International Journal of Communication Networks and Information Security (IJCNIS) Vol. 11, No. 3, December 2019

Exploiting Blockchains to improve Data Upload and Storage in the Cloud

Yassine El Khanboubi[1], Mostafa Hanoune[2]

[1,2]Faculty of Science Ben M'Sik, Hassan II University,Casablanca, Morocco

Abstract: Cloud computing is an information technology that enables different users to access a shared pool of configurable system resources and different services without physically acquiring them. So, it saves managing cost and time for organizations and can be rapidly provisioned with minimal management effort. Most industries nowadays such as banking, healthcare and education are migrating to the cloud due to its efficiency of services especially when it comes to data security and integrity. Cloud platforms encounter numerous challenges such as Data de-duplication, Data Transmission, Data Integrity, Virtual Machine Security, Data Availability, Bandwidth usage… etc. In this paper, we have adopted the Blockchain technology - which is a relatively new technology - that emerged for the first time as the cryptocurrency Bitcoin and proved its efficiency in securing data and assuring data integrity. It is mostly a distributed public ledger that holds transactions data in case of Bit coin. In our work, Blockchains are adopted in a different way than its regular use in Bitcoin. Three of the major challenges in Cloud Computing and Cloud services are Data De-duplication, Storage and Bandwidth usage are discussed in this paper.

Keywords: Deduplication, Blockchain, Chunking, Cloud Computing, Bandwidth, Execution Time, Genesis Block, Merkle Hash Tree, Storage space.

1. Introduction

Since the beginning of the IT revolution, resources management, security, data integrity and storage were (and still) the main challenges and focus of researchers and engineers. Cloud Computing was the pinnacle of years of researches and efforts of hundreds of researchers and engineers until it became the target of most regular users as well as all size of companies.
The National Institute of Standards and Technology (NIST) had classified Cloud Computing into four major patterns described as follow [1]:

- **Public Cloud:** All cloud services available to the public through the Internet. The resource offered in this type might be free of charge (such as Google Drive, Dropbox and so) or the services are offered to customers according to their needs and customization.
- **Private Cloud:** Is similar to Public Cloud and delivers similar benefits but through proprietary architecture, private cloud resources are dedicated to fill the needs and goals of a single organization.
- **Hybrid Cloud:** This type of cloud combines the advantages of public and private cloud services.
- **Community Cloud:** this Cloud infrastructure is a collaborative effort controlled, used and shared between several organizations with common concerns (security, compliance, policies, jurisdiction etc.).

Cloud Service Providers have to deal with a huge amount of data in terms of storage, on-demand access, performance, security and data integrity, which is a heavy task and a challenge that all CSP confront. According to statista.com data storage demand in the cloud has enormously increased in the few past years and continues to grow while the supply is struggling to catch up. The data storage supply and demand worldwide from 2009 to 2020 is presented in figure.1 below:

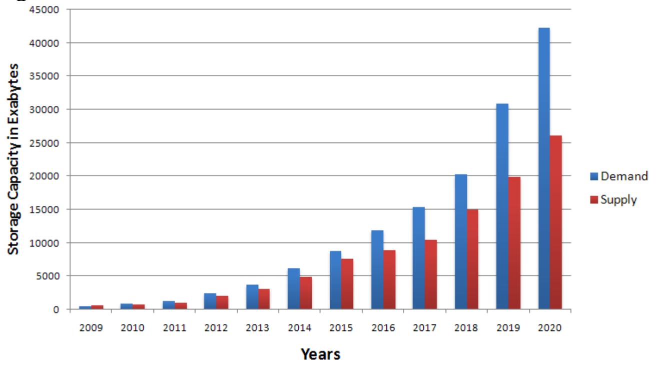

Figure 1. Data storage supply and demand worldwide from 2009 to 2020

Through time, many new concepts and researches helped improve cloud services in different areas (compression, data de-duplication, load balancing, etc.), a lot of work has been and still need to be done in order to reach the desired results in terms of security, strength and efficiency that every cloud user is looking forward to, especially with the continuous growth of cloud services and user needs.
According to RightScale.com, a survey conducted among 997 IT professionals, AWS (Amazon Web Service) remains the leader among the major cloud infrastructure providers. However, other cloud giants such as Google Cloud and IBM Cloud are quickly catching up. The survey is illustrated in the chart below:

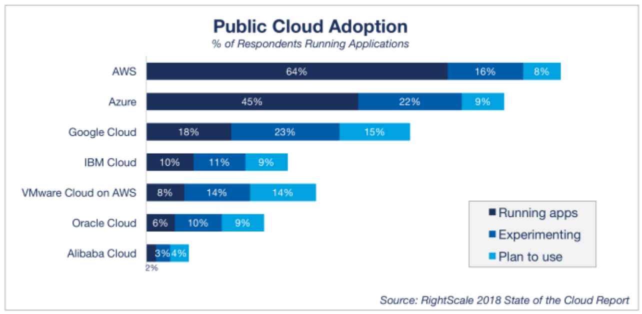

Figure 2. Public cloud adoption in 2018

Blockchain is a new concept that has been implemented for the first time in the cryptocurrency known as Bitcoin, and the concept seemed to be beneficial for cloud services.
The main idea about Blockchains is that they are only used in cryptocurrencies such as Bitcoin or Ethereum which is wrong as it is used in many different areas such as SmartContract, ID Systems …

Blockchains are mostly a public ledger or a distributed database that holds every event or transaction and then distributed to all participants, once received, most (or all) participants can then verify the transaction and then if it's valid, they add it to their local ledger. The data added to the Blockchain (called blocks) can never be erased or tempered with, and the data integrity is verified on the client side.

According to statista.com, the number of Blockchain wallets has been growing since the creation of the Bitcoin virtual currency in 2009, at the end of March 2019; Blockchain wallet users reached nearly 35 million. Blockchain is the technology that made Bitcoin popular and trusted by many major financial actors and it is a critical part of most cryptocurrencies. A "block" in this case refers to a record of the owners of all bitcoins, as well as the previous owners. Since this "chain" of ownership is distributed globally, it is extremely difficult to alter the ownership records.

A "wallet" in the digital sense defines a layer of security known as "tokenization". The user stores his private data with the wallet provider in a digital wallet. This provider sends payment information (in case of cryptocurrencies) to vendors in the shaped as a token. This token, instead of containing the personal information of the account owner, only holds enough information for the wallet provider to associate the transaction with the correct account, usually an account number.

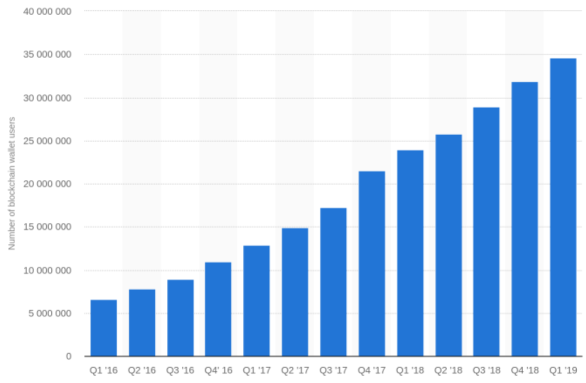

Figure3. Growth of Blockchain wallets from the first quarter of 2016 until the 1st quarter of 2019

In our approach, we have implemented Blockchains beyond the concept of its implementation in Bitcoin and other cryptocurrencies, with other concepts introduced in different papers such as Content-Aware Data chunking and Content Naming that we will be using but not discussing.

2. Related Works

S. Ghoshal and G. Paul [2] introduced in their paper a new model based on Blockchains similar to the approach described in our work but the concepts are very different, they proposed to split the original file into fixed-length file-blocks to generate the Merkle tree, in their work, they used block-chains to verify the integrity of a file, the described model doesn't manage the de-duplication files or upload, it only focuses on auditing and data integrity.

In [3], Andrea Margheri, and Vladimiro Sassone proposed to use a Third-Party Auditor, which operates as a gateway between the user and the CSP. The TPA's main role is to verify the file's integrity and decide whether to upload it or not, also, it generates the new blocks that will be added to the Blockchains. In comparison to our proposed work, this model has several drawbacks:

- The upload process is executed in 2 steps, the file is uploaded to the TPA and then the TPA uploads it to the CSP.
- If the TPA is down or out of service, the whole process is down.
- The TPA must be a trusted third party, if not; the data might be stolen or tempered with.
- The use of one Blockchain for all users and all up-loaded files.
- The model is not very beneficial in term of storage space since it does not manage the de-duplication on the server side.

3. Prerequisites

In this paper, we have adopted several concepts and models proposed in various papers and researches, and also technologies that were implemented to ensure our proposed model efficiency, so before getting into the discussion about our proposed model, here are the main concepts used in it.

3.1 Blockchain

3.1.1 Definition

Blockchain is a relatively new technology that has been introduced by Satoshi Nakamoto in 2008 [4] proved its efficiency in ensuring data integrity and security; it is mostly a public ledger or a distributed database that holds every event or transaction that is distributed to all participants, Blockchains can store any type of data, in our case we will be storing only references of file chunks and not the chunk itself, files will be stored in the file system, integrity will be verified before adding a block to the Blockchain, and re-verified during file read or update.

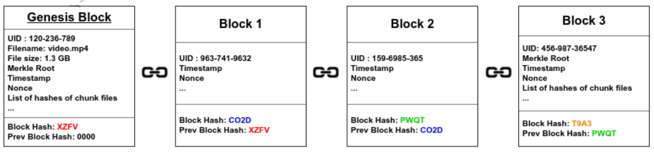

Figure4. Structure of a Blockchain used in our proposal

Each block in the Blockchain must have a reference to its precedent; this reference is the hash of the previous block [5].

The first block in a Blockchain is called Genesis Block (block **#0**) which has no previous block hash, this block can be very useful in our case, it will be used as a reference of all transaction, new owners, updates (which will be discussed in a future paper) … of a unique file on the server's file system.

Blockchain could be considered as a form of distributed databases. However, it is different from existing regular distributed databases (such as HBase, MondoDB…) by two main features:

1) <u>Cryptography</u>

Authenticity, user identity and the ledger integrity are achieved using encryption.

2) <u>Decentralization</u>

Unlike regular databases and ledgers, the Blockchain security and functionalities enforced in a public and distributed way instead of relying on a centralized server or a central authority.

3.1.2 Proof of Work

The proof of Work concept has emerged in 1993, when Cynthia Dward and Moni Naor published a paper where they

introduced a new method that tries to prevent spam emails.
A proof of work is a consensus algorithm in which it is costly and time-consuming to produce a piece of data, but it is easy for others to verify that the data is correct. Bitcoin is using a proof of work system called “Hashcash”.
For a block to be accepted by the network, miners have to complete a proof of work that comes in the form of an answer to a mathematical problem to verify all transactions in the block which in order to solve this problem nodes must run a long and random process. The difficulty of this work is not always the same, it keeps adjusting so new blocks can be generated every 10 minutes. There’s a very low probability of successful generation, so it is unpredictable which worker in the network will produce the next block.

3.2 Cloud service models

There are usually three models of cloud service to compare: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). Each model of these three has its benefits according to your needs, as well as variances, before choosing a model, it is necessary to understand the differences between PaaS, Saas and Iaas:

- **PaaS** (Platform as a Service) delivers a computing platform where the client can create, execute, deploy and manage their applications.
- **SaaS** (Software as a Service) is a model of software deployment where the software/applications are provided to the customers as a service through a program interface or a web browser. The Cloud's client does not need to install IT infrastructure such as network, servers, operating systems and application software inside his company because they are hosted and managed in supplier’s site.
- **IaaS** (Infrastructure as a Service) delivers computer infrastructure, typically a platform virtualization environment as a service. Rather than purchasing servers, software, storage, memory, processor or network equipment, clients buy those resources as fully outsourced services.

3.3 Merkle Tree

In cryptography, a Merkle Tree - also called Binary Hash Tree - [6]. It is a type of data structure that is used to verify the integrity of a large amount of data. Technically speaking, the Merkle tree is a binary tree where leaves are data blocks hashes, and every non-leaf node is the hash of the concatenation of two child blocks (using more than 2 child nodes is also possible but the majority of Merkle tree implementations are binary), the process is repeated several times according to the number of starting leaves until it gets a unique hash called MerkleRoot. Consider a file split into 4 parts P1, P2, P3 and P4.The hashes are computed as follow: H1 = h(P1), H2 = h(P2), H3 = h(P3), H4= h(P4), H5 = h(H1 + H2), H6 = h(H3 + H4), the Merkle root: H7 = h(H5 + H6),where h() is a double-hash function defined as h(x) = MD5(MD5(x)) and “+” means the concatenation hashes strings, the process will generate a hash tree of 3 levels.
The main advantage of the Merkle tree is that the slightest modification in one or more nodes will result in a whole new tree and the Merkle root will change completely, which will be very advantageous in verifying data integrity. A Merkle tree needs an even number of nodes in every level according to its depth in order to generate the hashes of a higher level in the tree, if an odd number of nodes is found while generating the tree, the last node is duplicated.
The tree depth is not (and can never be found) indicated by the MerkleRoot, this way a "second-preimage attack"[13] (where an attacker can create a file different than the original one that has the same Merkle hash root) can never be executed.
In our case, a file will be chunked into small parts and then the Merkle tree will be generated based on the resulted chunks (the number of levels in the Merkle tree depends on the initial number of chunks generated by the Content-Aware chunking algorithm) as shown in the figure bellow.
Any Hashing algorithm is applicable, we have decided to use MD5 algorithm.

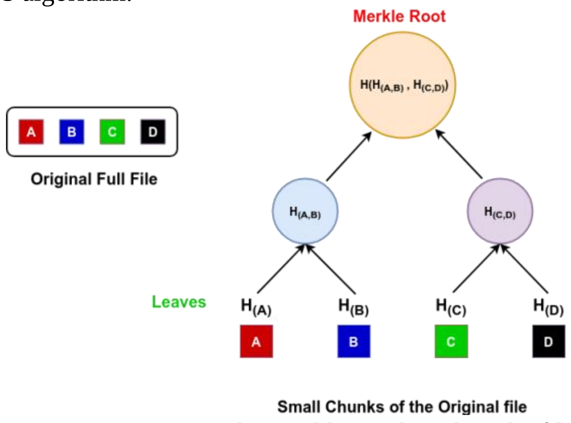

Figure5. Generation the Merkle tree based on the file chunks

3.4 Content-Aware Chunking

In our study, chunking is a crucial step to achieve data de-duplication and also to lighten the outgoing network traffic during file upload, as described in various papers, chunking is one of the main challenges in the de-duplication system and can be achieved using different methods[8]:
- File-level chunking.
- Fixed-size chunking.
- Robin Chunking.
- Content-aware chunking.
- Two Threshold Two Denominators (TTTD).
- Bimodal chunking.
- …

In the proposed model, a file chunk can be part of one or many files which will reduce the storage space needed; in this case a simple binary chunking (also known as Fixed-Size chunking that splits a file into equally sized chunks) will reduce the probability of having two files that share the same set of data. The chunks will be just a sequence of bits and boundaries are based on offsets like 4, 8 or 16 kB, which reduce the probability of using the same chunk in other files. Instead, using a Content-aware chunking algorithm will split the file based on its content, which will considerably improve the chunk re-usage probability.

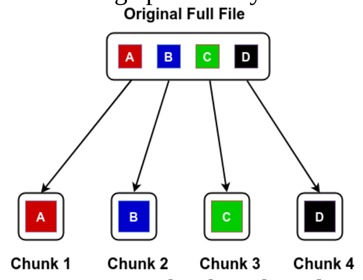

Figure 6. Content-Aware chunking based on Char type

3.5 Content Addressing (Content Naming)

Filenames are used to access the content of a file that is stored on the file system of a storage server, on a shared platform, several files (with different content) can be uploaded with the same name, in this case, the CSP must provide a way to generate a unique identifier of each file, hashing algorithms generates a unique string based on the file content, a different file will generate a unique hash, naming the files based on their hashes is called Content Naming which will make the file access more fast and no indexing is required, we have implemented this approach in our study to reduce file access time and computational power needed to retrieve the file.

4. Description of the Proposed Model

In this section we will describe our proposed model in which all the listed prerequisites described earlier were implemented as follow:

- Every user must be authenticated before submitting a file, we have chosen to implement a public/private key authentication to improve data security, password authentication is less secure and passwords can be bruteforced.
- Every file selected for upload in the client side is split using a Content-Aware chunking algorithm[9], which will generate a set of smaller files (chunks) of different sizes according to their content, the Cloud Service Provider defines a file size threshold based on the platforms performance, if a chunk oversize the defined threshold, it will be again chunked based on its content into smaller chunks, the generated files are named after their hashes which will enables an easy, fast and efficient access to the file in the file system (knowing that we only store references of the chunks in the Blockchain and not the whole file chunks).
- The client generates a Merkle Tree based on the hashes of all the chunks generated (hash of a chunk is called LEAF in the Merkle Tree terminology), in our case we have decided to use md5 hashing algorithm that generates a unique 32 chars string, our choice was based on the fact that MD5 algorithm is known to be much faster than SHAx algorithms (42% faster than SHA1 according to a test we performed on an Intel Core I5, 6th generation 2.53 GHz CPU), and the hash string length produced is smaller than any other hashing algorithm (32 chars for MD5, 40 chars for SHA1, 64 chars for SHA-256 and 128 chars for SHA-512).Also, even though MD5 is known to be broken in terms of pre-image resistance, it is still be very hard to perform such attack according to "Yu Sasaki and Kazumaro Aoki"[10], Other hashing algorithm may be applicable.
- Only the Merkle Root is sent to the CSP (which is basically a 32 chars string) as well as the user's unique ID (16 Bytes for a standard UUID), in the server side we only store a unique copy of each file (a Blockchain is generated for each unique file), we can determine if a file is already stored on the server by looking up his Merkle root knowing that the MerkleRoot is stored only in the first block of the Blockchain (Genesis Block) which will reduce the efforts needed to find it since we don't have to search the whole Blockchain.

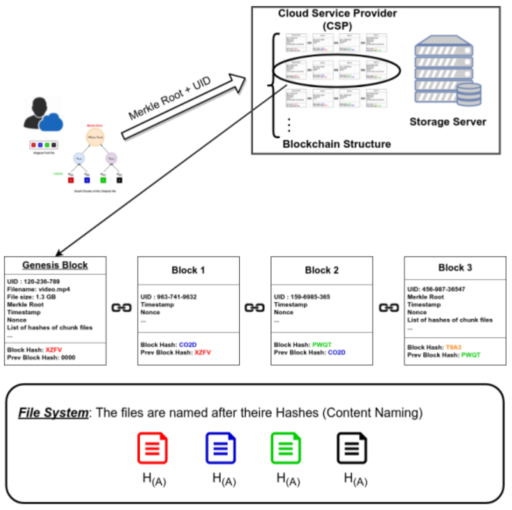

Figure 7. Server/Client communication and concept used in our model

A reel experiment of the proposed approach is graphically demonstrated in Figure 8 and Figure 9 below:

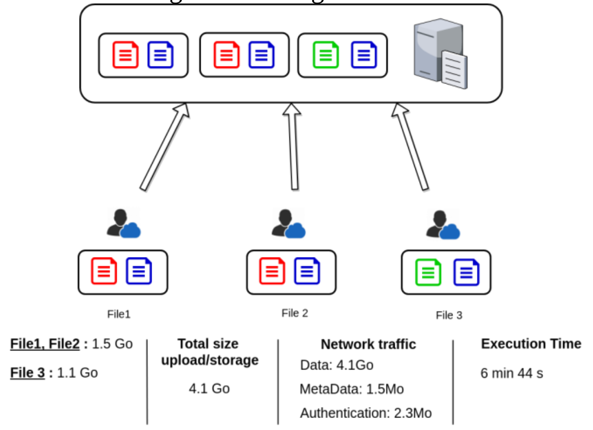

Figure8. Multi-file multi-user regular upload

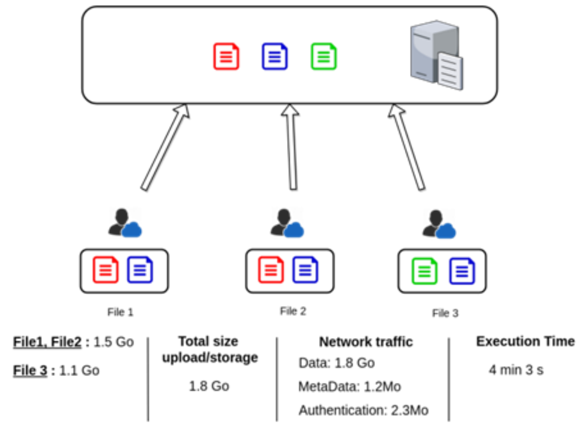

Figure9. Multi-file, Multi-user uploads according to the proposed model

4.1 Main Use Cases

Case 1: If the Merkle Root was found, then we verify ownership using the UID

a. If the user already owns the file, the user is then notified by his ownership and no further action is required, the file is not uploaded.

b. If the user does not own the file, a new block is added to the Blockchain containing the metadata of the new ownership:

1. UID: the new owner's ID which is a 16 Bytes long Unique User ID (UUID)

2. Timestamp: time of ownership in UNIX format.
3. Filename: the filename can be different of the original name.
4. Metadata: the user can add any metadata about the file in this field.
5. Nonce: an arbitrary and unique number that can only be used once.
6. The Previous Block hash: in Blockchains a new added block must have the previous block hash.
7. The MerkleRoot: each new block must have the MerkleRoot of the original file.
8. The Current Block Hash: after filling all fields in the new block, the block itself is hashed and added to the block, which will ensure the block's integrity.

At the end of the process, the new block is sent to all the users owning the same file (as described in the Blockchain documentation), this allows decentralized integrity verification; each user verifies the integrity of the file.

Case 2: If the Merkle Root was not found, it means that the original file was never uploaded, but the original file may share some of its parts with one or several files that were already uploaded. So, to reduce the storage space needed, reduce the bandwidth consumption and deal with de-duplication the server request the leaves of the Merkle tree from the client which sends the list of leaves hashes (32 x the number of leaves Bytes).

After receiving the leaves hashes, the server side application looks for each leave on the file system (chunks are named after their hashes as described in the "Content Naming" section) and generates a list of missing chunks hashes, the hashes of the existing chunks are regenerated and then compared to the leaves hashes to make sure that the file has not been tempered with, the list of the missing chunks is then sent to the client to be uploaded.

After uploading all the missing chunks, a new Blockchain is created; this newly generated Blockchain starts with a special type of blocks called The Genesis Block which holds all the data about the new file as follow:

1. UID: owners User unique ID which is a 16 Bytes long Unique User ID (UUID)
2. Filename: the name of the newly uploaded file
3. Filesize: the exact file size in Bytes of the original file, the sum of all file chunks sizes must match the original file size.
4. Timestamp: the Unix timestamp of the file upload
5. Nonce: an arbitrary and unique number that can only be used once.
6. MerkleRoot: the Merkle Root of the file generated by the client.
7. RefList: a list of all the chunks hashes that are part of this file, this field will be used to access and rebuild the original file.
8. Blockhash, after filling all fields in the new block, the block itself is hashed and added to the block, which will ensure the block's integrity.
9. PreviousBlockHash: the Genesis Block is the first block in the Blockchain therefore it has no previous block; this field will hold an all zero hash (00000000000000) as described in the BitcoinBlockchain. For every other block in the chain, it must contain the hash of the previous block.

The workflow above is described in Figure 10:

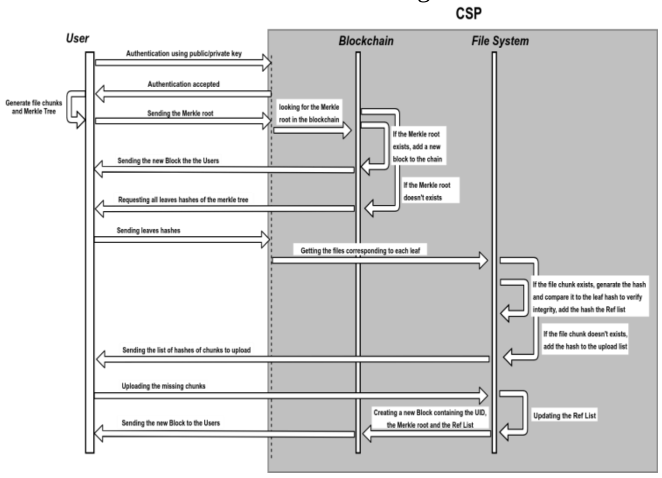
CSP
User
Blockchain
File System
Authentication using public/private key
Authentication accepted
Generate file chunks and Merkle Tree
Sending the Merkle root
looking for the Merkle root in the blockchain
If the Merkle root exists, add a new block to the chain
Sending the new Block the the Users
If the Merkle root doesn't exists
Requesting all leaves hashes of the merkle tree
Sending leaves hashes
Getting the files corresponding to each leaf
If the file chunk exists, genarate the hash and compare it to the leaf hash to verify integrity, add the hash the Ref list
If the file chunk doesn't exists, add the hash to the upload list
Sending the list of hashes of chunks to upload
Uploading the missing chunks
Updating the Ref List
Creating a new Block containing the UID, the Merkle root and the Ref List
Sending the new Block to the Users

Figure 10. Workflow of the proposed model

5. Implementation and Algorithms

The described model has been implemented and proven its efficiency, using the following application architecture:

Frontend Application: it's a JavaSE 8 implementation on the client side which has the following functionalities:

- Authentication using a public/private Key
- Choose, split and generate the Merkle tree leaves out of the chosen file.

```
emptyTempDir();
GenChunks();
For(i=0; i<listOfChunks.length; i++){
   Checksum = getFileCheksum(listOfChunks[i]);
   renameChunk(listOfChunks[i].getName(),
Cheksum);
}
```

Empty the temporary directory, generating chunks and renaming chunks by their hashes

- *Generating the Merkle Tree algorithm*

```
intcpt = 1;
while (leaves.size() > 1) {
   leaves = lvlhash(leaves);
   if (leaves.size() == 1) {
      tree.put("root", leaves);
   }else{
      tree.put("Level "+cpt,
      newArrayList<String>( leaves));
   }
   cpt++;
}
```

Generating the Merkle Tree on the client side

- The upload process

```
response = sendMerkleRoot();
If(response == exists){
   confirmUpload();
   saveNewBlock();
}else if(response == 'yours'){
   notifyUserOwnership();
}else of(response == 'empty'){
   Response = sendMerkleLeaves();
   If(response == 'empty'){
      confirmUpload();
      saveNewBlock();
   }else{
      List parts = getMissingParts();
      Foreach(parts as part){
         Upload(part);
      }
      confirmUpload();
      saveNewBlock();
        }
}
```

Perform an action based on whether a file exists on the server or not, and file ownership.

Backend Application: it's a PHP7/MySQL application where Blockchains are implemented as tables, which runs on the server side to perform the following tasks:

- Getting the Merkle Root from the client and checking if it exists on the Blockchains.
- Deciding whether to upload the whole file, parts of the file or nothing at all, the decision will be made based on the existence of the file (or parts of the file) on the server.
- Deciding whether to create a new Blockchain or just generate a new block and add it to an existing Blockchain.

```
Genesis_block = FindGenesisBlockByMerkleRoot();
If(Genesis_block != empty){
   If(Genesis_block.UID = PostedUID){
      callback = "yours";
   }else{
      addNewSimpleBlockToBlockchain();
      callback = "exists";}
}else{
   leaveHashes = getLeaves();
   List missingChunksList = findMissingChunks();
   sendMissingList();
   foreach(uploadedChunks as chunk){
   storeUploadedChunk();
}
   If(checkExistingChunksIntegrity()){
      updateRefList();
      createGenesisBlock(PostedUID);

        }else{
      Message = "Integrity Violation";}
}
```

Checking the existence of a file completely or partially, checking ownership, requesting upload and updating the Blockchain.

The algorithms above has been summarized and simplified for this paper.

6. Execution and Results

The implementation of the algorithms above has been tested on the following setup:

The client side application:

- The application runs on a Laptop computer with an Intel Core I5, 6th generation 2.53 GHz frequency CPU, 8 GBytes of RAM and 120 GBytes SSD Hard Drive.
- Ubuntu 18.04 OS and JDK 8 as the software environment.

The server side application:

- The application runs on a Desktop computer with an Intel Core I3 6th generation 3.70 GHz frequency, 4 GBytes of RAM and 500 GBytes SATA Hard Drive.
- Ubuntu 18.04 OS and JDK 8 as the software environment.

The client and the server communicate through a 100Mbps Ethernet network.

1- Use cases

- Regular upload (without the usage of the de-scribed model) of an mp4 video file of 1.1GBytes of size.
- First test: a totally new mp4 video file of 1.1GByte of size.
- Second test: an existing mp4 video file of 1.1GBytes owned by the same user.
- Third test: a partially existing mp4 video file of 1.1GBytes that shares 300 MBytes of chunks with other users.
- Last test: an existing file of 1.1GBytes owned by another user

2- ***Results*** (Splitting, hashing and renaming chunks are recalculated in each test)

The tests focus on three performance results, Bandwidth usage, storage space usage and execution time as described in **Table.1**:

Table 1. Results after performing 4 different tests on the proposed platform

Tests	Bandwidth (B)	Storage (MB)	Exec. Time(S)
1st Test	1181116054,4	1126.4	134,847
2nd Test	48	0	31,967
3rd Test	866610166,4	826.4	54,132
Last Test	48	0	22,053

The results are graphically represented bellow:

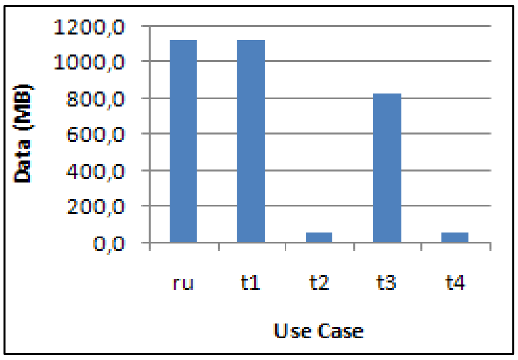

Figure 11. Comparison of data transferred over the network for different cases (***ru***: regular upload)

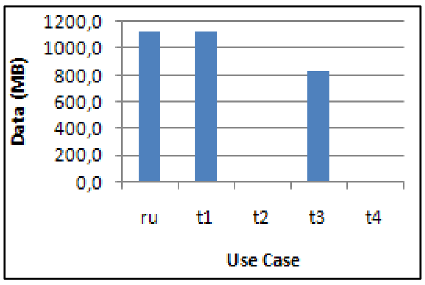

Figure 12. Comparison of the amount of data stored on the cloud (***ru***: regular upload)

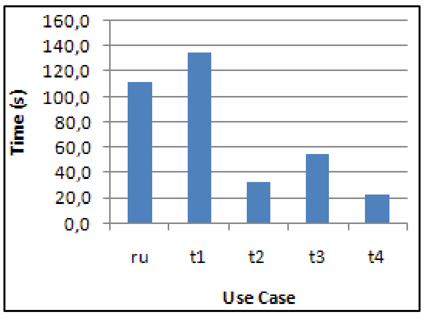

Figure 13. Comparison of the time needed for each case to execute the upload process (***ru***: regular upload)

After the analysis of the results obtained by executing four use cases and a regular upload we deduce the following:

1. In terms of bandwidth usage, the regular upload and the first test (a whole new file) uploads the same amount of data, but for all the other tests the amount of data upload is considerably reduced or not uploaded at all.
2. In terms of storage, the regular upload and the first test store the same amount of data, this is due to the inexistence of the original file (or parts of it) on the server, but for the rest of the tests the storage space needed is considerably reduced (or not stored at all).
3. In term of execution time, we notice that the first test takes more time than the regular upload, this is due to the chunking, hashing, renaming and generating the Merkle Tree, the time needed for the process to complete depends on the file size and the chosen hashing algorithm.

7. Conclusion and Future Work

At this stage, we have demonstrated the feasibility of using Blockchain technology to reduce bandwidth usage during a large file upload, storage space after upload, and a huge improvement of execution time, the numerical results has proven the efficiency of the proposed model and it can be more beneficial for big cloud platforms, the main improvements can be summarized as follow:

Bandwidth and Storage space usage: If a file already exists on the server, we only need to send the userID (16 Bytes for a standard UUID) and the Merkle root (32 Bytes).

If one or more chunks of a file already exist on the server, we only upload the non existing chunks

Execution Time: If a file is neither totally nor partially exists on the server, the time needed for the process to be completed is slightly greater than the time needed for a regular upload process; this is due to the chunking, hashing, renaming and generating the Merkle tree processes. In case of a file that is partially uploaded (one or more parts of the original file are already stored on the server), the time needed for the process to complete is considerably reduced compared to regular upload, the more parts exists on the server the less the upload time is needed.

In case of a file that was fully uploaded to the server before, the time needed for the whole process to complete is only the time needed on the client side to generate the Merkle Tree, for big files, we consider the duration of the process as negligible.

De-duplication: We only store 1 copy of each file (or chunk of file); multiple upload of the same file is managed by the ownership chain in the Blockchain. A chunk can be reused in many files based in its content, the probability of having chunks of files with the same content is considerably higher using a Content-Aware chunking algorithm instead of a Fixed-Size chunking algorithm (which is mostly binary).

Security and Integrity: The process starts with an authentication system based on Public/Private key, the whole network communication is encrypted.

The Blockchain and the Merkle Tree make sure that a file has not been tempered with and the integrity is then verified by all the owners the same file, this is due to the fact that the Blockchain is distributed to all users and every new added block is also distributed. The Blockchain does not allow deletion or modification of a block, in our case, a block references an ownership relationship between a user and a file.

Despite of all the issues improved by the proposed model, other issues and weaknesses emerged that need to be taken in consideration in order to achieve the desired efficient results of this work, some of the issues that emerged are described bellows:

File update and deletion: most of Cloud Service Providers allow users to delete update or change a file's metadata, in our case, a file (or parts of a file) can be owned by multiple users and the Blockchain doesn't allow any update or deletion once a block is added or created.

Computational Power: Chunking, Hashing, Renaming and generating the Merkle root of a file uses a lot more computational power than a regular upload process, especially in the client side which generally has limited hardware resources.

The described issues will be discussed in a future work.

References

[1] NIST, "NIST Definition of Cloud Computing," The National Institute of Standards and Technology (NIST), 2016. [Online]. http://www.nist.gov/itl/cloud/.

[2] S. Ghoshal and G. Paul, "Exploiting BlockChain Data Structure for Auditorless Auditing on Cloud Data," Information Systems Security Lecture Notes in Computer Science, pp. 359–371, 2016.

[3] E. Gaetani, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, and V. Sassone, "Blockchain-based database to ensure data integrity in cloud computing environments," CEUR Workshop Proc., vol. 1816, pp. 146–155, 2017.

[4] Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. [Online]. http://www.bitcoin.org/bitcoin.pdf, 2008

[5] Mohamed El Ghazouani, Moulay Ahmed El Kiram and LatifaEr-Rajy, "Blockchain& Multi-Agent System: A New Promising Approach for Cloud Data Integrity Auditing with Deduplication," IJCNIS, Vol. 11, No. 1, pp. 175–184, 2019.

[6] M. S. Niaz and G. Saake, "Merkle hash treebased techniques for data integrity of outsourced data," *CEUR Workshop Proc.*, vol. 1366, pp. 66–71, 2015.

[7] J.-Y. Ha, Y.-S. Lee, and J.-S. Kim, "De-duplication with Block-Level Content-Aware Chunking for Solid State Drives (SSDs)," 2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing, 2013.

[8] A. Venish and K. S. Sankar, "Study of Chunking Algorithm in Data De-duplication," *Proceedings of the International Conference on Soft Computing Systems Advances in Intelligent Systems and Computing*, pp. 13–20, Aug. 2015.

[9] X. Nie, L. Qin, and J. Zhou, "A content-awarechunkingscheme for data de-duplication in archival storage systems," High Technol. Lett., vol. 18, no. 1, pp. 45–50, 2012.

[10] I. Zikratov, A. Kuzmin, V. Akimenko, V. Niculichev, and L. Yalansky, "Ensuring data integrity using Blockchain technology," 2017 20th Conference of Open Innovations Association (FRUCT), 2017.

[11] J. L. Shawn Wilkinson, "Metadisk: Blockchain-Based Decentralized File Storage Application," *Liq. Cryst.*, vol. 14, no. 2, pp. 573–580, 1993.

[12] Y. Xu, "Section-Blockchain: A Storage Reduced Blockchain Protocol, the Foundation of an Autotrophic Decentralized Storage Architecture," 2018 23rd International Conference on Engineering of Complex Computer Systems (ICECCS), 2018.

[13] Y. Sasaki and K. Aoki, "Finding Preimages in Full MD5 Faster Than Exhaustive Search," Advances in Cryptology - EUROCRYPT 2009 Lecture Notes in Computer Science, pp. 134–152, 2009.

[14] Y. Yu, L. Xue, M. H. Au, W. Susilo, J. Ni, Y. Zhang, A. V. Vasilakos, and J. Shen, "Cloud data integrity checking with an identity-based auditing mechanism from RSA," Future Generation Computer Systems, vol. 62, pp. 85–91, 2016.

[15] L. González-Manzano and A. Orfila, "An efficient confidentiality-preserving Proof of Ownership for de-duplication," Journal of Network and Computer Applications, vol. 50, pp. 49–59, 2015.