Blockchain: Research and Applications 2 (2021) 100032

Contents lists available at ScienceDirect

Blockchain: Research and Applications

journal homepage: www.journals.elsevier.com/blockchain-research-and-applications

BlockHDFS: Blockchain-integrated Hadoop distributed file system for secure provenance traceability

Viraaji Mothukuri [a], Sai S. Cheerla [a], Reza M. Parizi [a], Qi Zhang [b], Kim-Kwang Raymond Choo [c,*]

[a] College of Computing and Software Engineering, Kennesaw State University, Kennesaw, GA, 30144, USA
[b] IBM Thomas J. Research Center, Yorktown Heights, NY, 10598, USA
[c] Department of Information Systems and Cyber Security, University of Texas at San Antonio, San Antonio, TX, 78249, USA

ARTICLE INFO

Keywords:
Big data
Hadoop
Blockchain
Hyperledger fabric
Hadoop distributed file system (HDFS)
Traceability
Security
Privacy

ABSTRACT

Hadoop Distributed File System (HDFS) is one of the widely used distributed file systems in big data analysis for frameworks such as Hadoop. HDFS allows one to manage large volumes of data using low-cost commodity hardware. However, vulnerabilities in HDFS can be exploited for nefarious activities. This reinforces the importance of ensuring robust security to facilitate file sharing in Hadoop as well as having a trusted mechanism to check the authenticity of shared files. This is the focus of this paper, where we aim to improve the security of HDFS using a blockchain-enabled approach (hereafter referred to as BlockHDFS). Specifically, the proposed BlockHDFS uses the enterprise-level Hyperledger Fabric platform to capitalize on files' metadata for building trusted data security and traceability in HDFS.

1. Introduction

Hadoop distributed file system (HDFS) [1,2], one of the most widely used file systems in organizations that deal with big data applications, is mainly used for batch processing of data. It is known for its high throughput data accessibility with low latency. There are several alternatives to HDFS, such as Ceph [3], GPFS [4], and Hydra [5]. Unlike other file systems, HDFS was developed by the Apache Software Foundation with the specific aims to store large datasets in a distributed manner to address hardware failures and to perform Map Reduce [6] operations for data analytics. MapReduce is a pre-built framework in HDFS. The users can use one or more files in MapReduce to map and reduce (sort) the data inside files to obtain the desired output. Java code snippets are, however, required to be written to perform MapReduce operations. MapReduce considers a file in the input directory, and the desired output can be made to write in an output directory. What makes HDFS so popular today is, perhaps, its architecture. HDFS is similar to UNIX file system architecture, but the data is distributed among several hard disks and can be run on commodity hardware, making it cheaper to use and maintain. Apache Hadoop is an open-source framework that runs on top of HDFS.

Being a distributed file system, Apache Hadoop is effective. Right from the beginning, the motivation of Hadoop has been to build a high-throughput system with fail-safe features. There are two ways to upload a file into HDFS, namely: through the command line and using APIs. Both ways can be accessed using SSH if there is a need for a remote connection, and the command line and API have multiple commands for different file operations. HDFS supports creating, appending, opening, deleting, renaming, status, and listing functions. Once a file is uploaded into HDFS, various ecosystem frameworks (e.g., Spark [7], Hive [8], HBase [9]) can be used to perform the required data analytics. These ecosystem applications generally use the Hadoop Main API to perform data operations from and to the HDFS.

Inspired by the unique security-by-design and tamper-resistant characteristics of blockchain technology in contemporary application domains [10–14], we propose a new approach, called BlockHDFS, that incorporates blockchain functions into the HDFS by capitalizing on file metadata stored in tamper-proof blocks. In our work, we use Hyperledger Fabric [15] to get the data from HDFS and store them inside Hyperledger as an asset. Hyperledger Fabric is a permissioned blockchain with a distributed ledger that supports smart contracts [16]. Only authenticated users are able to see the data inside Hyperledger, unlike other public blockchains such as Ethereum, Bitcoin, or Monero. Using Hyperledger

* Corresponding author.
E-mail addresses: vmothuku@students.kennesaw.edu (V. Mothukuri), scheerla@students.kennesaw.edu (S.S. Cheerla), rparizi1@kennesaw.edu (R.M. Parizi), q.zhang@ibm.com (Q. Zhang), raymond.choo@fulbrightmail.org (K.-K.R. Choo).

https://doi.org/10.1016/j.bcra.2021.100032
Received 27 May 2021; Received in revised form 14 September 2021; Accepted 6 October 2021

Fabric also requires less computation power to run, as everything resides on a private business network suited for organizations and enterprise needs. Consequently, this eliminates the necessity of blockchain mining.

Storing metadata of the files into the blockchain can be implemented on almost every filesystem as every file system needs to maintain metadata. HDFS uses CRC32 or CRC32C, a 32-bit Cyclic Redundancy Check based on the Castagnoli polynomial algorithm to generate the hash of the files. However, when the same hash is accessed through Hadoops API, the MD5 version of hash is produced, which is MD5-of-MD5-of-CRC32C. This hash can be used to check if there is a change in the file after it is stored in HDFS. To the best of our knowledge, our work is one of the first attempts in the research community to integrate blockchain into HDFS for better data security and trusted traceability. Overall, our results indicate positive impacts on the security and traceability of files in HDFS, which makes the proposed approach promising and noteworthy.

The rest of this paper is structured as follows. Section 2 introduces the reader to the required background and provides an overview of the related approaches. In Section 3, the threat model is presented. Section 4 presents the proposed BlockHDFS. Section 5 discusses the experimental study and the results. Finally, Section 6 gives the concluding remarks.

2. Background and related work

2.1. Hadoop Distributed File System (HDFS)

HDFS [2], a distributed file system for parallel big data processing, uses the master-slave architecture to schedule the tasks and manage the data transfer among different computing nodes. In HDFS, the master node is called NameNode and the slave node is called DataNode. Multiple NameNodes can be used to maintain high availability using an active-passive relationship. NameNodes are responsible for performing block operations, and they store metadata related to the file system, such as where each chunk of a file is located. HDFS has been widely used by distributed data processing frameworks such as Map Reduce [6] to perform efficient big data analytics jobs. A MapReduce job consists of Map tasks that perform the same map operation on different data chunks in parallel and reduce tasks that use shuffling and reducing functions to generate desired outputs. Different MapReduce jobs can share the same cluster of machines and are coordinated by a job tracker.

HDFS can be implemented on commodity hardware with decent computing power. HDFS maintains replicas of a file in the form of blocks. The blocks are present in different DataNodes in order to recover files in case of a crash of a DataNode, which makes HDFS highly fault-tolerant. The read and write operations performed on HDFS are collaboratively managed by both NameNodes and DataNodes. In addition, NameNodes maintain logs of all the operations and store them inside an image file at checkpoints. A checkpoint is a time where all the logs are combined into a file called fsImage. When a NameNode is turned back on after being switched off, all the information about the blocks is loaded from the fsImage file.

2.2. Hyperledger Fabric

Hyperledger Fabric [15] is an open-source and openly governed collaborative effort to advance cross-industry blockchain technologies for business and the Linux Foundation hosts it. Hyperledger Fabric is a blockchain framework implementation and one of the Hyperledger projects intended as a foundation for developing applications/solutions with a modular architecture suited for enterprise-level solutions. The main components of Hyperledger Fabric are briefly described below.

2.2.1. Distributed ledger

The Fabric ledger has two parts: State Data and Transaction Logs. The ledger is the sequenced, tamper-resistant records of all state transitions. State transitions are the result of chain code invocations (“transactions”) submitted by participating parties. Each transaction results in a set of asset key-value pairs that are committed to the ledger as creates, updates, or deletes. Each transaction has a unique ID, its time-stamp, and contains signatures of every endorsing peer, and is submitted to the ordering service.

The ledger is comprised of a blockchain to store the immutable, sequenced records in blocks, as well as a state database to maintain the current Fabric state. There is one ledger per channel. Each peer maintains a copy of the ledger for each channel in which they are a member.

2.2.2. Nodes

The concept of ‘node’ is common in all blockchain technologies. In a public blockchain like Ethereum [17], anyone can participate as a node by downloading the node client software. A node becomes the communication endpoint in the blockchain network. Peer-to-peer protocols are used to keep the distributed ledger in sync across different nodes in the network. In a permissioned blockchain, such as Hyperledger Fabric, nodes need a valid certificate to be able to communicate to the network, where the participant's identity is not the same as the node's identity.

2.2.3. Channel

Members can participate in multiple Hyperledger blockchain networks. Transaction in each network is isolated and this is made possible by way of what is referred to as the channels. Channel is a data partitioning mechanism to control transaction visibility only to stakeholders. Non-registered members on the network are not allowed to access the channel and will not see transactions on the channel.

2.3. Related work

There are several works [18–20] that discuss the integration of distributed big data platforms and blockchain in literature. The main intention of HDFS is to accelerate big data processing. Therefore, very few security features were embedded in HDFS at its early stage. Later on, Kerberos Authentication Protocol [21] was created to improve HDFS authentication. For example, HDFS started to support ACL (Access Control List) for file authorizations. It also supports auditing for system security [22], which, however, lacks a format to make such auditing easy to read or be unified. Some ecosystems running on HDFS, such as Apache NiFi [23], support system auditing more productively, but there is no direct way to get hold of the audit logs in a standard format. Coming to the main security feature in HDFS, which is Encryption, most of the data sent between the HDFS and its clients is done through RPC (Remote Procedure Call), which is encrypted using SASL (Simple Authentication and Security Layer) RPC. There are two types of data encryptions in HDFS which are data-at-rest and data-in-transit. Data-in-transit refers to the data being transmitted from the clients to HDFS, and it can be encrypted using RPC, as mentioned above. However, data-at-rest, which refers to the data stored in HDFS, is still at high risk. There have been very few approaches to secure data at rest. Project Rhino [24], for example, is designed to help encrypt the data-at-rest, but this technique requires tweaking the components of HDFS such as configuration files, adding new properties, and changing ports as it is installed system-wide. Orthogonal to the approach proposed by Rhino [24], our solution targets providing an immutable set of logs transparently. This approach does not require any change to the user application code. While such immutable logs can serve as a trusted source of information to help users investigate what has gone wrong when the data in HDFS is compromised.

3. Threat model

Many hackers have targeted HDFS mainly because of the values of the data it holds. The data in HDFS is mostly organizational data that contains much information, such as sales, employees, and analytical market data. Fig. 1 shows our proposed threat model in HDFS. The dotted line in this figure represents Trust Boundary. On the left of the dotted line is the users, who connect to the HDFS, which are on the right side of the dotted

line, via the API component. Examples of such API components can be WebHDFS [25], or HttpFS [26], which are responsible for all the data retrieval and modifications. Thanks to approaches such as SASL RPC, data being uploaded to HDFS is hard to be altered before it reaches HDFS.

Concretely speaking, Hadoop uses open ports to project the API to the Internet or a network. Every open port has its own purpose. For example, open port 23 is used for telnet services. Hackers can exploit such open ports to perform vulnerability scans, which eventually enables them to access the data in the DataNodes illegally. However, for data on HDFS, a hacker can exploit the API and modify the data inside the DataNodes. In most cases, the attacker performs the footprinting, using software such as NMap [27] and Zen Map [28], to get the IP address of the target network and then carries out port scanning to find information such as the operating systems, the running services, or system architecture of the machines on this network. Since an open port with a process running in the background will always listen for input from other processes in the network, hackers will find these processes and exploit them. For instance, WebHDFS uses a default port of 50070 for NameNode, and organizations use this port by default. A malicious user who gains access to such a port can easily modify the data inside the DataNode, and all it takes is the name of the user who is using the HDFS.

To tackle such attacks, our solution takes the advantage of blockchain to create an immutable set of logs for HDFS, which can serve as a trusted source of information for an effective investigation when something has gone wrong in HDFS. Even though our solution cannot completely stop a hacker from attacking the HDFS data via an open port, it provides a way to log the data modifications in a permanent and tamper-proof way, which helps to guide the investigators to identify the hackers.

4. BlockHDFS

In this section, we introduce our proposed solution named BlockHDFS, which enhances the security of HDFS in a user transparent way by storing metadata of files from HDFS in a blockchain.

4.1. Architecture of BlockHDFS

Fig. 2 presents the proposed architecture of BlockHDFS, which consists of three components: an HDFS cluster including the NameNodes and DataNodes, a permissioned blockchain network such as Hyperledger Fabric, and a NodeJS Client which acts as a bridge connecting the HDFS cluster and the blockchain network.

BlockHDFS is designed in a user transparent way, thus it shares many features and operations with traditional HDFS. For instance, a user uploads a file to BlockHDFS by using either the API or command line, and the file transfer is secured using DEK encryption [29] when data is at rest. The file residing in the DataNode is then replicated into a number of other DataNodes based on the preset replication factor. In our design, Hyperledger Fabric [30] is used to store file metadata, which includes but is not limited to the hash value, access time, and modification time of a file into the blockchain, while HDFS still manages the files themselves. In addition, a NodeJS client periodically extracts the file metadata from HDFS using the WebHDFS REST API and then stores them in the ledger of Hyperledger Fabric. The user can always query the Hyperledger Fabric to understand what has happened to a given file, as all the related information is stored on the blockchain ledger, which provides an immutable and trusted traceability.

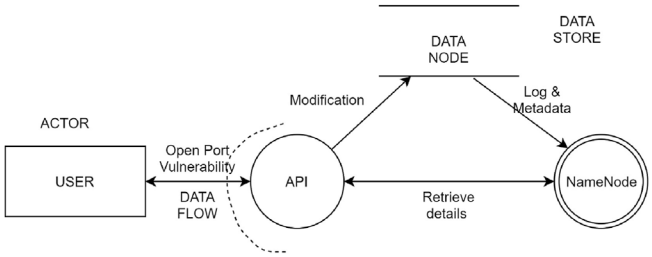

Fig. 1. Data flow diagram of threat model.

It is worth noting that the client in Hyperledger Fabric is only used to store the file data, also known as metadata including access time, modification time, and hash value—note, the current architecture allows adding multiple metadata beyond those listed here. The files themselves are still managed by HDFS itself. The file metadata is provided by the WebHDFS REST API. The client in between is responsible for getting these three values from the REST API and storing them in Hyperledger Fabric. The user can always make a query to the Hyperledger Fabric to read changes that happened to a given file as all the information is stored in the form of secured transactions on the blockchain, improving security and traceability.

In BlockHDFS, the blockchain is responsible for storing the metadata of the files. Overheads incurred by storing the HDFS file metadata into the blockchain are from two aspects. First, the WebHDFS REST API needs to read the metadata, such as the hash value, of a file from HDFS. Retrieving file metadata from HDFS can introduce some latency. However, as will be shown in the experiments, such latency is not significant. Second, it takes additional operations to store the metadata into the blockchain. However, since metadata size is usually small, such overhead will neither introduce high latency for HDFS operations nor require a large amount of storage on the blockchain. Hyperledger client component is used to communicate between Hyperledger Fabric network and enable the interaction with the ledger. When a file is added to the file system, the client will get the filename using LIST A DIRECTORY operation and retrieve metadata, such as the hash value, access time, and modification time of a file, from HDFS using WebHDFS API. Then the JSON data is added to Hyperledger using the NodeJS client. The NodeJS client will notify users in case of failing to write the metadata into the blockchain.

Following, we describe the main components of BlockHDFS and its functions in reaching our security goals.

4.1.1. User

In BlockHDFS, a user is considered an administrator or one of the employees of an organization who has already been given privileges and access to the HDFS. A user can add, modify, and delete the data inside HDFS based on ACL rules. For example, using the WebHDFS API, a user can add a file to HDFS as follows:

```
curl -i -X PUT "http://<HOST>:
<PORT>/webhdfs/v1/<PATH>? op=CREATE"
```

To add, modify, or delete a file, a user will not be aware of the existence of the Hyperledger Fabric. All the file operations are performed using Hadoop's API only. The user will not directly interact with the Hyperledger unless she/he wants to check the information on the blockchain.

4.1.2. WebHDFS REST API

This REST API can be used to perform file operations and find information about the files in HDFS. If the security is turned off on a local server by disabling the Kerberos or by turning Safe Mode off, the REST API operations can be performed by anyone on the network without authentication. If the security is turned on, then the authentication is taken care of by Kerberos [21]. Authentication using SSH [31] is needed to access Hadoop API if the NodeJS client is hosted on a remote server instead of the local Hadoop server.

4.1.3. File checksum

HDFS uses the CRC32 hashing algorithm [32], while the checksum produced by WebHDFS REST API is the MD5 of CRC32. When a user uploads a file to HDFS, she/he should check the authenticity of the file by validating if the checksum in HDFS matches the checksum on her/his local machine. Even though the file upload is secured by encryption in

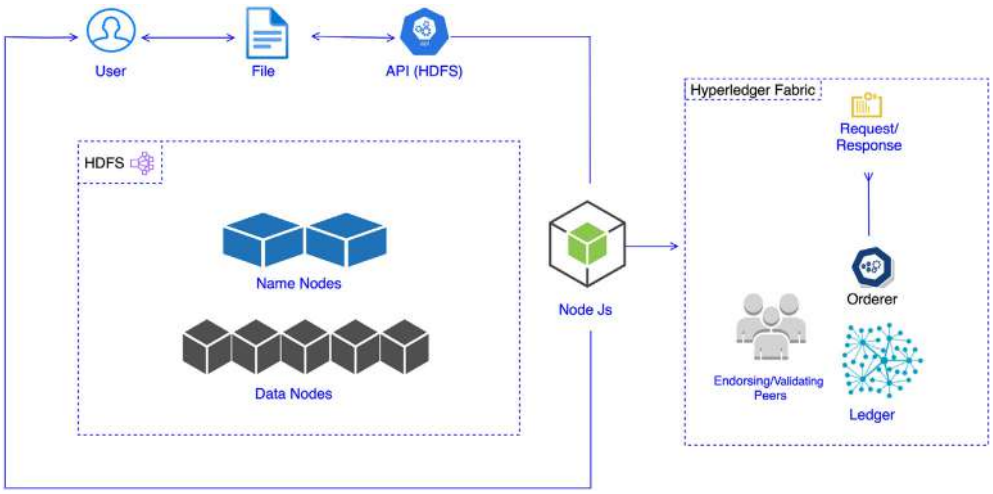

Fig. 2. BlockHDFS architecture model. HDFS: Hadoop Distributed File System.

RPC [33], it is usually beneficial to verify if the file is uploaded correctly without any errors. Therefore, such validation is included in BlockHDFS, which gets the checksum of a file using REST API by using the below command:

```
curl -i -X PUT "http://<HOST>:
<PORT>/webhdfs/v1/<PATH>? op=GETFILECHECKSUM"
```

4.1.4. NodeJS client

The NodeJS Client is an application that is used to obtain the file-related metadata, such as checksum, access time, and modification time of a file, from HDFS. This application also sends the data received from HDFS to the Hyperledger Fabric. In practice, this application can be written in any language which supports REST API operations, while we used NodeJS. If this NodeJS client runs on a remote server, authentication is needed to access the WebHDFS REST API server.

4.1.5. Hyperledger Fabric blockchain

Hyperledger Fabric is a permissioned blockchain suitable for enterprise applications, which requires user authentication. In BlockHDFS, only users of HDFS will be given access to Hyperledger Fabric, which reduces the risk of exposing the blockchain ledger to unauthorized users. HDFS file metadata sent by NodeJS clients, such as file name, hash value, access time, and modification time of a file, is stored in the blockchain ledger in the form of assets. The ledger can be used as an immutable and trusted source of logs by the user if she/he suspects something has gone wrong with the file, which provides an easy way to trace what has happened to a file over time.

4.2. Implementation of BlockHDFS

The client application in between the HDFS and the Hyperledger Fabric is implemented in NodeJS. We use the Hadoop WebHDFS REST API client library for NodeJS to support asynchronous functions, improving the code's performance and responsiveness by enabling parallel processing. We are using a client library called webhdfs-npm, which supports various file system functions. Here in the code, we can use asynchronous functions to execute the WebHDFS API functions faster without waiting for other functions. When a file is uploaded to the HDFS, the NodeJS client checks if a new file has been uploaded to the HDFS using WebHDFS REST API by executing the LISTDIRECTORY operation. If there is a new file, then the NodeJS client retrieves the hash value of this file using GETFILECHECKSUM as well as the access time and modification time of this file using GETFILESTATUS. All such data is parsed from a JSON format. The NodeJS client then writes these values into the Hyperledger Fabric as an asset.

Hyperledger Fabric v1.1 and Hadoop v3.1.2 are used in the prototype. A script was developed to automate all the tasks needed to get the BlockHDFS up and running. As an example of how BlockHDFS works, three files of different sizes are uploaded onto HDFS. The MD5 hash values of these files are generated by WebHDFS and are written into the Hyperledger Fabric using a REST API. The NodeJS client is responsible for sending the GET call to WebHDFS and PUT response to the Hyperledger Fabric. Fig. 3 shows what the JSON data from WebHDFS is like, where the directory structure along with file names and other data can be seen. Such data can be received by sending the LISTDIRECTORY API call to the WebHDFS.

Fig. 4 shows the checksum JSON data from one of the folders inside HDFS. This piece of data comes from WebHDFS as well.

As mentioned earlier, the metadata of a file is sent to the Hyperledger Fabric as an asset with the help of the NodeJS client. An example of such data stored in the blockchain is shown in Fig. 5. If there is a change to this file, a new asset will be added along with the date and time by the NodeJS client. Such assets on the blockchain can be used as an immutable and trusted log by the user to track the history information of a file. In our design, the NodeJS client checks the file changes periodically using a feature called CRON JOBS [34] in Linux. Fig. 5 shows the data in the blockchain ledger in JSON format. The asset Id field contains both hash value and modification time of a file, while the value field contains the filename (Note that the actual length of the hash value was trimmed to make it fit into the figure).

5. Evaluation

This section presents the evaluation results of BlockHDFS in terms of its performance and security capabilities.

5.1. Experimental setup

We carried out the experiments on a single machine with an Ubuntu Operating System running on Oracle VirtualBox. The setup was of Single Node type where there were one NameNode and one DataNode. The specifications of the hardware used are Intel i9-9980HK CPU@4.8Ghz, 16 GB of RAM, and 350 GB of SSD. The services running during the experiments include Hadoop (i.e., HDFS with MapReduce), Docker, Docker-Compose, NodeJS, and Hyperledger Fabric. The Hadoop was installed in a standalone version, whereas the Hyperledger Fabric was installed using Docker. For the blockchain setup, we used Hyperledger Fabric. Specifically, we first created a business network with orderers and peers set to 1 as an experimental setup. With more data or with a need for more peers, they can be easily added based on requirements. This is a

```
FileStatuses:
  FileStatus:
    0:
        accessTime:       1573421133621
        blockSize:        134217728
        childrenNum:      0
        fileId:           16391
        group:            "supergroup"
        length:           1044
        modificationTime: 1573240851529
        owner:            "dr.who"
        pathSuffix:       "C2ImportSchoolSample.csv"
        permission:       "644"
        replication:      1
        storagePolicy:    0
        type:             "FILE"
    1:
        accessTime:       1573421133621
        blockSize:        134217728
        childrenNum:      0
        fileId:           16389
        group:            "supergroup"
        length:           187182221
        modificationTime: 1573202084650
        owner:            "blockhdfs"
        pathSuffix:       "hundredmbsample.csv"
        permission:       "644"
        replication:      1
        storagePolicy:    0
        type:             "FILE"
```


Fig. 3. WebHDFS JSON data.


```
FileChecksum:
    algorithm:  "MD5-of-262144MD5-of-512CRC32C"
    bytes:      "000002000000000000004000001f293a9e4ddf21eabfab88df300538600000000"
    length:     28
```


Fig. 4. WebHDFS checksum data.


```
{
  "$class": "org.example.testnetwork.SampleAsset",
  "assetId": "000001f293a9e4ddf21eabfab88df3005386 , 2019-11-15 01:15:39",
  "owner": "resource:org.example.testnetwork.SampleParticipant#blockhdfs",
  "value": "hundredmbsample.csv"
},
{
  "$class": "org.example.testnetwork.SampleAsset",
  "assetId": "0000a92742488697a07749b1e292a81a640d , 2019-11-24 01:25:48",
  "owner": "resource:org.example.testnetwork.SampleParticipant#blockhdfs",
  "value": "50gb.txt"
},
{
  "$class": "org.example.testnetwork.SampleAsset",
  "assetId": "0000e580f374ea9b81cb726cf7a8c99b3288 , 2019-11-15 01:15:29",
  "owner": "resource:org.example.testnetwork.SampleParticipant#blockhdfs",
  "value": "onembsample.csv"
},
```


Fig. 5. File data in blockchain.

private network where only authenticated users can manage the data on the blockchain. The NodeJS client acts as a mediator to get data from WebHDFS and send it to the Fabric Ledger. As for Hadoop, the WebHDFS REST API is a global client, which means the JSON data is the same irrespective of the number of NameNodes and DataNodes. We used different file sizes ranging from 1 MB to 100 GB of different types, such as .txt, .csv and .dat files.

5.2. Performance

To compare the performance difference between the HDFS (i.e., the baseline scenario) and BlockHDFS, we first ran the Wordcount program using MapReduce with input data of 5.5 GB when no blockchain-related services are running. Then, we ran the same MapReduce program in BlockHDFS, where additional services such as Docker, Hyperledger Fabric are running along with Hadoop. The results from this evaluation are shown in Fig. 6. It can be observed that BlockHDFS brings little performance overhead to the Wordcount job in terms of execution time. We also measured the memory utilization during the experiment, which shows that running Wordcount in HDFS consumed 5.46 GB, while that in BlockHDFS consumed 6.16 GB, which is an acceptable trade-off for security.

We measured how the NodeJS client performs when BlockHDFS is under different loads. We uploaded 6 files to the BlockHDFS, the size of which ranges from 1 MB to 100 GB. The execution time being measured here includes the total time taken by the NodeJS client to obtain file metadata from WebHDFS and send it to the Hyperledger Fabric's client. The performance of the NodeJS client is compared with and without running the Wordcount job in the BlockHDFS, and the results are presented in Fig. 7.

In the above figure, the without load case means that no MapReduce job is running on the BlockHDFS and the with load case has the Wordcount job running. Compared with the without load case, the WebHDFS took a lot more time to generate the hash value in the with load case, especially for larger files, i.e., the 100 GB file. This is because for the mapping process, more disk usage is needed to read the input data file, and at the same time, WebHDFS is trying to read the 100 GB file from BlockHDFS, which imposes much pressure on the disk and thus increases the read time of the file.

In BlockHDFS, the NodeJS client periodically extracts file metadata

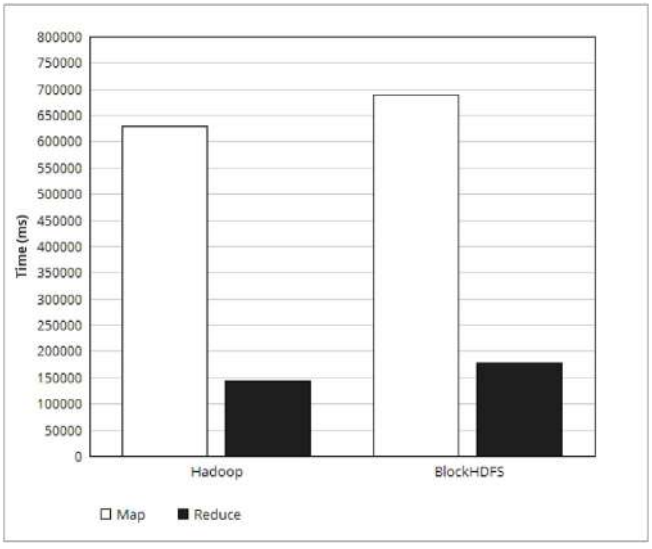

Fig. 6. Execution time of Wordcount in HDFS (Hadoop Distributed File System) and BlockHDFS.

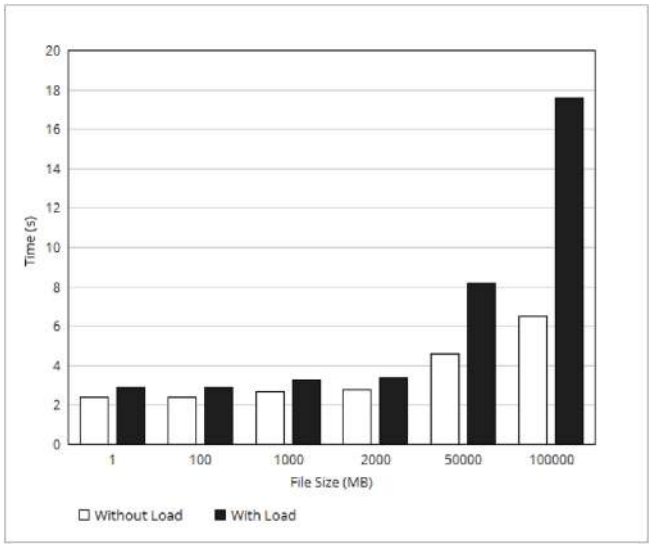

Fig. 7. NodeJS Client execution time with and without Wordcount running in BlockHDFS (Hadoop Distributed File System).

from HDFS. We were interested in understanding how the NodeJS client's running frequency can affect the performance of the jobs running in the same cluster. Therefore, in this set of experiments, we measured the execution time of Wordcount when the NodeJS client runs every 1 s, 3 s, and 5 s, respectively. Fig. 8 gives the results of the MapReduce execution times for the Wordcount program for each frequency. As can be seen from the figure, the execution time of a Map task under 1 s frequency is higher than the other two cases. This is because when NodeJS runs under 1 s frequency, the WebHDFS tries to read the filesystem more frequently than when the NodeJS runs under 3 s frequency and 5 s frequency. But the reduced task time increased gradually as we increased the execution frequency. This is because, as mentioned before, the client only executes again when the current execution is completely finished. Therefore, even though the frequency is set to 1 s, which is trying to read the filesystem more, it is not occupying the RAM more because WebHDFS is taking time to read the filesystem and vice versa.

While experimenting, we found that setting the execution frequency to 1 s did not yield better results as the NodeJS client imposed more disk usage. Each execution of NodeJS client takes around 3 s and executing

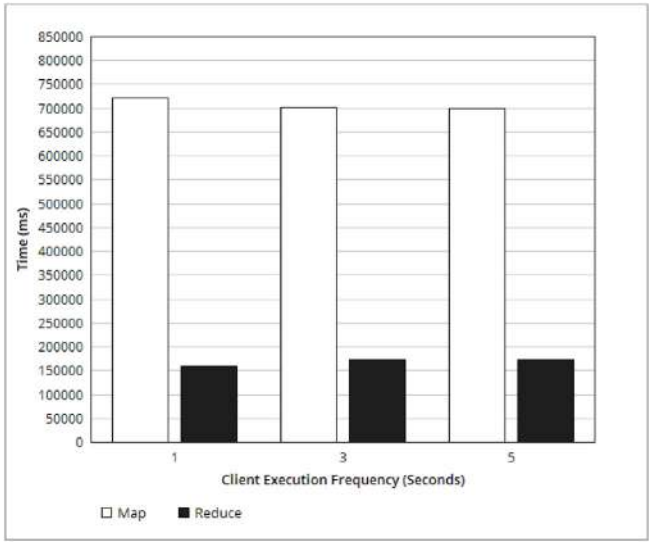

Fig. 8. Execution time of Wordcount in BlockHDFS (Hadoop Distributed File System) with different NodeJS client execution frequencies.

the NodeJS client after 1 s is not feasible. Therefore, using an execution frequency of more than 5 s between executions is reasonable.

5.3. Security implications

BlockHDFS is created to make HDFS more secure by logging the relevant metadata into the blockchain. Accessing HDFS by API is a normal practice, but it can be hacked in many ways, such as Reverse Engineering [35], Man-in-the-middle attack [36], User Spoofing [37], and Session Replays [38]. One concrete attack is DemonBot which gives access to Remote Code Execution (RCE) in HDFS. Let us assume that a hacker has gained access to an HDFS cluster and modified the files in it. In order to investigate what has happened to the files in HDFS, currently, the administrator can only get file metadata such as access time and modification time in terms of the most recent activity or the last change to the file, but not in terms of a trusted, complete log. This would not be enough or feasible to monitor in case of rapid file changes in a span of time. Furthermore, such metadata is also located in the HDFS, which may have already been compromised, thus making it completely unreliable. To overcome these problems, our solution offers a transparent, reliable, and low-cost approach to capturing the historical information of the HDFS files and recording them in the blockchain. As the data on the blockchain is immutable, these changes are recorded permanently in a non-repudiation manner, providing an excellent chain of custody. The HDFS administrator can check these changes by going to the NodeJS client. Any change to the files inside a specified folder can be recorded to the blockchain. In this way, BlockHDFS provides a convenient and trusted way to log the file changes, which makes the investigation job easier when an administrator wants to check the file authenticity.

If we consider the attack scenario of the Hadoop file system in our proposed approach, BlockHDFS, the attacker intrusion details are logged in detail on the blockchain ledger. This serves as a traceback to recover the file system before the attack and unauthorized changes. Moreover, as the ledger is impossible to alter, the attacker would be unable to make any changes to cover up the traces of the attack. The admin of the BlockHDFS can access the log trace, roll back the changes, and design recovery strategies according to the attack's impact.

5.4. Current limitations

Although the BlockHDFS has a generic and highly scalable architecture, its current implementation has some limitations. First, it is possible to create folders and sub-folders inside HDFS to store files. The depth of the LISTDIRECTORY is only 1 in HDFS, which means only the files present inside the top level of a folder are read by the WebHDFS client. All the files inside sub-folders are therefore ignored. This is because there is no recursive way to list filenames inside the sub-folders of a folder. Without the filename, the current implementation of BlockHDFS cannot pass a request to the REST API to calculate the hash and other required parameters. Second, the current NodeJs client, which is responsible for getting the data from WebHDFS and sending it to the Hyperledger fabric ledger, is set to execute periodically for a set amount of time. Then, the Hyperledger creates a new asset when there is a change in data of file values such as modification time, hash, and access time. In the live production version, the NodeJs client should be made to work in real-time instead of executing it periodically by integrating it directly into Hadoop, which is a part of our future work.

6. Conclusions and future work

Integrating blockchain with distributed file systems such as HDFS can potentially improve security and traceability. In the context of this paper, the original design of Hadoop is more optimized for file processing instead of security-by-design. Hence, in this paper, we proposed a new approach to introduce blockchain (and more specifically, Hyperledger) to enhance the security of the HDFS ecosystem. In the current

implementation, we have only added minimal metadata to the blockchain, but with BlockHDFS, one can easily add more features suited to their application needs. For future work, BlockHDFS can be extended to work in real-time with the file system and track all data between NameNode and DataNodes of the HDFS in the secure ledger with multiple nodes.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Apache hadoop, URL, http://hadoop.apache.org, 2006.
[2] K. Shvachko, H. Kuang, S. Radia, R. Chansler, The hadoop distributed file system, in: 2010 IEEE 26th Sym- Posium on Mass Storage Systems and Technologies (MSST); 3–7 May 2010; Incline Village, NV, USA, IEEE, Piscataway, NJ, USA, 2010, pp. 1–10.
[3] S.A. Weil, S.A. Brandt, E.L. Miller, D.D.E. Long, C. Maltzahn, Ceph: a scalable, high-performance dis- tributed file system, in: Proceedings of the 7th Symposium on Operating Systems Design and Implementation, OSDI '06; 6–8 Nov 2006; Seattle, WA, USA, USENIX Association, Berkeley, CA, USA, 2006, pp. 307–320.
[4] F. Schmuck, R. Haskin, Gpfs: a shared-disk file system for large computing clusters, in: Proceedings of the 1st USENIX Conference on File and Storage Technologies, FAST '02; 28–30 Jan 2002; Monterey, CA, USA, USENIX Association, Berkeley, CA, USA, 2002.
[5] C. Ungureanu, B. Atkin, A. Aranya, et al., HydraFS: A high-throughput file system for the hydrastor content-addressable storage system, in: Proceedings of the 8th USENIX Conference on File and Storage Technologies, FAST'10; 23–26 Feb 2010; San Jose, CA, USA, USENIX Association, Berkeley, CA, USA, 2010, pp. 225–239.
[6] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
[7] M. Zaharia, R.S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M.J. Franklin, et al., Apache spark: a unified engine for big data processing, Commun. ACM 59 (11) (2016) 56–65.
[8] Apache hive (2013). URL https://hive.apache.org/.
[9] M.N. Vora, Hadoop-hbase for large-scale data, in: Proceedings of 2011 International Conference on Computer Science and Network Technology; 24–26 Dec 2011; Harbin, China, IEEE, Piscataway, NJ, USA, 2011, pp. 601–605.
[10] T. Alladi, V. Chamola, R.M. Parizi, K.R. Choo, Blockchain applications for industry 4.0 and industrial iot: a review, IEEE Access 7 (2019) 176935–176951, https://doi.org/10.1109/ACCESS.2019.2956748.
[11] E. Nyaletey, R.M. Parizi, Q. Zhang, K.-K.R. Choo, Blockipfs—blockchain-enabled interplanetary file system for forensic and trusted data traceability, in: 2019 IEEE International Conference on Blockchain (IEEE Blockchain-2019); 14–17 Jul 2019; Atlanta, GA, USA, IEEE, Piscataway, NJ, USA, 2019, 00012.
[12] A. Yazdinejad, R.M. Parizi, A. Dehghantanha, K.-K.R. Choo, P4-to-blockchain: a secure blockchain-enabled packet parser for software defined networking, Comput. Secur. 88 (2020), 101629, https://doi.org/10.1016/j.cose.2019.101629.
[13] A. Yazdinejad, R.M. Parizi, A. Dehghantanha, K.R. Choo, Blockchain-enabled authentication handover with efficient privacy protection in sdn-based 5g networks, IEEE Trans. Netw. Sci. Eng. 8 (2) (2019) 1120–1132, https://doi.org/10.1109/TNSE.2019.2937481.
[14] A. Yazdinejad, R.M. Parizi, A. Dehghantanha, H. Karimipour, G. Srivastava, M. Aledhari, Enabling drones in the internet of things with decentralized blockchain-based security, IEEE Internet of Things J. 8 (8) (2021) 6406–6415.
[15] E. Androulaki, A. Barger, V. Bortnikov, et al., Hyperledger fabric: a distributed operating system for permis- sioned blockchains, in: Proceedings of the Thirteenth EuroSys Conference, EuroSys '18; 23–26 Apr 2018; Porto, Portugal, ACM, New York, NY, USA, 2018, pp. 1–15, 30:1–30:15.
[16] R.M. Parizi, Amritraj, A. Dehghantanha, Smart contract programming languages on blockchains: an empirical evaluation of usability and security, in: S. Chen, H. Wang, LJ. Zhang (Eds.), Blockchain–ICBC 2018, Springer, Cham, Switzerland, 2018, pp. 75–91.
[17] G. Wood, Ethereum: a Secure Decentralised Generalised Transaction Ledger. Ethereum project yellow paper 151, 2014, pp. 1–32 (2014).
[18] X.F. Zhang, Y.M. Wang, Research on intelligent medical big data system based on hadoop and blockchain, 2021, EURASIP J. Wirel. Commun. Netw. (1) (2021) 7, https://doi.org/10.1186/s13638-020-01858-3.
[19] N. Abdullah, A. Hakansson, E. Moradian, Blockchain based approach to enhance big data authentication in distributed environment, in: 2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN); 4–7 Jul 2017; Milan, Italy, IEEE, Piscataway, NJ, USA, 2017, pp. 887–892.
[20] M.S. Sahoo, P.K. Baruah, Hbasechaindb—a scalable blockchain framework on hadoop ecosystem, in: R. Yokota, W. Wu (Eds.), Supercomputing Frontiers, Springer, Cham, Switzerland, 2018, pp. 18–29.
[21] B.C. Neuman, T. Ts'o, Kerberos: an authentication service for computer networks, IEEE Commun. Mag. 32 (9) (1994) 33–38.
[22] P. Koopman, Embedded system security, Computer 37 (7) (2004) 95–97.
[23] Apache nifi, URL, https://nifi.apache.org/.
[24] P.P. Sharma, C.P. Navdeti, Securing big data hadoop: a review of security issues, threats and solution, Int. J. Comput. Sci. Inf. Technol. 5 (2) (2014) 2126–2131.
[25] Webhdfs rest api, URL, https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/WebHDFS. html.
[26] O. Kiselyov, A network file system over http: remote access and modification of files and files, in: USENIX Annual Technical Conference, FREENIX Track, 1999, pp. 75–80.
[27] Nmap. URL https://nmap.org/.
[28] Zenmap Introduction. URL https://nmap.org/zenmap/.
[29] M. H. Etzel, D. W. Faucher, D. N. Heer, D. P. Maher, R. J. Rance, Data encryption key management system, US Patent 6,577,734 (Jun. 10 2003).
[30] E. Androulaki, A. Barger, V. Bortnikov, et al., Hyperledger fabric: a distributed operating system for permissioned blockchains, in: Proceedings of the Thirteenth EuroSys Conference, EuroSys '18; 23–26 Apr 2018; Porto, Portugal, ACM, New York, NY, USA, 2018, pp. 1–15.
[31] T. Ylönen, SSH–secure login connections over the internet, in: 6th USENIX Security Symposium, Focusing on Applications of Cryptography; 22–25 Jul 1996; San Jose, CA, USA vol. 37, 1996.
[32] X. Yu, Cyclic redundancy check computation algorithm, 802,038, US Patent 6 (Oct. 5 2004).
[33] B.J. Nelson, Remote Procedure Call. Ph.D Thesis, Carnegie Mellon University, Ann Arbor, MI, USA, 1981.
[34] M.S. Keller, Take command: cron: job scheduler, Linux J. 1999 (65es) (1999) 15.
[35] E.J. Chikofsky, J.H. Cross, Reverse engineering and design recovery: a taxonomy, IEEE Software 7 (1) (1990) 13–17.
[36] A. Ornaghi, M. Valleri, Man in the middle attacks, in: Blackhat Conference Europe; 14–15 May 2003; Amsterdam, the Netherlands, Blackhat Conference Europe, vol. 1045, 2003.
[37] E.W. Felten, D. Balfanz, D. Dean, D.S. Wallach, Web spoofing: an internet con game, Software World 28 (2) (1997) 6–8.
[38] M. Rouse, M. Rouse, What is Session Replay? - Definition from whatis.com. URL https://searchsecurity.techtarget.com/definition/session-replay.