

**VAULT - A SHARED DISTRIBUTED AND REDUNDANT STORAGE SOLUTION**

**Project ID: 19-002**

**Project Proposal**

**B.Sc. (Honors) Degree in Information Technology**

**Submitted on 08-03-2019**

**VAULT**

**A shared distributed and redundant storage solution**

**Project ID:19-002**

**Authors:**

|  |  |  |
| --- | --- | --- |
| **Student ID** | **Name** | **Signature** |
| IT16106420 | T.R.N.R. Peiris |  |
| IT16158528 | K.V.A. Sachintha |  |
| IT16016026 | B.A. Ganegoda |  |
| IT16091276 | W.M.U.K.M.T. Bandara |  |

**Supervisor**

**………………………………….**

**Mr. Amila Nuwan Senarathne**

**DECLARATION**

We declare that this is our own work and this project proposal does not incorporate with acknowledge any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of our knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

|  |  |
| --- | --- |
| ……………………………………. T.R.N. R Peiris | ……………………………………. K.V. A Sachintha |
| …………………………………….  B.A Ganegoda | ……………………………………….  W.M.U.K.M.T Bandara |

**ABSTRACT**

An ideal distributed storage solution must have the ability to provide the users with redundant, reliable, shared and secure access to the user’s data and nevertheless must have the ability to scale and descend according to the storage space available while maintaining a higher performance level.

So far organizations have spent millions of dollars per year and are still paying on Cloud storage services. However, the security of the data that is stored on the cloud is uncertain with few exceptions. **VAULT** plans to address this problem in a decentralized methodology. It eliminates the authoritative figure of a cloud service provider such as Google or Amazon thus completely assuring privacy of stored data. **VAULT** will encrypt, segment and store data in multiple redundant locations (throughout the **VAULT** swarm / network) in the pooled storage space and use a custom blockchain to map the data hash to the actual location of the file segments. The blockchain will act as a DHT (Distributed Hash Table) in this scenario. The application will fragment and store file blocks throughout the peer to peer network in a redundant manner. The redundancy of data fragments is achieved by implementing the Reed-Solomon Error Correcting codes that fragments the file into data and parity blocks. These blocks can then be used to recover the original file to a maximum of N/2 peer failures (N – Number of peers in the Vault network) depending on the Reed-Solomon configuration.

**Table of Contents**

[1 Introduction 1](#_Toc3891973)

[1.1 Background 1](#_Toc3891974)

[1.2 Literature Review 2](#_Toc3891975)

[1.2.1 Past work 2](#_Toc3891976)

[1.2.2 Similar Products 4](#_Toc3891977)

[1.3 Research Gap & Research Problem 6](#_Toc3891978)

[1.3.1 Research Gap 6](#_Toc3891979)

[1.3.2 Research Problem 7](#_Toc3891980)

[2 Objectives 8](#_Toc3891981)

[2.1 Main Objective 8](#_Toc3891982)

[2.2 Specific Objectives 8](#_Toc3891983)

[3 Research Methodology 9](#_Toc3891984)

[3.1 Prerequisites 9](#_Toc3891985)

[3.2 Research Architecture 10](#_Toc3891986)

[3.3 Programming Structure 13](#_Toc3891987)

[3.4 Tools and Techniques 15](#_Toc3891988)

[4 Description of Personnel and Facilities 16](#_Toc3891989)

[5 Gannt Chart 18](#_Toc3891990)

[6 Budget 19](#_Toc3891991)

[7 References 20](#_Toc3891992)

[8 Appendices 22](#_Toc3891993)

[8.1 Work Breakdown Structure 22](#_Toc3891994)

**I List of Figures**

[Figure 1: MooseFS 4](file:///C:\Users\Amod\Desktop\Project%20Proposal\vault%20-%20latest%203.0.docx#_Toc3891995)

[Figure 2: BWFS 5](file:///C:\Users\Amod\Desktop\Project%20Proposal\vault%20-%20latest%203.0.docx#_Toc3891996)

[Figure 3: Process overview - Upload 10](#_Toc3891997)

[Figure 4: Process Overview - Download 11](#_Toc3891998)

[Figure 5: NodeJS Module based high-level implementation 13](file:///C:\Users\Amod\Desktop\Project%20Proposal\vault%20-%20latest%203.0.docx#_Toc3891999)

**II List of Tables**

[Table 1: Feature Comparison 7](#_Toc3892000)

[Table 2: Personnel and Facilities allocation 16](#_Toc3892001)

**III List of Abbreviations**

|  |  |
| --- | --- |
| API | Application Programming Interface |
| DFS | Distributed File System |
| DHT | Distributed Hash Table |
| DSS | Distributed Storage Solution |
| GZIP | GNU Zip |
| HTTP | Hypertext Transfer Protocol |
| LZ4 | Lempel Ziv 4 |
| NoSQL | Not only SQL |
| ORM | Object Relational Mapping |
| OSX | Macintosh Operating System X |
| PKI | Public Key Infrastructure |
| POSIX | Portable Operating System Interface |
| RUP | Rational Unified Process |
| SAN | Storage Area Network |
| SCSI | Small Computer System Interface |
| SHA | Secure Hash Algorithm |
| UNIX | Formerly UNICS for Uniplexed Information Computing System |

# Introduction

## Background

A huge amount of storage space sits idly unused in mobile devices, hard drives etc. in the world. Large scale corporations that 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.

A Distributed Storage System (DSS) is formed by networking together a large number of, inexpensive and unreliable storage devices and making the storage devices reliable and highly available with the capability to expand. Cloud storage is a model of computer data storage in which the digital data is stored in logical pools. The physical storage spans multiple servers (sometimes in multiple locations), and the physical environment is typically owned and managed by a hosting company. These cloud storage providers are responsible for keeping the data available and accessible, and the physical environment protected and running. Corporate environments buy or lease storage capacity from the providers to store user, organization, or application data. Which is an expensive thing to do to protect the organizations data. When it comes to the security of the data that is stored in the cloud, a company will have only a handful of people who can access those data. But when it comes to cloud storage, there may be thousands of employees and many customers. Which will drastically reduce the security of the data that is being stored in the cloud.

We are proposing a comprehensive counter approach to cloud storage solutions which applied properly could obsolete cloud data storage solutions in a corporate environment. This solution plans to address the problem in a decentralized methodology. It eliminates the authoritative figure of a cloud service provider such as Google or Amazon thus completely assuring privacy of stored data.

## Literature Review

This section depicts a literature survey carried out on existing platforms with nearly similar capabilities and each members’ individual functionalities. Some of the eminent researches and products are reviewed here.

### Past work

#### 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 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.

#### Reed-Solomon Codes [2]

Reed-Solomon Codes are used to correction of burst errors and have a wide range of applications in digital communication and data storage. In coding theory, Reed-Solomon (RS) codes are the subset of BCH 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 analyze the performance and efficiency of Reed-Solomon (RS) Codes.

#### Frangipani: A Scalable Distributed File System [3]

This new application called Frangipani is a new Distributed File System (DFS) that has a scalable option. This system was built on a two-layer structure. The lowest layer known as the Petal which is a Virtual disk which is like a large block device. This system provides a sufficient amount of scalability and availability. However, this system lacks a redundancy mechanism which is a needed step in case of a data loss. In the second layer, cluster of machines that has Frangipani system runs the Frangipani system code on the shared petal disks. This uses a distributed lock service in order to guarantee that the whole system acts as a single system. Frangipani is programed to run on an administration which acts as a centralized system. Which will become a single point of failure if any incident occurs. The Frangipani system works assuming that the machines trust each other. However, that will lead to a variety of several information security vulnerabilities.

#### The Hadoop Distributed File System [4]

This DFS was initially designed to store a huge number of data in a distributed server which are connected to each other or acts as clusters and stream the data that was stored to applications with a high bandwidth. With the ability to distribute the storage and computation across clusters of servers, the Hadoop system has the ability to grow and expand. However, the Hadoop system has few disadvantages, security of the data is a main issue.

#### The Google File System [5]

The Google File System is a DSS that is scalable that can be used for distributed applications that are both large and data intensive. The main highlight of this is that the Google DSS provides a fault tolerance although the system runs on inexpensive or unreliable hardware, and it delivers high aggregate performance to a large number of clients. However, the service is deployed within Google only. The storage is accessed by thousands of clients with the exception of that the clients don’t really have the control over their own data that they are storing in the storage allocated to them. The main concern in this also is the security threat that the data faces. A DSS solution should provide security since data is the main integral part of technology.

#### Coda: A Highly Available File System for a Distributed Workstation Environment [6]

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.

### Similar Products

#### MooseFS

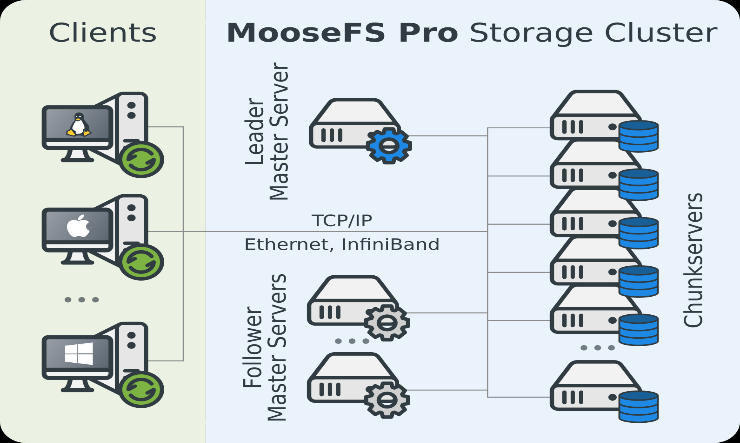


Figure 1: MooseFS

MooseFS is a fault-tolerant, highly available, highly performing, scaling-out, network distributed file system which distribute 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. [7]

#### 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. [8]

#### 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. [9]

#### OCFS2

OCFS2 is a file system for Linux which provides both high performance and high availability. It can be used with almost all applications and cluster-aware applications can make use of cache-coherent parallel I/Os from multiple nodes to scale out applications easily. [10]

#### OrangeFS

A scale-out network file system designed for use on high-end computing systems that provides very high-performance access to multi-server-based disk storage, in parallel. The OrangeFS server and client are user-level code, making them very easy to install and manage. [11]

#### Blue Whale File System (BWFS)

Figure 2: BWFS

BWFS is a SAN file system developed by Blue Whale for the FC SAN/IP SAN environment. It can integrate multiple FC or iSCSI disk arrays into a cluster storage space to support multi-server concurrent processing. It is designed to provide high-performance and expandable file sharing services and supports various applications of multi-server work flows or under the cluster environment. [12]

#### 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. [13]

## Research Gap & Research Problem

### Research Gap

Even though there are various kinds of commercial level and open source products available to assist in the process converting office cloud environment to an on-premises distributed file system environment most of them are made for server based large scale deployments and cannot be used for small scale organizations. It also does not give much in terms of a solution for the matter that this system is proposed which is to get maximum of the unused/idle storage. Some unique features of the proposed system **VAULT** are as follows.

* Data redundancy
* Use of less storage media for recover data when file crashes
* Security
* Platform Independency
* Easy Installation & Setup
* File Sharing
* Use of Blockchain

Table 1: Feature Comparison

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Features** | **MooseFS** | **IBM Spectrum Scale** | **OCFS2** | **OrangeFS** | **BWFS** | **Minio** | **Ceph** | **VAULT** |
| High Availability | ✓ | ✓ | ✓ | ✓ |  | ✓ | ✓ | ✓ |
| Scalability | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Minimal Investment | ✓ |  |  | ✓ |  | ✓ | ✓ | ✓ |
| Big Data Support | ✓ | ✓ | ✓ | ✓ |  | ✓ | ✓ | ✓ |
| Data encryption |  | ✓ |  |  |  | ✓ | ✓ | ✓ |
| Data Recovery | ✓ | ✓ |  | ✓ | ✓ | ✓ | ✓ | ✓ |
| Platform independent | ✓ | ✓ |  |  |  |  |  | ✓ |
| Security |  | ✓ | ✓ | ✓ |  | ✓ | ✓ | ✓ |
| Data Redundancy |  |  |  |  | ✓ | ✓ |  | ✓ |
| Minimum additional storage for backup |  |  |  |  |  |  |  | ✓ |
| Blockchain integration |  |  |  |  |  |  |  | ✓ |
| Easy to setup and run |  |  |  |  |  |  |  | ✓ |
| File Sharing |  |  |  |  |  |  |  | ✓ |

### Research Problem

The Large-scale corporations that use thousands of workstations for their employees have hundreds of terabytes of unused free space in their hard drives and the amount of money that these large-scale companies pay for cloud storages are increasing day by day.

This unnecessary spending costs companies around the world an estimated $62 billion annually and this is a serious issue that every company trying to overcome.

The security of the data that is stored in the cloud, a company will have only a handful of people who can access those data. But when it comes to cloud storage, there may be thousands of employees and many customers. Which will drastically reduce the security of the data that is being stored in the cloud and there is the threat of data exfiltration from organization computers where employees work is also a major concern.

In some cases, the clouds unavailability will disrupt the work process of many organizations. Thus, a solution is needed to reduce the unavailability and increase the availability of the data.

# Objectives

## Main Objective

Creating an easy to use, secure, non-repudiable, distributed application to make use of idle/unused storage space on networked computers with performance and redundancy gains that exceed current market products.

## Specific Objectives

* Blockchain involvement

A blockchain will be used as an intermediate database to track file locations to assist in the process of file segmentation and recreation.

* Redundancy  
  Active file reconstruction in case of node failures without making duplicates of the same file over and over.
* File sharing while keeping the security

A mechanism to share files like in cloud storages without revealing the locations of the file strips are in, will be develop under this objective.

* Upload/Download speed improvement

Parallel / non-blocking execution.

* Storage allocation using both dynamic and static mechanisms

Implementation of a mechanism to dynamically and statically allocate free storage to the storage pool in order to maintain availability.

* Security as a bonus

Security will be automatically increased hence one file won’t be stored inside one location and every file will be encrypted before them went in to the segmentation process and system won’t allow anyone to view the locations of the file strips.

# Research Methodology

This section will explore the research components and the methodology in which the research workload is carried out to build the distributed software product **Vault**. Since the final outcome of the research is a software product, we plan to adhere to software development practices introduced in Rational Unified Process (RUP) to iterate and develop a working product.

## Prerequisites

Since the research is in the domain of distributed computing, and the final software is designed to work in a peer to peer fashion, the development environment has to be configured such that a single codebase can be pushed to multiple peers simultaneously. Therefore, the requirements mentioned below are followed.

* The development environment has no strict conditions. It could either be Windows, Linux or OSX, only constraint being it supports dockerization.
* The selected development environment should have Docker installed. We plan to use Docker to spin up multiple containers (docker-swarm) to act as standalone peers in a constrained private network for testing.
* Since running multiple Docker containers is resource extensive, a powerful computer with abundant memory is required.

## Research Architecture

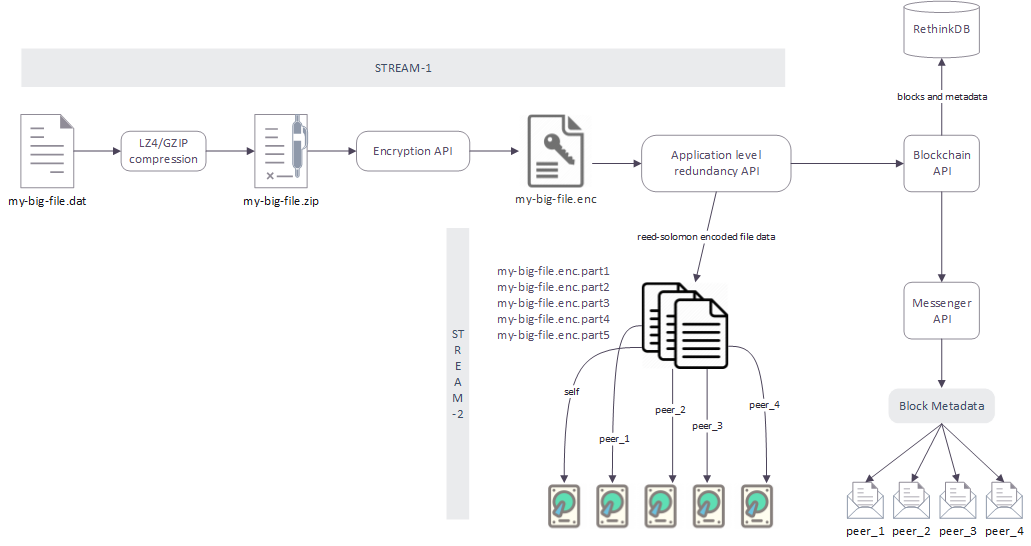


Figure 3: Process overview - Upload

A high level overview showing the main components of **Vault** and the procedure in which file redundancy is achieved is show above. An input file is compressed, encrypted and fragmented in a redundant fashion after which fragments are pushed to multiple nodes in the peer to peer network. Metadata of the original file and fragments are first stored on the original host in the custom blockchain after which they are relayed to all peers irrespective of them receiving a fragment. This method will create and maintain a consistent ledger of files that were added and/or removed from the Vault storage space throughout the Vault swarm’s lifetime.

The process can be further expressed as follows.

* LZ4/GZIP Compression – The input file stream from the uploaded file is read into the compression function to create a compressed output file stream.
* Encryption API – The compressed file stream enters the encryption API in which the data is encrypted using the user keystore. The encryption process is done in a way that a user can easily share the file to another user such that it doesn’t result in revealing the original key.
* Redundancy API – The file stream is encoded using Reed-Solomon codes to accomplish redundancy. The Reed-Solomon algorithm must be implemented in NodeJS to support stream processing and asynchronous execution.
* The Reed-Solomon encoded file fragments are then pushed to select peers in the Vault swarm asynchronously. This will make use of the entire network bandwidth available to that node.
* Simultaneously, the file and fragment metadata are stored in the custom blockchain implemented on the local RethinkDB NoSQL database. The same file and fragment metadata are also sent to all peers in the Vault swarm as well. This will ensure that the entire swarm will have a consistent/mirror copy of the blockchain.

The above flow of events expresses the upload function of a particular file. The image shown below will lay out the procedure in which a file is assembled from fragments that are stored in different peers in the swarm.

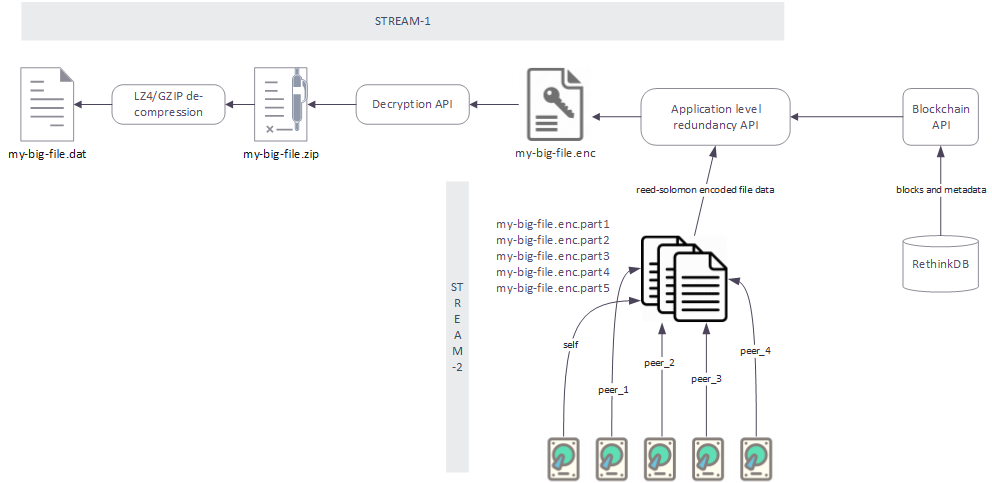


Figure 4: Process Overview - Download

* The user requests a file to be downloaded.
* The file and fragment metadata are extracted from the local chain and then the application requests the relevant files from the peers in the swarm asynchronously and downloads them to a temporary location.
* After the download is complete, the fragments are then sent through the Redundancy API to be decoded to the encrypted version of the file. After verifying the integrity of the encrypted file, it is then decrypted using the user’s key and then streamed to be downloaded via the decompression function.

If the amount of fragments that are resolved and downloaded doesn’t match the actual number of fragments, the Redundancy API will try to reconstruct the file to its original state. This may fail due to a couple of reasons,

* Number of fragments received are below the tolerance configuration for the Vault swarm. – This condition is catastrophic since the Redundancy API will not be able to decode and reconstruct the original file due to the original Reed-Solomon configuration at encoding.
* The fragment is corrupted – The Redundancy API will re-download the file and try reconstruction again.

## Programming Structure

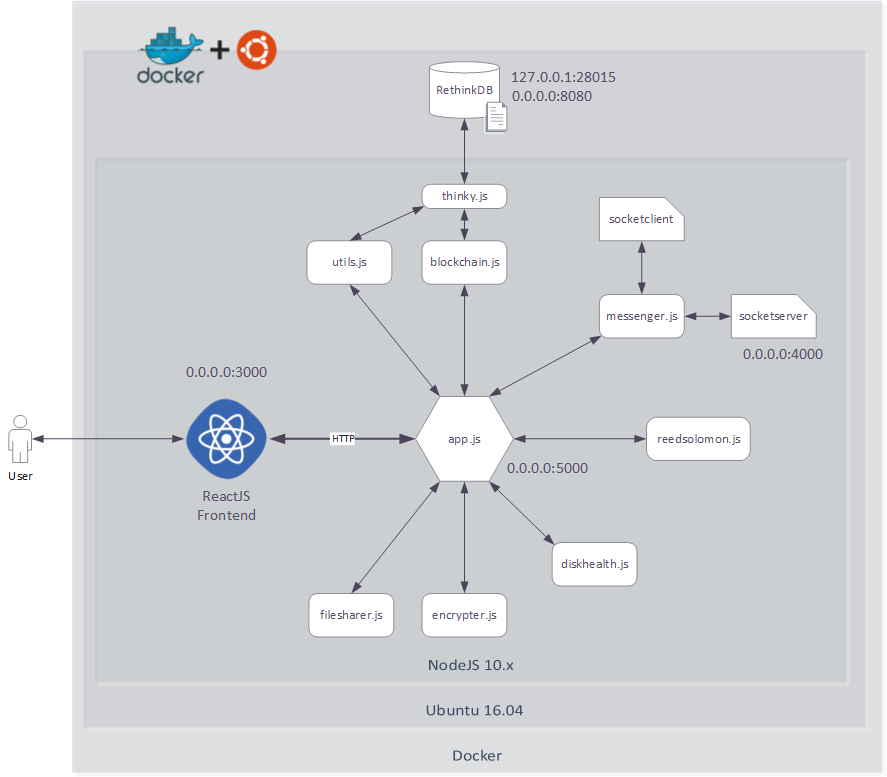
Since the research components can be designed, developed and tested out individually, the individual components are developed as modules which then can then be imported to assemble the complete software. The module based approach is as follows.

Figure 5: NodeJS Module based high-level implementation

Each module/library should be implemented in a standalone fashion such that the developer can test and update the module as necessary. The functions tasked to each module is as follows.

* utils.js – exposes an API of helper functions for user authentication, signup, application settings and host decommission handling.
* blockchain.js – contains the core blockchain logic. Block creation and validation is implemented here. It exposes an API to interact with the local base-chain.
* thinky.js – Contains data models that are replicated in the RethinkDB database. Exposes an API to interact with the database via the thinky ORM (Object Relational Mapping) library.
* messenger.js – peer to peer messaging library
  + socketserver - serves an socket.io endpoint for peers to connect to.
  + socketclient – saves and manages client socket connections to peers in the swarm.
* reedsolomon.js – Contains the NodeJS implementation of Reed-Solomon Erasure Codes. Exposes functions to encode and fragment files as well as decode and reassemble them.
* diskhealth.js – Functions to monitor and report host disk status and health is implemented here. It will also contain functionality to dynamically and statically allocate space for Vault to store fragment data.
* encrypter.js – Exposes an API to handle stream encryption and decryption asynchronously.
* filesharer.js – Implements file sharing capability.
* app.js – all modules are integrated here. It also exposes a HTTP API for the ReactJS frontend.

## Tools and Techniques

Listed below is a non-exhaustive list of tools and techniques to be used in this research.

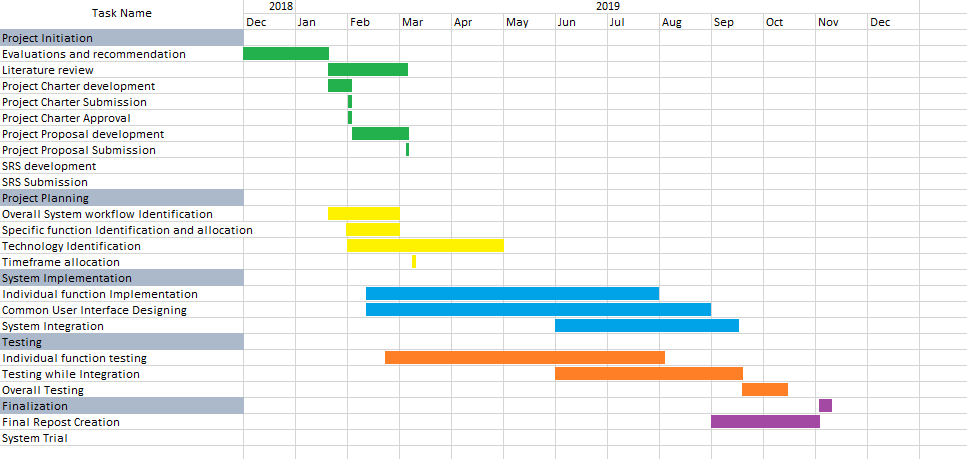
* Tools
  + docker
  + docker-compose
  + NodeJS
    - cryptoJS – SHA256 hashing library
    - socket.io – popular websocket library
    - thinkyJS – ORM for RethinkDB
  + Ubuntu docker image
  + RethinkDB NoSQL database
  + ReactJS
* Techniques
  + Stream Processing
  + Asynchronous / non-blocking execution
  + PKI (Public Key Cryptography)

# Description of Personnel and Facilities

Table 2: Personnel and Facilities allocation

|  |  |  |
| --- | --- | --- |
| **Member** | **Research Component** | **Tasks** |
| IT16106420  T.R.N.R. Peiris | Redundancy handling and hardware monitoring | * Implementation of a mechanism to act in the middle process of the system which take one file as an input and segment them into pieces in a redundant manner by implementing the Reed-Solomon error correcting codes. * A module/component to monitor and report drive health throughout the cluster will be implemented here as well. * Purpose of this process is that, even though one or two pieces of the scattered file were loss in the reconstruction process that file can be re generated. |
| IT16158528  K.V.A. Sachintha | Custom blockchain for key value storage with web socket based peer pairing | * Implementation of a blockchain to act as a DHT (Distributed hash table) which helps the process of storing segmentation and location data of a file. * The blockchain would also function as an immutable ledger, such that that it would contain the entire history of the Vault cluster. * Purpose of this process is to assist the process of segmentation and recreation of files in a secure manner in a decentralized file system. |
| IT16016026  B.A. Ganegoda | Identity management, key derivation for file sharing and encryption | * Implementation of a secure authentication mechanism which works efficiently and scale well in a distributed network and a suitable mechanism to encrypt the files before sending them in to the process of segmentation. * The sharing of a file by a particular user is also handled here. The receiving user should be able to decrypt the file which was encrypted by the sender’s node. * Purpose of this process is to control the access to the system using a secure authentication mechanism and increasing the security of a file by pre-segmentation encryption. |
| IT16091276  W.M.U.K.M.T. Bandara | Storage allocation mechanism and device removal protocol | * Implementation of a mechanism to dynamically allocate free storage to the system and a protocol to act in a situation of device removal. * The scenario in which a voluntary removal of a device is initiated is handled here. The existing data in that particular node should be copied back to the cluster nodes whilst handling the invalidation of previous hash-location records. * Purpose of this process is to manage the storage pool of an organization and planning the exit strategy. |

# Gannt Chart



# Budget

|  |  |
| --- | --- |
| Description | Estimated Cost (Rs.) |
| Traveling | 5000.00 |
| Printing Materials | 2000.00 |
| Miscellaneous | 1000.00 |
| Total | **8000.00** |

# 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] | U. P. S. Priyanka Shrivastava, "Error Detection and Correction Using Reed Solomon Codes," *International Journal of Advanced Research in Computer Science and Software Engineering ,* vol. Volume 3, no. Issue 8, August 2013 . |
| [3] | T. M. E. K. L. Chandramohan A. Thekkath, "Frangipani: A Scalable Distributed File System," Systems Research Center, Digital Equipment Corporation , 130 Lytton Ave, Palo Alto, CA 94301, 1997. |
| [4] | K. . Shvachko, H. . Kuang, S. . Radia and R. . Chansler, "The Hadoop Distributed File System," , 2010. [Online]. Available: https://cs.uwaterloo.ca/~david/cs848s13/alex-presentation.pdf. [Accessed 6 3 2019]. |
| [5] | S. . Ghemawat, H. . Gobioff and S.-T. . Leung, "The Google File System," *Operating Systems Review,* vol. 37, no. 5, p. , 2003. |
| [6] | 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 . |
| [7] | MooseFS, "moosefs.com," MooseFS, 03 2019. [Online]. Available: https://moosefs.com/. [Accessed 02 03 2019]. |
| [8] | Ceph, "ceph.com," Ceph, 03 2019. [Online]. Available: https://ceph.com/ceph-storage/file-system/. [Accessed 01 03 2019]. |
| [9] | 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]. |
| [10] | Oracle, "oracle.com," Oracle, 03 2019. [Online]. Available: https://oss.oracle.com/projects/ocfs2/. [Accessed 03 2019]. |
| [11] | The OrangeFS Project, "orangefs," The OrangeFS Project, 03 2019. [Online]. Available: http://www.orangefs.org/. [Accessed 03 2019]. |
| [12] | BlueWhale, "bwstor.com," BlueWhale, 03 2019. [Online]. Available: www.bwstor.com.cn/templates/T\_product\_EN/index.aspx?nodeid=150&page=ContentPage&contentid=402. [Accessed 02 03 2019]. |
| [13] | Minio, "minio," Minio, 02 2019. [Online]. Available: https://www.minio.io/. [Accessed 03 2019]. |
| [14] | 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]. |

# Appendices

## Work Breakdown Structure

