A Tamper-Resistant and Decentralized Service for Cloud Storage Based on Layered Blockchain

Fuxiao Zhou, Haopeng Chen(✉), and Zhijian Jiang

Shanghai Jiao Tong University, Shanghai, China
{zhoufuxiao,chen-hp,jiangzhijian}@sjtu.edu.cn

Abstract. With the ever-increasing scale of data, IP traffic of data centers will gradually suffer from a serious shortage. Centralized clouds may no longer deliver satisfactory storage services. To alleviate network pressure, transmitting source data to the edge of network instead of the remote cloud is becoming more and more important. Based on the blockchain technology, we propose a decentralized cloud storage service with tamper-resistant function. Our service provides users with metadata storage and management capabilities. In order to overcome the inherent defects of blockchain, in our design, the blockchain network has a layered and collaborative structure. After the storage capacity problem is solved, our service allows users to upload digests of source data to make the query function more user-friendly. For privacy protection, we subjoin the access management function as well. Finally, the experimental results show that the performance and availability of the blockchain network are able to support our service to provide efficient functions to users.

Keywords: Cloud storage · Collaboration · Decentralization · Tamper · Blockchain · Metadata

1 Introduction

The emergence of cloud computing has tremendously changed the way people deal with data [1]. A growing number of users are willing to process their data on the cloud. The use of cloud-based storage services for storing data has also become a popular alternative to traditional local storage systems [2]. As a result, storage as a service (STaaS) is raising widespread concern. However, with the advent of big data era, cloud storage has also been facing new challenges. As estimated by Cisco Global Cloud Index [3], nearly 850 zettabytes data will be generated by all people, machines and things by 2021. Simultaneously, the global data center IP traffic will only reach 20.6 zettabytes by that time. Though most of the ephemeral data will be consumed instantly, the remaining data which is worth saving still occupies a large part. Given the scarcity of data center IP traffic, it is arduous to upload all the relatively important data to cloud centers.

H. Gao et al. (Eds.): CollaborateCom 2020, LNICST 350, pp. 482–493, 2021.
https://doi.org/10.1007/978-3-030-67540-0_31

Under the restriction of scarce network resources, centralized clouds may no longer own the ability to deliver satisfactory storage services to users. Fortunately, edge or fog computing might help bridge this gap [3]. Given a finite network bandwidth, the large volume of data which needs to be stored and shared can be transmitted to the edge of the network instead of the remote cloud, so as to solve the data transfer bottleneck. While the metadata [4] with which users can locate and inspect the data actually needed should be uploaded and administrated. For the purpose of storing metadata on the cloud, several requirements ought to be satisfied.

Ensuring data integrity and validity is essential. To protect data from tampering, some archival storage systems such as Amazon Glacier [5] follow a Write Once, Read Many (WORM) paradigm, which treats all archive data as read-only. Even if supported by high fault tolerance strategies, there is a certain risk that data can be maliciously modified without notification [6,7].

When users propose to apply a traditional centralized cloud storage service, it means that the supplier of the service is considered to be a trusted third-party. Nevertheless, this is a questionable proposition. In addition to the trust crisis, a centralized service may inevitably lead users to suffer from other intrinsic defects such as performance bottleneck caused by central server and the risk of data unavailability caused by data center failure.

Briefly, a decentralized cloud storage service secured from tampering is quite in need. As it happens, the prevalence of Bitcoin [8] has raised great attention, we found the underlying technology blockchain just in line with our needs. If only its inherent drawbacks overcome, with key characteristics like decentralization, persistency, anonymity and auditability [9], blockchain can play a pivotal role in this scenario. For instance, in Bitcoin blockchain, the information once entered can never be erased. As each node in the Peer-to-Peer (P2P) network is required to store the entire history information, which restricts the capacity substantially.

Under this condition, we develop a decentralized service based on blockchain for cloud storage in order to supply users with a tamper-resistant function. Supported by security mechanisms, our service will take charge of management and preservation of the meta information. When actually used, data can be positioned, checked and verified with the information extracted from our service. To cross the barrier of capacity limit and scalability, we come up with a layered and collaborative structure. In our design, the mainchain stores indexes of metadata while subchains allow users to upload metadata and digest of source data.

In this paper, we have made the following contributions:

- A simple storage strategy to relieve the contradiction between huge data volume and limited network resources of cloud.
- A design of decentralized cloud storage service based on blockchain which provides powerful tamper resistance and access management function.
- A layered structure design of blockchain to expand the storage capacity without sacrificing scalability.

The rest of this paper is organized as follows: Sect. 2 presents the detailed design of our tamper-resistant and decentralized service. Section 3 gives an intro-

duction of workflows, together with the deployment strategy. In Sect. 4, we evaluate our service based on the simulation by Hyperledger Fabric. Section 5 discusses the related works. Finally, we end with a conclusion in Sect. 6.

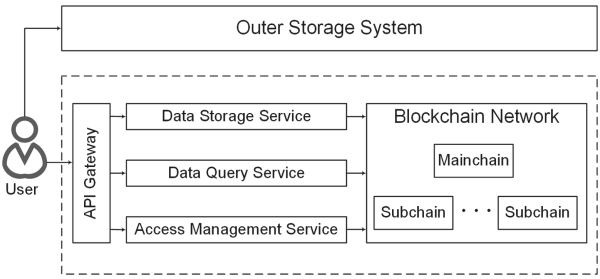


Fig. 1. Service architecture

2 Design

2.1 Overview

Due to the application scenario, we implement our ideas as a form of cloud service. Collaborating with outer storage systems, our service is capable of providing reliable functions to users. Figure 1 demonstrates the service architecture. The layered design of blockchain network increases the storage capacity and enhances

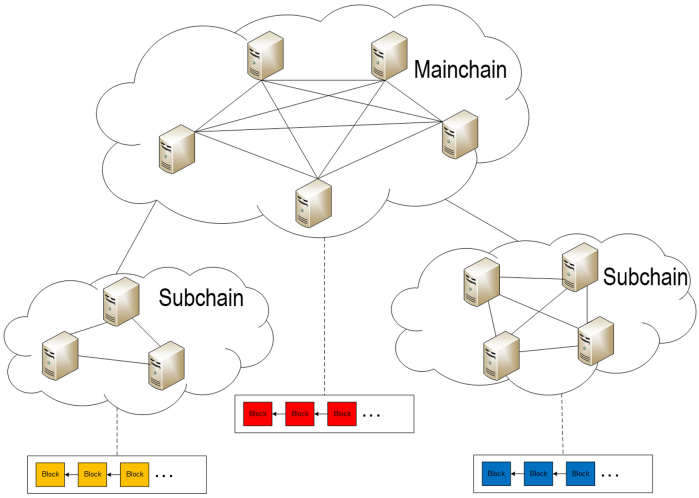


Fig. 2. Layered and collaborative blockchain architecture

scalability. Subchains can be added in any position according to actual demand. All subservices keep contact with the blockchain network for tamper-resistent, persistency, fault-tolerance and other features.

2.2 Blockchain Network

Blockchain network is a core part of our design. In order to ensure the tamper-resistent ability of our service, any data that needs to be persisted should under the protection of a reliable mechanism. Although it provides a strong guarantee for security, the traditional blockchain cannot fully meet our requirements for two reasons.

The first reason is capicity limit. Because of the decentralization feature, blockchain is a peer-to-peer network which requires every single node to keep a copy of all blocks and stay in synchronization with each other at any time. This constraint limits the amount of data across the network extremely. For this reason, uploading large data to the traditional blockchain network is not allowed. However, with compact data, the query experience will be very unpleasant.

The second reason is the scalability defect of blockchain. It is well known that high scalability is hard to achieve in a blockchain network because of consensus issue. So putting all operations on a single blockchain network is easy to cause insufficient throughput. With a traditional blockchain network, our service will lose the ability to make a guarantee of concurrency.

Because of the considerations above, we design a two-layer structure for our blockchain network. Of course, with more layers, the blockchain network may achieve higher capacity and scalability. We found it unworthy owing to it brings much more complexity at the same time. The architecture is shown in Fig. 2. There is a mainchain and serval subchains in our network.

The unique mainchain is responsible for direct interactions with the data query service. The mainchain maintains a record of mapping from subchains to node IP address list as well. When users query data, the data query service can get the metadata from any IP address in the list of corresponding subchain. Which node will be chosen is determined by the network delay and the remaining bandwidth of each node.

Subchains take charge of direct interactions with the data storage service and access management service. A special subchain is designed for the access management service to store authentication information of users. Each of the subchains has a unique id and any change of its node IP address will be synchronized to the mainchain. In order to expand capacity, a new subchain can be added in any position.

2.3 Data Storage Service

When users upload new data or update old data, the data storage service will be called. A complete data insert operation involves uploading the metadata and the digest of source data to a subchain and transmitting the index to the mainchain.

We design a unified data structure to store metadata and digest of source data in subchains. As shown in (1), we use six parameters to represent specific information.

$$UUID : \{validation, metadata, digest, readerset, writer\_set\} \tag{1}$$

- *UUID*: UUID is a hash value calculated from the metadata uploaded by the user.
- *validation*: validation is a boolean value whose default value is true.
- *metadata*: metadata is uploaded by users.
- *digest*: digest is also uploaded by users. It is a relatively detailed description of the source data and is represented by a string.
- *reader_set*: readerset is a set of user_id.
- *writer_set*: similar to reader_set, writer_set is used for access management as well.

As shown in (2), we design another unified data structure to store indexes of metadata in the mainchain. In general key-value databases like RocksDB [10], the insert operation of a record will cover the old record with the same key. This can also be considered a form of tampering. In order to solve this problem, in our design, key is allowed to appear repeatedly. For each key, the mainchain maintains an array of pairs composed of UUID and subchain_ID. subchain_ID is globally unique. Obviously, the data structure is particularly compact to relieve the capacity burden of the mainchain.

$$KEY : [\{UUID, subchain\_ID\}, ...] \tag{2}$$

The data storage service's interfaces were shown in (3), (4) and (5).

$$UUID\ PUT\_S(subchain\_ID, metadata, digest) \tag{3}$$

PUT_S is used to insert records into subchains. It accepts three parameters include subchain_ID, metadata and digest. The UUID calculated from metadata will be returned. Another usage of UUID will be explained in the next subsection.

$$void\ PUT\_M(KEY, UUID, subchain\_ID) \tag{4}$$

PUT_M is usually called after PUT_S, to upload the UUID and subchain_ID to the mainchain. It accepts three parameters include KEY, UUID and subchain_ID. As a suggestion, KEY should be a string related to the contents of source data for the convenience of inquiries. After PUT_M executes successfully, the record will be visible globally.

$$void\ REMOVE(subchain\_ID, UUID) \tag{5}$$

REMOVE is designed for deleting records in subchains. It accepts two parameters include subchain_ID and UUID. When it is called, if there exists a record with the same UUID in the corresponding subchain, its validation value will be turned to false.

2.4 Data Query Service

When users need to perform a query operation, the data query service will be called. Different from the data storage service, the data query service does not interact directly with subchains. This mechanism will bring greater security to our service and will eliminate the impact of subchain failure on the overall situation. Since the mainchain usually contains more nodes, the throughput of query is easier to meet requirements.

$$result\ GET(KEY, UUID) \tag{6}$$

The interface exposed by the data query service is shown in (6). Between the two parameters accepted by GET, UUID is optional. With both KEY and UUID, the result returned will be a pair of metadata and digest. Users have already obtained the UUID while inserting records into subchains.

$$[\{UUID, \{metadata, digest\}\}, ...] \tag{7}$$

In the case of UUID default, the return value will be an array, as shown in (7). Due to the UUID is unknown, the data query service is unable to provide accurate queries and chooses to return all records of the KEY instead.

2.5 Access Management Service

Until now, records are public. All users are free to access any record they want, which will definitely cause privacy problems. As a result, we subjoin the access management service for privacy protection.

The implementation of this service is based on the two parameters in records which are stored in subchains. They are reader_set and writer_set. Records can only be accessed by users in their reader_set. Before each return of data query service, the reader_set will be checked. Compared with reader_set, writer_set is much more important. Users in the writer_set own the permission to modify parameters except UUID and metadata in a record, including reader_set and writer_set. Consequently, it should be cautious to add any user to the writer_set.

$$group\_id\ CREATE\_GROUP([user\_id, ...]) \tag{8}$$

To save storage space, the record size should be compressed as much as possible. Group_id is designed to represent a batch of users. The group_id can be used as a special user_id. As mentioned before, a special subchain is built to store authentication information of users. With the help of this subchain, building user groups is not a difficult task.

$$void\ UPDATE\_GROUP(group\_id, [user\_id, ...]) \tag{9}$$

The access management service exposes four interfaces as shown in (8), (9), (10) and (11). CREATE_GROUP is used to create a new group. It accepts a set

of user_id and returns a unique group_id. The first user in the set will have the administrator privilege.

$$void\ UPDATE\_RSET(KEY, UUID, [user\_id, ...]) \tag{10}$$

UPDATE_GROUP is used to modify an existing group. Only the first user in the existing group will be allowed to update the user_id group.

$$void\ UPDATE\_WSET(KEY, UUID, [user\_id, ...]) \tag{11}$$

UPDATE_RSET and UPDATE_WSET are used to update the reader_set and the writer_set of a record stored in subchains. The writer_set will be checked before each operation executed.

3 Workflow and Deployment

In this section, we discuss the usage of our service, including general workflows and the deployment strategy.

3.1 Read Workflow

We will give an explanation of the read workflow as follows.

a) **Search indexes with KEY** Assigned a KEY, indexes stored in the main-chain can be find. If no indexes were found, return directly.
b) **Filter indexes by UUID** If UUID was assigned, use it to filter the indexes. Otherwise, skip this step. If no records left, return.
c) **Query records in subchains** With the help of indexes, query records in the corresponding subchains.
d) **Check reader_set** For each record, check its reader_set. If the user had the read access, append metadata and digest to the result.
e) **Return with result** If the result was not empty, return with it. Otherwise, just return.

In the read workflow, users do not interact directly with subchains. The reason is that subchains are usually composed of relatively few nodes, consequently, more vulnerable. This design can effectively isolate errors. Even if some subchains failed, the biggest loss is just missing some data.

3.2 Write Workflow

The write workflow is shown as follows.

a) **Select subchain**: Select the target subchain. In order to economize on network resources, the nearest subchain should be chosen.
b) **calculate UUID** Calculate the hash value of the metadata as UUID.

c) **Query index with KEY and UUID** With KEY and UUID, query the index from the mainchain. If not found, skip to step *g*.
d) **Query record** With the index, query the record in the corresponding subchain. If not found, skip to step *g*.
e) **Check writer_set** Check the writer_set. If the user did not own the write access, return.
f) **Check repeat** With UUID, check if there is already a valid record with the same UUID and the user does not have write access. If so, change a subchain until fulfilling the requirement and update the subchain_ID. If no such subchain found, return.
g) **Insert record** Insert a new record to the selected subchain.
h) **Upload index** Upload the index to the mainchain.
i) **Return with UUID** Return with the UUID.

In the write workflow, choosing subchain is a key point. For taking up less public bandwidth and lower latency, the nearest one should been selected. Another benefit is that when a subchain works normally, it means that there is no large area collapse occurring nearby. The metadata acquired by users is more valuable, for the source data has a high probability of being available.

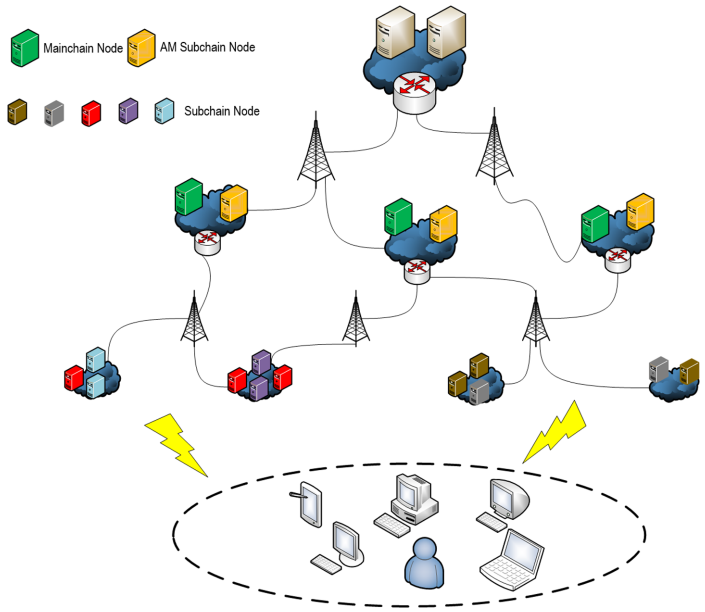


Fig. 3. Deployment strategy

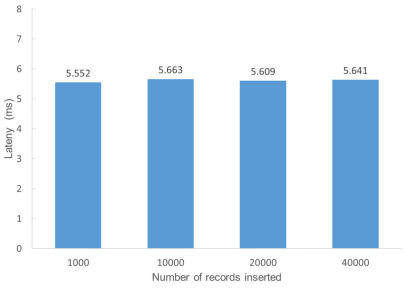
3.3 Deployment Strategy

Our service is designed specifically for the cloud environment. Due to the complex network structure of cloud, the deployment of our service ought to meet certain principles as well. Figure 3 illustrates the deployment strategy of our service.

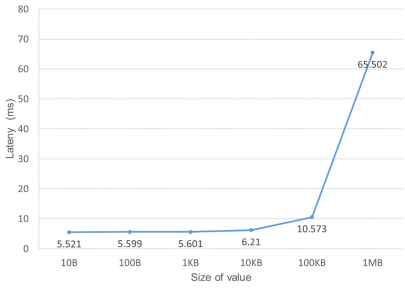
The mainchain nodes should be deployed closer to the cloud center, to cover a larger range. Nodes of subchains should be adjacent to the outer storage systems which are deployed in edge nodes. However, nodes of the special subchain designed for the access management service called AM Subchain Node ought to be neighbors of the mainchain nodes, for lower latency.

4 Evaluation

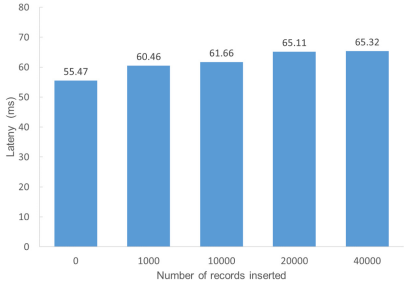
We built a little private blockchain that contains four peer-to-peer nodes based on Hyperledger Fabric. Through simulation, we evaluated the performance of our blockchain network and the availability of our service.



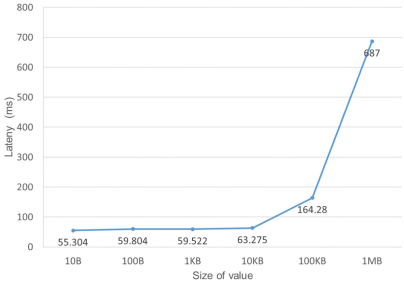
(a) Read latency with different number of records inserted



(b) Read latency with different record size



(c) Write latency with different number of records inserted



(d) Write latency with different record size

Fig. 4. Read and write latency of records with different number or size

4.1 Read Latency

To measure the latency of read operations, we randomly selected 1000 records to read in each node. In order to eliminate random errorterm, the result is the average value of all 4000 measured latency values. As shown in Fig. 4(a), with different number of records inserted, from 1000 to 40000, the read latency keeps around 5.6 ms. This performance is enough to guarantee the user experience of our service.

The size of records may also affect read latency. We inserted records with different value size before starting the reading test. The result is shown in Fig. 4(b). When the value size increases from 10 B to 10 KB, the change of read latency is not obvious. When the value size reaches 100 KB and 1 MB, it causes a large growth in read latency.

4.2 Write Latency

Similarly, we measured the write latency. As shown in Fig. 4(c) and Fig. 4(d), the result is average value as well. Despite the write latency is nearly 10 times higher than the read latency, 65 ms is still in an acceptable range. It is worth noting that as the value size increases, the latency of write operations will become a fairly large value. Therefore, both for performance and capacity considerations, the size of digest uploaded by users should be limited to under 10 KB.

4.3 Availability

A significant advantage of decentralized systems is high availability. Suppose we built a system with a mainchain and two subchains, the mainchain has four nodes while each subchain has three nodes. Table 1 lists the probability of each chain being inaccessible when a certain number of arbitrary nodes have crashed.

Due to all chains own at least three nodes, in the case where the number of crashed nodes is less than three, the availability of any chain will not be

Table 1. Failure probability of each chain when random nodes crash

Number	Mainchain	Subchain_1	Subchain_2
<3	0	0	0
3	0	0.83%	0.83%
4	0.48%	3.33%	3.33%
5	2.38%	8.33%	8.33%
6	7.14%	16.67%	16.67%
7	16.67%	29.16%	29.16%
8	33.33%	46.67%	46.67%
9	60%	70%	70%
10	100%	100%	100%

influenced. From the results we can discover that although the mainchain merely has one more node than subchains, it is more robust evidently. If half of the nodes crashed, the inaccessible probability of each chain is under 10%. Even if only one tenth of the nodes is working properly, the mainchain still has a 40% inventory possibility. Briefly, the blockchain network has extremely high availability and adding more nodes will greatly enhance it.

5 Related Works

The InterPlanetary File System (IPFS) [11] is a peer-to-peer distributed file system that seeks to connect all computing devices with the same system of files. With the help of a distributed hashtable [12], an incentivized block exchange, and a self-certifying namespace, IPFS has overcome shortcomings of centralized storage systems like network congestion and waste of storage space. However, IPFS has no tamper resistance which is quite important in our scenario.

BigchainDB [13] is a decentralized database with large scale. BigchainDB inherits features of modern distributed databases and adds blockchain characteristics. Of course, getting this property comes at a price. BigchainDB is an implementation of private blockchain. Compared to our service, adding new nodes in BigchainDB is troublesome. Although with quite high throughput and capacity, it is not realistic to deploy BigchainDB in the edge environment.

S. Ali and G. Wang [14] applied blockchain to PingER [15], a worldwide end-to-end Internet performance measurement project. With the help of permissioned blockchain and Distributed Hash Tables (DHT), the data storage and access framework stores metadata of the file on the blockchain, whereas the actual files are stored at multiple locations. Compared to our service, it specifies the storage location that will not help reduce network overhead. Besides, it does not support uploading digest, thence it is more suitable to store specific data like daily PingER data that does not need complicated query function.

6 Conclusion

With the explosive growth in data volume, cloud storage service is facing a serious situation that the network traffic of data centers is gradually unable to meet user needs. In order to solve this problem, storing source data in edge nodes as well as uploading metadata to clouds is a good choice.

However, storing metadata in traditional cloud storage systems has serval shortages. Blockchain, an underlying technology of Bitcoin can help to solve these problems. Based on blockchain, we propose a decentralized cloud storage service with high tamper-resistant.

For the purpose of expanding storage capacity, a layered and collaborative structure of blockchain has been designed. The blockchain network of our service allows users to upload metadata and digest of source data to subchains. To deal with query operations, the mainchain preserves indexes of records. Another function provided by our service is access management. We demonstrate the read

and write workflow of our service. A deployment strategy is also recommended in this paper. To evaluate the performance and availability, we simulate the blockchain network by Hyperledger Fabric. In conclusion, the blockchain network is capable of providing the features we need.

Acknowledgment. This paper is supported by Project 213.

References

1. Joseph, A.D., Katz, R., Konwinski, A., Gunho, L., Patterson, D., Rabkin, A.: A view of cloud computing. Commun. ACM **53**(4), 50–58 (2010)
2. Waibel, P., Matt, J., Hochreiner, C., Skarlat, O., Hans, R., Schulte, S.: Cost-optimized redundant data storage in the cloud. Serv. Oriented Comput. Appl. **11**(4), 411–426 (2017). https://doi.org/10.1007/s11761-017-0218-9
3. Cisco Visual Networking Index: Forecast and methodology, 2016–2021. http://www.cisco.com
4. Duval, E., Hodgins, W., Sutton, S., Weibel, S.L.: Metadata principles and practicalities. D-lib Mag. **8**(4), 1082–9873 (2002)
5. Amazon Glacier. https://aws.amazon.com/glacier/. Accessed Nov 2017
6. Renner, T., Müller, J., Kao, O.: Endolith: a blockchain-based framework to enhance data retention in cloud storages. In: 2018 26th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pp. 627–634. IEEE (2018)
7. Lee, B., Awad, A., Awad, M.: Towards secure provenance in the cloud: a survey. In: Proceedings of the 8th International Conference on Utility and Cloud Computing, pp. 577–582. IEEE Press (2015)
8. Nakamoto, S., et al.: Bitcoin: a peer-to-peer electronic cash system (2008)
9. Zheng, Z., Xie, S., Dai, H.-N., Chen, X., Wang, H.: Blockchain challenges and opportunities: a survey. Int. J. Web Grid Serv. **14**(4), 352–375 (2018)
10. Borthakur, D.: Under the hood: building and open-sourcing RocksDB. Facebook Engineering Notes (2013)
11. Benet, J.: IPFS-content addressed, versioned, P2P file system. arXiv preprint arXiv:1407.3561 (2014)
12. Naor, M., Wieder, U.: A simple fault tolerant distributed hash table. In: Kaashoek, M.F., Stoica, I. (eds.) IPTPS 2003. LNCS, vol. 2735, pp. 88–97. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45172-3_8
13. McConaghy, T., et al.: BigchainDB: a scalable blockchain database. White Paper, BigChainDB (2016)
14. Ali, S., Wang, G., White, B., Cottrell, R.L.: A blockchain-based decentralized data storage and access framework for PingER. In: 2018 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE), pp. 1303–1308. IEEE (2018)
15. Matthews, W., Cottrell, L.: The PingER project: active Internet performance monitoring for the HENP community. IEEE Commun. Mag. **38**(5), 130–136 (2000)