VAULT

A Shared Distributed And Redundant Storage Solution

T.R.N.R. Peiris
Department of Computer Systems Engineering,
Sri Lanka Institute of Information Technology,
Malabe, Sri Lanka
peiris1996@gmail.com

K.V.A. Sachintha
Department of Computer Systems Engineering,
Sri Lanka Institute of Information Technology,
Malabe, Sri Lanka
amod077@gmail.com

B.A. Ganegoda
Department of Computer Systems Engineering,
Sri Lanka Institute of Information Technology,
Malabe, Sri Lanka
asviganegoda@gmail.com

W.M.U.K.M.T. Bandara
Department of Computer Systems Engineering,
Sri Lanka Institute of Information Technology,
Malabe, Sri Lanka
ukmihiran@gmail.com

***Abstract*—An ideal distributed storage solution must have the ability to provide redundant, reliable, shared and secure access to user data without compromising the ability to scale and descend while maintaining performance. VAULT is an attempt to avert the negatives of the cloud in a local environment using a decentralized methodology. VAULT makes use of individual idle storage space on a network of peer-to-peer nodes which is then provided to an end user to store files in the pooled space. VAULT implements redundancy by the use of Reed-Solomon codes and maps file fragment locations using a blockchain as a distributed ledger. Fragment distribution is optimized using a machine learning approach where node characteristics are used to determine the reliability of each node. The aggregation of above features makes VAULT and ideal solution for corporate environments where consumer hardware and infrastructure is already allocated.**

Keywords—distributed ledger, blockchain, socket.io, websockets, regression, reed-solomon erasure code, brain.js, redundancy, hashing, encryption, merkle tree, peer-to-peer

I. Introduction

A huge amount of storage space sits idly unused in mobile devices, hard drives etc. in the world. Large scale corporations that routinely use thousands of workstations for their employees have hundreds of terabytes of unused free space in their hard drives. If there was a way to make use of this free space, it would greatly reduce costs of operating and maintaining costly storage infrastructure and increase efficiency in the process as well. The threat of data exfiltration from organization computers where employees work is also a major concern. If an unauthorized person were to gain access to the computer, he/she would have access to all data in that device. So far organizations have spent millions of dollars annually and are still paying for Cloud Storage services. However, the security of the data that is stored on the cloud cannot be guaranteed. VAULT plans to address this problem in a decentralized methodology. Vault will encrypt, segment and store data in multiple locations in a pooled storage space and use a blockchain to map hashes to actual locations. The blockchain will act as a DHT (Distributed Hash Table) in this scenario. The application will fragment and store file blocks throughout a peer to peer network in a redundant manner. This redundancy is achieved by implementing the Reed-Solomon Error Correcting codes. These blocks can then be used to recover the original file up to a maximum of N-2 peer failures (N –number of peers one file will be distributed) depending on the Reed-Solomon configuration.

II. Literature Review

A. Past work

1) Authentication in Distributed Systems: Theory and Practice [1]

This paper describes both a theory of authentication in distributed systems and a practical system based on the theory. It also uses the theory to explain several other security mechanisms, both currently existing and proposed.

The researchers made several assumptions regarding the channels based on encryption and the hardware and local operating system on each node. This proposed method however does not include blockchain mechanisms in any way.

2) Reed-Solomon Codes [2]

Reed-Solomon Codes are used to correct burst errors and have a wide range of applications in digital communications and data storage. In coding theory, Reed-Solomon (RS) codes are the subset of BCH (Bose–Chaudhuri–Hocquenghem) codes that are one of the most powerful known classes of linear cyclic block codes. At the present, Reed-Solomon Codes is the most efficient and powerful method used for error detection and correction. This paper describes and analyzes the performance and efficiency of Reed-Solomon (RS) Codes.

3) Coda: A Highly Available File System for a Distributed Workstation Environment. [3]

Coda is a DSS for a large-scale environment that has UNIX workstations. Coda uses two methods in order to provide resiliency towards data loss happens due to network and server failures. Like any other DSS the Cods system will store the data at multiple servers making it less likely to be susceptible to data loss due to network failures and server failures and as an additional protection method the cache site acts as a replication component. This leads to more availability of the data. However, the paper suggest that the system is still in the development phase and has a lot of integrations.

B. *Similar Products*

1) *MooseFS*

MooseFS is a fault-tolerant, highly available, highly performing, scaling-out, network distributed file system which distributes data over several physical servers that are visible to the user as one virtual disk. It is POSIX compliant and acts like any other Unix-like file system. [4]

2) *Ceph*

Provides data safety for mission-critical applications. Provides virtually unlimited storage to file systems. Applications that use file systems can use CephFS with POSIX semantics without any integration. Automatically balances the file system to deliver maximum performance. [5]

3) *IBM Spectrum Scale*

Spectrum Scale is a commercial level file system which provides higher performance by striping blocks of data from individual files over multiple disks and reading and writing these blocks in parallel. Other features provided by Spectrum Scale include high availability, disaster recovery and security. [6]

4) *Minio*

Minio is a high-performance distributed object storage server, designed for large-scale private cloud infrastructure. Minio is widely deployed across the world with over 196.7M+ docker pulls. [7]

III. METHODOLOGY

A. *Distributed Ledger*

Since the days of yore, ledgers were central in driving economic growth, recording transactions to provide context into the economic activities of the day. The technology that started from using clay tablets and then papyrus leaves, made a huge leap with the invention of paper. Over the last 50 years, computers have transformed the process of record keeping and ledger maintenance at an astonishing pace. Today with the surge of interest in distributed data systems, the information stored on computers is taking an entirely different form, which is cryptographically secured, fast and most importantly, decentralized.

A single database that is consensually shared and synchronized across multiple institutions, geographies or on a smaller scale, nodes (computers) is called as a Distributed Ledger. It eliminated the need of a central authority to remain responsible for the data and its security. Information stored in the distributed ledger / blockchain is securely and accurately stored with the help of proven cryptographic techniques used for hashing and sometimes encryption which can be accessed using keys and cryptographic signatures.

A participant in the peer to peer network for a distributed leger has access to the shared data repository and owns and stores an identical copy of it locally. Any changes or additions made to the ledger are copied to all participants such that synchrony is achieved in a matter of seconds or minutes. Underlying the distributed ledger technology is the blockchain, which is the technology that underlies Bitcoin, Ethereum etc.

The functionality of the distributed ledger in Vault is that it provides an immutable record of files uploaded to Vault from day zero. It also records actions taken on the file such as share or delete. This allows the application to track each individual file and its activity.

The distributed ledger or blockchain inside Vault will hereafter be called as “Fragchain”. Fragchain is a combination of two main and specific components.

1. Local blockchain logic
2. Messenger module

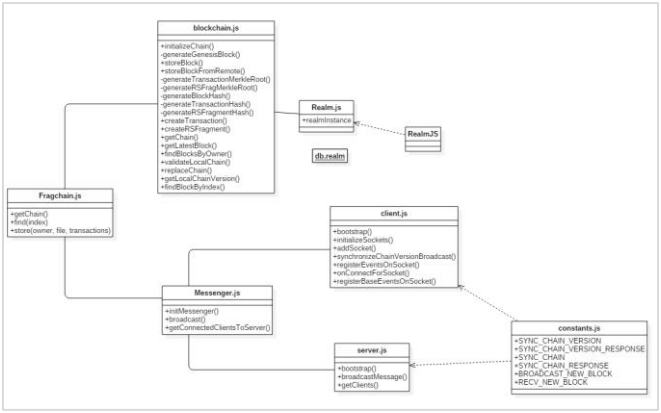

Figure 1: Fragchain Architecture

1) *Local Blockchain Logic*

The logic governing the local chain is defined in this module. It exposes an API that is used to manipulate the local blockchain. Vault’s Fragchain uses the Realm database for storage. It is a widely used key value mobile database which is compact and easy to use.

It should be noted that the current data structure can be easily modified to be used for a different use case altogether. Fragchain was developed in a way which allows a developer to mutate the data structure without breaking blockchain specific functionality, giving the developer the freedom to store any kind of data in the distributed ledger. This allows Fragchain to be used in scenarios where a distributed ledger is required.

a) Data Structure

Data that needed to be persisted for Vault to function were mapped into a specific data structure that uniformly represent the fragmenting hierarchy. The data written to the blockchain is structured as follows.

A Block contains a list of transactions which in turn contains a list of fragments. The figure above depicts the

cardinalities of components of the data structure. The entire block is then saved on to the chain with references to the authenticated user and details concerning the original file. Once a file is uploaded, the file is chunked into pieces whose size depends on the host's capabilities. These chunks are then processed and fragmented further using Reed Solomon Erasure Coding. The chunks map to Transactions and the pieces generated from erasure coding are mapped into fragments. Therefore, a Block can contain one or more Transactions and a Transaction always contain 6-8 Fragments depending on the Reed Solomon Configuration [8].

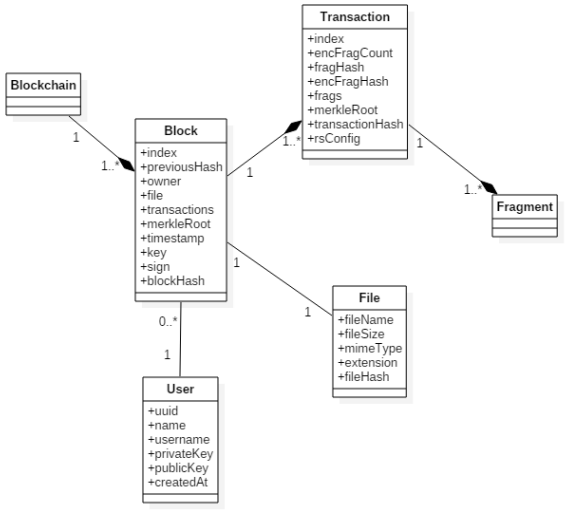

Figure 2: Block Data Structure

b) Security

INTEGRITY

A Merkle Root is calculated for Transactions and Fragments individually. This step ensures the integrity of data stored to an excessive degree. If a block is modified voluntarily or involuntarily, the validation mechanisms that verifies the hash tree would detect the abnormality and fail. Merkle Root calculations are carried out twice when a block is generated. Firstly, the fragments are hashed individually, and their hashes are used to calculate the FragmentMerkleRoot which is stored in the Transaction as 'merkleRoot'. Then the transactions are hashed individually, and the transaction hashes are used to generate the TransactionMerkleRoot which is subsequently stored in the Block as 'merkleRoot'. The Block's hash is then derived using parameters such as its index, previous hash, owner, timestamp and importantly Merkle Root. This single value signifies the integrity of Transactions and Fragments when aggregated.

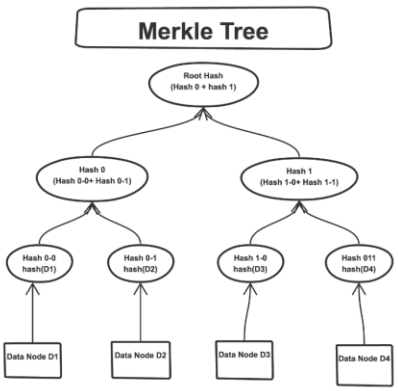

Figure 3: Merkle Tree

ORIGIN AUTHENTICATION

Digital signing is a technique used to validate or verify the authenticity and integrity of data or information. It also provides the added assurance of evidence of origin and identity which prevents the problem of tampering and impersonation in digital communications. Fragchain uses digital signatures to sign individual blocks before they are added to the distributed ledger.

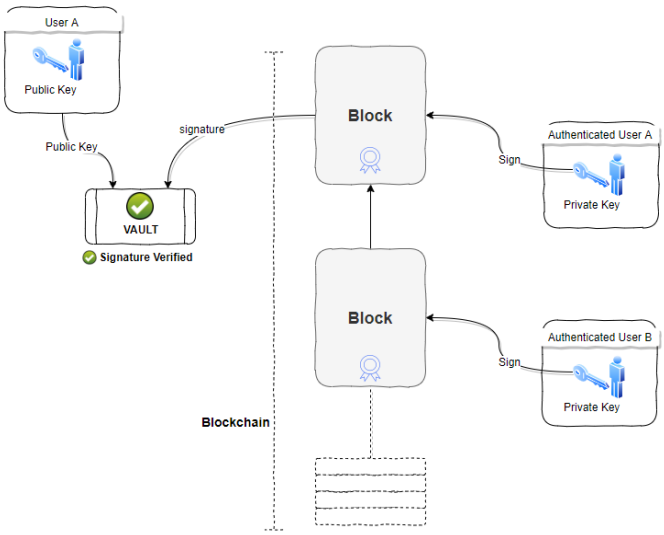

Figure 4: Block Signing

DISTRIBUTED LOCKING TO SOLVE CONSENSUS

It is extremely important that the shared ledger is modified in a mutually exclusive manner. When multiple nodes try to perform modifications to the ledger, those modifications should be performed synchronously such that it isn't subject to mutations intermittently. If write access to the ledger were not mutually exclusive, the resultant ledgers of each node will be different from one another. This causes the consistency of the entire shared ledger to fail.

To counter said circumstances, mutual exclusiveness was implemented in the form of a distributed lock when performing integrity changing operations such as modifications or additions on the ledger. A DLM (Distributed Lock Manager) was implemented in JavaScript (NodeJS) that emulates the DLM of Redis, Redlock [8] [9]. The following properties [9] are achieved as minimum guarantees so that distributed locks are utilized in an effective way.

1. Safety – Mutual Exclusion. Only one participant / node can hold the lock at any given time.
2. Deadlock Free – It should eventually be always possible to procure a lock, albeit the node that had the lock crashes or fails.
3. Fault Tolerant – Nodes should be able to procure and release locks as long the majority of nodes are live.

In the scenario where a host doesn't hold the lock, it tries to acquire the lock from all nodes in the swarm. Only after acquiring the lock, the system can make persistent changes to the ledger. The token used in the DLM has a TTL (Time to Live) value. This value dictates the lifetime of a token. The main objective of using a TTL value with the token is to prevent deadlocks. If a node while holding the lock crashes, the rest of the peers will not have access to the shared ledger unless the lock is released. However, with the TTL implemented, the lock can expire allowing the nodes to access the shared resource albeit with a delay.

OTHER FEATURES

These data structures, specifically the ones that hold file and fragment data contain fields that are designated to store file digests. When fragments are generated, Message Digest 5 (MD5) hashes are computed for each file and stored alongside the Fragment. These are then queried when a file is reconstructed to validate the integrity of the regenerated file.

The underlying database is encrypted using AES-256 using a peer-key [10], which is configured by the administrator of the swarm when configuring for the first time. In the event someone with physical access to the computer / device steals the raw database file, they won't be able to read the contents of it.

2) Messenger Module

Communication is a key factor in distributed systems to maintain state and propagate changes. The Messenger module of Fragchain works closely with the local blockchain logic to propagate changes made to the ledger to every node in the peer to peer network. The messenger module uses Socket.io to maintain real-time, bidirectional, event-based communications.

The communication happens between a socket.io-client and a socket.io-server. Each Fragchain node houses a socket.io-client and a socket.io-server due to the peer to peer architecture of Fragchain. Socket.io switches protocols and upgrades to better transports such as WebSocket without the intervention of a user or an external component. It also tries to auto reconnect forever when a remote server is crashed or non-responsive. This feature allows Fragchain to automatically self-recover in the event of peer failures. A heartbeat mechanism is also implemented allowing the server and the client to know when the corresponding end is non-responsive.

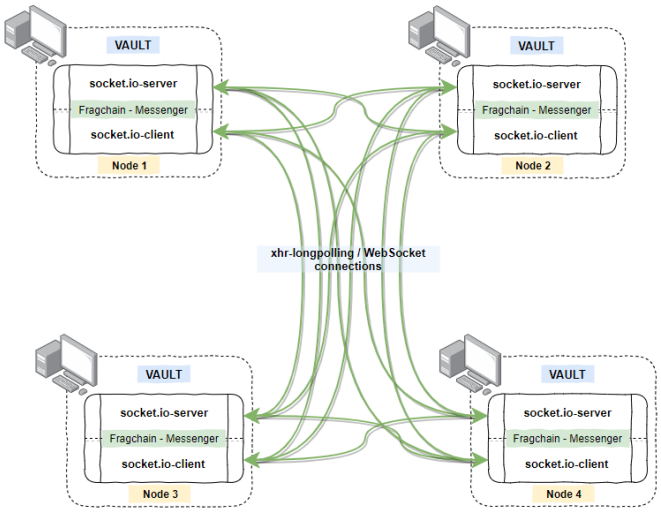

Figure 5: Messenger Module

The event driven paradigm of socket.io lets the messenger module to respond to events that are sent or broadcasted over the peer to peer network. The events the Messenger module respond to are as follows.

1. SYNC_CHAIN_VERSION
2. SYNC_CHAIN_VERSION_RESPONSE
3. SYNC_CHAIN
4. SYNC_CHAIN_RESPONSE
5. BROADCAST_NEW_BLOCK
6. RECV_NEW_BLOCK

a) SYNC_CHAIN_VERSION

When Fragchain starts for the first time, or when it recovers from a crash, a SYNC_CHAIN_VERSION event is broadcasted. This event requests the chain version of all the peers in the peer to peer network. Upon receiving chain version responses from peers, the Messenger module checks the local chain version and compares it with the received version to determine synchrony. If the chain versions do not match, the entire blockchain is requested from a peer in the network which is then swapped with the local chain. The Messenger module internally calls the local blockchain API to replace the blockchain.

b) SYNC_CHAIN_VERSION_RESPONSE

Fragchain nodes that listens to SYNC_CHAIN_VERSION respond back with SYNC_CHAIN_VERSION_RESPONSE with the local chain version. The Messenger module locally calls the blockchain API to resolve the chain version which is then communicated to the requesting peer.

c) SYNC_CHAIN

When a peer receives a SYNC_CHAIN_VERSION_RESPONSE from a remote peer, the chain version is compared, and the local chain is updated if the versions do not match. The SYNC_CHAIN event is emitted to the peer who responded with the higher chain version. The emitting node requests the blockchain of the listener so that is can be swapped with the obsolete chain.

d) SYNC_CHAIN_RESPONSE

Upon receiving SYNC_CHAIN event from a remote peer, the system queries the local chain and creates a stream-able object that contains the local blockchain. This is then streamed to the peer that requested the latest blockchain with SYNC_CHAIN_RESPONSE as the event name.

Once the SYNC_CHAIN_RESPONSE is received by the peer, it starts to asynchronously save the incoming blockchain data by swapping the outdated blockchain with the latest version. During the swapping process, the peer is prevented from making writes to the chain until it ends. Any newly created or received blocks are held in a buffer until the swap succeeds.

e) BROADCAST_NEW_BLOCK

When a file is uploaded to Vault, Fragchain creates the block, and after acquiring the lock, the block is saved to the local chain. The BROADCAST_NEW_BLOCK event is then fired, and block is then broadcasted to the entire swarm of peers.

f) RECV_NEW_BLOCK

Upon receiving a new Block from a remote peer, Fragchain tries to acquire the lock before saving the block to the local chain. When the lock is acquired, the received block is saved to the local chain after which the lock is released.

The events described above are the bare minimum required for nodes to communicate and maintain a unified distributed ledger. This can be modified and extended to other use cases by simply defining relevant events and listeners.

3) Results

Fragchain differs from other blockchain solutions available because it does not use computationally extensive PoW (Proof of Work) calculation to achieve consensus [10]. Fragchain is a distributed ledger that who's main function is to store data securely. Therefore, the implementation of a PoW function was redundant [11]. This allows Fragchain to converge and achieve consensus within a matter of seconds.

Blockchain	Time taken to converge
Bitcoin	~10 minutes
Ethereum	~ 12 seconds
Fragchain	~2 seconds

Table 1: Time taken to Converge

Furthermore, Fragchain does not contain alternate histories (ancient states) where the blockchain is branched and left stale. It has a single linear history that starts from the genesis block and spans forward in time. This has allowed Fragchain to be compact thereby utilizing the bare minimum storage required to store the blockchain locally.

Fragchain automatically recovers from a hard fail and synchronizes the blockchain to the swarm default version when recovering. The messenger client of all peers in the swarm polls the crashed servers until it comes back up. This allows the swarm to self-heal without 3rd party intervention

B. Redundancy Function

1) Introduction to Reed-Solomon Codes

In 1960 RS Codes were first introduced as an error correcting code to be applied in data transmission to ensure that the messages sending through communication networks are error free. Before transmitting the message, if there are *k* number of digits to be transmit, RS code will increase it to *n* number of digits by adding some extra *s* number of parity digits to the message, where $n = k + s$. By using RS code assurance of error freeness of a message can be given even if *s* number of digits are corrupted while transmission. [8]

2) Reed-Solomon erasure codes for data storage

RS erasure coding technology can be also used as a standard to store data reliably within datacenter environments. The advantage of using RS coding in datacenter environment is even though some parts of the data is missing the original data can be recovered. Backblaze already have built their own java library that suites for datacenter environment. [9]

3) JavaScript libarary for RS code in VAULT

Since VAULT is a distributed file system when a user uploads a file to the system it will be compressed, if not already compressed, encrypted and then send to the function called Redundancy function where in the system RS coding will be used to segment the files into shards in a redundant manner. Redundancy function of the VAULT mainly tries to achieve two goals.

- Reduce the amount of storage space needed for backups
- Increase the assurance of data availability

The redundancy function was written from the scratch using JavaScript and currently it is configured to split a file into four shards while generating two parity shards. Mainly the program contains two separate sections called encoder and decoder, which will be used for file segmentation and regeneration.

a) Encoder

Basically, the encoder takes one file as an input and returns six files as an output. It will be used in the file uploading process. When a user uploads a file it will be compressed, if not already and then encrypted. The encrypted file will be temporarily saved into the local machine, once it is saved the system will automatically invoke the encoder function. Encoder function reads the file and loads it into four different same size

buffers. Then those four buffers will be directly saved into four files and will be used to generate the other two parity buffers.

File (Hex format): 00 01 02 03 04 05 06 07 08 09 10 0A 0B 0C 10 11 12 1A 1B AA

Data buffer 01	00	01	02	03	04
Data buffer 02	05	06	07	08	09
Data buffer 03	10	0A	0B	0C	10
Data buffer 04	11	12	1A	1B	AA

Table 2:Data Buffers

When building the parity buffers one byte from each data buffer with the same index will be selected at a time and then apply them into two simple equations to generate two parity buffers byte by byte.

Equation 01 (to generate parity 01)

P1 = d1 + d2 + d3 + d4

Equation 02 (to generate parity 02)

P2 = (2 x d1) + (4 x d2) + (d3 x 6) +(8 x d4)

When reading a byte in JavaScript it will be automatically convert it into a decimal integer value between 0 to 255. Assume d1, d2, d3, d4 all are equals to hexadecimal value of 0xAA. In such cases value of P1 and P2 will be greater than 255, which then cannot be convert into a one hexadecimal value, because of that reason finite fields will be used in the program to map the output value into a value between 0 to 255.

```
const PrimeField = require('rye').PrimeField;
const field = new PrimeField(257);

let parity1Byte = field.add(field.add(shard1[i], shard2[i]), field.add(shard3[i], shard4[i]));

let parity2Byte = field.add(field.add(field.mul(2, shard1[i]),field.mul(4,shard2[i])),field.add(field.mul(6,shard3[i]),field.mul(8, shard4[i])));
```

Since there are some values in the equation that need to be divided by even numbers JavaScript finite field function cannot map them in between 0 to 255 in such cases, instead of mapping into a value between 0 to 255 mapping will be done between 0 to 256.

When output values are map between 0 to 256, value of 256 cannot be saved into the buffer, hence the program save it to the buffer as 0 and maintain a Boolean array which keeps the status of each byte which will be added to the parity buffer by checking whether the byte is a 0 or 256. At the end, those bits stream will be appended to the end of the parity buffers, which make parity shard is equals to size of a one data shard plus size of a data shard divided by 8.

Size of a parity shard = size of a data shard + size of a data shard / 8

	Hexadecimal values					Bit values				
Index	0	1	2	3	4	0	1	2	3	4
Parity buffer 01	38	35	46	50	199	0	0	0	0	0
Parity buffer 02	252	230	49	69	215	0	0	0	0	0

Table 3: Parity Buffers

Once the encoding or else segmentation process is over, total number of six files will be distributed among the local network and the details such as hash values of each shard will be pushed to the blockchain.

b) Decoder

Functionality of the decoder is to reverse the process performed by the encoder, hence it takes 4 files as an input and returns one file which holds the same hash as the original file the user wants to download. When a user request or else press download button of a file, details related to that file will be taken out from the blockchain in a secure manner and then system will search for the six shard which was distributed among the network by the encoder. Once the system has found four available shards from the network it will automatically invoke the decoder function.

First, decoder function checks the index value of each shard. Index values will be given to the shards when the encoding process happens based on the order the original file will be segmented.

Shard	**Index value**						
Data shard 01	1	00	01	02	03	04	
Data shard 02	2	05	06	07	08	09	
Data shard 03	3	10	0A	0B	0C	10	
Data shard 04	4	11	12	1A	1B	AA	
Parity shard 01	5	26	23	2E	32	C7	00
Parity shard 02	6	FC	E6	31	45	D7	00

Table 4: Index Values

Once the decoder function has found out the index value of the shards it will check whether how many numbers of data shards are missing, if all the four data shards are available decoder will concatenate the four shards together based on their index values and re-generate the original file right away.

```
fs.writeFileSync(decodedFile,Buffer.concat(data));
```

Regenerated File (Hex format): 00 01 02 03 04 05 06 07 08 09 10 0A 0B 0C 10 11 12 1A 1B AA

If any data shards are missing based on the available parity shards the missing data shards will be regenerated byte by byte and stored inside a buffer.

When regenerating the data shards with the help of the parity shards the bit stream which was appended to the end of the parity shard will be taken out and saved in a Boolean array, and when each byte is generating if the generated byte value is equals to 0 the program decide whether it is actually a 0 or 256, based on the Boolean value in the array with the same index value as the parity byte.

Below equations will be used when recreating the data shards from the available parity shards by the decoder and in each addition, division or multiplication or subtraction finite fields will be used to map output values to a value between 0 to 255. Assume that one data shard is missing, and the available parity shard is parity 01.

Missing data shard	Equation to be used
Data 01	D1 = P1 – D2 – D3 – D4
Data 02	D2 = P1 – D1 – D3 – D4
Data 03	D3 = P1 – D1 – D2 – D4
Data 04	D4 = P1 – D1 – D2 – D3

Table 5:Recovery Equations 01

Assume that one data shard is missing, and the available parity shard is parity 02.

Missing data shard	Equation to be used
Data 01	D1 = (P2 – 4xD2 – 6xD3 – 8xD4)/2
Data 02	D2 = (P2 – 2xD1 – 6xD3 – 8xD4)/4
Data 03	D3 = (P3 – 2xD1 – 4xD2 – 8xD4)/6
Data 04	D4 = (P2 – 2xD1 – 4xD2 – 6xD3)/8

Table 6:Recovery Equations 02

Assume that two data shards are missing, and two parity shards are available.

Missing data shards	Equations to be used
Data 01, 02	D1 = (4xP1 + 2xD3 + 4xD4 - P2)/2
	D2 = (P2 - 4xD3 - 6xD4 – 2xP1)/2
Data 01, 03	D1 = (6xP1 – 2xD2 + 2xD4 – P2)/4
	D3 = (P2 – 2xD2 – 6xD4 – 2xP1)/4
Data 01, 04	D1 = (8xP1 – 4xD2 – 2xD3 – P2)/6
	D4 = (P2 – 2xD2 – 4xD3 – 2xP1)/6
Data 02, 03	D2 = (6xP1 – 4xD1 + 2xD4 – P2)/2
	D3 = (P2 + 2xD1 – 4xD4 – 4xP1)/2
Data 02, 04	D2 = (8xP1 – 6xD1 – 2xD3 – P2)/4
	D4 = (P2 + 2xD1 – 2xD3 – 4xP1)/4
Data 03, 04	D3 = (8xP1 – 6xD1 – 4xD2 – P2)/2
	D4 = (P2 + 4xD1 + 2xD2 – 6xP1)/2

Table 7:Recovery Equations 03

The advantage of the parity shards is that even though there are two shards are missing from the total of six shards, regardless of the index value of the shard the original file can be regenerated.

4) Comparing Regular file systems with VAULT based on storage space needed for backups.

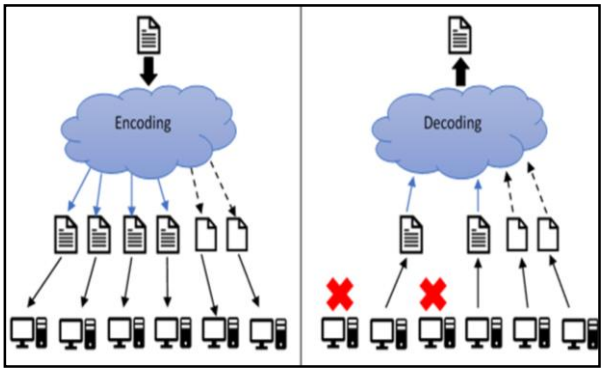

Figure 6 : VAULT Redundancy Function

The advantage of using RS coding-based technology to segment the files has its perks, one of those perks is that it will reduce the storage space requires for backups.

In a regular distributed file system when a file will be segmented into four shards, either the system will have to store two copies of each shard or two copies of the original file to recover the original file from a two-node failure situation.

Storage space required for backups = 2 x Size of the original file
Total storage space required = 3 x Size of the original file

In VAULT, since RS coding will be used when segmenting the file, the amount of storage space required to recover from a two-node failure situation is low comparing to the regular systems.

Storage space required for backups = 9 x (Size of the original file / 16)
Total storage space required = Size of the original file + 9 x (Size of the original file / 16)

Example: File size of 8 GB (8000 MB)

Regular system		
Backup space	2 x 8000	16 000 MB
Total Space	3 x 8000	24 000 MB

Table 8:Regular File System Storage Usage

VAULT		
Backup space	9 x (8000 / 16)	4500 MB
Total space	8000 + 9 x (8000/16)	12 500 MB

Table 9:VAULT Storage Usage

Approximately the total amount of storage space required by the VAULT to recover from a two-node failure situation is half of the size that will be required by the regular file systems.

C. Encryption and user authentication

1) Privacy and Secure Data Exchange

With the evolving technologies and the immense possibilities to use cloud storages to store user data and use web services apart from everyday using personal computers suddenly took everyone to a path of tiptoeing on a path of sensitivity and privacy. A secure design has to be established in order to use VAULT, since the application involves the use of a custom blockchain to store user information and furthermore the application offers a file share functionality with the option of both public and private.

Computer security is concerned with protecting an automated information system in order to attain the applicable objectives of preserving the confidentiality, integrity, and availability of information system resources, e.g., data, software, and hardware. [10]

a) Confidentiality

Implementing this requirement will be used to declare that the confidential information that is stored in the application cannot be disclosed to an unauthorized party.

b) Integrity

Integrity can be divided into two parts as data integrity and the system integrity. The first being the assurance of the data not being manipulated intentionally or unintentionally during the storage and the transmission process. System integrity ensured that the application acts as intended conducting the intended functions without any unaffected manner.

c) Availability:

This ensures that the application and all its functions and services can be reached as intended and is not denied any of the allowed users.

VAULT is designed and developed to comply with the said CIA triad.

2) User authentication

```
return  crypto.randomBytes(20).toString('hex');
```

Figure 8:Random Key Generation

Each user will be registered with a user defined password and a username, this way the user authentication in the application is covered. The user's detail such as passwords and usernames are also stored in the blockchain itself. However, the user credentials such as passwords are stored as non-reversible hash values making any attacks to gain access to the passwords futile.

3) Encryption

The Vault application is equipped with two Cryptographic principals which helps the VAULT application to fulfill its duties as a cryptographically secure application.

(1) Symmetric encryption.

(2) Public key cryptography.

The Cryptography feature helps to maintain confidentiality and integrity in the scope of private file sharing and private file storage.

As soon as a file chosen to be uploaded to the application, the application compresses that file thus making the file rather small and compact compared to the initial size of the file. Then a random certain length key is then generated from within the system, which is then later used as the input key for the AES – 256 file encryptions.

The crypto module is used a module to randomly generate key that is a random byte value which then converted to hex format then returned to the encryption module. Since the crypto module generates only random bytes, the bytes are not predictable.

```
if (checkExtension(filename)) {

    const cipher = crypto.createCipher('aes-256-cbc', keypair.tempkey());
    file.pipe(cipher).pipe(fs.createWriteStream('./temp/' + filename + '.enc'));

} else {

    const zip = zlib.createGzip();
    const cipher = crypto.createCipher('aes-256-cbc', key.tempkey);
    file.pipe(zip).pipe(cipher).pipe(fs.createWriteStream('./temp/' + filename  + '.enc'));

}
```

Figure 7:Compression and the encryption

The figure 7 depicts that the file is being checked for the extension before being compressed and encrypted. CBC mode is used as the mode in the AES encryption algorithm.

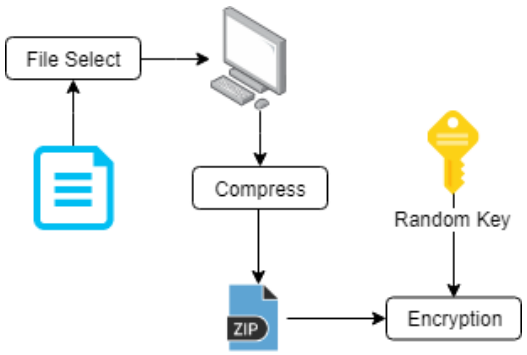

Figure 9:File encryption

The speed of the encryption may defer depending on the hardware that the application runs and the size of the file. The main problem that will be faced with this method is to share the secret key with the two parties. The Random key has to be distributed between the sender and recipient and must be only known to each of those users. A typical solution being the

sender and recipient must physically share the random key with each other, however that method is not the correct solution. The other solution is to exchange the random key over an unsecure open channel.

To avoid being vulnerable in the open channel, the random key that was generated depicted in the figure 8 will be stored securely in the blockchain using public key cryptography.

As a user registers to the system, a public and a private key will be created. And the public key is later stored in the blockchain that is used to identify the users individually. This public key is used to encrypt the random key that was used to encrypt the uploaded file and later that encrypted random key is stored in the blockchain along with the file details.

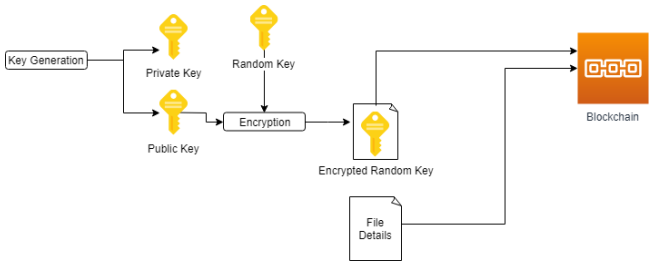

Figure 10:Random Key encryption

Advantages of using a random secret key is that of its fixed length and the use of less space. The Encryption and the decryption speed are based upon the key length and also the data size of the file to encrypt. Based on this input, the possible remediation to faster system runtime is to encrypt the Random key with the receivers' public key. Decreasing the time that is used to encrypt and decrypt drastically increases the total performance of the application.

Summarizing, three steps are necessary to enable an index for private sharing:

- Create Random Key to encrypt the compressed file
- Create two key pairs (Public and private) and encrypting the generated Random key with the sender's public key.
- Store the Encrypted Random key in the blockchain along with the file details.

Downloading the file works by applying the process reversely. At first the user will select the file that needs downloading and after that the user will be able to download the file using the index information stored in the blockchain. When the file is being downloaded, simultaneously the encrypted Random key will be downloaded. That key will be decrypted using the downloading parties' private key. That private key will be used to decrypt the file that was previously encrypted using the Random key.

Hashing values will be used to compare the hash values during the upload and download process in order to secure the integrity of the files.

4) Share Functionality

A main component of VAULT is a decentralized version of a share functionality based on a Peer to Peer network. The share functionality covers two modes:

1. Public Sharing
2. Private Sharing.

Both components will be explained using diagrams to sum up the methodology that is used and again depict the purpose and application of the two modes.

a) Public Sharing

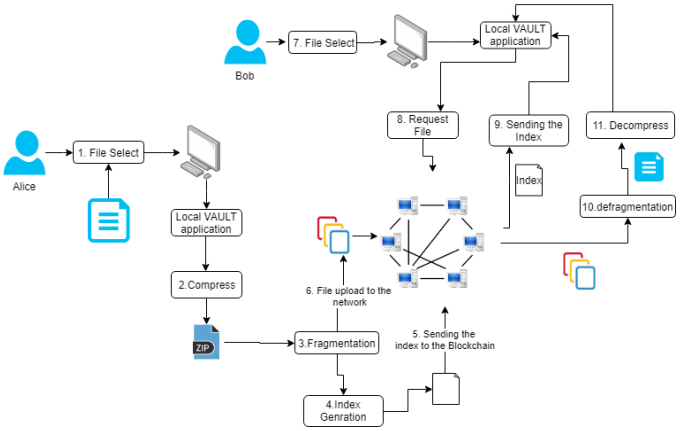

Figure 11:Public File sharing

Between the two modes that was mentioned above, the public sharing mode is the simplest one. The mode requires very little complex functions (i.e. No cryptography to encrypt the files) to store and share the files with the public. The figure 11 depicts the user Alice sharing a file with the public.

As the first step of the process the Alice wish to upload a file to the network and this file is a public file. Alice logs on to the application and the file is then selected. During the step two, the file is being checked whether if it is not compressed or not, if not the file is then compressed. The file is then fragmented into parts and hash values are then generated for each of these files' parts. The next part of the process is the index generation, the index that is generated will be stored in the blockchain for file downloading reference. The file is then uploaded to the network. Any user who has access to the application and the network can download the file. Since the file uploaded was public.

The downloading process is as the uploading process. When a file is selected, the index will be requested, that index will be used to defragment the file and then later the file will be decompressed.

b) Private Sharing

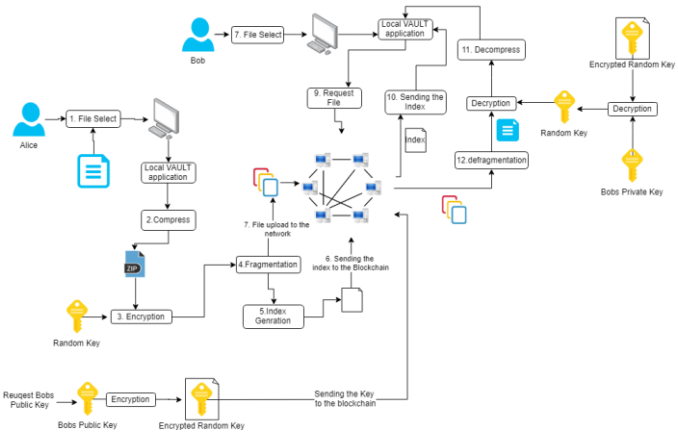

Figure 12:Private File sharing

Figure 12 scenario depicts steps that are in the private sharing function between two VAULT users Alice and Bob.

Same steps are carried out as the public sharing function until the step 2. As explained earlier in the cryptography section, a Random key will be used to encrypt the file. The file is then fragmented using the fragmentation module. Simultaneously the Random key that was used will be encrypted using the intended parties public key. Say if Alice needs to share the file with Bob, Alice encrypts the random key with Bobs public key. Say that Alice wants to keep the file to herself, the Random key is encrypted with the Alice's Public key. That way, only Alice has access to the file. The encrypted Random key is then sent to the blockchain with the index details.

The download process is same as the upload process. Bob gets a notification that a file has been shared with Bob by Alice and then Bob can use Bobs' private key to decrypt the encrypted Random key. The bob will get the index which is publicly available in the blockchain and download the file. The file is then defragmented and uncompressed and then decrypted using the Random key.

D. Machine learning for storage module

Storage management provides the facility to create and manage storage space for users to store the data. the storage space will create from the virtual storage pool. This virtual storage pool will create by collecting physical HDD or SDD drives via the network. Data storage module implements a dynamic disk allocation mechanism. Machine learning algorithms have an excellent ability to learn to manage with such a dynamic environment. Therefore, storage module consists of machine learning techniques.

1) Node Selection

Machine learning techniques aims to learn from data to solve problems such as prediction and estimation. Before storing the data all the nodes in the network will analysis and weighted. Also, will give the results which nodes can be storing the data. Primarily, machine a learning algorithm will give prediction details about all the nodes in the network.

The simplest definition of machine learning is collection of computational methods that use experience, previous information available to the system, to improve performance or to make predictions. The storage module uses a regression algorithm within the general machine learning literature. The node selection process performed by '`ml-regression-multivariate-linear`' java script library.

```
const predict = (node) => {
    return new Promise((resolve, reject) => {
        try{
            if(node[4] === 0){
                reject(0);
            }
            else{
                const mlr = new MLR(x, y);
                let results = mlr.predict(node);
                newInput = inputs;
                newInput.push(node);
                newOutput = outputs;
                newOutput.push(results);
                resolve(results);
            }
        }
        catch(error){
            reject(-1);
        }
    })
};
```

Figure 13: Sample Algorithm

When a node is requiring distributing a file among the network it collects information such as available storage space, free storage space, used storage space and health status from all the available nodes in the system. Then the information will be feeding into the regression algorithm and weights will be assigned to each and every node available in the system. Based on those weights system will decide which nodes are most suitable to hold the shards.

2) Node availablility through messenger module

This module act as a call center between nodes. Each time when a file will be scattered into shards and distributed among the nodes or else when a file will be regenerating from shards which was previously distributed among the nodes, the particular node where the owner or the download/upload requester rest triggers the model to check whether the particular nodes are up and running or down. And this module uses Socket.io to maintain real-time, bidirectional communication.

IV. Conclusion

It is evident from the functions and features of VAULT illustrated above that it successfully achieves and outperforms current applications in the niche market of storage solutions. Since privacy is inbuilt into the application environment, it is ideal to be used in privacy critical organizations. The fact that VAULT is intended to run on existing general-purpose hardware makes it even more appealing to corporate environments with hundreds of

workstations allowing them to save money on costly storage infrastructure. The decentralized nature of VAULT inherently eliminates a single point of failure thus defaulting to a much more resilient design. Intelligent resource allocation achieved via Machine Learning optimizations allow VAULT to perform file distribution much more effectively resulting in fewer reconstructions of files and even fewer files becoming unavailable.

V. Acknowledgement

We thank our supervisor Mr. Amila Senarathne who provided insight and expertise that greatly assisted the research. We heartily extend our thanks to Mr. Kavinga Yapa Abeywardena for guiding us in refining our presentation slides for the project.

VI. References

[1] M. A. M. B. E. W. BUTLER LAMPSON, "Authentication in Distributed Systems: Theory and Practice," *ACM Trans. Computer Systems,* vol. 10, no. 4 , pp. 265-310, (Nov. 1992.

[2] S. B. Wicker and V. K. Bhargava, "An Introduction to ReedSolomon Codes," 1994. [Online]. Available: http://ieeexplore.ieee.org/book/0780353919.excerpt.pdf. [Accessed 17 3 2019].

[3] M. I. J. J. K. P. K. M. E. O. E. H. S. A. D. C. S. MAHADEV SATYANARAYANAN, "Coda: A Highly Available File System for a Distributed Workstation Environment," *IEEE TRANSACTIONS ON COMPUTERS,* vol. VOL. 39, no. NO. 4, APRIL 1990 .

[4] MooseFS, "moosefs.com," MooseFS, 03 2019. [Online]. Available: https://moosefs.com/. [Accessed 02 03 2019].

[5] Ceph, "ceph.com," Ceph, 03 2019. [Online]. Available: https://ceph.com/ceph-storage/file-system/. [Accessed 01 03 2019].

[6] IBM, "www.ibm.com," IBM, 03 2019. [Online]. Available: https://www.ibm.com/us-en/marketplace/scale-out-file-and-object-storage. [Accessed 03 2019].

[7] Minio, "minio," Minio, 02 2019. [Online]. Available: https://www.minio.io/. [Accessed 03 2019].

[8] M. Kleppmann, "How to do distributed locking," 06 Feb 2016. [Online]. Available: https://martin.kleppmann.com/2016/02/08/how-to-do-distributed-locking.html. [Accessed July 2019].

[9] antirez.com, "Is Redlock safe?," 2017. [Online]. Available: http://antirez.com/news/101. [Accessed August 2019].

[10] realm.io, "Realm Database," [Online]. Available: https://realm.io/products/realm-database. [Accessed July 2019].

[11] BlockSplain, "Does blockchain size matter?," November 2018. [Online]. Available: https://blocksplain.com/2018/02/22/blockchain-size/. [Accessed August 2019].

[12] bitcoin.it, "Proof of work," bitcoin.it, 2018. [Online]. Available: https://en.bitcoin.it/wiki/Proof_of_work. [Accessed 2019].

[13] I. S. Reed and G. Solomon, "Polynomial Codes Over Certain Finite Fields," *Journal of the Society for Industrial and Applied Mathematics,* 1960.

[14] Backblaze, "Backblaze Reed-Solomon," Backblaze , [Online]. Available: https://www.backblaze.com/open-source-reed-solomon.html.

[15] E. A. R. Barbara Guttman, "An Introduction to Computer Security: the NIST Handbook," National Institute of Standards & Technology, Gaithersburg, MD, United States, 1995.

[16] A. Schoedon, "The Ethereum-blockchain size will not exceed 1TB anytime soon.," dev.to, 2017. [Online]. Available: https://dev.to/5chdn/the-ethereum-blockchain-size-will-not-exceed-1tb-anytime-soon-58a. [Accessed Aug 2019].

[17] L. Kahn, "How to Mine Ethereum. Guide for beginners," cointelegraph.com, 2018. [Online]. Available: https://cointelegraph.com/ethereum-for-beginners/how-to-mine-ethereum-guide-for-beginners#a-little-about-ethereum-mining. [Accessed 2019].