Journal of computer science and software applications
https://www.mfacademia.org/index.php/jcssa
ISSN:2377-0430
Vol. 5, No. 2, 2025

Toward a Decentralized Cloud Storage and Internet Service Network: Challenges and Future Directions

Carsten Whitaker[1], Adele Laurent[2], Arvind Patel[3]
[1]Department of Computer Science, University of New Mexico, Albuquerque, USA
[2]Department of Computer Science, University of New Mexico, Albuquerque, USA
[3]Department of Computer Science, University of New Mexico, Albuquerque, USA
*Corresponding Author: Carsten Whitaker, carsten93@unm.edu

Abstract:

The reliance on large corporations for high-performance and reliable cloud services presents vulnerabilities related to data security, privacy, and infrastructure failure. Decentralized networks offer a promising alternative by distributing resources and computational power across participants, mitigating dependence on centralized authorities. This paper investigates the development of a decentralized network for cloud storage and internet services, discussing key methods such as file encryption, distributed hash tables, and blockchain-based protocols. Although current solutions provide foundational progress, further research is needed to refine data structures, integrate Proof of Spacetime mechanisms, address potential attacks, and enhance fault tolerance and self-recovery. Additionally, the potential for transforming decentralized storage networks into a permanent, immutable web is highlighted, with future work proposed to address DNS resolution and robust file-sharing solutions for web integration.

Keywords:

Decentralized storage network, Blockchain, Decryption, Distributed hash table, Proof of replication.

1. Introduction

Advances in cryptographic algorithms are the cornerstone of modern interconnecting web network. Almost all the high performance and reliable web services are currently provided by large corporation, who have the money and resource capability to cope with such high demand for fast and reliable cloud services. However, there is not guarantee that data and privacy is secured when they are stored in the hand of large corporations and central authorities, and such high dependability could lead to vulnerability and unreliability once the corporation's infrastructure failed.

To solve this problem, many decentralized services have emerged over the past decades. These services aimed for utilizing the recourses and compute power of the entire network, agreeing on central algorithm and consensus, to fulfill a need to an extend comparable to those of the large corporation.

One such network, BitTorrent [1], has been extremely successful. As of now, it has become one of the most popular way of downloading contents. A typical BitTorrent download session download the file, block by block, from all connected peers that owns the desired file. This peer-to-peer download structure can be very reliable and efficient with increased number of connected peers.

This paper attempts to research on the possible solution for a decentralized network that provides reliable cloud storage and internet services

2. File Handling

2.1 Envelop Sealing

Because of the relative compute intensive nature of the asymmetric encryption [2] (public key encryption), a common solution in the industry is to encrypt the file using symmetric encryption, and then encrypt the key asymmetrically. For example, a file could be encrypted using AES-256, whose Key is encrypted using RSA-256. This sealing of symmetric encryption key in asymmetric encryption is called Envelop Sealing.

Envelop sealing could be extremely useful in DSN, where large files are split into multiple small file blocks. Depending on the actual sizes of the file and the block, the resulting blocks may be large in quantity, significantly increasing the overhead of encrypting all the blocks.

2.2 File Encryption

Implementing the concept of envelop sealing, the method used here for file encryption is a combination of AES-256 and RSA-256.

2.2.1. AES symmetric encryption

This encryption method is used for encrypting all file block. The encryption key, *AES_ Key* [3], can be user specific or are randomly generated if not specified.

2.2.2. RSA public key encryption

This encryption method is used for encrypting the *AES_ Key* [4], using the hash of each node's address as public key. The private key can either be set manually or randomly generated.

2.2.3. General Scheme

The general scheme of encrypting data block are the following:

$$m_i \in \{block_1, block_2, \ldots, block_i\}$$

$$M_i = AES(m_i, K_{AES})$$

$$K_{AES\,Encrypted} = RSA(K_{AES}, Hash(Adr_{Node}))$$

2.3 Sharing of Files

Sharing of files require not only the transmission of data, but also the transferring of private key that can be used to decrypt those encrypted data. Unfortunately, for security reasons, the transmission of private key on internet should be strictly avoided. Therefore, the *AES_ Key* should be re-encrypted using the hash of target node's address, and then transfer the encrypted data and *AES_ Key*.

2.3.1. Hosting Websites

For files that should be publicly accessible, like hosting a website on DSN, related data should not be encrypted, but should have proper permission setup to prevent any modification from non-administrative users.

3. Content Addressing

3.1 Content Identifier

Each content saved in decentralized network will have a unique identifier, which is called content identifier (CID). Contrary to conventional way of identifying files by its location like in URL, The CIDs are based on the content's cryptographic hashes, which are called "key". It means that any difference in content will lead to a different CID.

However, as CIDs are generated according to the content instead of physical location, it needs a specific method to decide where contents are saved and how to find them by CIDs.

3.2 Distributed Hash Tables

To create a lookup service that seeks the desired content, a Distributed Hash Table is maintained to keep track of the CID and the UIDs, user identifier, that store them.

3.2.1. Directed Acyclic Graph

Directed Acyclic Graph, or DAG, refers to a Merkle tree [5] whose nodes are the hash values of data blocks in the root directory. Each file is split into data blocks and are linked to a single node representing the file. Each node's hash is a combined result of all its branches. Because of this, starting from each node, there is no sequential links that traces back to initial node itself, because a branch of that node cannot contain that node at the same time, resulting in an acyclic tree. In this way, a traditional location-based directory is converted to a content-based structure.

Merkle trees are essential in many of the distributed application, where mutual trust and security of data should be embedded in the algorithm.

3.2.2. Kademlia DHT

Kademlia DHT provides a solution for the distribution of hash tables across users.

The users in decentralized network are treated as nodes. Each node has a unique node ID. And these IDs are in the same form as the form of the content's cryptographic hash. Both of node IDs and content's keys are 160-bit strings in Kademlia distributed hash table (DHT). The main idea of content saving is that information of content is saved in the nodes whose node ID are closest to its key.

In Kademlia DHT, distance is computed according to XOR metric [6]. For example, assuming the length of node IDs are 4, here are 2 node IDs: "100", "010". The XOR result of them is "110", which equals to 6 in decimal. So, the distance between these 2 nodes are 6.

Each content can be saved in the nodes in the form as <key, value>. The key is a hash string generated by the content. And for small content, the value is its content, otherwise the value is the hash of a node which saves the content.

3.3 Content Retrieval

This section covers the process of requesting retrieval of content.

3.3.1. Wantlist

First, a Wantlist, containing the desired CIDs, are send out to connected nodes in proximity. The nodes that receive the Wantlist will compare it with CIDs hosting in their own repositories, and send back the data with the corresponding CIDs. The transmission of data is done recursively, which means that other data blocks linked to the requested block are also sent.

3.3.2. Hash Recipe

Along with the data blocks, each of them is also attached a Recipe used verify the content's hash. This recipe would include any information required to regenerate the hash value of that block, such as the hash function and encoding method used. If the regenerated hash matches any of the desired CID in the Wantlist, this data block is accepted and preserved in user's repository. Otherwise, this data is discarded immediately for safety reasons, regardless of whether it is sent accidentally or intentionally.

3.3.3. Seeking content

When a node receives the Wantlist, but does not possess any of the desired content, the node will send back a list of potential nodes that may fulfill the Wantlist. Based on the principle of Kademlia DHT [7], a node whose hash is similar to the CID might have a higher chance of owning the CID in its DHT. The initial node requesting for the content would continue communicating with nodes that have a higher chance of having the CID. A reliable way of seeking content in DHT is established.

3.4 Finding nodes by node ID:

We have mentioned above that the length of node ID is 160-bit. Each node will maintain 160 buckets which are used in routing. Each bucket saves k nodes information: <IP address, UDP port, Node ID>. For example, k-th bucket will save the nodes whose Node ID have (k-1)-bit same prefix with user's node ID. If the node ID of user is "1xxx…", the node whose ID likes "0xxx…" will be save in the bucket [0].

The process of finding nodes by its node ID is that:

user first computes *user's ID* XOR *target ID*. Assuming result is *r*. Then user reads bucket [*r*]. If bucket [*r*] is not filled, user must read neighboring bucket to get *k* nodes.

user choose *a* nodes from *k* nodes and ask each of them to return information about another *k* nodes it knows which is closest to the target.

while user gets new nodes information, he will compute the distance between new node and target node. If new node is closer to target than some node in k nodes do, user will update the k nodes list.

If all node in k nodes list have been asked, jump to 5, otherwise jump to 2

user now have k nodes which is closest to the target node. If target node exists, it will in the nodes list.

The difference between finding values and finding nodes is that, during the process, if some node contains the <key, value>, it will return value to the user and searching process will terminate immediately.

4. Storage Market

4.1 properties of DSN and blockchain

Blockchain is defined as a constantly growing chain of blocks which have a timestamp, and a link to the previous block. Blockchain technology resides on a P2P network. It physically cannot work with a single computer or point-of-connection Instead, it requires a slew of other computers to join in, in order to complete a specific task on the network. [8]

Decentralization is defined as the transfer of authority from a central entity to a more localized and 'liberal' system. In contrast to a centralized platform's single point-of-data, decentralized platforms exist within multiple-points-of-data.

4.2 The benefits of using blockchain

4.2.1. Enhanced Data Security

Using blockchain in DSN makes the whole system more difficult to hack because the information is not only timestamped, it is also stored in such a way that you would have to get into every single computer at the same time in order to hack into the network. Because centralized networks have a single point of data collection, they are extremely susceptible to hacking. Blockchain technology, and in turn decentralization, is an effective way to work around this weakness. So, storing information on a peer-to-peer network is best in terms of security. This also reduced the influences of censorships. Unlike a centralized server, who has fixed domain names and absolute control over every data hosting on their server, peer-to-peer file sharing between numerous nodes avoids censorships. The cryptographic nature of these file blocks makes them difficult to identify, and no central authority can block all connections and take down censored data from all thousands of self-hosted nodes.

4.2.2. Avoid centrally hold the data

Using blockchain in DSN eliminates several risks that come with data being held centrally by storing data across its peer-to-peer network. The decentralized blockchain may use ad hoc message passing and distributed networking. Peer-to-peer blockchain networks lack centralized points of vulnerability that computer crackers can exploit; likewise, it has no central point of failure.

4.2.3. Less expensive

Traditional cloud computing, especially the hyperscale cloud providers, consists of a few large companies – Amazon, Google, Microsoft, Alibaba. They have central control over thousands of machines, used by millions of users. DSN pluses a couple of thousands of hosting providers, but they are much smaller than the global hyperscale giants. This blockchain-regulated market of cloud storage and incentivization in the form of cryptocurrency enable the network to be readily scalable. Millions of computers connected without central control. Pricing of storing and retrieving files are automatically adjusted through market demand and competition between service providers, which, in

this case, refer to miners providing storage spaces and traffic bandwidth, resulting in a very competitive pricing compared with traditional centralized cloud storage providers. [9]

4.3 Protocol

Three parties participate in this block chain market: *Clients, Storage Miners, Retrieval Miners*. Their rules are explained in section 4.4.1.

4.3.1. Necessary Terms

There are several terms that are necessary to understand the protocol of the market.

Incentives – This is the amount of cryptocurrency that Miners can gain for successfully completing requested job (either storing the data for required amount of time or serving the data).

Collateral – The amount of cryptocurrency required for Miners to deposit along with the stored data. Acting as insurance for Clients in case the stored data is corrupted or lost.

4.3.2. Order

Orders are negotiations between Clients and Miners. There are three types of Order: *bid* order, *ask* order, and *deal* order.

bid order – a request of storing data from Clients. It includes *Incentives*, *Collateral* required, sizes of data and time required to store.

ask order – a response to *bid* order, indicating that the Storage Miner has accepted the *bid* order.

deal order – after Client agree on the *ask* order, a *deal* order is created. It includes the cryptographic hashes reference to the *bid* and *ask* order, and the hash of data that Client is storing.

This process is very similar to that in a free-market biding. Clients have certain jobs and are willing to pay certain money for it. After competition, a certain party won the biding and are responsible for the job offered by the client.

4.4 Block Chain Market

The primary purpose of implementing block chain into a DSN is to transfer the network into a market, in which people with excessive storage spaces can lend those spaces to customers who want to trust the security and reliability of their data.

4.4.1. Participants

Clients are customers that have data to be stored. When a Client wants to store certain data, after negotiated with Storage Miners through *Protocol*, the replica of the data is sent to Storage Miners for storing, along with *Incentives*. Clients retrieve the stored data by negotiating to Retrieval Miners in similar manners.

Storage Miners provide storage spaces to store the replica of Client's data and provide computing power for creation of new blocks. Storage Miners need to generate valid *proofs* that replica is indeed stored, along with small amount of *Collateral.* Storage Miners can be Retrieval Miners at the same time.

Retrieval Miners provide data traffic to serve the stored replica from Storage Miners to Clients.

4.4.2. Ledger

Similar to Bitcoin and many other block chain systems, each block in the chain is essentially a ledger, keeping track of every transactions. However, the demand for a publicly verifiable market means that, in DSN, the ledger should contain more information.

Previous section mentions that Clients would negotiate with multiple Storage Miners for pricing through orders. These negotiations are in-chain orders, meaning that all orders are created in blocks and can be accessed publicly. In this way, pricing are transparent through the market, balancing the market through market-demand.

Storage Miners' *proofs* is also recorded in the Ledger. The history of transactions enables every node

in the network to verify if certain proof is valid.

5. Proof of Replication

5.1 Purpose of PoRep

We research on *Proof of Replication* (PoRep) [10], which is a variant of proof of storage and it allows a prover P to convince Verifier V that Replica R, which is a physical copy of Data D, is correctly stored. In the context of DSN market, it ensures that Client's data is securely stored in the sectors provided by Storage Miners.

This proof is performing satisfyingly when applying on Decentralized Storage Network. It includes multiple benefits including transparency, efficiency, and protection towards unfriendly attacks.

There is no doubt that most of the decentralized storage network is viewed as untrusted cloud storage because normally they are not running by large, authoritative companies like Google or Amazon. Therefore, the Proof of Replication plays a crucial role in this circumstance to provide dependability to both parties.

5.2 Algorithm

The algorithm of PoRep is based on three parts--Set Up, Prove, and Verify--as the syntax provided below:

$$PoRepSetup\ (1^{\lambda}, D) \rightarrow R^{D}, S_p, S_v$$

$$SP, SV - scheme - specific\ setup\ variables$$

$$R - replica\ of\ D$$

$$PoRep.\ Prove\ (S_p, R^{D}, c) \rightarrow \pi^{C}$$

$$c - challenge$$

$$\pi^{C} - proof$$

$$PoRep.\ Verify\ (S_V, c, \pi^{C}) \rightarrow \{Accept, Reject\}$$

In the Setup Phrase (sealing), the prover will generate multiple replicas of data by a key choosing by the verifier. The verifier sends a challenge to the prover periodically. The prover receives the challenges and generates proofs that based on the data replicas. Finally, the verifier checks the proof and decides whether to accept or reject the proof. Furthermore, time limits apply to PoRep to ensure security. Assuming the setup phrase takes abundant time to generate the replicas which is longer than the proving time. An attacker can hardly produce the proofs within the time given without storing the replicas.

Proof of Replication can defend multiple attacks including the generation attacks. Since the Decentralized Storage is profitable, attackers may generate a large amount of junk data to get paid. Proof of Replication can prevent this attack in its setting. The attack cannot make up replicas in time. An honest prover has sealed the specific data at the beginning since it takes time. There is no way to generate the proof without unique replicas of the data. This also distinguishes honest prover and false prover.

5.3 Comparison with other Proof

Comparing the Proof of Space normally used, Proof of Replication has an advantage in short verification time. Proof of Space requires the verifier to frequently sends challenges to the prover. However, Proof of Replication avoids such circumstances by its long sealing time. The proving time is relatively short compares to time requires in sealing. A verifier can send the challenge randomly over a period of time.

6. Conclusion

First, this report covers the methods about encrypting and sharing files, the certain use of distributed has table in decentralized storage network, and the basic protocol of a blockchain market. However, the integration of blockchain and proofs into the network is also vague. Future research should add more details on the actual data structure used in the protocol, and a more specific scheme of its process. Proof of Spacetime can also be adopted in the future. Potential attacks and their corresponding preventions should be covered in future research, as well as solution for self-recovery and fault tolerance in case the protocol experience undefined behaviors. There are also potentials for converting a storage network into a permanent web because of its immutability of data stored. In such scenario, more details of web integration should be covered, such as DNS resolves and a more elegant and robust solution for file sharing.

References

[1] Cohen B. (May 2003):Incentives build robustness in bittorrent. Workshop on Economics of Peer-to-Peer System, Berkeley, USA.

[2] Bellare M., Rogaway P.(1995):Optimal asymmetric encryption. Lecture note on Computer Science, 92- 111.

[3] Gary C.Kessler(Updated version, 3 March 2016): An Overview of Cryptography. http://www.garykessler. net/library/crypto.html.

[4] Gary C.Kessler(Updated version, 3 March 2016): An Overview of Cryptography. http://www.garykessler. net/library/crypto.html.

[5] Szydlo M. (2004):Merkle Tree Traversal in Log Space and Time. Lecture Notes in Computer Science, 541-554.

[6] Maymounkov P., Mazieres D. (2002):Kademila: A Peer-To-Peer Information System Based on the XOR Metric. Lecture Note in Computer Science, 53-65.

[7] Steiner M.,En-Najjary T., Biersack E.W.(2009): Long Term Study of Peer Behavior in the kad DHT. IEEE/ACM Transactions on Networking, 17(5),1371-1384.

[8] Conte de Leon D., Sralick A.Q.,Jillepalli A.A., Haney M.A., Sheldon F.T.(2017): Blockchain:properties and misconceptions. Asia Pacific Journal of Innovation and Entrepreneurship.

[9] Emiliano Pagnotta, Andrea Buraschi(2018): An equilibrium valuation of bitcoin and decentralized network assets. SSRN Electronic Journal, SSRN – 3142022.

[10] Benet Juan, David Dalrymple, Nicola Greco(2017): Proof of replication. Protocol Labs, July 27 (2017):20.