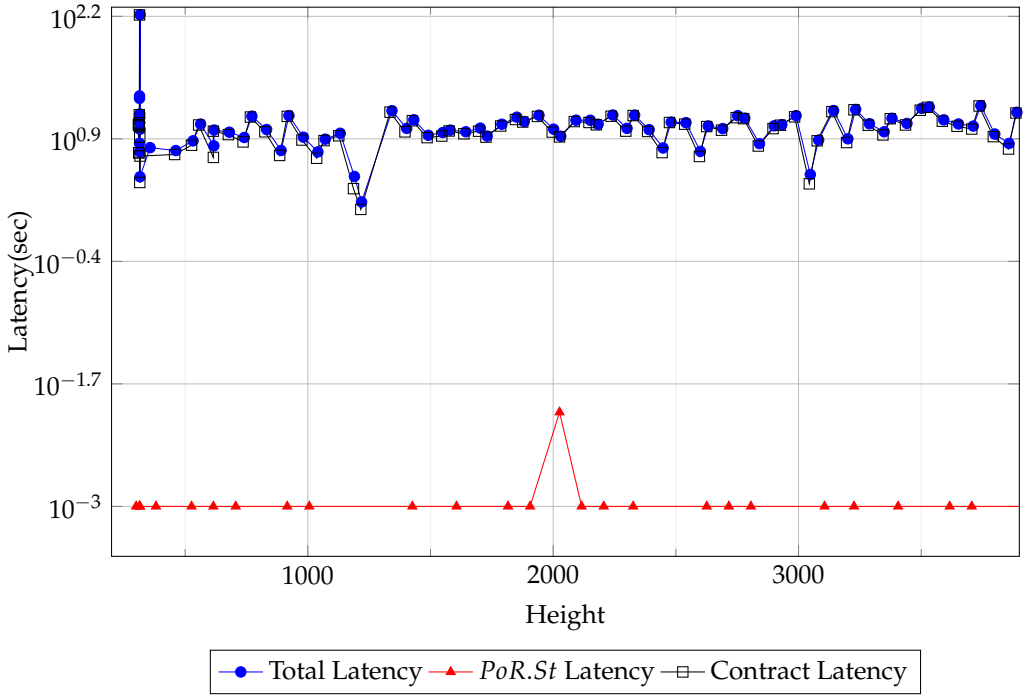

Figure 6. The time latency of the upload phase.

For the first 400 blocks, the overall latency is higher than most of the time. The cold blocks stacking in the cold pool and deployed when nodes are available causes the result. On average, the latency of deploying the blocks to IPFS nodes overcomes 40 s, which takes only 6.67% of the mining time of the Bitcoin Mainnet (10 min per block on average). We consider the latency to be acceptable. Furthermore, the latency caused by the proof of retrievability was illustrated. The average latency of *PoR.St* is 0.0015 s. Compared with the overall latency, it is efficient to compute.

Secondly, we conducted the latency of the challenge phase, which is shown in Table 5. Contract broadcasting consumes most of the time. Finally, the performance of the retrieval phase was evaluated. The average time of retrieving from IPFS is twice that of the local storage. However, it is acceptable to retrieve from IPFS in 0.01 s. Unlike the original blockchain, our framework increases the latency incurring from broadcasting and mining blocks. Nevertheless, our framework guarantees block integrity and a robust network.

Table 5. Average latency in each phase.

Upload Phase			Retrieve Phase		Challenge Phase
Tag	**Contract**	**Total**	**Local**	**IFPS**	
0.0015 s	13.304 s	14.038 s	0.00563 s	0.01 s	23.274 s

5.2. Throughput

To avoid increasing the overloading of the network, the throughput of our framework was evaluated. We executed the experiments using the same parameters as described in the previous section, which are shown in Table 4. To focus on the throughput effect of our framework, the blockchain that does not contain transactions other than contracts were conducted. As shown in Figure 7, we compare the results in two different heights of source Bitcoin. Firstly, the throughputs when the height of Bitcoin is under 4000 were compared. Our throughput performance is similar to the throughput of Bitcoin. Secondly, we compare our framework with Bitcoin with height between 700,000 and 710,000. The increased throughput in our framework takes only 0.167% of the original Bitcoin. The number of transactions is extremely low in our network, which means our framework

does not increase the overhead to influence blockchain performance. It shows that our framework is scalable to be deployed in different applications.

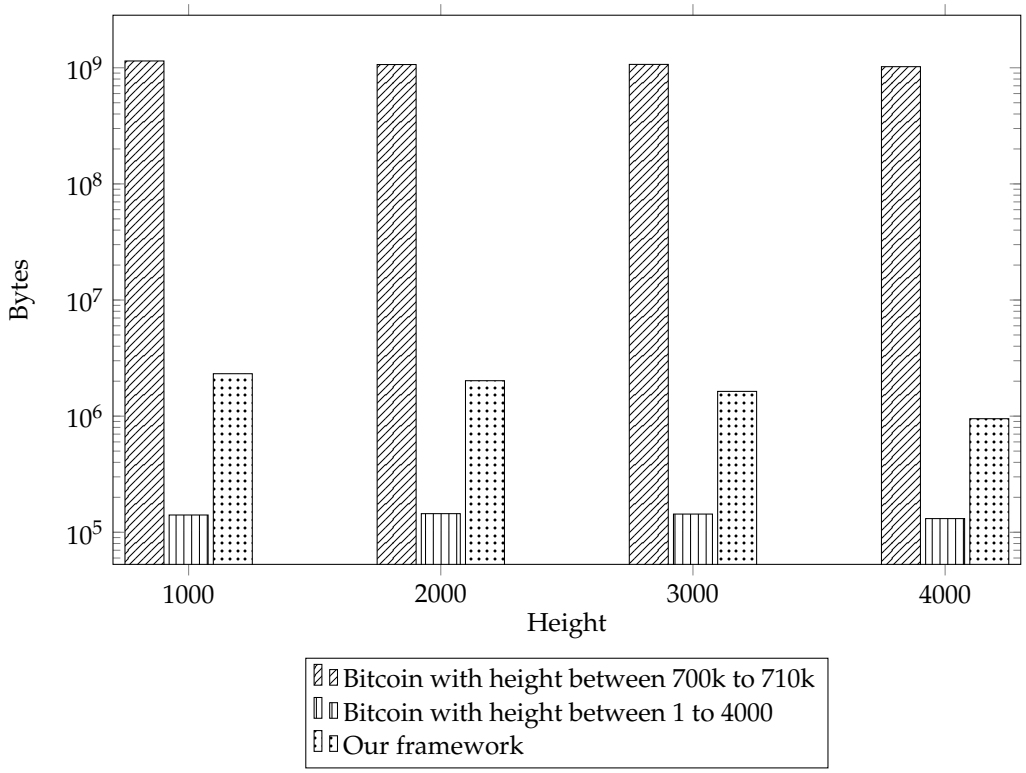

Figure 7. The throughputs between different heights of bitcoin and our framework.

5.3. Full Node Access Rate

Although the full node is provided to decrease the risk of missing blocks, the frequency of requesting blocks from the full node is as low as possible with the efficiency pricing strategy. These experiments are conducted to show the frequency of retrieving the blocks from full nodes. The experiments were conducted 100 times and randomly chose the offline IPFS nodes. The blockchain height is 10,000 and the number of IPFS nodes is 100. The average size of the blocks is 223.8345 bytes. To distinguish the effect of storage size on access rate, two experiments were carried out with different storage sizes. In the first experiment, we limited the storage size of IPFS nodes between 50 KB and 100 KB and randomly assigned different sizes. The result of this experiment is shown in Figure 8.

The experimental results show that the access rate of the balancing strategy is lower than the access rate of the non-balancing strategy. When 20 nodes go offline, the access rate of the non-balancing strategy and the balancing strategy is 0.8254 and 0.0193. The access rate of the balancing strategy is 42 times higher than that of the non-balancing strategy. However, in the second experiment, in which we limit the storage size to between 2000 KB and 4000 KB, the access rate of the balancing strategy is lower than the non-balancing strategy. Figure 9 presents the results of the second experiment.

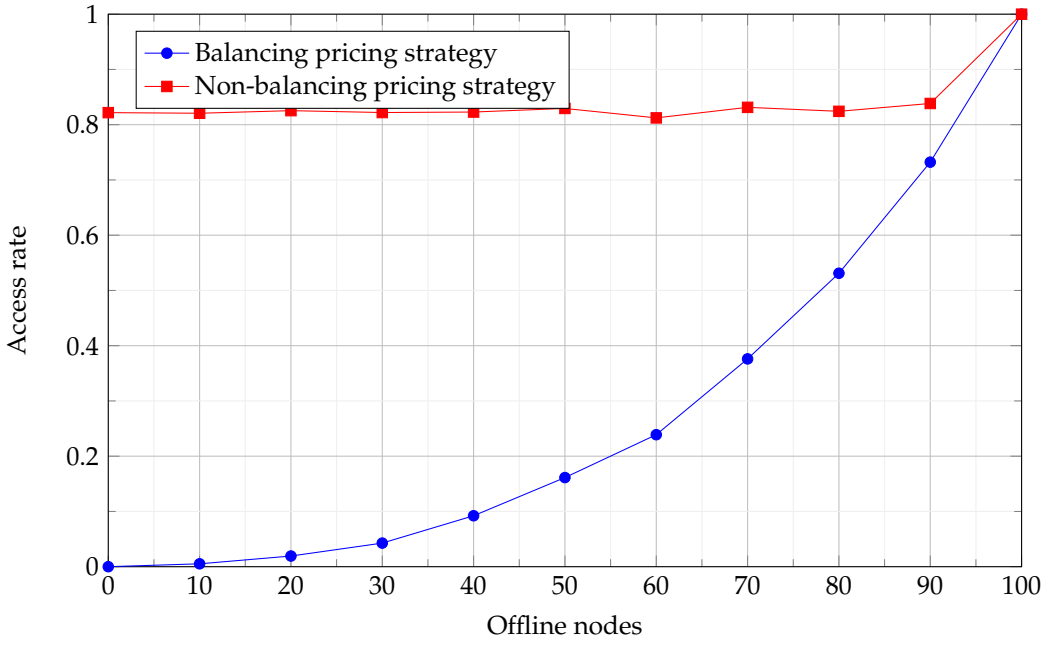

Figure 8. The access rate of different pricing strategies when the storage size is limited between 50 KB and 200 KB.

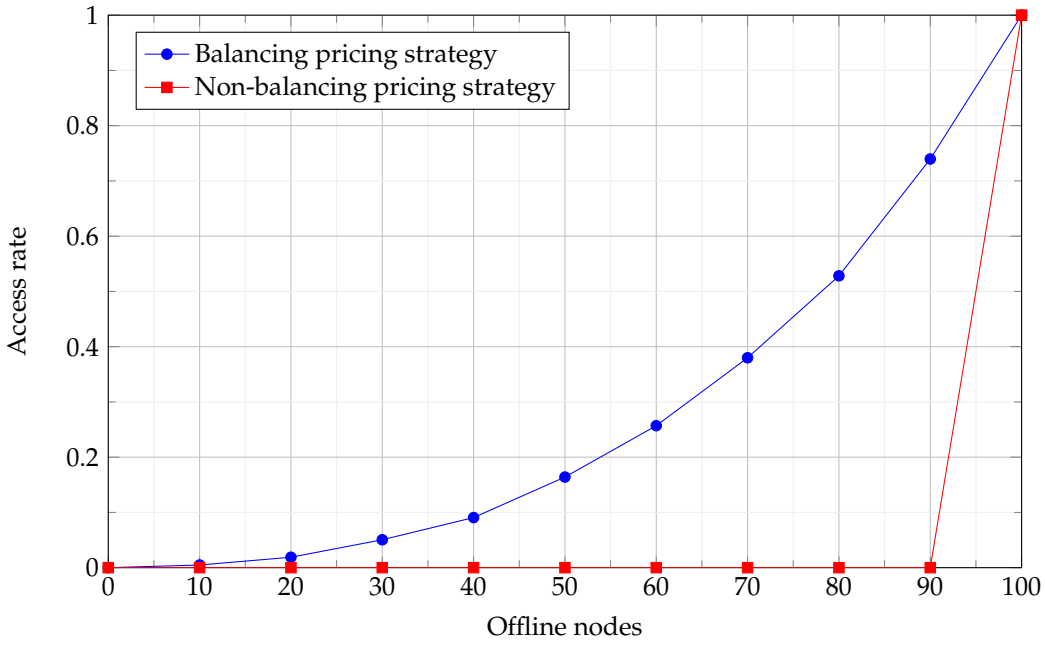

Figure 9. The access rate of different pricing strategies when the storage size is limited between 2000 KB and 4000 KB.

In the second experiment, the storage size is sufficient for the nodes to store more blocks. In our assumption, the nodes store as many blocks as possible. This behavior improves the replicas in the network and decreases the access rate. However, the storage size of each node is fulfilled with the non-balancing strategy in the future, which is the situation of the first experiment. The blocks are able to be stored in the network and become lost at last. The balancing strategy guarantees the availability at least. Furthermore, the overall storage consumption is much greater for the non-balancing strategy. Table 6 presents the different storage consumptions of the two strategies. The load-balancing pricing strategy reduce 97% of the storage size, compared to the non-load-balancing pricing strategy.

The overall storage consumption of the non-balancing strategy is 33 times higher than the balancing strategy. Storage consumption is very important for IPFS nodes, as it affects the cost of keeping them and reduces the credit drop due to failure to provide blocks. To determine the effect of different storage sizes in the non-balancing strategy, our experiments were conducted in four conditions: 50–100 KB, 100–200 KB, 1000–2000 KB and 2000–4000 KB, which is shown as Figure 10. The access rate is significantly negatively related to the storage size of the nodes.

Table 6. Storage consumptions.

	Balancing Strategy	**Non-Balancing Strategy**
Overall	6.305 MB	210.159 MB
Average per node	66.122 KB	2269.804 KB

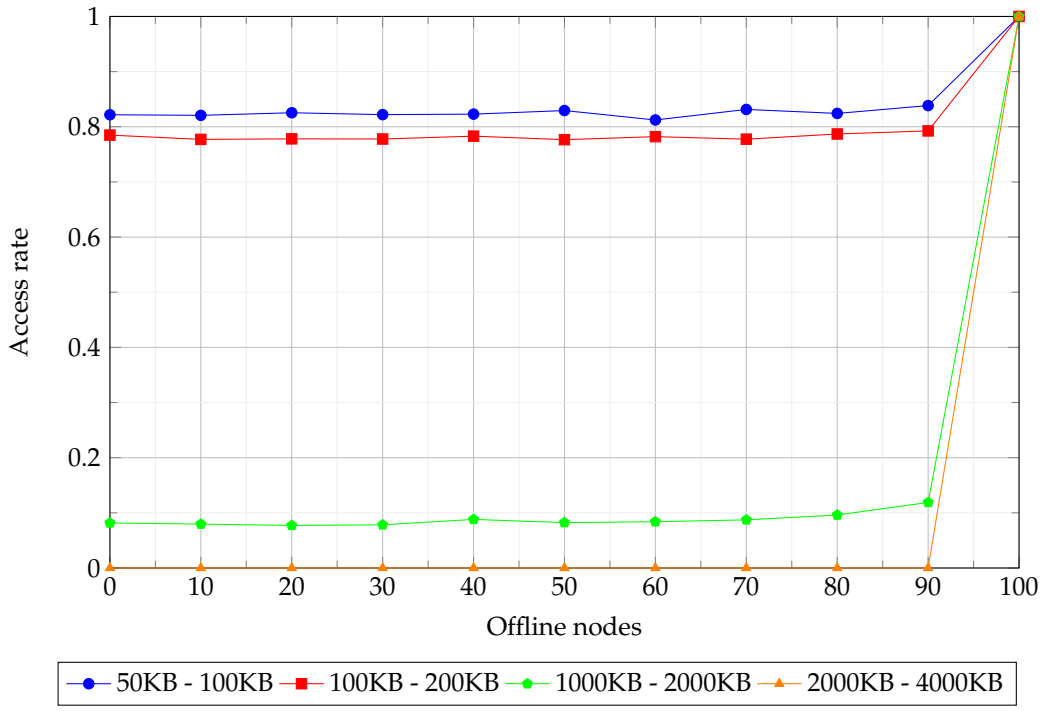

Figure 10. The access rate of non-balancing pricing strategy when storage size limited in different sizes.

6. Conclusions and Future Work

In this paper, we propose a framework to reduce the storage size of the IPFS-based blockchain nodes using the smart contract. Based on the contract with the proof of the retrievability scheme and the pricing strategy, IPFS nodes are rewarded and regulated so that the risk of losing blocks is decreased. Although our framework has the latency and additional throughput incurring from broadcasting and mining blocks, the latency of our framework takes only 6.67% of the mining time of the Bitcoin Mainnet, and the increased throughput in our framework takes only 0.167% of that produced by the original Bitcoin. As a result, the design of the proposed framework is efficient, scalable, and durable. In future work, we will attempt to research the source of the reward which is paid to the contract. This research may improve the sustainability of providing the rewards and decrease the inflation effect of the contract. In addition, the pricing strategy of full nodes needs to be proposed to improve their durability.

Author Contributions: Conceptualization, P.-H.K., Y.-L.H. and C.-W.H.; methodology, P.-H.K., Y.-L.H. and C.-W.H.; software, P.-H.K.; validation, P.-H.K., Y.-L.H. and C.-W.H.; formal analysis, P.-H.K., Y.-L.H. and C.-W.H.; investigation, P.-H.K., Y.-L.H. and C.-W.H.; writing—original draft preparation, P.-H.K. and Y.-L.H.; writing—review and editing, P.-H.K. and Y.-L.H.; visualization, P.-H.K., Y.-L.H. and C.-W.H.; supervision, Y.-L.H. and C.-W.H.; project administration, Y.-L.H. and C.-W.H.; funding acquisition, Y.-L.H. and C.-W.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded in part by the National Science and Technology Council under the grant 110-2221-E-002-006 and 111-2221-E-194-045.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: This work was supported by the Advanced Institute of Manufacturing with High-tech Innovations (AIM-HI) and the Center for Innovative Research on Aging Society (CIRAS) from The Featured Areas Research Center Program within the framework of the Higher Education Sprout Project by Ministry of Education (MOE) in Taiwan.

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

References

1. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 20 June 2022).
2. Wood, G. Ethereum: A Secure Decentralised Generalised Transaction Ledger. 2022. Available online: https://ethereum.github.io/yellowpaper/paper.pdf (accessed on 20 June 2022).
3. Androulaki, E.; Manevich, Y.; Muralidharan, S.; Murthy, C.; Nguyen, B.; Sethi, M.; Singh, G.; Smith, K.; Sorniotti, A.; Stathakopoulou, C.; et al. Hyperledger fabric: A distributed operating system for permissioned blockchains. In Proceedings of the Thirteenth EuroSys Conference, Porto, Portugal, 23–26 April 2018; pp. 1–15. [CrossRef]
4. Kshetri, N.; Voas, J. Blockchain-Enabled E-Voting. *IEEE Softw.* **2018**, *35*, 95–99. [CrossRef]
5. Hjalmarsson, F.P.; Hreioarsson, G.K.; Hamdaqa, M.; Hjalmtysson, G. Blockchain-Based E-Voting System. In Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 2–7 July 2018; pp. 983–986. [CrossRef]
6. Lu, X.; Guan, Z.; Zhou, X.; Du, X.; Wu, L.; Guizani, M. A Secure and Efficient Renewable Energy Trading Scheme Based on Blockchain in Smart Grid. In Proceedings of the 2019 IEEE 21st International Conference on High Performance Computing and Communications, Zhangjiajie, China, 10–12 August 2019; pp. 1839–1844. [CrossRef]
7. Wang, N.; Chau, S.C.K.; Zhou, Y. Privacy-Preserving Energy Storage Sharing with Blockchain. In Proceedings of the Twelfth ACM International Conference on Future Energy Systems, Virtual Event, Italy, 28 June–2 July 2021; pp. 185–198. [CrossRef]
8. Treleaven, P.; Gendal Brown, R.; Yang, D. Blockchain Technology in Finance. *Computer* **2017**, *50*, 14–17. [CrossRef]
9. Guerar, M.; Merlo, A.; Migliardi, M.; Palmieri, F.; Verderame, L. A Fraud-Resilient Blockchain-Based Solution for Invoice Financing. *IEEE Trans. Eng. Manag.* **2020**, *67*, 1086–1098. [CrossRef]
10. Wang, G.; Shi, Z.J.; Nixon, M.; Han, S. ChainSplitter: Towards Blockchain-based Industrial IoT Architecture for Supporting Hierarchical Storage. In Proceedings of the 2019 IEEE International Conference on Blockchain (Blockchain), Atlanta, GA, USA, 14–17 July 2019; pp. 166–175. [CrossRef]
11. Niya, S.R.; Schiller, E.; Cepilov, I.; Maddaloni, F.; Aydinli, K.; Surbeck, T.; Bocek, T.; Stiller, B. Adaptation of Proof-of-Stake-based Blockchains for IoT Data Streams. In Proceedings of the 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), Seoul, Korea, 14–17 May 2019; pp. 15–16. [CrossRef]
12. Malik, S.; Dedeoglu, V.; Kanhere, S.S.; Jurdak, R. TrustChain: Trust Management in Blockchain and IoT Supported Supply Chains. In Proceedings of the 2019 IEEE International Conference on Blockchain (Blockchain), Atlanta, GA, USA, 14–17 July 2019; pp. 184–193. [CrossRef]
13. Musigmann, B.; von der Gracht, H.; Hartmann, E. Blockchain Technology in Logistics and Supply Chain Management—A Bibliometric Literature Review From 2016 to January 2020. *IEEE Trans. Eng. Manag.* **2020**, *67*, 988–1007. [CrossRef]
14. Juma, H.; Shaalan, K.; Kamel, I. A Survey on Using Blockchain in Trade Supply Chain Solutions. *IEEE Access* **2019**, *7*, 184115–184132. [CrossRef]
15. Chang, A.J.; El-Rayes, N.; Shi, J. Blockchain Technology for Supply Chain Management: A Comprehensive Review. *FinTech* **2022**, *1*, 191–205. [CrossRef]
16. Chang, J.; Katehakis, M.; Shi, J.; Yan, Z. Blockchain-Empowered Newsvendor Optimization. *Int. J. Prod. Econ.* **2021**, *238*, 108144. [CrossRef]

17. Chang, J.; Katehakis, M.; Melamed, B.; Shi, J. Blockchain Design for Supply Chain Management. *SSRN Electron. J.* **2018**. [CrossRef]
18. Blockchain.com. The Total Size of the Blockchain Minus Database Indexes in Megabytes. Available online: https://www.blockchain.com/charts/blocks-size (accessed on 20 June 2022).
19. Qi, X.; Zhang, Z.; Jin, C.; Zhou, A. BFT-Store: Storage Partition for Permissioned Blockchain via Erasure Coding. In Proceedings of the 2020 IEEE 36th International Conference on Data Engineering (ICDE), Dallas, TX, USA, 20–24 April 2020; pp. 1926–1929. [CrossRef]
20. Du, Z.; Qian, H.f.; Pang, X. PartitionChain: A Scalable and Reliable Data Storage Strategy for Permissioned Blockchain. *IEEE Trans. Knowl. Data Eng.* **2021**. [CrossRef]
21. Chou, I.T.; Su, H.H.; Hsueh, Y.L.; Hsueh, C.W. BC-Store. In Proceedings of the 2020 2nd International Electronics Communication Conference, Singapore, 8–10 July 2020; pp. 33–38. [CrossRef]
22. Zheng, Q.; Li, Y.; Chen, P.; Dong, X. An Innovative IPFS-Based Storage Model for Blockchain. In Proceedings of the 2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI), Santiago, Chile, 3–6 December 2018; pp. 704–708. [CrossRef]
23. Dai, M.; Zhang, S.; Wang, H.; Jin, S. A Low Storage Room Requirement Framework for Distributed Ledger in Blockchain. *IEEE Access* **2018**, *6*, 22970–22975. [CrossRef]
24. Rupasena, J.; Rewa, T.; Hemachandra, K.T.; Liyanage, M. Scalable Storage Scheme for Blockchain-Enabled IoT Equipped Food Supply Chains. In Proceedings of the 2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit), Porto, Portugal, 8–11 June 2021; pp. 300–305. [CrossRef]
25. Sohan, M.S.H.; Mahmud, M.; Sikder, M.A.B.; Hossain, F.S.; Hasan, M.R. Increasing Throughput and Reducing Storage Bloating Problem Using IPFS and Dual-Blockchain Method. In Proceedings of the 2021 2nd International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST), Dhaka, Bangladesh, 5–7 January 2021; pp. 732–736. [CrossRef]
26. Yin, H.; Zhang, Z.; Zhu, L.; Li, M.; Du, X.; Guizani, M.; Khoussainov, B. A Blockchain-Based Storage System With Financial Incentives for Load-balancing. *IEEE Trans. Netw. Sci. Eng.* **2021**, *8*, 1178–1188. [CrossRef]
27. Ateniese, G.; Burns, R.; Curtmola, R.; Herring, J.; Kissner, L.; Peterson, Z.; Song, D. Provable data possession at untrusted stores. In Proceedings of the 14th ACM Conference on Computer and Communications Security—CCS '07, Alexandria, VA, USA, 2 November–31 October 2007; pp. 598–609. [CrossRef]
28. Renner, T.; Muller, J.; Kao, O. Endolith: A Blockchain-Based Framework to Enhance Data Retention in Cloud Storages. In Proceedings of the 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), Cambridge, UK, 21–23 March 2018; pp. 21–23. [CrossRef]
29. Li, Y.; Yu, Y.; Chen, R.; Du, X.; Guizani, M. IntegrityChain: Provable Data Possession for Decentralized Storage. *IEEE J. Sel. Areas Commun.* **2020**, *38*, 1205–1217. [CrossRef]
30. Xu, Y.; Ren, J.; Zhang, Y.; Zhang, C.; Shen, B.; Zhang, Y. Blockchain Empowered Arbitrable Data Auditing Scheme for Network Storage as a Service. *IEEE Trans. Serv. Comput.* **2019**, *13*, 289–300. [CrossRef]
31. Xu, Y.; Zhang, C.; Wang, G.; Qin, Z.; Zeng, Q. A Blockchain-enabled Deduplicatable Data Auditing Mechanism for Network Storage Services. *IEEE Trans. Emerg. Top. Comput.* **2020**, *9*, 1421–1432. [CrossRef]
32. Swarm. SWARM: Storage and Communication Infrastructure for a Self-Sovereign Digital Society. 2021. Available online: https://www.ethswarm.org/swarm-whitepaper.pdf (accessed on 20 June 2022).
33. Trón, V.; Fischer, A.; Nagy, D.A.; Felföldi, Z.; Johnson, N. Swap, Swear and Swindle: Incentive System for Swarm. 2016. Available online: https://ethersphere.github.io/swarm-home/ethersphere/orange-papers/1/sw%5E3.pdf (accessed on 20 June 2022).
34. Labs, P. Filecoin: A Decentralized Storage Network. 2017. Available online: https://filecoin.io/filecoin.pdf (accessed on 20 June 2022).
35. Yu, H.; Hu, Q.; Yang, Z.; Liu, H. Efficient Continuous Big Data Integrity Checking for Decentralized Storage. *IEEE Trans. Netw. Sci. Eng.* **2021**, *8*, 1658–1673. [CrossRef]
36. Shen, M.; Duan, J.; Zhu, L.; Zhang, J.; Du, X.; Guizani, M. Blockchain-Based Incentives for Secure and Collaborative Data Sharing in Multiple Clouds. *IEEE J. Sel. Areas Commun.* **2020**, *38*, 1229–1241. [CrossRef]
37. Storj Labs, I. Storj: A Decentralized Cloud Storage Network Framework. 2018. Available online: https://storj.io/storjv3.pdf (accessed on 20 June 2022).
38. Liang, W.; Fan, Y.; Li, K.C.; Zhang, D.; Gaudiot, J.L. Secure Data Storage and Recovery in Industrial Blockchain Network Environments. *IEEE Trans. Ind. Inform.* **2020**, *16*, 6543–6552. [CrossRef]
39. Shacham, H.; Waters, B. Compact Proofs of Retrievability. *J. Cryptol.* **2013**, *26*, 442–483. [CrossRef]
40. Rivest, R.L.; Shamir, A.; Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. In *Communications of the ACM*; Association for Computing Machinery: New York, NY, USA, 1978; Volume 21, pp. 120–126. [CrossRef]
41. Vogelsteller, F.; Buterin, V. EIP 20: ERC-20 Token Standard. 2015. Available online: https://eips.ethereum.org/EIPS/eip-20 (accessed on 20 June 2022).