International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 21 (2017) pp. 11374-11382

Simplified HDFS Architecture with Blockchain Distribution of Metadata

Deepa S. Kumar
Research Scholar, Karpagam Academy of Higher Education
Karpagam University, Coimbatore, Tamilnadu, India.
Orcid: 0000-0001 9479-8508

M. Abdul Rahman
Pro-Vice Chancellor, APJ Abdul Kalam Technological University
Thiruvananthapuram, Kerala, India.

Abstract

Big data storage becomes one of the great challenges due to the rapid growth of huge volume, variety, velocity and veracity of data from various sources like social sites, Internet of Things, mobile users and others. These data cannot be processed by the traditional database systems. Hadoop is a distributed and massively parallel processing system for big data whereby the storage is based on the distributed file system called HadoopDistributed File System (HDFS). HDFS is organized as a collection of Data Nodes which stores data as blocks and the information regarding the data is kept as metadata which is stored in an expensive, reliable hardware known as Name Node, which serves as a master server. The existing HDFS architecture suffers from single point of failure due to the existence of single Name Node, where the metadata creation is being done. Metadata is the whole information regarding the distribution of data blocks, replica management and block size information.This paper proposed a simplified architecture for big data storage which eliminates the concept of master node called Name Node with the functionalities of the Name Node being distributed using blockchain technology. Metadata creation and blockchain placement was implemented and tested in a cluster of nodes. The code for metadata creation in python runs successfully for the proposed architecture. Theimplementation of blockchain in the proposed methodology results in low metadata access delay and thereby improves the execution time.

Keywords : Hadoop Distributed File System, Name Node, Data Node, Metadata, Blockchain.

INTRODUCTION

The data is growing in an exponential rate especially from the last decade. Big data is a huge pool of various kinds of data in various formats, which contains hidden potentials for better business decisions. The emergence of huge volume of data and its processing has evolved in 2000. Organizations are producing data at an astonishing rate. It is reported that Face book alone collects approximately 750 terabytes a day [1]. Hadoop is an open source framework overseen by Apache software foundations for the storage and processing of large data sets. Hadoop uses HDFS to store the large data across different commodity hardware and maintains a distributed file system. Hadoop stores data in a distributed and massively parallel computing environment [2, 3] and the files are scattered across the cluster. There should be an efficient file system to manage the files in Hadoop. The file system used in Hadoop is called Hadoop Distributed File System (HDFS). HDFS is configured to have a Name Node and a cluster of Data Nodes. This architecture is scalable and fault tolerant. HDFS is inevitable when terabytes or peta bytes of data need to be stored and processed due to the limitation of hard disk capacity of local machines. Hence the client data may be split into blocks and stored over a cluster of commodity hardware's storage rack. Since the data are distributed across different data nodes, the information regarding the storage and retrieval should be maintained in an organized way. The information regarding the distribution of data, block size, replica maintenance is kept in a file called metadata. The meta data is maintained in a specially designated, reliable hardware known as Name node. Each block of data maintains a replication factor, by default 3, as a backup mechanism. The data block size is expanded to 64MB (default) or 128 MB in size in latest versions of Map Reduce framework. A study is being conducted to explore the Hadoop ecosystem component, the storage architecture known as HDFS, the processing frameworks namely, Map Reduce and YARN and the versions of Hadoop. The first version of Hadoop was released in 2007 and Hadoop 1.2.1 documentation was done in 2013 and the second version of Hadoop 2.x series (beta) in 2014 and 2016 and the latest version alpha is released in 2017[4]. Big data processing involves storage and processing components. The various components are reviewed. Patil et.al [5] presented all the Hadoop components in detail. They have pointed out that the HDFS differs from the other distributed file system by the way of fault tolerance, by maintaining three replicas on each block. Ashish and Patil [6] focused on paradigm of Big Data technology, components of Hadoop, architecture of HDFS and replica management. Chandra et al. [7] focused on how the replicas are managed in HDFS and also focus on possible failure affect the Hadoop Cluster. Dhavapriya, and Yasodha [8] described the HDFS architecture and propose various methods for catering to the problems through Map Reduce

framework over HDFS. Gayathri Ravichandran [9] and Bijesh and Anurag [10] were describing the HDFS components and the processing framework and its various stages. Suresh and Ramlal [11] discussed various aspects of big data, Hadoop, its overall architecture, technical details. The usage of big data in social media, IoT and Online Marketing were highlighted and found the shortcomings. They concluded that the manipulations are necessary in storage and processing on the data. Varsha Bobade [12] described the various components of Hadoop ecosystem and various challenges of big data. Tanvi and, Handa [13] found that HDFS is mainly designed to store the large amount of data for extreme rapid computation on a cluster of Data Nodes and the files in HDFS are managed by a single server, the Name Node. Name Node stores metadata, in its main memory, for each file stored into HDFS. As a consequence, HDFS suffers performance degradation with increased number of small files. Storing and managing a large number of small files imposes a heavy burden on the Name Node. Chandrasekar et al [14] built an indexing mechanism to access the individual files from the corresponding combined file. Also, they proposed to remove the single point of failure and increasing the storage capacity of the architecture, by a special node called Avatar Node [14]. From the review, the storage component of Hadoop is HDFS. The components are cheap commodity hardware for actual data storage called Data Nodes and the whole data distribution , management and block creation, metadata creation are authorized and controlled by the single , expensive server known as Name Node. There are not many changes in storage architecture, whereas the processing architecture upgraded from Map Reduce / YARN and Spark stream processing. The workload increases for the Name Node when large number of smaller files is to be stored and the problem was resolved using the introduction of Avatar node, still there exist the problem of single point of failure with the single Name Node in the architecture. When huge amount of data is to be processed, elimination of Name Node is difficult. Here it is proposed to eliminate the Name Node from the architecture in small sized clusters.

RELATED WORKS

Kon Stantin Shvachko et. al. investigates the HDFS in detail and pointed out the merits and drawback of HDFS architecture[15] . HDFS, originated from GFS [16] and both store data in Data Nodes and meta data in a single node called Name node. The name node acts as a single point of failure as it goes down; the entire application data becomes unresponsive. Lustre file system, a High performance computing data parallel file system [17] designed for large scale data storage and metadata management. The metadata was distributed across a cluster of metadata servers in Lustre file system. Limitation was the size of the machine, due to which the Lustre clients were limited. When the metadata was distributed to a cluster of servers, load balancing becomes another major challenge in Federated HDFS [18]. It scales metadata, but individual metadata servers were not scalable. Next evolution was to fully distribute metadata file system in PVFS [19] and Ceph [20] with a dynamic distribution policy among the metadata servers. Another approach on shared metadata was through the shared disk space in GPFS [21]. CalvinFS [22] adopts a shared nothing policy in Local area networks and replicates the information on Wide Area Networks. Giraffa file system [23] was designed with the aim of dynamically distributing the metadata. IndexFS [24] was a distributed file system with scalable metadata and client cache mechanism for metadata whereas ShradFS [25] was designed for optimized concurrency control among multiple transactions.

The storage and processing were the two most important challenges in big data processing. The Hadoop ecosystem components provide both these functionalities. The processing framework of Hadoop is Map Reduce/ YARN. The storage architecture of Hadoop is HDFS. But for relatively smaller sized clusters of nodes, maintaining an expensive Name Node is not cost effective. Under such circumstances, it is proposed to distribute the functionalities of the central Name Node to the Data Nodes and eliminating Name Node from the HDFS architecture. The proposed architecture is named as simplified HDFS architecture (SHDFS). The most important function of Name Node is creating and recording metadata which includes the information regarding the location of blocks stored the size of the files, permissions and hierarchy. The other functions include maintaining and managing Data Nodes, receiving heartbeat and block report from Data Nodes.

In SHDFS, all these functionalities will be perfectly handled by the Data Nodes. The metadata creation and recording is also distributed across all the Data Nodes using the blockchain technology. The metadata is stored in encrypted form in blockchain. Blockchain technology is the underlying mechanism used in Bit coin, the digital crypto currency. Here it is used to keep the metadata information in a distributive manner.

Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) is designed to store huge volume of data sets for user applications. When it is required to store peta bytes or zeta byte under big data technologies, the HDFS architecture is efficient, with costly reliable hardware and a cluster of thousands of commodity hardware. This paper explained the architecture of HDFS and proposes a modified architecture for comparatively smaller sized clusters with terabytes of storage for private data.

Architectural overview of HDFS

The HDFS is for storing the data sets in any form- structured, semi structured and unstructured, required to be processed for various user application and also to make Business Intelligence decisions. HDFS is fulfilling the storage with clusters of commodity hardware and with streaming access patterns with write-once-read-many strategy. HDFS is specially designed for the storage, normally in hard disk as files and directories and the block size is enhanced from 4MB to 64MB/128MB, in order to reduce the size of metadata and with a proper reutilization of memory through dynamic reallocation. Yahoo originally released GFS (Google File System) and modified to HDFS.

The architectural components of HDFS are five. It includes

1) *Name Node:* HDFS cluster consists of a single Name Node, a master server that manages the metadata which regulates access to files by clients. The Name Node is responsible for keeping track of complete file system by two things: Name Space Image and EditLog. The Name space image contains the metadata about the files and directories and it contains data about all the blocks, to which data nodes they are associated and on which data node it resides. Editlog contains the log of activities on HDFS performed by the client. Editlog keeps on growing as the activities on HDFS performed by the client goes on. The Name space image and Editlog combines the complete file system image (FSImage) giving details of all the files and blocks on HDFS. The FSImage is kept in the main memory of the Name Node, for faster lookup. The block information is updated by the Name Node as and when the Data Node joins the network. As soon as the Data Node boots up and connected to the network, it would send Name Node, the information about the blocks it has and the Name Node updates information to Name Space Image. When a client requests for storage to the Name Node, it will create the metadata. The metadata consists of the original file name, block filenames, and file size for each block, address of the Data Node and the address of the Data Nodes on which the replicas are to be stored. The metadata once written cannot be altered, since it follows write-once-read-many architecture. The subsequent updates, if any, will be kept as separate metadata and all the transaction logs are kept in a file called Editlog. The existence of single Name Node in a cluster greatly simplifies the architecture of the system, but the chance of single point of failure is more. Hence the Name Node is created with expensive reliable hardware. The Name Node maintains the file system namespace. Any change to the file system name space or its properties is recorded by the Name Node. An application can specify the number of replicas of a file that should be maintained by HDFS. The number of copies of a file is called the replication factor of that file. When the client request to store huge data, the data is divided into blocks of 64MB and each block is distributed to different Data Nodes within the cluster and the metadata is stored in Name Node. HDFS defines a replica factor 3, by default, as backup which should also be maintained in Data Node and metadata information in Name Node. The Name Node makes all decisions regarding replication of blocks. The Data Nodes periodically send a Heartbeat to make the Name Node aware of its existence. The Data Nodes will also send a Block report to the Name Node after storing the data. Receipt of periodic Heartbeat implies that the Data Node is functioning properly. A Block report acknowledges the receipt of the data blocks by the Data Nodes. The Name Node keeps the repository for all HDFS metadata. The system is designed in such a way that user requests through the Name Node into HDFS storage.

   The metadata holds a major concern in storing and processing data in HDFS. The security of metadata is another concern in Hadoop system. The security is immature and rapidly evolving part of the Hadoop ecosystem.

2) *Secondary Name Node:* Secondary Name Node has not the same functionality, but the same specification as that of Name Node. As the Name Node suffers from single point of failure, in HDFS some resilient additions are to be performed to recover the data. For the same, it is recommended to transform the Name Space Image to some remote NFS mount device, from time to time. The secondary Name Node is used to combine the Name Space Image and the Edit log called checkpoints and writes it to a file. This helps the Name Node to release the main memory occupied by the edit log till the point of last checkpoint. In case of failure, the Hadoop administrator has to boot up the new Name Node and the last information from the remote NFS mount should be retrieved manually and configure the new node.

3) *Job tracker:* JobTracker process runs on a separate node and not usually on a Data Node, which is a part of MapReduce execution in MRv1. The next version replaces it by Resource Manager/Application Master. It receives the requests from the client and talks to the Name Node to determine the location of the data. JobTracker finds the best TaskTracker nodes to execute tasks based on the data locality to execute a task on a given node. JobTracker monitors the individual TaskTrackers and submits back the overall status of the job back to the client. JobTracker process is critical to the Hadoop cluster in terms of MapReduce execution. When the JobTracker is down, HDFS will still be functional but the MapReduce execution cannot be started and the existing MapReduce jobs will be halted.

4) *Data Node:* Data Nodes are cheaper commodity hardware

with storage and processing capability which are grouped across different clusters. HDFS exposes a file system namespace and allows user data to be stored in files. The input file is split into a number of files blocks (blocks of 64MB or 128MB) and the blocks are stored in a set of Data Nodes as per the metadata information from the Name Node. The Name Node executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to Data Nodes. The Data Nodes are responsible for serving read and write requests from the file system's clients. The Data Nodes also perform block creation, deletion, and replication upon instruction from the Name Node. When a Data Node starts up, it scans through its local file system, generates a list of all HDFS data blocks that correspond to each of these local files and sends this report to the Name Node, which is the Block report. Heartbeat is the signal sent by all the data nodes periodically to the Name Node for maintaining the connection.

5) *Task tracker:* TaskTracker runs on all Data Nodes. It is replaced by Node Manager in MRv2. The Mapper and Reducer tasks executed, as part of Map reduce programming, on Data Nodes are administered by TaskTrackers. TaskTrackers will be assigned Mapper and Reducer tasks to execute by JobTracker. TaskTracker will be in constant communication with the JobTracker signaling the progress of the task in execution.

The working principle of HDFS follows the master slave configuration. Master services include Name Node, secondary Name Node and job tracker and the slave services include Data Node and task tracker. Master node and slave nodes can communicate with each other. The slave nodes communicate with each other. The job tracker can communicate with the Task Tracker as depicted in figure 1.

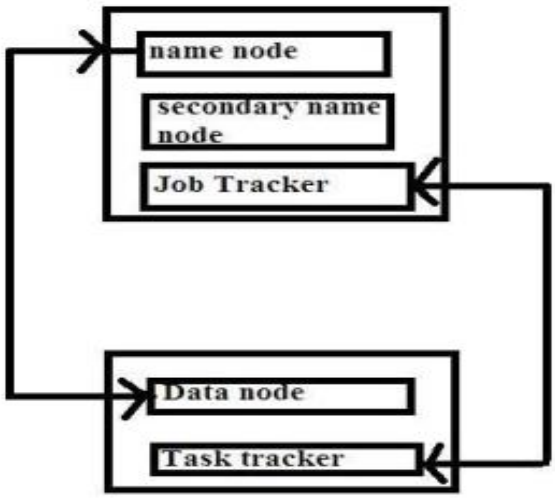

Figure 1: Components of HDFS architecture.

FEATURES OF HDFS ARCHITECTURE

The features of the present architecture are explained in detail. HDFS architecture is a distributive storage mechanism and it differs from other distributive systems with a better fault tolerant mechanism and an efficient replica management.

1) *Name node & Data Nodes Failure*: Hardware failure may occur in both the central Name Node and distributed Data Nodes. Data recovery till the last checkpoint, which is stored in a remote NFS mounted file, is handled by secondary Name Node. The unreachability of data due to the failure of data nodes will be compensated by maintaining the duplicate copies (By default, 3 copies) in other Data Nodes across the clusters.

2) *Write-once-read-many access:* For the coherence of data which is distributed across a cluster of nodes, HDFS applications need a write-once-read-many access model for files. A file once created, written, and closed need not be modified. The consistency of data is difficult to maintain in a distributed architecture. This assumption simplifies data coherency issues and easy data access.

3) *Moving Computation to Data:* A computation requested by an application is much more efficient if it is executed near the data it operates on. This is especially true when the size of the data set is huge. This minimizes network congestion and increases the overall throughput of the system. The assumption is that it is often better to migrate the code to where the data is located rather than moving the data to where the application is running. HDFS provides interfaces for applications to move themselves closer to where the data is located.

4) *Portability:* HDFS has been designed to be easily portable from one platform to another. This facilitates widespread adoption of HDFS as a platform of choice for a large set of applications. Shrading enables the expansion of servers within the cluster and this can be easily done because the HDFS architecture is designed for hardware and software portability.

The major drawback of the existing architecture is that the existence of central Name Node leads to single point failure. Another issue is that the entire workflow in HDFS architecture is based on the metadata, which is not secured enough. Basic authentication mechanism using KERBROS is implemented in the present architecture. It is possible that a block of data fetched from a Data Node arrives corrupted. This corruption can occur because of faults in a storage device, network faults, or buggy software. The HDFS client software implements checksum checking on the contents of HDFS files. When a client creates an HDFS file, it computes a checksum of each block of the file and stores these checksums in a separate hidden file in the same HDFS namespace. When a client retrieves the file contents it verifies that the data it received from each Data Node matches the checksum stored

in the associated checksum file. If not, then the client can opt to retrieve that block from another Data Node that has a replica of that block. But privacy and anonymity is not being addressed in the architecture.

BLOCKCHAIN FOR METADATA STORAGE

A blockchain is a distributed database of records or public ledger of transactions or digital assets which is executed and shared among the authorized set of users. The blockchain establishment is based on creating a distributed consensus which allows the peer to peer users on the network to confirm that a digital event or record had been created and it is verifiable and irrefutable. Another characteristic of blockchain technology is anonymity through which the technology hides the identity of the parties involved in the P2P network. Blockchain technology has specific applications in both financial and non financial sectors. Bit coin is the most popular example tied to blockchain technology. It uses cryptographic proof instead of trusting a third party for the execution of a transaction by two parties over the internet. Each transaction is protected by a signature scheme. The transactions are verified and broadcasted to every other node in the P2P network. Blockchain can eliminate the double spending problem by linking all the blocks which contains two hash values, current hash computed from the current block and previous hash from the previous block. The previous hashes links each block with one another and which makes the entire blockchain immutable.

1) *Proof-of-work***:** In order to deal with unconfirmed transactions, a mathematical mechanism known as proof-of-work is being introduced. In this method, each node generating the block in the network has to prove that it can put sufficient computing power to solve the mathematical puzzle. For the same, each node has to find a nonce value which when hashed with the data records and the previous hash attached in that block should produce a specific number of leading zeros. There are nodes which donates their computing resources to find the nonce for building the block. They are called as Miners and they will be paid for their efforts.

2) *Proof-of-Existence*: Current digital economy is based on trusting a single trusted authority. Then facts can be hacked, manipulated or compromised. In such situations, blockchain as a revolutionary change can muddle through, by implementing smart contracts, smart property etc. In such cases, validating the existence and possession of documents can be provided as proof-of-Existence which serves the online proof of existence of any document. For the same, blockchain has to store message digest of the file and the timestamp associated with each document, which allows anyone on the decentralized network can certify the existence of a document that existed at a particular time.

Metadata storage in cloud using blockchain

A blockchain is one of the technologies for the safe distribution of metadata over a distributed system. In cloud computing, there are various solutions such as Dropbox, Google drive and One Drive are being utilized to store data, video, audio etc. The cloud file storage faces security, privacy and data access challenges since the users have to trust up on the third party for storing their confidential files. In cloud platform, blockchain technology assisted project known as Storj which provides a blockchain for P2P distributed storage. As a result, due to the lack of central control, significant improvement in security and privacy achieved in cloud.

Replacing metadata from the Name Node in HDFS

The data which is stored in the blockchain database is not stored in any single location; instead it is being distributed on every computer. The contents of the blockchain are verifiable, but immutable. From the literature of HDFS architecture 5, 6 the existing architecture is build with an expensive master node called Name Node which acts as the central point of failure. One of the main functionality of the master node is to build and safeguard the metadata. The metadata is not secured though. Hence, in HDFS the blockchain technology can be adopted for the storage of meta data in a distributive manner in a P2P network of Data Nodes.

METHODOLOGY

Simplified HDFS (SHDFS) architecture was proposed. In this architecture, there is a collection of Data Nodes called HDFS_DNs which are organized as a cluster of commodity hardware. The Name Node is eliminated in this architecture, which makes the design simpler and cheaper. Now the metadata storage and management is an important concern in SHDFS design in the absence of the central authority-the Name Node. HDFS should have the capability to store, manage, and access file metadata reliably, and provide fast access to the metadata store. The storage of metadata is done by distributing it across all the Data Nodes by a master node called SHDFS_MINER (one among the Data Nodes) and is stored in an encrypted form using the SHDFS_Clients' public key. SHDFS_MINER can be assigned to any Data Nodes based on any arbitration mechanism in a master/slave configuration.

Blockchain is a technology used for linking the data blocks in different Data Nodes serially for easy access. The block diagram of blockchain is shown in figure 2. The ordering of data blocks are ensured using the Time stamping attached to the blocks. The data blocks are attached with two hash values (H1- previous data block hash; H2-current data block hash). For the first data block, H1 is from metadata and H2 computed from the current data block. Second block onwards,

H1 is the previous hash value received from the corresponding Data Nodes, and H2 is computed by the local Data Node itself. The different data blocks of the same file which are scattered over different Data Nodes are linked together using the hash (H1) of the previous block. The current hash value (H2) stored in each block which preserves data integrity. The distributed consensus is achieved in blockchain by finding the proper nonce value as a proof-of-work.

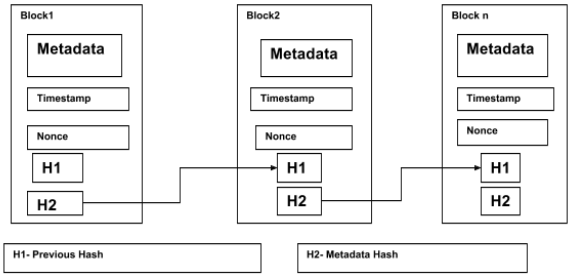

Figure 2: Blocks in the blockchain.

The components of SHDFS are

1) SHDFS_Client: who makes requests for the storage and performs integrity check after retrieval.

2) SHDFS_DN: Specialized Data Nodes whose primary objective is to store the blocks of data and a memory area reserved for storing the metadata blocks.

3) SHDFS_MINER: One or more specially designated nodes from the Data Nodes called as MINER nodes for running metadata creation along with data block storage.

An important property of this proposed architecture is to store metadata securely across the cluster of Data Nodes in an encrypted form. The details are listed in the activity table. In the SHDFS architecture, the concept of Name Node is not being used. Activity table which explains the SHDFS architecture as defined in table 1 and the block diagram of SHDFS Architecture is shown in figure 3.

Table 1: Activity table-Simplified HDFS

1.	SHDFS_Client requests the Admin to get the key pair for encrypting the corresponding metadata as well as data blocks, if needed.
2.	Using the method of Token passing or any other Arbitration mechanism, selects one SHDFS_MINER from the data nodes.
3.	After receiving the key pair, SHDFS_Client requests the SHDFS_MINER for data storage.
4.	When a client is writing data to an HDFS file, this data is first written to a SHDFS_MINER. Then SHDFS_MINER creates the metadata which contains the details of Data Nodes in which the data blocks and its replica blocks to be stored and the metadata will be stored in a temporary storage. The default block size and replication factor are specified by Hadoop configuration. An application can specify block size, the number of replicas, and the replication factor for a specific file at its creation time. Heartbeats, failure report and Block report. • All peer Data Nodes are sending a heartbeat signal mutually to identify the failure nodes within the cluster. If a particular node doesn't receive the heart beat within the specified time, peer node will sent a failure report to the MINER and the MINER will create the new metadata. • The Data Nodes send Block report after the successful storage of the data blocks to the MINER.
5.	When all the block reports are received in MINER, it will send the copy of metadata encrypted using the SHDFS_Client's public key to all the associated Data Nodes. • The replica management is done with the principle of Rack Awareness. The replica copies are limited to 3, with the property that, two replicas are placed on one rack and the third one is in a different rack.
6.	Blockchain construction: Each block of the blockchain is constructed by attaching two hash values (H1, H2) along with the data fragment (64MB/128MB). The first hash value (H1) stores the hash of the previous block and second hash (H2) stores the hash computed from the current data fragment. The first hash value (H1) is used to chain different data fragments in different SHDFS_DNs.
7.	SHDFS_Client can read the meta data from SHDFS_MINER stored in encrypted form, which is to be decrypted using its private key and reading data from the corresponding Data Node. After getting the first block of data, it is easy to access the subsequent blocks through the blockchain.
8.	Failure of SHDFS_MINER, Data Nodes: • If the MINER node fails, through arbitration mechanism, another node will act as SHDFS_MINER. • If the Data Node fails, it is noticed through the failure report to the SHDFS_MINER and it changes the metadata.

FEATURES OF SHDFS

Most important features of the proposed architecture is described below.

1) *No centralized official copy exists*: In SHDFS, as the central server being proposed to eliminate, there is no official copy exists, instead it is being distributed. Meta data are broadcast to the network using software and the messages are delivered on a best effort basis.

2) *Structure of Metadata***:** The FsImage and the EditLog are central data structures of HDFS. As in the existing architecture, there will be a metadata block creator which stores all the information of the file like number of blocks, block size, address of data nodes for storing and keeping replicas etc. Incremental updates are treated as separate metadata block.

3) *Data integrity*: Correctness of each data block can be verified through the two hash values (H1, H2) attached with each block in a blockchain. The SHDFS_Client has to compute the hash value for the data fragment read from the data nodes and compare with the second hash value(H2) attached in each block. If it matches, accepts the data block.

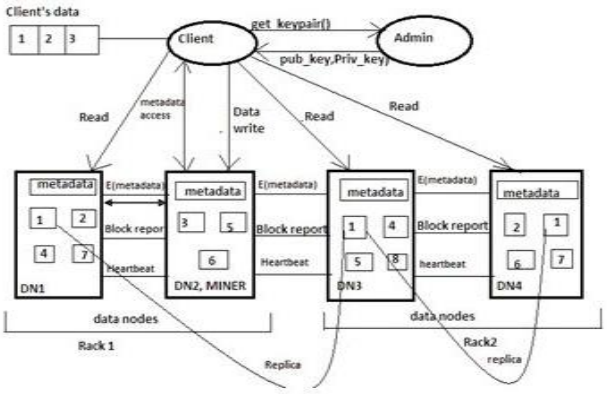

Figure 3: SHDFS architecture.

RESULTS & DISCUSSION

In the proposed architecture, the expensive and reliable hardware, Name Node is eliminated as pointed out in [18-25]. The functionalities of the Name Node are being distributed among the cheap hardware, the set of Data Nodes in the cluster. The prime responsibility of creating and maintaining metadata is implemented in the specialized Data Node, called SHDFS_Miner and the metadata information is kept in blockchain in a distributive manner. Blockchain implementation eliminates the possibility of single point of failure in the Name Node. Since the proposed architecture eliminates the Name Node and Secondary Name Node especially in small sized clusters, the architectural configuration will be comparatively cheaper.

```
Create_metadata (input_file)
        Blocks=input_file.size/blocksize
        Read the client_file
        For each block do
                Copy the block to Data_Node1 as per rules
                Copy block from Data_Node1  to Data_Node2.
                Copy block from Data_Node1  to Data_Node3.

Create metadata_block ()
Broadcast metadata_block to all the cluster nodes.
```

Figure 4: Algorithm for the creation of HDFS_Metadata.

The creation of metadata is defined in figure 4. Algorithm accepts the file size and if the file size > 64MB, then split the file into file partitions of block size 64MB, except for the last file block and assign names to the sub files.

The algorithm also implements Rack awareness for assigning Data nodes for replica maintenance, by default 3 copies.

Table 2: Sample metadata blocks in blockchain.

Original-filename	File-partition-name	Blocksize(MB)	Replica1(DN_No)	Replica2(DN_No)	Replica3(DN_No)
f.txt(200MB)	f1.txt	64	DN1	DN4	DN5
	f2.txt	64	DN2	DN6	DN7
	f3.txt	64	DN1	DN3	DN5
	f4.txt	8	DN4	DN6	DN7

A sample metadata is shown in Table 3. The block of information constitutes a new block in the existing blockchain in the P2P network.

```
def createblock(metadata)
        Compute the genesis block.
        Compute prevHash  and datahash (SHA-265)
        Compute Nonce.
        Recompute prevHash and datahash (data+nonce)
        While True:
                Header = prevHash + datahash + nonce
                Hash = SHA256 (header)
            Break
                Nonce += 1

        Header = prevHash + time + datahash + Nonce
        Block = header + data
```

Figure 5: Algorithm for maintaining the HDFS_metadata in blockchain.

The pseudo code of Blockchain implementation as shown in figure 5. The blocks are built by adding Timestamp, computed previous hash and data hash along with metadata and nonce. The blocks are linked using previous hash value in each block.

HDFS distributed storage is organized on a rack-to-rack basis in a cluster. The metadata distribution in blockchain as shown in figure 6. Instead of maintaining metadata in all the Data Nodes, it is being distributed in one Data Node per rack. Within the same rack, the metadata can be retrieved from a single Data Node without utilizing network bandwidth if that Data Node fails, the blocks can be retrieved from other Data Nodes in other racks. Such kind of implementation of blockchain on rack-by-rack basis in the proposed SHDFS reduces the number of metadata copies to be maintained in the blockchain.

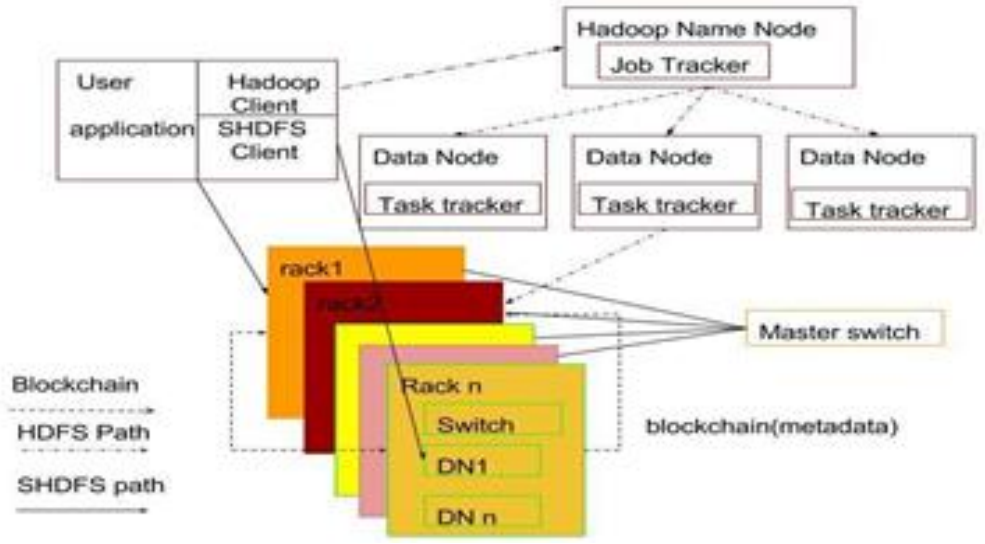

Figure 6: Distribution of SHDFS storage on rack-by-rack.

The proposed architecture completely eliminates single point of failure like Giraffa [25], IndexFS [23] and ShradFS [24]. SHDFS needs further research on scalability of metadata in blockchain as the cluster capacity was limited. Here application data and metadata were sharing the same HDFS cluster, so that scalability is being one of the challenges in large scale data processing, especially when large number of small files is to be maintained. Since the SHDFS architecture distributes the metadata across the cluster of nodes, the HDFS clients can directly the Data Nodes for processing and also job scheduler delay can be minimized.

CONCLUSION

The current HDFS architecture suffers from single point of failure and a new architecture was proposed for small sized cluster of nodes to alleviate the problem of single point of failure. Also the secondary name node is used to create checkpoints to the remote NFS mounted device. Both are maintained using expensive reliable hardware. In the new architecture both expensive hardware is being eliminated and hence it is a cost effective one. Implementation of Algorithm 1 has done the metadata creation along with replica management and which can be executed in any Data Node. The particular Data Node upon creating the metadata, it will broadcast to other Data Nodes. Other functionalities of the Name Node are compensated in the Activity Table.

Implementation of Algorithm 2 is for the creation of blocks for metadata distribution as blockchain. The blockchain stores copies of metadata on a rack-by-rack basis with one Data Node / rack and hence reduce the number of copies and cost of maintaining blockchain. With the implementation of blockchain technology, there is no point of recovery due to the existence of multiple copies of the metadata being distributed across one Data Node per rack. Hence the secondary name node is not required in the proposed SHDFS architecture.

ACKNOWLEDGEMENT

I would like to express my profound gratitude to Research Guide Dr. Prof. M. Abdul Rahman for his valuable suggestions and constant support.

REFERENCES

[1] Kon stantin Shvachko *et.al.* "The Hadoop Distributed File System, Yahoo! "Sunnyvale, California USA, 978-1-4244-7153-9, 2010.

[2] T. White, *Hadoop: The Definitive Guide*. O'Reilly Media, Yahoo! Press, June 5, 2009.

[3] http://hadoop.apache.org/releases.pdf.

[4] Poonam S. Patil *et.al.* "Survey Paper on Big Data Processing and Hadoop Components". *International Journal of Science and Research*, ISSN (Online): 2319-7064,3(10): 585-590,2014.

[5] Ashish A. Patokar *et.al.* "Analysis of Big Data by using Hadoop in Cloud Computing by Map Reducing", *National Conference on Innovative Trends in Science and Engineering (NC-ITSE'16)* ISSN: 2321-8169 Volume: 4(7): 378 – 381,2016.

[6] S. Chandra Mouliswaran *et.al.* "Study on replica management and high availability in hadoop distributed file system (HDFS)". *Journal of Science*, 2(2): 65-70, 2012.

[7] M. Dhavapriya *et.al.* "Big Data Analytics: Challenges

and Solutions Using Hadoop, Map Reduce and Big Table", *International Journal of Computer Science Trends and Technology (IJCST)* – 4(1): ISSN: 2347-8578, 2016.

[8] Gayathri Ravichandran, International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056 Volume: 04(2): 448-451, 2017.

[9] Bijesh Dhyani *et.al.* "Big Data Analytics using Hadoop", *International Journal of Computer Applications* (0975 – 8887), 108(12):1-5, 2014.

[10] Suresh Lakavath *et.al.* "A Big Data Hadoop Architecture for Online Analysis", *International Journal of Computer Science and Information Technology & Security (IJCSITS)*, ISSN: 2249-9555 4(6):149-153, 2014.

[11] Varsha B. Bobade, "Survey Paper on Big Data and Hadoop", *International Research Journal of Engineering and Technology (IRJET)* e-ISSN: 2395-0056 3(1):861-863, 2016.

[12] Tanvi Gupta *et.al.* "An Extended HDFS with an AVATAR NODE to handle both small files and to eliminate single point of failure". *International Conference on Soft Computing Techniques and Implementations- (ICSCTI)* Department of ECE, FET, MRIU, Faridabad, India, Oct 8-10, INSPEC Accession Number: 16072425,2015 .

[13] S. Chandrasekar *et.al.* "A novel indexing scheme for efficient handling of small files in Hadoop Distributed File System". *International Conference on Computer Communication and Informatics*, INSPEC accession no: 13357231, 2013.

[14] HDFS. Hadoop distributed file system. http://hadoop.apache.org/.

[15] S. Ghemawat *et.al.* "The Google file system", *In Proceedings of the 19th ACM symposium on operating systems (SOSP)*, 2003.

[16] Lustre. *Lustre file system*. http://www.lustre.org/.

[17] B. Welch *et.al.* "Scalable performance of the panasas parallel file system", *In Proceedings of the 6th USENIX conference on file and storage technologies (FAST)*, 2008.

[18] P.H. Carns *et.al.* "Pvfs: A parallel file system for linux clusters", *In Proceedings of the 4th annual Linux showcase and conference*, pages 391–430, 2000.

[19] S. A. Weil *et. al.* "Ceph: A Scalable, High-Performance Distributed File System", *In Proceedings of the 7th symposium on operating systems design and implementation (OSDI)*, 2006.

[20] F. Schmuck *et. al.,* "GPFS: A Shared-Disk File System for Large Computing Clusters", *In Proceedings of the 1st USENIX conference on file and storage technologies (FAST)*, 2002.

[21] A. Thomson *et.al.* "CalvinFS: consistent wan replication and scalable metadata management for distributed file systems", *In Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST).*2015.

[22] Konstantin V Shvachko *et.al.* "Scaling Namespace Operations with Giraffa File System", *SUMMER 2017* VOL. 42, NO. 2, 27-30, 2017.

[23] Lin Xiao *et.al,* "ShardFS vs. IndexFS: Replication vs. Caching Strategies for Distributed Metadata Management in Cloud Storage Systems", SoCC '15, August 27-29, USA.*ACM* 978-1-4503-3651-2/15/08.http://dx.doi.org/10.1145/2806777.2806844, 2015.

[24] ShardFS. Shardfs code release. http://www.pdl.cmu.edu/ ShardFS/ index.shtml.