HIGH PERFORMANCE DISTRIBUTED BIG FILE CLOUD STORAGE

A Master's Project Presented to
Department of Computer and Information Sciences
SUNY Polytechnic Institute
Utica, New York

In Partial Fulfillment of the Requirements for the
Master of Science Degree

by

Anusha Shakelli
May 2016

HIGH PERFORMANCE DISTRIBUTED BIG FILE CLOUD STORAGE

Except where reference is made to the work of others, the work described here is my own or was done in collaboration with my advisor and/or members of the advisory committee. Further, the content of this work is truthful in regards to my own work and the portrayal of other's work. This work, I claim, does not include proprietary or classified information to the best of my knowledge.

Anusha Shakelli

Anusha Shakelli

HIGH PERFORMANCE DISTRIBUTED BIG FILE CLOUD STORAGE

Master of Science Project in Computer and Information Sciences
Department of Computer Sciences
SUNY Polytechnic Institute

Approved and recommended for acceptance as a project in partial fulfillment of the requirements for the degree of **Master of Science in Computer and Information Sciences**

May 9, 2016
DATE

Sam Sengupta, Ph.D. (Adviser)

Joshua White, Ph. D.

Chen-Fu-Chiang, Ph.D.

HIGH PERFORMANCE DISTRIBUTED BIG FILE CLOUD STORAGE

Except where reference is made to the work of others, the work described here is my own or was done in collaboration with my advisor and/or members of the advisory committee. Further, the content of this work is truthful in regards to my own work and the portrayal of other's work. This work, I claim, does not include proprietary or classified information to the best of my knowledge.

_____

Anusha Shakelli

Abstract

Cloud storage services are growing at a fast rate and are emerging in data storage field. These services are used by people for backing up data, sharing file through social networks like Facebook [3], Zing Me [2]. Users will be able to upload data from computer, mobile or tablet and also download and share them to others. Thus, system load in cloud storage becomes huge. Nowadays, Cloud storage service has become a crucial requirement for many enterprises due to its features like cost saving, performance, security, flexibility.

To design an efficient storage engine for cloud based systems, it is always required to deal with requirements like big file processing, lightweight metadata, deduplication, high scalability. Here we suggest a Big file cloud architecture to handle all problems in big file cloud system. Basically, here we propose to build a scalable distributed data cloud storage that supports big file with size up to several terabytes.

In cloud storage, system load is usually heavy. Data deduplication to reduce wastage of storage space caused by storing same static data from different users. In order to solve the above problems, a common method used in Cloud storages, is by dividing big file into small blocks, storing them on disks and then dealing them using a metadata system [1], [6], [19], [20]. Current cloud storage services have a complex metadata system. Thereby, the space complexity of the metadata System is O(n) and it is not scalable for big file. In this research, a new big file cloud storage architecture and a better solution to reduce the space complexity of metadata is suggested.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The goal of this project is to achieve an efficient storage engine for cloud based systems taking into account the limitations faced by previously existing cloud storage systems like Dropbox [6], Personal cloud storage services [5] like Google drive, One drive, Cloud drive etc., and overcoming them in the design of Big file cloud (BFC) storage. In BFC, certain limitations like metadata complexity, data deduplication faced by the previous cloud based storage systems have been resolved. As an extension, in this project, file compression [11] is done which reduces file sizes and enhances space utilization in storage. The project is extendable to even accommodate file decompression [11].

The project designs a simple metadata [23] in BFC in order to create a high performance cloud storage. In BFC, metadata is independent of the number of chunks for any size of file whether it's a small or a huge file, as the solution will store the first chunk id and the number of chunks produced from the original uploaded file. A chunk is a data segment which is generated from the uploaded file. Depending on the size of uploaded file, it will be split into number of fixed size chunks (except last chunk which may have same or smaller size) if the file size is bigger than the configured size. The contiguous ID range of chunks makes it easy to distribute and scale-out storage system. BFC will also support global deduplication mechanism [21] in order to save storage space and network traffic when many users try to upload same static data. A simple method SHA2 hash function [22] to detect duplicate files in the whole system while uploading is utilized.

1.2 Motivation

Cloud based storage serves several users with storage capacity and each user will reach various terabytes of data. The Cloud Storage providers are in charge of keeping the information accessible and available and the physical environment to be ensured and running. Cloud storage is used by several people in many cases, like backing up data, share file to others through social networks such as, facebook [3], zingme [2]. Uploading data from different devices like computer, mobile phone or tablet and then downloading or sharing to others. Thereby, the system load in cloud storage is huge. So, in order to assure excellent quality service for users, the system has to deal with numerous requirements like efficient storage and management of big files, data deduplication so as to reduce the wastage of storage space which is due to storing the same static data from different users.

In conventional systems, there are many challenges faced by the system in order to manage large number of big files like, scale system for incredible growth of data, distributing data in large number of nodes, load balancing, fault tolerance etc. In order to solve these problems a

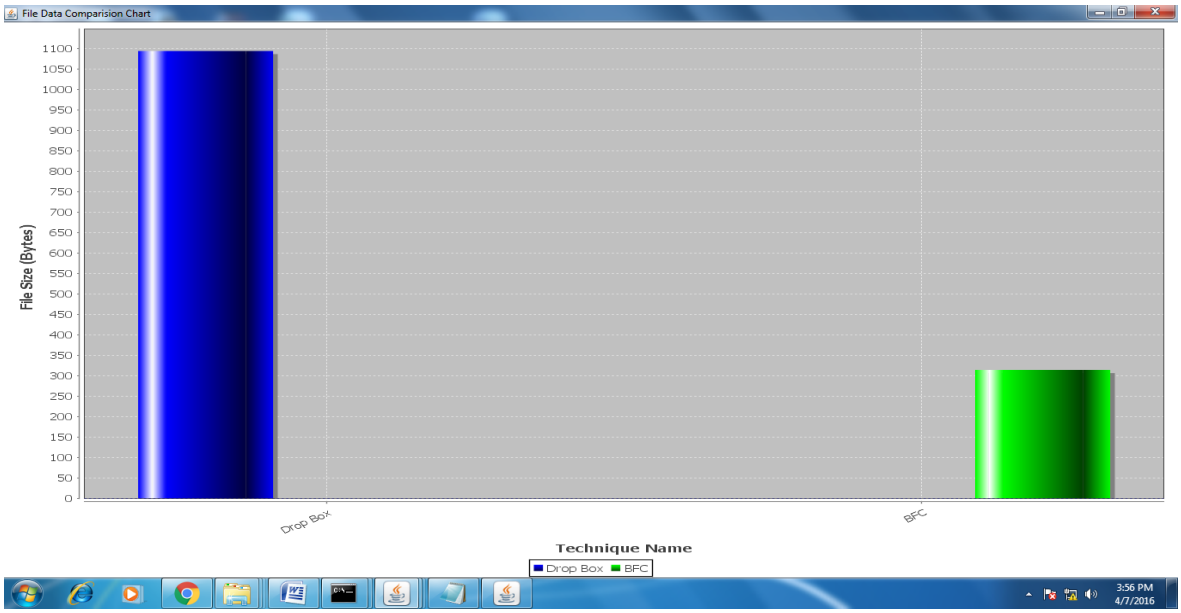
common method used in many cloud storages is dividing big file to smaller chunks, storing chunks on distributed nodes and managing them by using a metadata system. [1], [6], [19], [21].

In one such cloud storage system Dropbox [1], metadata size will proportionately increase with the original file size, leading to difficulty when the file size is big. There are significant challenges like efficient storage of chunks and metadata design faced by cloud storage providers. Thereby, the current cloud storage services have a complicated metadata system that, the file size of every file is directly proportional to the size of metadata at least. The space complexity of this metadata system is O(n) in Dropbox [1], HDFS [21]. Thereby, the space complexity of these meta-data system is not scalable for huge files. In this research, a new big-file cloud storage architecture and improved solution to reduce the space complexity of metadata is suggested. Here, in BFC we came up with a solution where the metadata size is independent of the number of chunks regardless of any file size: small or big. In case of extension of my project, file compression is used, where we first compress a huge file and then store on cloud server.

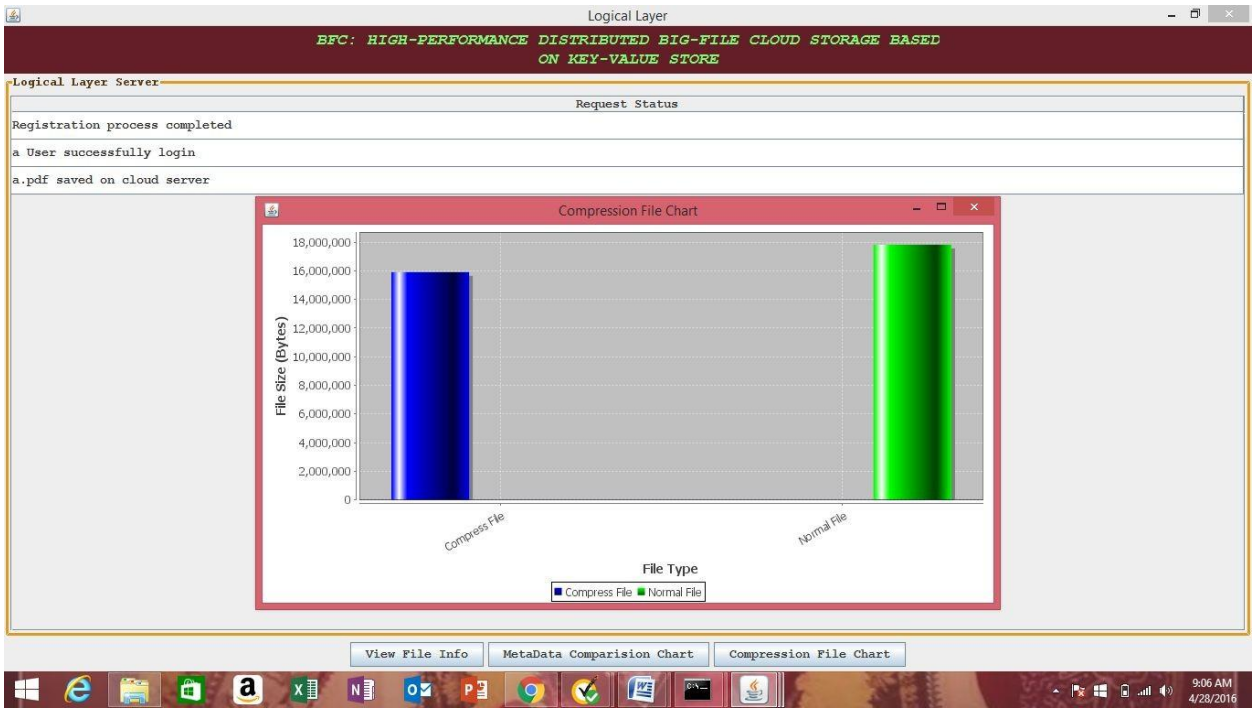
1.3 What are Issues

The quality of performing extraordinarily well, better, faster and more efficient than others is high performance. If we try to measure the high performance in processors we will see which processor works faster than the other. Based on which processor works faster, that one will have high performance. High performance will mainly depend on the response time and query execution time.

High performance: Here, in this project in order to perform data deduplication we are using metadata of the files which are stored on the cloud. BFC supports lightweight metadata system wherein, the size of metadata in BFC is smaller than the existed cloud storage systems like Dropbox [6]. Thereby, it will be easier for us to check data duplication at the server side so; it will result in reducing the execution and response time of the query. The metadata in BFC consists of first chunk and last chunk that's it whereas, in existed cloud storage systems like Dropbox, the metadata consists of series of all chunks i.e., from first chunk to last chunk. This can be seen in the metadata size comparison chart between existing and proposed system in the figure 1 below. Thereby, the limitations of the previously existing cloud storage systems: Dropbox [6], Google Drive [5] like metadata complexity, data deduplication are overcome by the proposed system BFC in this project. Additionally, file compression [11] discussed in 1.6 below is also implemented as project extension which will result in reducing storage requirements. The compressed file and normal file comparison is shown below in figure 2. Also, the transmission of compressed data over a medium will be resulted in an increase in the rate of information transfer. The project is extendable to accommodate file decompression[11]. Therefore, we can conclude the proposed system as high performance.



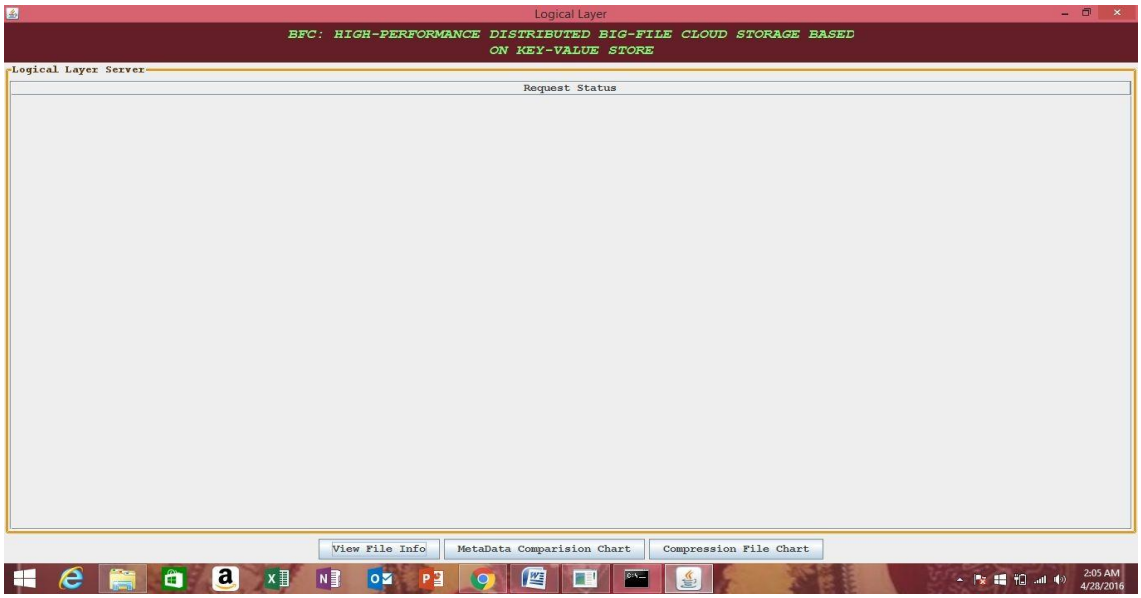
Metadatasizecomparison



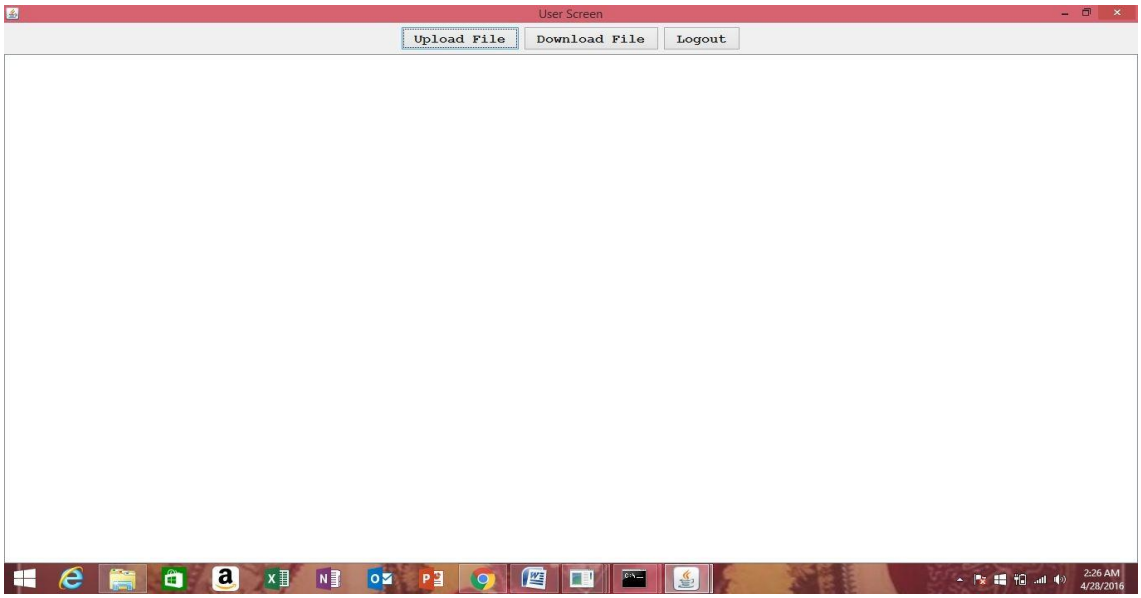
File Compression comparison chart

Distributed system: A system is said to be a distributed system if it has ability to share resources among multiple systems which can also be in different locations in order to serve millions of users. The project BFC is a distributed application [18] with a client-server architecture explained in the Distributed system section 1.4.4. Here, in this project a server application (Logical layer) and a client application (Application layer) are developed, we will be able to run the server

application in one system and client application in many systems which are connected through Local Area Network (LAN).



Server application(Logical layer)



Client application (Application Layer)

Scalability: A system is said to be scalable when it has the potential to expand in order to accommodate the growing amount of work. The scalability of the system can be measured through an algorithm, design, program. A system is scalable when it is efficient and practical when applied to situations like large input data set, a large number of outputs or users, in order to conclude whether a system is efficient and practical when applied to situations like large input

data set, a large number of outputs or users, or a large number of participating nodes in the case of a distributed system.
In BFC, file-id and chunk-id are integer keys that can be automatically incremented. A simple hash function hash(key) = key is utilized for consistent hashing [27]. In this case, it will be easier in order to scale-out the system.

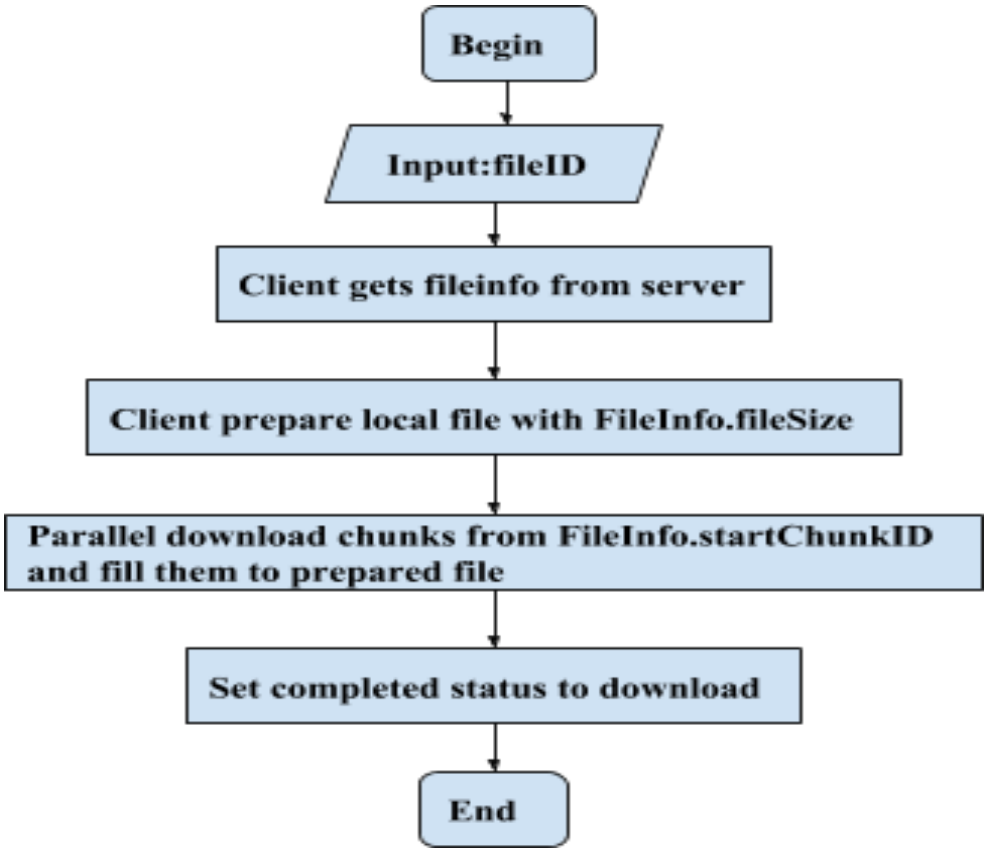


Figure 1: Downloading Algorithm of application [8]

Chunk optimum size: A chunk is a data segment generated from a file uploaded by the user. If the file size is bigger than the configured size, it will be split into a list of chunks. All the chunks which are generated from a file, except the last chunk will have the same size (except the last chunk which may have an equal or small size). After that, the ID generator will generate id for the file and the first chunk with auto-increment mechanism. Next chunk in the list of chunks will be assigned an ID gradually in an increasing manner until the last chunk.

```
public long getChunkSize(File file){
        long length = file.length();
        tot_blocks=0;
        long size=0;
        if(length>= 1000){
```

```
            size = length/10;
            tot_blocks = 10;
   }
if(length < 1000 && length > 500){
            size = length/5;
   }
if(length < 500 && length > 1){
            size  = length/3;
            tot_blocks = 3;
}

return size;
}
```

1.4 BFC Architecture

BFC Architecture [8] comprises of four layers namely: Application layer, Storage Logical layer and Object Store layer. Each layer comprises of various coordinated components.

Application layer consists of indigenous software on desktop systems, mobile devices and web-interface, which grants user to upload, download and share their own files. This layer utilizes API which is from Storage Logical layer and applies various algorithms for the purpose of downloading and uploading process.

Storage Logical layer contains numerous queuing services and worker services, ID-Generator services and all logical API for Cloud Storage System. This layer will implement business logic part in BFC. The most essential components of this layer is upload and download service. This layer will provide a CloudApps [28] service which will serve users requests. The Storage logical layer will stores as well as recovers information from Object Store Layer.

Object Store Layer is the most essential layer which stores and caches objects. This layer maintains data of all objects in the system which will include client data, metadata in particular. It consists of many distributed backend services. Object Store Layer has two important services namely, FileInfoService and ChunkStoreService. Information of files is stored in FileInfoService. ChunkStoreService will store data chunks which are built by splitting the original files that client has uploaded.

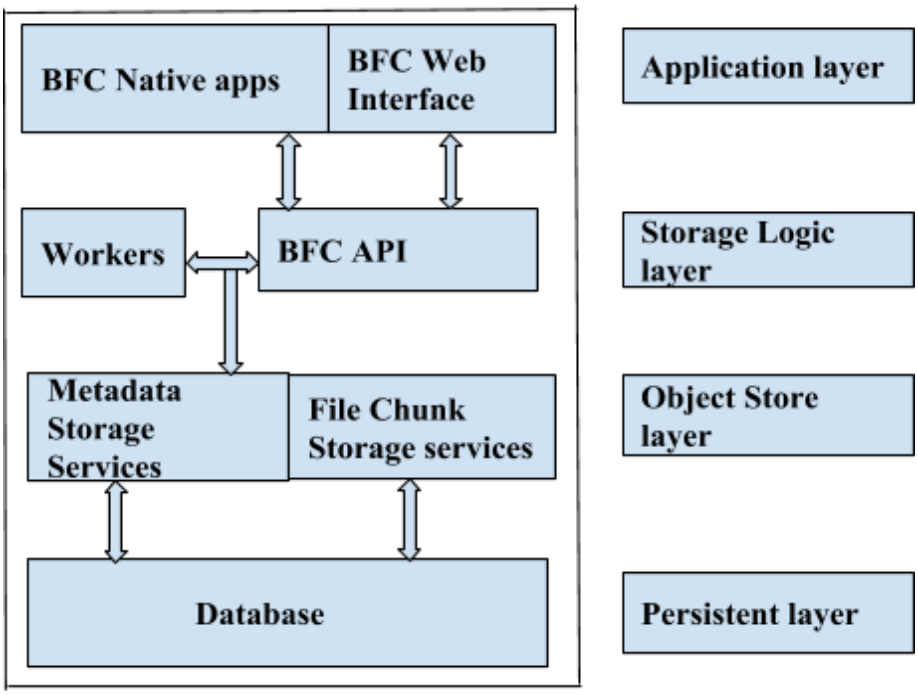


Figure 2: BFC Architecture

In BFC system, we achieved few enhancements to make low complicated metadata. Metadata represents a file and how it is composed as a list of small chunks. The fundamental element in the BFC cloud storage system is chunk. Here, the original large file which is uploaded by the client is split into a list of chunks. This brings a great deal of advantages. Above all, it is simple to store, distribute and replicate chunks, efficient storage of small chunks, also uploading and downloading file parallel and resumable. It is hard to achieve the above benefits with a huge file in local file system.

1.4.1 Metadata

Metadata [23] is the data which describes about original data. It consists of a series of elements, and each element has information such as, chunk size, hash value of chunk. The length of the series will be equivalent to the number of chunks from file, making it difficult when the file size is large.

BFC suggested a solution in which the size of metadata will not be dependent on the number of chunks regardless of file size(very small file or big file). Here, the solution suggests to store the id of the first chunk, and the number of chunks which are generated by splitting original file. Since the id of chunk is progressively assigned from the first chunk, we can undoubtedly calculate the ith chunk id by the following formula:
Chunkid[i] = fileInfo.startChunkID + i.

Metadata is primarily represented in FileInfo structure which comprise of following fields: *fileName* (name of the file); *filedId* (identification of file) *Ref FileID* (Id of file that have earlier existed in the system) *refFileId* is accurate if it is greater than zero. startchunkID will represent the first chunk of file, the next chunk will have id as startChunkID + 1, *numchunk* represents number of chunks, filesize is the size of file in bytes. Status shows the status of the file, it will

have one in four states namely, *EUploading* file - when chunks are uploading to the server, *ECompleted* file - when all chunks are uploaded however, it is not checked as consistent , *ECorrupted* file - when all chunks are uploaded to server however, after checking it is not consistent. *EGoodCompleted* - when all the chunks are uploaded to server and obtained good result after consistent checking. The below model is utilized in interpreting into programming code.

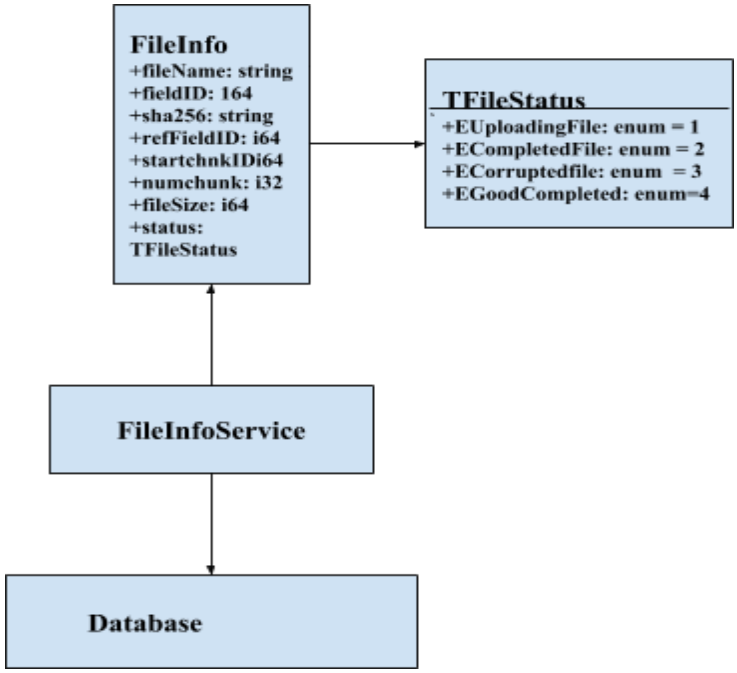


Figure 3: Metadata Structure

Thereby, the metadata of a file (size of Fileinfo object) will be almost the same for all files in the system independent of size of the file huge or small. (The one and only dissimilarity metadata of files is the length of filename). By utilizing this approach, we have created a lightweight metadata while designing a big file storage system.

1.4.2 Data deduplication

Cloud storage faces a complicated challenge in overcoming data duplication by eliminating identical data. In BFC data deduplication is utilized. Data deduplication can work on client or server side [17]. In BFC, we have implemented on the server side. Data deduplication is one of the crucial mechanism for minimizing identical copies of similar data. In order to enhance effective usage of storage space, indistinguishable data are found then only one copy of data is saved and is interchanged with other duplicates with reference that addresses first duplicate. In this way, we can reduce identical file being stored in cloud by different users thereby, reducing data duplication.

The method used to detect duplicate data is SHA2 [22] hash function while uploading. Let us say, that the file selected by the client to be uploaded is X. Then the SHA value is computed. Afterwards, the necessary information of file is created, which includes file name, file size, SHA value. If the data deduplication is implemented at server side then, the SHA value will be utilized

to find related fileID. Suppose, if at all there is a fieldID with the SHA value found in the system we call it as Y. This implies that file X and file Y are absolutely same. Therefore, we commonly refer file X to file Y and assign the id of file Y to *refFileID* field of file: a field which is used to describe that a file is referenced to another file. Now, the necessary information will be sent to the client and the upload will be finished. Thereby, the storage space is not wasted and is effectively utilized. There is another case, where there is no fileID related to SHA value of file A or suppose data deduplication not enabled, then new fields for the file like id of file, id of first chunk, number of chunks computed from file size and chunk size. This information will be utilized by the client to upload file content to the server. After uploading all the chunks, then the SHA value computed by client is compared with the SHA value in the server. Then, the status of property FileInfo is set to EGoodCompleted if everything goes well.

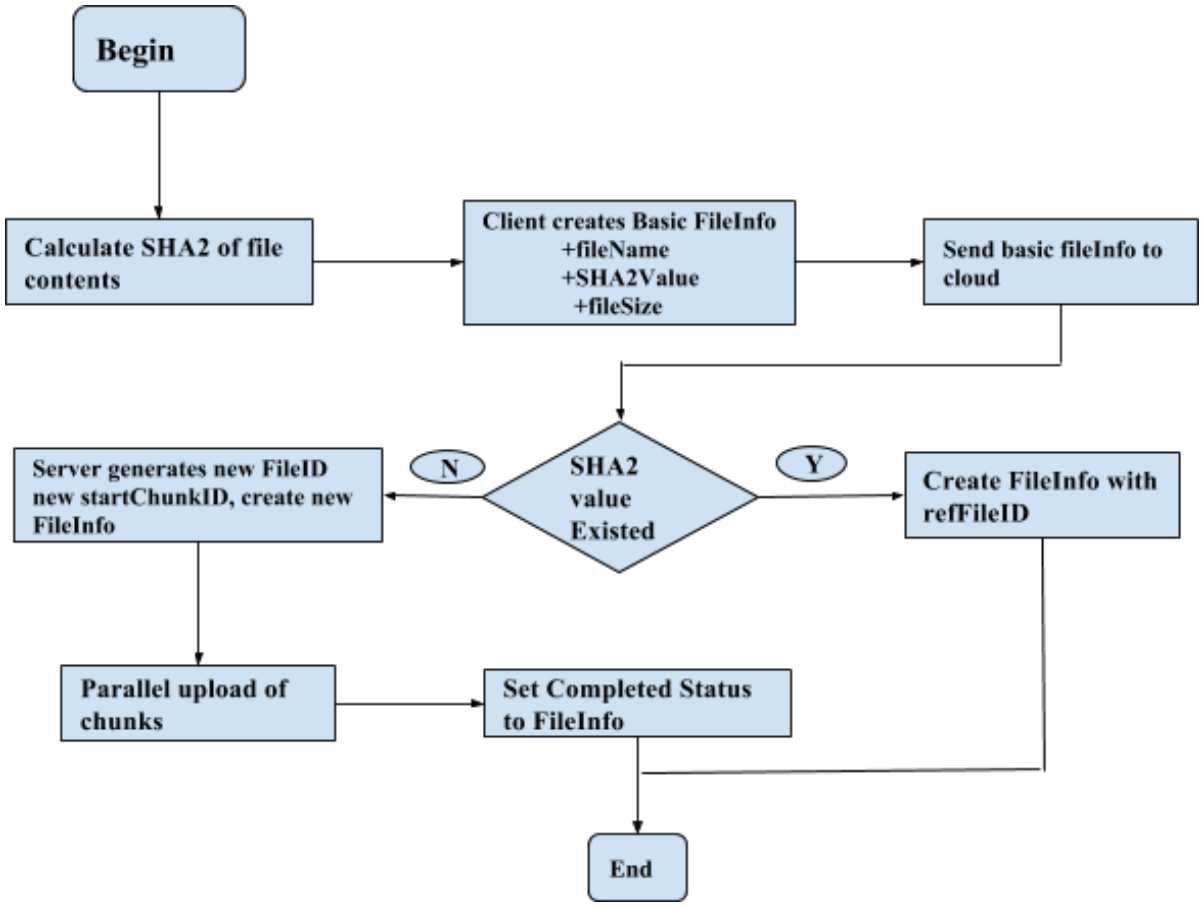


Figure 4: Uploading and Deduplication Algorithm

1.4.3 Data Security In BFC

Data Security in BFC in this project is provided by using AES algorithm.
Encryption is the method of converting plaintext to cipher-text through applying mathematical transformations. These transformations are known as encryption algorithms and they require an encryption key. Decryption is a reverse process of obtaining the original data from the cipher-text using a decryption key.

AES [4] depends on a design principle called as Substitution permutation network. It is fast in software as well as hardware. It has a fixed block size of around 128 bits and a key size of 128, 192, or 256 bits. AES will operate on a 4*4 matrix of bytes known as state. Most of the AES computations are performed in a special finite field. AES cipher [4] is indicated as various repetitions of transformation rounds that will transform the input data into the final output of ciphertext. Each round involves a few processing steps. Including the one that depends on encryption key. A set of reverse rounds are applied in order to transform ciphertext to the original text by using the same encryption.

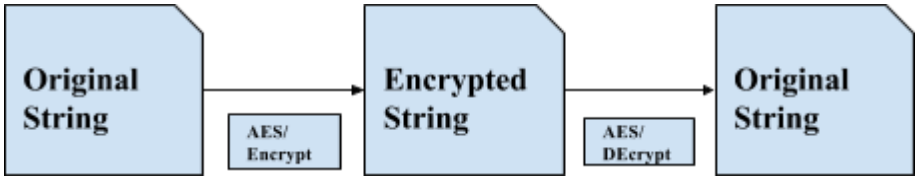


Figure 5: Block diagram for Encryption and Decryption

Algorithm description:

1. Key Expansion - Round keys are derived from the cipher key utilizing Rijndael's key schedule (it expands into various separate round keys).

2. Initial Round
   a. AddRoundKey - every byte of the state is combined with the round key using bitwise xor.

3. Rounds
   a. SubBytes - a non-linear substitution step where every byte is replaced with the other depending on a lookup table.
   b. ShiftRows - a transposition step where each row of the state is shifted cyclically a certain number of steps.
   c. MixColumns - it's a mixing operation which operates on the columns of the state, by combining four bytes in every column.
   d. AddRoundKey

4. Final Round (no MixColumns)
   a. SubBytes
   b. ShiftRows
   c. AddRoundKey

From the above, we can say that BFC architecture is a high performance system as it has overcome major limitations like complex metadata system, data deduplication faced by previous cloud storage systems.

1.4.4 Distributed System

BFC supports distributed systems. A Distributed system[18] is a system that grants several users to access the same file and it also allows essential operations like create, delete, write, modify. Each file is split into multiple chunks, which will be stored on different machines thereby, facilitating applications in executing parallely. Many enterprises will be able to store and access their data which is stored remotely in the same way as they do locally.

In BFC, client-server architecture is built which grants file sharing between the remote machines on a network in the same way when they are located nearby. In BFC metadata is stored and distributed. File Info service and Chunk Storage service which we have seen in BFC architecture to distribute data uses consistent-hashing [27]. BFC can be designed so that it can operate on network environment like Local Area Network (LAN), Internet. Here, data can be distributed to server so that it can serve various clients.

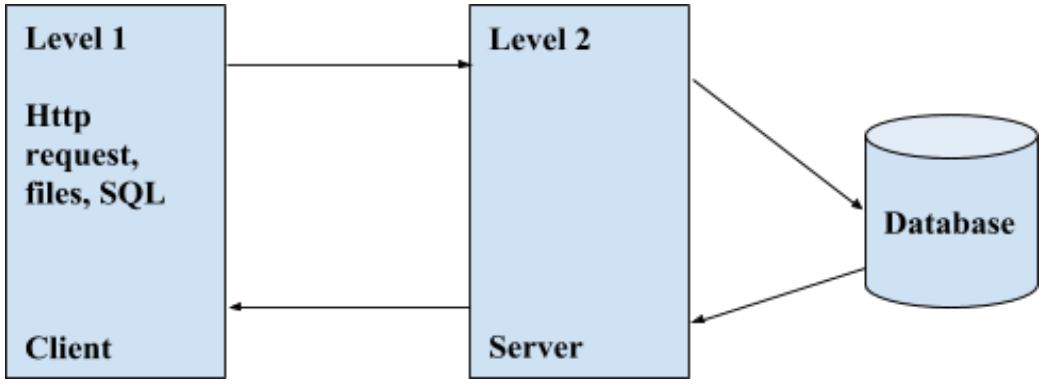


Figure 6: Client-Server architecture

In BFC, Client - server architecture is implemented as a standalone that is, client and server are in the same system. Client is implemented at the application layer and server at the logical layer. MySQl Database [31] is used in order to store data and data can be retrieved whenever needed. This standalone application can also be implemented as a network where different systems can be connected through LAN (server is located in other system and several clients can access the server). When any client sends requests we can locate the IP address of the particular client in the server (logical layer).

1.5 Comparison between existing systems and BFC

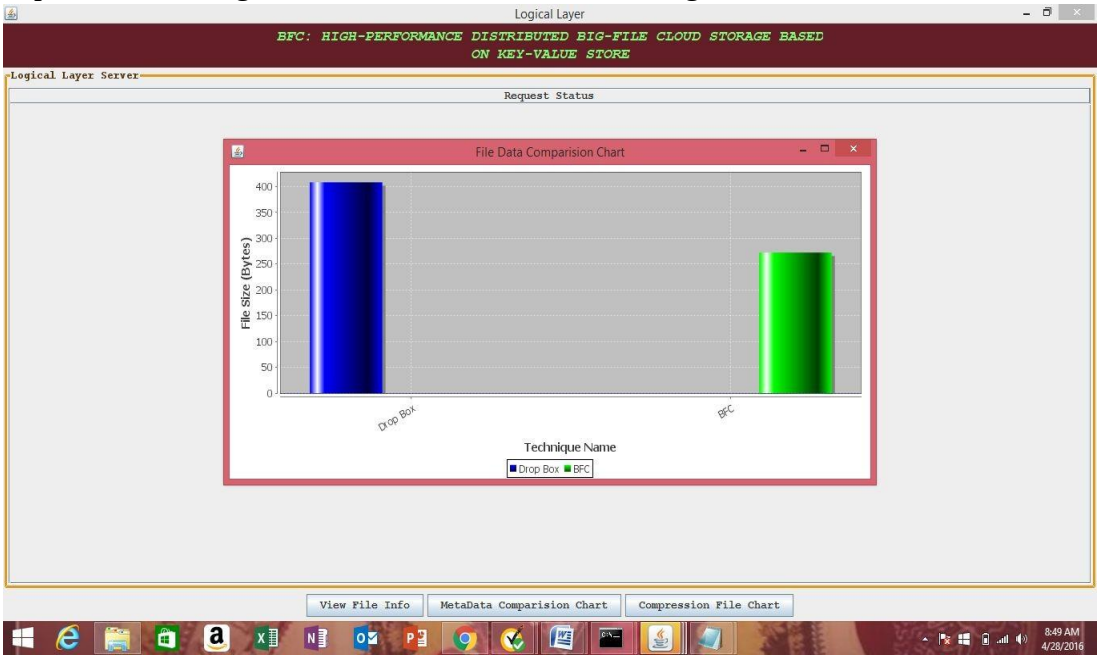
The limitations faced by existing systems like metadata complexity, data duplication, scalability etc are overcome in this project and they are compared below. To overcome the issue of metadata complexity, BFC with metadata of fixed size for each file is implemented. Data deduplication mechanism is also supported in BFC.

1.5.1 Metadata comparison

Dropbox [1] is cloud based storage system that will grant users for storage of documents, videos and other files. Dropbox is accessible on Web interface, and various types of client softwares on desktop and mobile operating systems. The user interface aids synchronization and sharing between different devices with personal storage. Dropbox was mainly written in Python.

The necessary object in the Dropbox is a chunk of 4MB data. If a file is huge than the configured size it will be split into multiple chunks. Each chunk is an independent element and is determined by a SHA256 value and will be stored in Amazon Simple Storage(S3). In dropbox, metadata consists of a series of SHA 256 of its chunks [1], [6]. Thereby, it's size is proportional to the file size. In case of huge files, metadata is big as it is comprised of a series of all the chunks (from first to last).

In order to overcome the issue discussed above, BFC with metadata of fixed size for each file is implemented. Here, instead of storing all the chunks metadata stores only the first chunk and the last chunk thereby reducing space leading to effective utilization of storage space. This also helps in minimizing the amount of data for transferring metadata between clients and servers.



Metadata Comparison

We have also seen metadata comparison in Dropbox and BFC which can be done at the Logical layer in our implementation which can be seen above.

1.5.2 Deduplication Comparison

In previously developed cloud storage systems, Dropbox [1], Google Drive [5], One Drive do not provide deduplication mechanism. In BFC data deduplication mechanism is supported. Here, let us say first user uploads a file into the cloud. If another user tries to upload the same file it is not stored again in the cloud as the same file is already available at server side. It will not upload the same file the next time instead, it will tag to the already existing file. Thereby, reducing the space complexity.

1.6 Compression(Project extension)

In Cloud storage, various files may contain redundant data or information that adds little to the data. This will lead to huge amount of data being transmitted between client and the server. One such solution which can overcome this problem is to use additional storage devices in order to enhance the existing communication. However, this will result in increased operation costs for the organization.

One best solution in order to overcome the problem of increased data storage and information transfer is by using efficient code. This can be implemented by using Java applications which provides java.util.zip package [11]. We have existing applications for data compression such as WinZIP [24], gzip [25] and jar [26] however we cannot use them because they are standalone.

Figure 7: Block diagram of data compression

By implementing data compression, there is reduction in storage in the cloud thereby, enhancing the utilization of storage space. We can also say that the rate of data transfer increases as compressed data is transferred between client and server systems.

Data compression is performed by using java.util.zip package provided by Java. Zip oriented data compression can be performed by using java.util.zip package [11]. This package consists of classes to create, read and modify files which bear gzip and zip formats. On the whole, the package consists of one interface, 14 classes, and 2 exception classes. The java.util.zip package consists of classes for compression and decompression. This package consists of a ZipInputStream [11] class in order to read ZIP files. It is created like any other input stream.We

shall be using the following four classes from the java.util.zip package in order to work with the Zip file format:

- ZipEntry
- ZipInputStream
- ZipOutputStream
- ZipFile

A ZipEntry object depicts an entry in an archive in a ZIP file format. A zip entry can be compressed or uncompressed.
ZipEntry class has methods to set and get information regarding an entry in a ZIP file.
ZipInputStream will be able to read data from a ZIP file for each entry.
ZipOutputStream will be able to write data to a ZIP file for each entry.
ZipFile is a utility class which is used to read entries from a ZIP file.

The ZipEntry class explains about a compressed file which is stored in a ZIP file. The different methods in this class can be utilized to set and get pieces of information regarding the entry.
We will be able to implement data compression on the existing hardware by using software or by the use of special hardware devices which comprise of compression techniques. The code written using this package is discussed in the code for file compression in following chapter 5.

Compression is implemented as the extension of my project. My project is extendable to accommodate decompression.

1.7 The Scientific Contributions of the project

1. The project will overcome the difficulties faced by the previous cloud storage systems[6], [20], namely metadata complexity. This is overcome by implementing a lightweight metadata in BFC.
2. The second contribution is to develop a Big file cloud storage which provides data deduplication[10].
3. The third contribution extension of the project, is incorporating the concept of Data compression[11] of files before their storage in the cloud.
4. The fourth contribution is that the data security is implemented through performing encryption and decryption [4].
5. The fifth contribution is front end user interface is implemented at the Application layer, server (back end) is implemented at the logical layer. Also, MySQL Database [31] is designed for storage of information. Thereby, High performance BFC is implemented.
6. Finally, we also see the metadata comparison in BFC and Dropbox [6](previous cloud storage), deduplication, compression comparison in the results. Thereby, enhancement of BFC is observed.

1.8 Project Structure

Chapter 2 deals with aims and objectives of the project. They form the objectives to design and implement our project.

Chapter 3 deals with the analysis of existing cloud systems and their limitations, summary of limitations overcome by the project.

Chapter 4 deals with the analysis and design of the project and the requirements for the project design and implementation, the UML diagrams which are interpreted into programming code.

Chapter 5 deals with implementation of the project, Java technologies used in writing code for project implementation and execution.

Chapter 6 deals with the results of effectiveness of the system: Screenshots obtained when the project is executed, test cases. From screenshots, comparison is shown between the existing systems and the project which indicate that the limitations have been overcome by the high performance distributed BFC.

Chapter 7 deals with the security provided in BFC which is implemented in the project, conclusion and future scope of cloud based storage systems.

The project concludes with a small summary of the findings and a discussion, and the conclusion drawn from the project will be placed at the end of this report.

Chapter 2

Aims and Objectives of the Project

While, it has already been specified in the abstract of the project:

"To design an efficient storage engine for cloud based systems, it is always required to deal with requirements like big file processing, lightweight metadata, deduplication, high scalability. Here, in this paper, we suggest a Big file cloud architecture to handle all problems in big file cloud system. Basically, here we propose to build a scalable distributed data cloud storage that supports big file with size up to several terabytes.
In cloud storage, system load is usually heavy. Data deduplication[10] to reduce wastage of storage space caused by storing same static data from different users. In order to solve the above problems, a common method used in Cloud storages, is by dividing big file into small blocks, storing them on disks and then dealing them using a metadata system. Current cloud storage services have a complex metadata system[6],[5]. Thereby, the space complexity of the metadata System is O(n) and it is not scalable for big file. In this research, a new big file cloud storage architecture and a better solution to reduce the space complexity of metadata is suggested. In order to reduce space complexity, while we try to store a huge file in cloud we are first compressing and then storing on the cloud server."

From the above, the primary objective is to be able to design BFC with its calculations and construction modeling in order to solve most of the issues. It is completed by suggesting settled size metadata outline which backings rapid and simultaneous, dispersed record I/O, a few calculations for resumable transfer, download and basic information deduplication for static information. The outcomes can be utilized for building flexible appropriated information distributed cloud storage that supports huge document with size up to a few terabytes.

The secondary objective is to implement the above developed design, test and compare with the previously existing models. This is to ensure that the issues faced by the existing models are solved in order to reach the demands of rapidly developing current cloud storage in the information technology field. The project is extended by developing and implementing file compression before cloud storage.

2.1 Aim

The aim of the project is to create a high performance big file cloud storage system which will overcome all the limitations faced by cloud storage systems and also to see that the other issues which arise from the above limitations should also be resolved, and the people should enjoy the benefits of the cloud storage systems.

2.2 Summary

The results of the project that have been mentioned are being accomplished by working on the primary and secondary objectives. The design will be tested against existing systems so that there can be comparison made to either or not come to a conclusion that the issues have been resolved based on the analysis that would be done. The scope of this project is vast as it covers many areas like security, networking, big data and many more which will enable further study to be done in this area of cloud storage systems. Cloud storage systems can have many benefits like low cost, reliability, flexibility, reduction in space complexity and network traffic etc.

Chapter 3

Literature Survey

3.1 Introduction

In the past, there are many limitations [6],[5]which need to be overcome. Presently in information technology an efficient cloud storage system can be developed, by analyzing the design of already existing systems and evaluating their impact of design choices on performance.

3.2 Dropbox

Nowadays, personal cloud storage services are gaining importance. It is believed that cloud storage generate huge amount of Internet traffic as there is increase in the number of providers to enter the market and an increasing offer of cheap storage space. The understanding of architecture, performance of such systems and their workload is essential in order to design efficient cloud storage systems and predict their impact on the network. Here, we will present a characterization of Dropbox[6], which is a leading solution in personal cloud storage in our datasets. Dropbox is the most widely used cloud storage system which accounts for a volume equivalent to one third of the youtube traffic at campus networks. If we analyze the usage of Dropbox on the internet, it shows that there is an increasing interest on cloud-based storage systems. Major companies like Google, Apple, Microsoft are offering this service. Drop box service performance is highly impacted by the distance between the clients and the data-centers, which are located in the U.S. In the client protocol the usage of per-chunk acknowledgements combined with small chunk sizes will deeply limit the effective throughput of the service. We have identified two possible improvements to the protocol in this paper: a) the usage of a chunk building scheme. b) introduction of delayed acknowledgements. The recent deployment of bundling mechanism has improved the system performance dramatically. We can expect that the overall performance can be improved by the deployment of other data-centers in different locations.

3.3 The Google File System

Google File system is a distributed file system for large distributed data-intensive applications. It provides fault tolerance when running on economical commodity hardware, and it delivers high overall performance to a huge number of clients. While sharing most of the same goals as earlier distributed file systems, our design has been motivated by consideration of our application workloads and technological environment, both present and predicted that reflect a marked deviation from few earlier file system expectations. This led to reexamine conventional choices

investigate thoroughly different design points. The file system has strongly met our storage needs. It is mostly utilized within google as the storage platform for the generation and processing of data used by our service besides research and development efforts that require huge data sets. The largest cluster at present, will provide hundreds of terabytes of storage beyond thousands of disks over a thousand machines, and it is simultaneously accessed by hundreds of clients.

The Google File System[7] signifies the qualities necessary in order to support large-scale data processing on hardware. While some design conclusions are particular to our unique setting, many may be applicable to data processing tasks of a same magnitude and cost awareness. The conventional file system assumptions are reexamined here in light of our present and predicted application workloads and technological environment. Our examinations have led to thoroughly different points in the design space. In order to improve the overall system, component failures are treated as standard rather than exception optimize for large files, then enhance the standard file system interface. Fault tolerance is provided by our system by constantly monitoring, reproducing critical data, rapid and automatic recovery. Chunk replication will allow us in order to tolerate chunkserver failures. The commonness of these failures encouraged a novel outline repair mechanism that will regularly repair the damage and will repay for lost replicas soon.

3.4 Personal Cloud Storage Services

Personal Cloud storage services are data-intensive applications creating huge amount of Internet traffic. Let us analyze 5 of these services[5]. Dropbox implements most of the checked capabilities, and its sophisticated client clearly boosts performance, although some protocol tweaks seem possible to reduce network overhead. However, duplication of data, metadata complexity become major problems giving rise to wastage of storage space and increasing network overhead. In Cloud Drive, wastage of bandwidth has a magnitude which is higher than other services, lack of client performance results in bottlenecks. SkyDrive also shows some limitations in performance like network latency. In One Drive we can see limitation of data duplication. Google Drive follows a different approach resulting in a mixed picture: it enjoys the benefits of using Google's capillary infrastructure and private backbone, which reduce network latency and speed up the system. However, protocols and client features limit performance, especially when multiple files are considered.

3.5 Overall development by the Project

Above, we have seen what has been done in existing systems. Here, we discuss about the limitations of previous systems and how the contribution in this project relates to overall development.

Analysis of the usage of existing Cloud storage systems on the internet shows an increasing interest on cloud based storage systems. For instance, one of the cloud storage systems Dropbox is most widely used which accounts for a volume equivalent to one third of the youtube traffic at campus networks on some days. The existing Cloud storage systems have tried to implement distributed cloud storage and improve their performance. However, there are certain limitations

like metadata complexity, data duplication which in turn lead to other issues discussed below. In our project implementation, we will look forward to overcome the issues faced by the existing cloud storage systems.

The above discussed limitations like metadata complexity, data duplication are overcome in our project implementation by designing a high performance distributed big file cloud based storage system. It provides lightweight metadata system and data deduplication and in turn an extension, data compression is also implemented thereby, achieving effective utilization of storage space. Data security through encryption and decryption is also provided.

Chapter 4

Model Analysis and Design

4.1 Introduction

The Software Development Life Cycle (SDLC)[11], in information system, software engineering and systems engineering is the process of creating systems, models and methodologies that people will use in order to develop these systems. The SDLC concept in software engineering supports various kinds of software development methodologies. These methodologies in turn would form the framework for planning the creation of an information system in the software development process.

4.2 Analysis

4.2.1 Existing System

Cloud storage is used by people for daily demands, for instance backing up data, sending file to friends through social networks. Users will also possibly upload data from different types of devices and they can download or share with others. In Cloud storage system load is usually very heavy. Thereby, in order to assure excellent quality of service for users, the system will have to deal with certain problems and requirements.

Disadvantages

- Efficiently storing, retrieving and managing big files in the system.
- Data duplication in order to reduce wastage of storage space which is due to storing same static data from different users.
- Parallel and resumable upload and download.

4.2.2 Proposed system

A common method which is used for solving these problems is by dividing big file to multiple smaller chunks, storing them on disks and then managing them by using a meta data system. Cloud storage providers have to face significant problems like, storing chunks and meta-data effectively and designing a light weight meta data. After a long time, current cloud storage services have a complex meta data system; somewhat the size of metadata will be linear to the file size for every file. Thereby, the space complexity of these meta data system is not scalable for big file. In this research, we implement a big file cloud storage architecture and also a superior solution to reduce the space complexity of meta-data.

Advantages

- A lightweight metadata design for big file. Every file has approximately the same size of metadata.
- A logical contiguous chunk-id of chunk collection of files that makes it manageable in order to distribute data and scale out the storage system.
- File compression which overcomes the problem of increased data storage and information transfer.

4.2.3 Software Requirement Specification

External Interface Requirements:

User Interface: The User interface is a user friendly Java Graphical User Interface.

Hardware Interface: The interaction between the user and the console is achieved through Java capabilities.

Hardware Requirements:

- Processor : Intel(R) Core(TM) i5
- Speed : 1.1 GHz
- RAM : 256 MB(min)
- Hard Disk : 20GB

Software Requirements:

- Operating System : Windows XP
- Programming Language : JAVA
- Front End : AWT, Swing
- Back End : MySQL

4.3 Design

The Unified modeling language allows the software engineer so as to express an analysis model using the modeling notation which is governed by a set of syntactic semantic and practical rules.

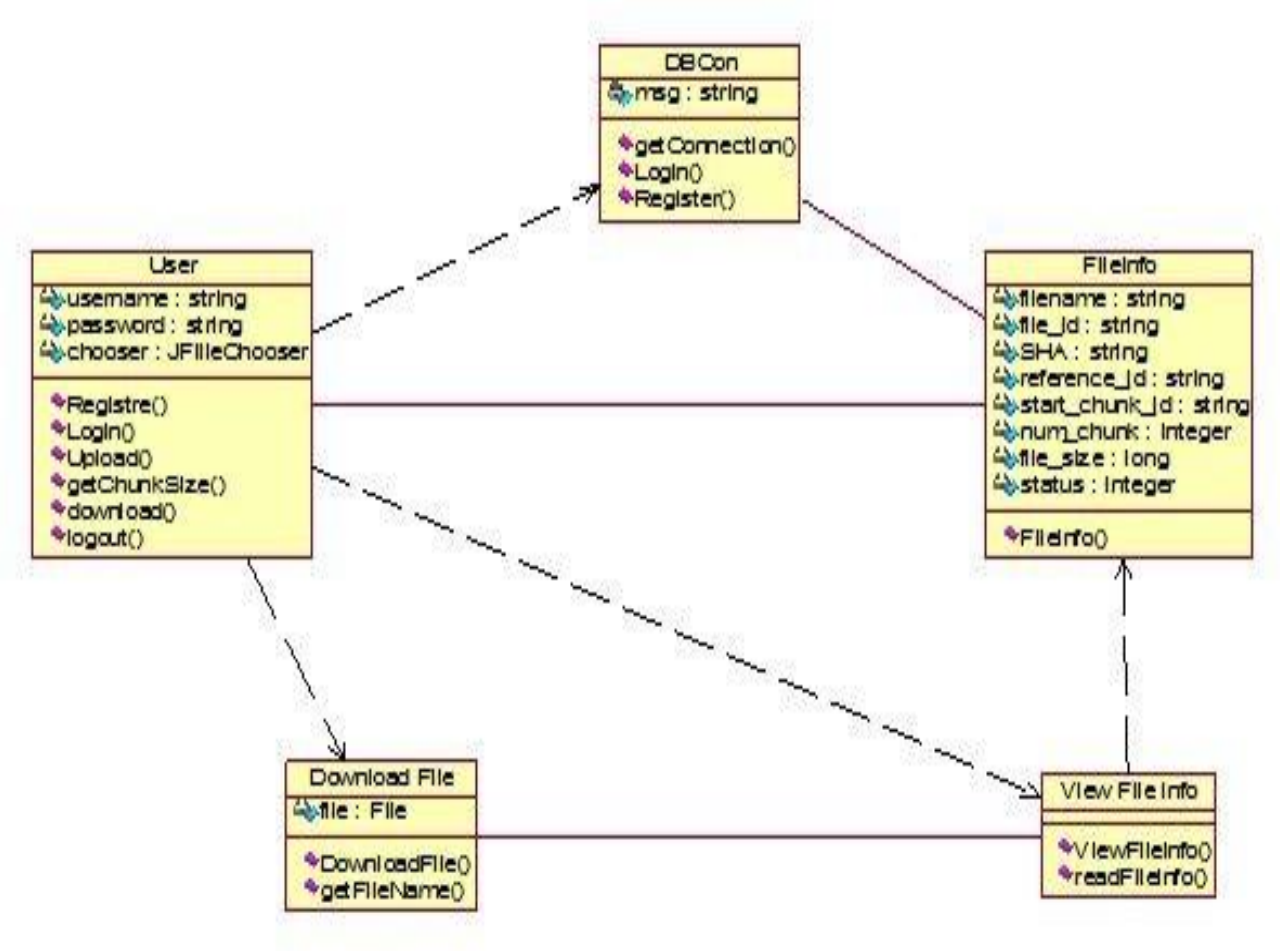
4.4 UML diagrams

4.4.1 Class diagram

The class diagram is the vital building block of object oriented modeling. It can be used both for general theoretical modeling of the systematic of the application, and also for a itemized modeling utilized for making interpretation of the models into programming code. In the outline, classes with three sections are portrayed with boxes which comprise of three parts:

a) The upper part represents the name of the class.

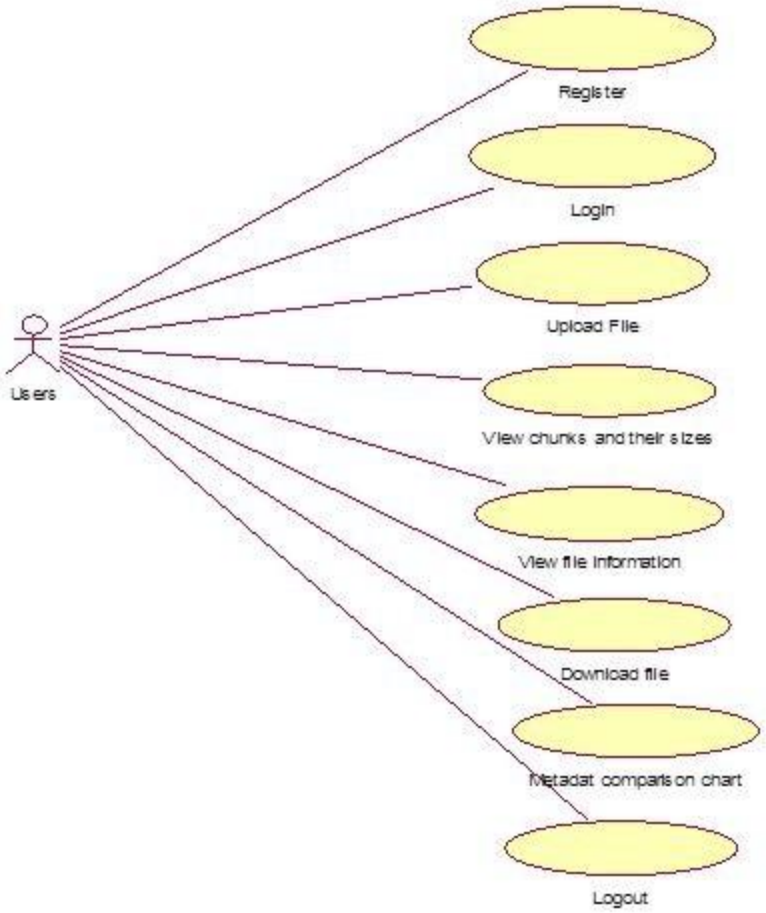
b) The middle part includes the attributes of the class.
c) The bottom part will give the methods or operations the class can initiate.



Class Diagram

4.4.2 Use case diagram

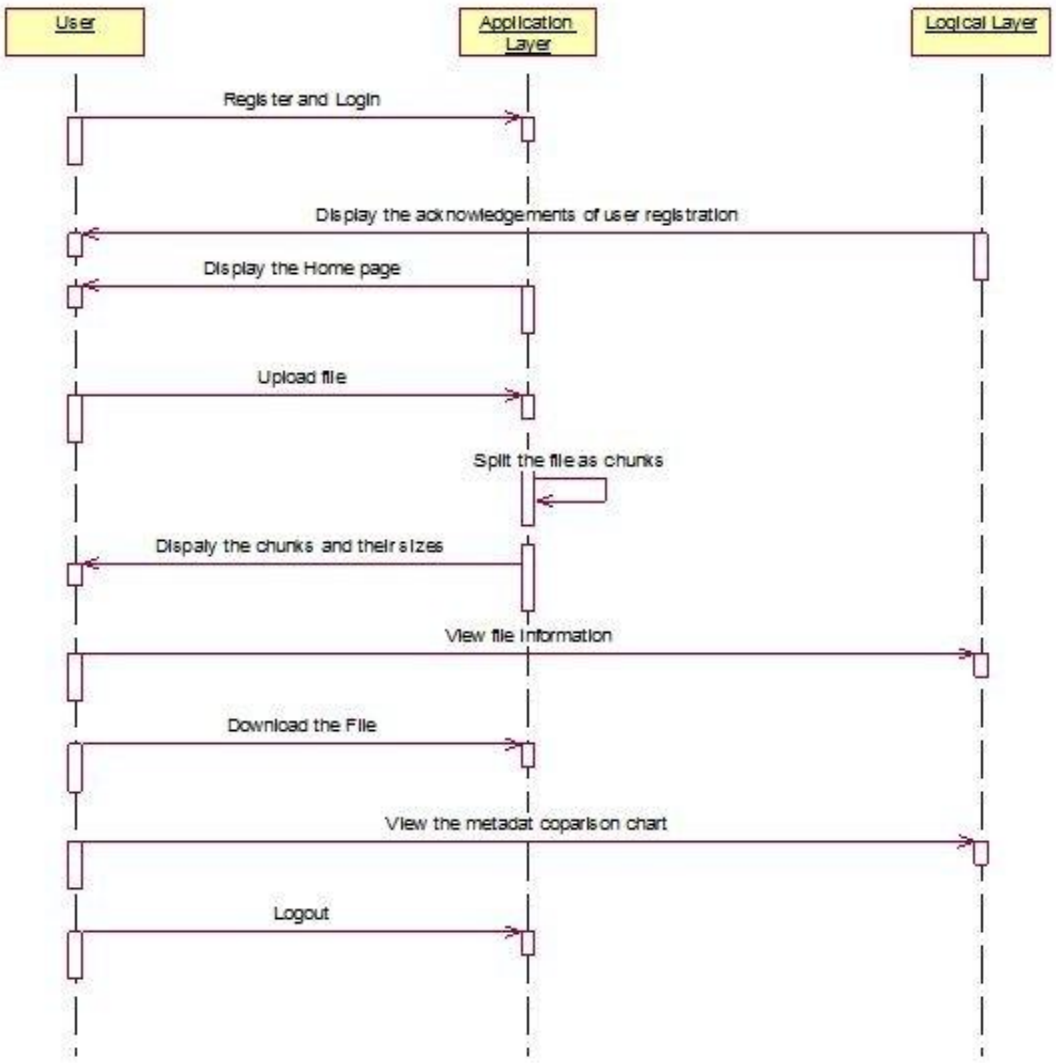
A use case diagram is a depiction of a client's interaction with the system and it represents the details of a use case. A use case diagram can represent the various types of users of a system and the few ways that they interact with the system. This diagram is utilized in coincidence with textual use case and is accompanied by other types of diagrams as well.



Use case diagram

4.4.3 Sequence Diagram

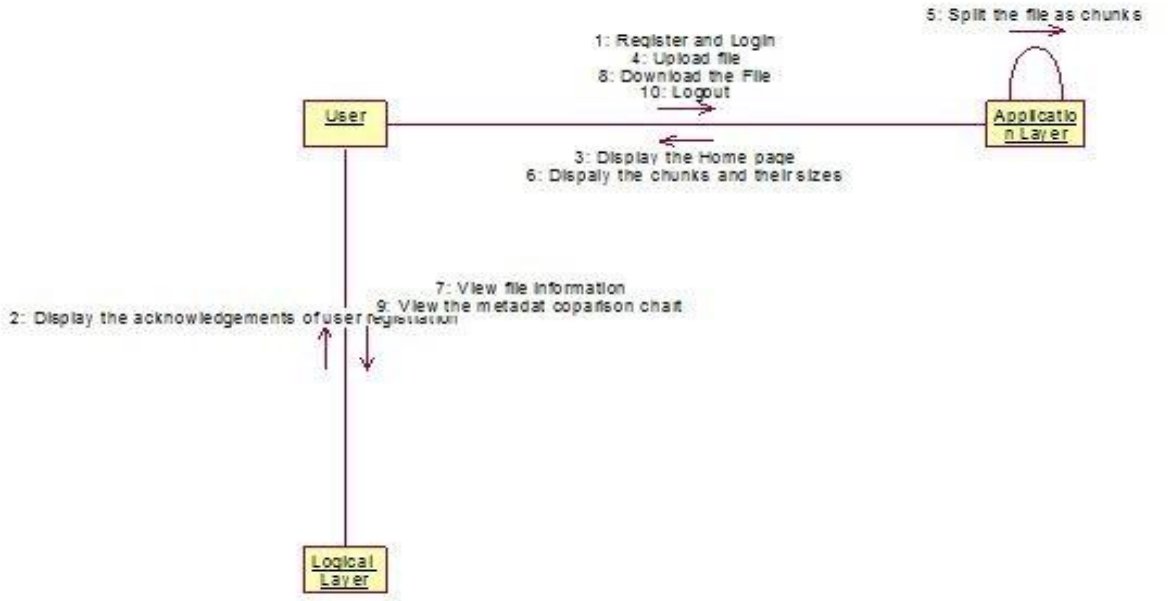
A **sequence diagram** is an interaction diagram which demonstrates how processes operate with each other and in what order. It is built of a Message Sequence Chart. A sequence diagram will demonstrate object interactions arranged in time sequence. It represents the objects and classes involved in the scenario and the sequence of messages exchanged between the objects which are required to carry out the functionality of the situation. Sequence diagrams are connected with use case realizations in the logical perspective of the system under development. Sequence diagrams are also called event diagrams, event scenarios, and timing diagrams.



Sequence Diagram

4.4.4 Collaboration diagram

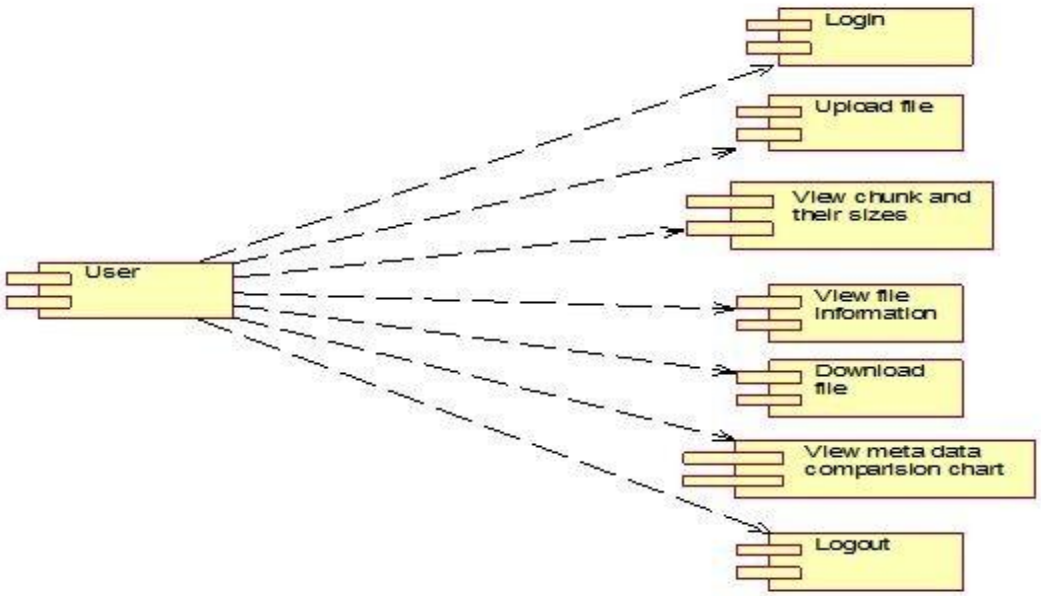
A collaboration diagram will describe interactions among objects in terms of sequenced messages. This diagram represents a combination of information which is taken from class, sequence, and use case diagrams which depicts both the static structure and dynamic behavior of a system.



Collaboration Diagram

4.4.5 Component diagram

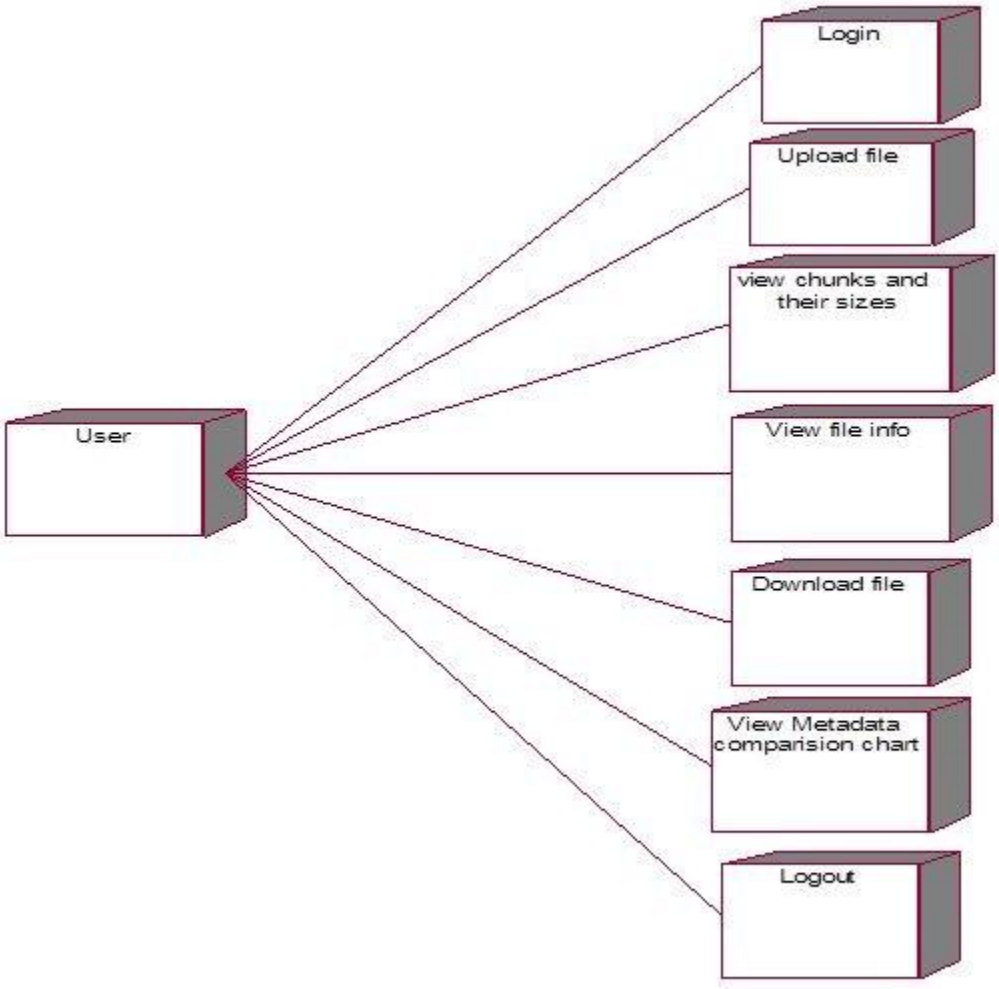
In the Unified Modeling Language, a component diagram will depict how components are wired together keeping in mind the end goal is to form large components or software systems. They are utilized to provide the structure of arbitrarily complex systems. Components are wired together using an assembly connector to connect the required interface of one component with the provided interface of another component.



Component Diagram

4.4.6 Deployment diagram

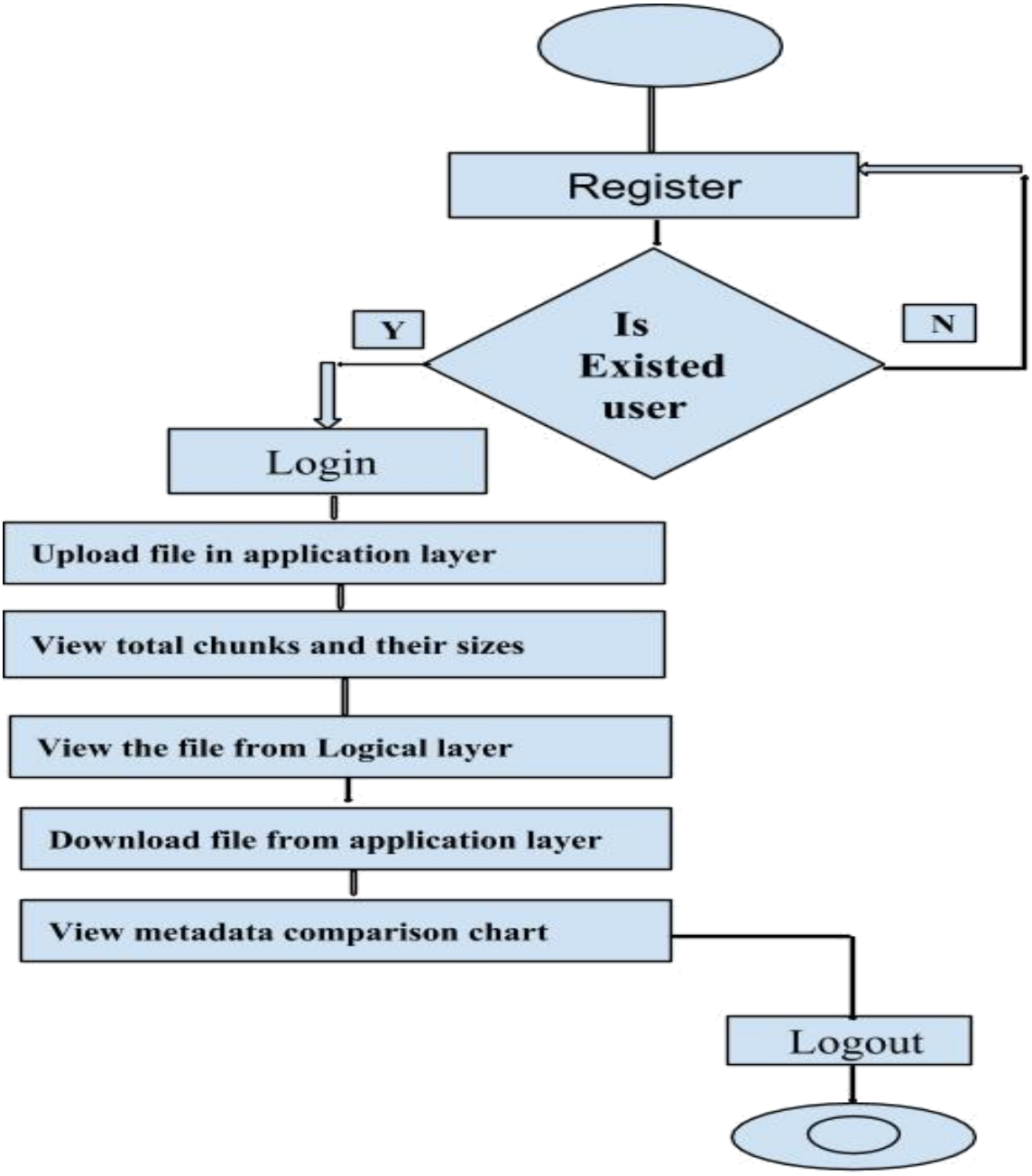
A deployment diagram in the Unified Modeling Language will demonstrate the physical deployment of artifacts on nodes. In order to describe a website, for example, a deployment diagram will show what hardware components (nodes) exist (eg., web application, database), and how different pieces are connected (eg., JDBC, REST, RMI). In the figure, the nodes appear as boxes, and the artifacts which are allocated to each node will appear as rectangles within the boxes. Nodes can have sub nodes, which will appear as nested boxes. A single node in a deployment diagram, in terms of concept may show multiple physical nodes, such as a cluster of database servers.



Deployment Diagram

4.4.7 Activity Diagram

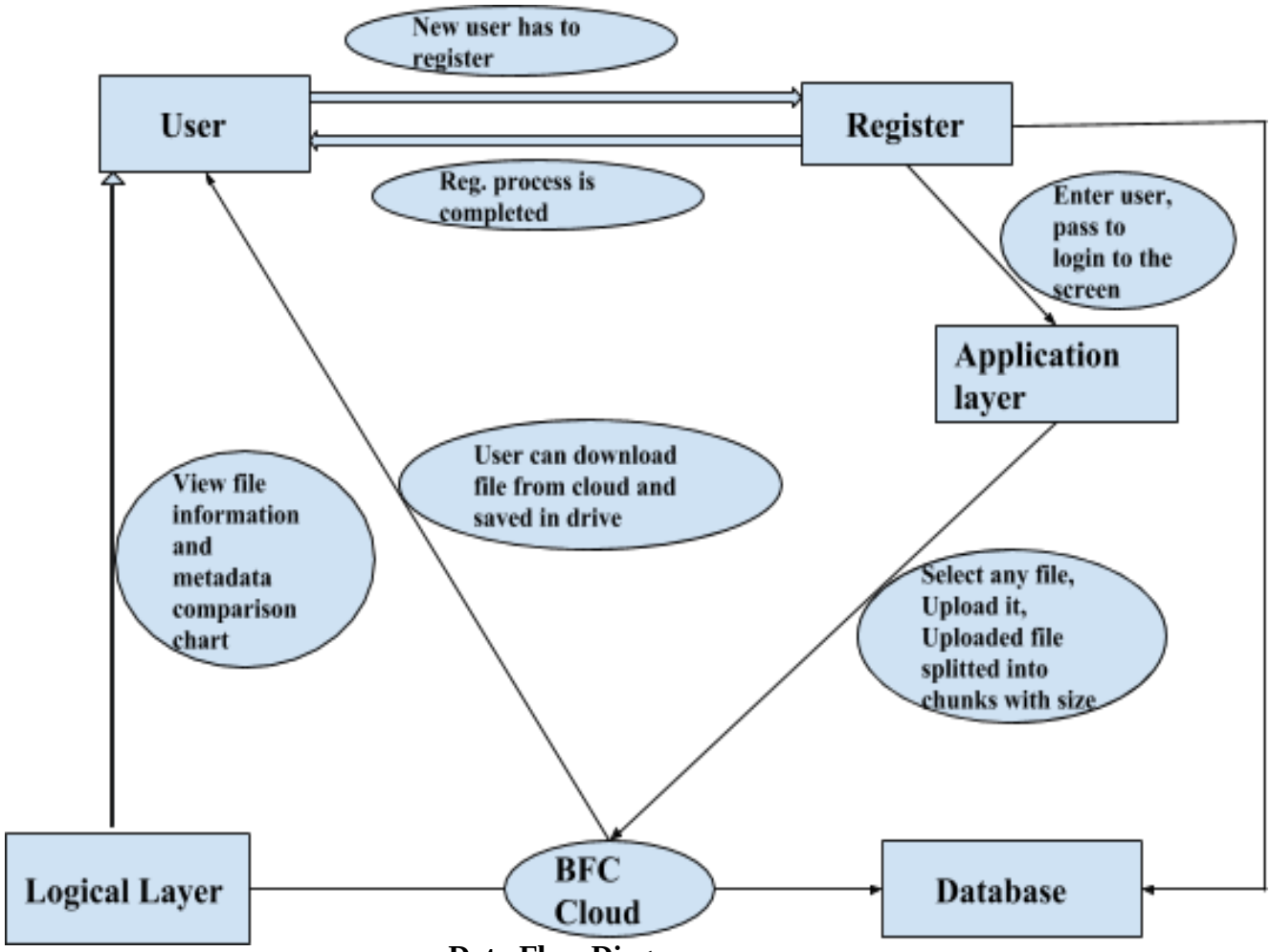
The important diagram in UML is Activity diagram which depicts dynamic aspects of the system. It is essentially a flow chart which shows the flow from one activity to another activity. The activity can be distinguished as an operation of the system. So the control flow in the flow chart is shown from one operation to another. This flow can be sequential, branched or concurrent.



Activity Diagram

4.4.8 Data Flow Diagram

Data Flow diagram is a demonstration that explains the passage of information in a process. A DFD can be easily drawn by using simple symbols. Complicated processes can be easily automated by creating DFDs, easy to use, free downloadable diagramming tools. A DFD is a model for building and analyzing information processes. DFD represents the flow of information in a process which depends upon the inputs and outputs. A DFD can also be called as a Process Model. A DFD reveals business or technical process with the support of the outside data saved, in addition to the data flowing from the process to another and the end results.



Data Flow Diagram

Chapter 5

Implementation

It deals with how high performance distributed BFC architecture is implemented, technologies used in coding in Java.

5.1 BFC Architecture Modules

1. Application Layer
2. Storage Layer
3. Object Store Layer

5.1.1 Module Description

Application Layer: It consists of indigenous software on desktop systems, mobile devices and web-interface, which grants user to upload, download and share their own files.

Storage Logical Layer: It contains numerous queuing services and worker services, ID-Generator services and all logical API for Cloud Storage System. This layer will implement business logic part in BFC.

Object Store Layer: It consists of many distributed backend services. Object Store Layer has two important services namely, FileInfoService and ChunkStoreService. Information of files is stored in FileInfoService. ChunkStoreService will store data chunks which are built by splitting the original files that user has uploaded.

5.2 Introduction of technologies used

5.3 About Java

Java is a platform independent language that can be used to create software to be embedded in different consumer electronic devices. Java has a great impact on Internet because Java gives rise to a various objects that can move around freely in Cyberspace. There are two categories of objects namely, passive information and Dynamic active programs that are transmitted between the server and the personal computer. This has led to a new form of program called the Applet.

Applications and applets

A program that will run on a computer under the operating system of that particular computer. It is similar to one that is created using C or C++. The ability of Java to create applets makes it more crucial. An applet is an application which is designed in order to be transmitted over the Internet and can be executed by a Java compatible web browser.

Java Architecture

Java architecture comprises of a simple, robust, object-oriented high operating environment for development. Java supports portability by compiling byte codes for Java virtual machine, which is later interpreted on each of the platform by the run-time environment. Java is a dynamic system which can be able to load code when required from a machine across the planet.

Compilation of code

When a code is compiled, the Java compiler will create a machine code (byte code) for a imaginary machine called Java Virtual Machine (JVM). JVM is created in order to overcome the issue of probability. The code will be written and compiled for one machine i.e., JVM and it is interpreted on all machines. During the runtime the Java interpreter will trick the byte code file that it been run on a Java Virtual Machine. This could be an Intel pentium or Apple Macintosh running system and all will be able to receive code from any device through internet and can run applets.

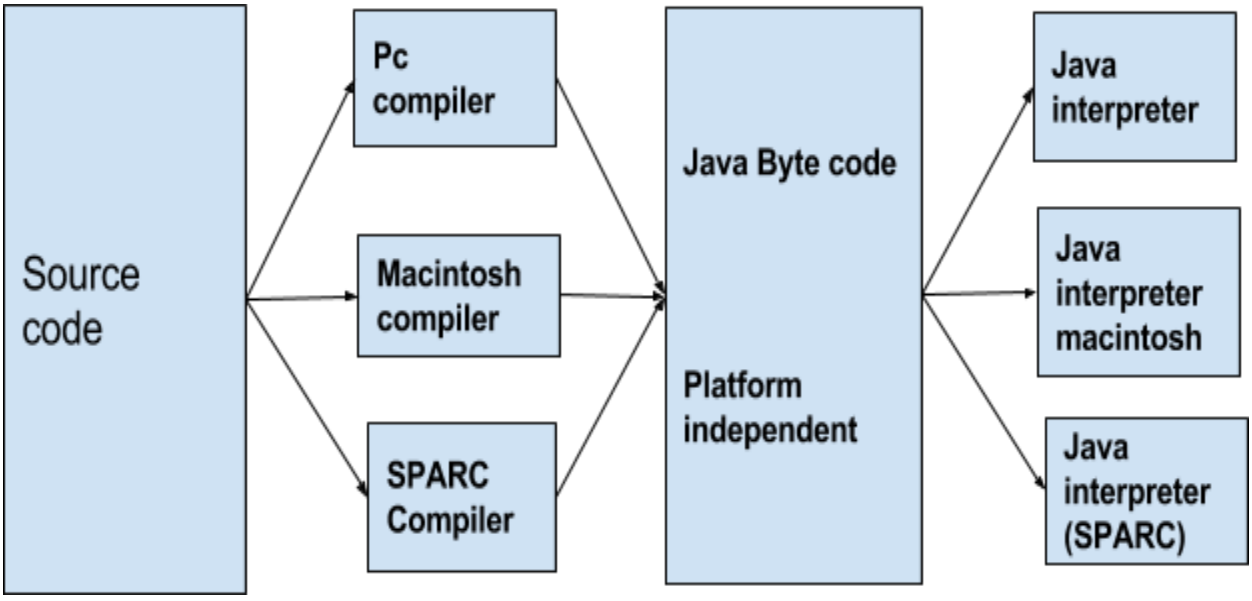


Figure 8: Block diagram for code compilation and Interpretation of Java Source Code

5.3.1 AWT and Swings

AWT: Users can interact with the program through the user interface. Graphical User Interface (GUI) is one such type of user interface that would grant users in order to interact with electronic devices through images other than text commands. Java provides a class library called as Abstract Window Toolkit (AWT) used to write graphical programs. AWT is not platform independent as Java, and it provides a common set of tools for the design of graphical user interface. A graphical user interface is comprised of graphical elements known as components. A component is an object that has a graphical representation and can be displayed on the screen, through which a user can interact with the program and provide input to the program. These user interface components are instances of class Component. Typical class components include items such as buttons, scrollbars and text fields.

Swings: Swing is essential to develop Java with a graphical user interface (GUI). Swing Toolkit is the fundamental tool for building GUI in Java. Various components used in Swing Toolkit are list controls, buttons, labels, checkboxes. Java Swing will be able to handle all the AWT flexible components. Swing has many advanced features such as JTable, JTabbedPane and JTree which are not available in AWT. Swings are developed by java language whereas, AWTs are developed by using C and C++.

5.3.2 Code

Register.java : This code is used for capturing the user data and processing it for registration.

```
package BFC.
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.net.Socket;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;
import net.miginfocom.swing.MigLayout;
import java.awt.Color;
import javax.swing.JFrame;
public class Register extends JFrame{
   JPanel p1;
  Font f1;
```

```
    JLabel l1,l2,l3,l4,l5;
    JTextField tf1,tf2,tf3,tf4,tf5;
    JButton b1,b2;
    Dimension d;
    int w,h;
    int center;
public Register(){
        super("New User Registration Screen");
    d = Toolkit.getDefaultToolkit().getScreenSize();
    w = (int)d.getWidth();
    h = (int)d.getHeight();
    center = w/30;
        setBackground(Color.white);
    p1 = new JPanel();
        p1.setBackground(Color.white);
    p1.setLayout(new MigLayout("wrap 2","","[]15[]"));
    f1 = new Font("Verdana",Font.PLAIN,14);
    p1.add(new JLabel(),"cell 0 30");
    p1.add(new JLabel(),"cell 1 30");

    l1 = new JLabel("Username");
    l1.setFont(f1);
    p1.add(l1,"gap left "+center);
    tf1 = new JTextField(12);
    tf1.setFont(f1);
    p1.add(tf1,"gap left 30");

    l2 = new JLabel("Password");
    l2.setFont(f1);
    p1.add(l2,"gap left "+center);
    tf2 = new JPasswordField(12);
    tf2.setFont(f1);
    p1.add(tf2,"gap left 30");


    l3 = new JLabel("Contact No");
    l3.setFont(f1);
    p1.add(l3,"gap left "+center);
    tf3 = new JTextField(12);
    tf3.setFont(f1);
    p1.add(tf3,"gap left 30");

    l4 = new JLabel("Email ID");
    l4.setFont(f1);
    p1.add(l4,"gap left "+center);
    tf4 = new JTextField(12);
    tf4.setFont(f1);
```

```
    p1.add(tf4,"gap left 30");

    l5 = new JLabel("Address");
    l5.setFont(f1);
    p1.add(l5,"gap left "+center);
    tf5 = new JTextField(30);
    tf5.setFont(f1);
    p1.add(tf5,"gap left 30");


    p1.add(new JLabel(""));
    b1 = new JButton("Register");
    b1.setFont(f1);
    p1.add(b1,"split 2");
    b1.addActionListener(new ActionListener(){
        @Override
        public void actionPerformed(ActionEvent ae){
            process();
        }
    });


    b2 = new JButton("Clear");
    b2.setFont(f1);
    p1.add(b2);
    b2.addActionListener(new ActionListener(){
        @Override
        public void actionPerformed(ActionEvent ae){
            clearFields();
        }
    });

    add(p1);

}
public void process(){
    try{
        String uname=tf1.getText();
        String pass=tf2.getText();
        String contact=tf3.getText();
        String email=tf4.getText();
        String address=tf5.getText();
        if(uname.length() <=0 || uname == null){
            JOptionPane.showMessageDialog(this,"Username must be enter");
            tf1.requestFocus();
            return;
        }
```

```
if(pass.length() <=0 || pass == null){
   JOptionPane.showMessageDialog(this,"Password must be enter");
   tf2.requestFocus();
   return;
}
if(contact.length() <=0 || contact == null){
   JOptionPane.showMessageDialog(this,"Contact No must be enter");
   tf3.requestFocus();
   return;
}
if(contact.length() <=0 || contact == null){
   JOptionPane.showMessageDialog(this,"Contact no must be enter");
   tf3.requestFocus();
   return;
}
            if(!validatePhoneNumber(contact.trim())){
                  JOptionPane.showMessageDialog(this,"Enter valid contact no");
   tf3.requestFocus();
   return;
            }
if(email.length() <=0 || email == null){
   JOptionPane.showMessageDialog(this,"Email id must be enter");
   tf4.requestFocus();
   return;
}
            if(!CheckMail.checkMail(email)){
                  JOptionPane.showMessageDialog(this,"Enter valid mailid");
                  tf4.requestFocus();
                  return;
            }
            if(address.length() <=0 || address == null){
   JOptionPane.showMessageDialog(this,"Address must be enter");
   tf5.requestFocus();
   return;
}
Socket socket=new Socket("localhost",1200);
ObjectOutputStream out=new ObjectOutputStream(socket.getOutputStream());
ObjectInputStream in=new ObjectInputStream(socket.getInputStream());
Object req[]={"register",uname,pass,contact,email,address};
out.writeObject(req);
out.flush();
Object res[]=(Object[])in.readObject();
            String msg = res[0].toString();
            if(msg.equals("Registration process completed")){
                  JOptionPane.showMessageDialog(this,msg);
                  setVisible(false);
            }else{
```

```
                        JOptionPane.showMessageDialog(this,msg);
                }
    }catch(Exception e){
        e.printStackTrace();
    }
}
public void clearFields(){
    tf1.setText("");
    tf2.setText("");
    tf3.setText("");
    tf4.setText("");
    tf5.setText("");
}
private static boolean validatePhoneNumber(String phoneNo){
        //validate phone numbers of format "1234567890"
    if(phoneNo.matches("\\d{10}"))
                return true;
        //validating phone number with -, . or spaces
    else if(phoneNo.matches("\\d{3}[-\\.\\s]\\d{3}[-\\.\\s]\\d{4}"))
                return true;
        //validating phone number with extension length from 3 to 5
        else if(phoneNo.matches("\\d{3}-\\d{3}-\\d{4}\\s(x|(ext))\\d{3,5}"))
                return true;
        //validating phone number where area code is in braces ()
    else if(phoneNo.matches("\\(\\d{3}\\)-\\d{3}-\\d{4}"))
                return true;
        //return false if nothing matches the input
    else
                return false;

}
}
```

Login.java : This code is used for authenticating the users when they login

```
package BFC;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.ImageIcon;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

```
import java.awt.BorderLayout;
import java.awt.Color;
import javax.swing.JPasswordField;
import javax.swing.UIManager;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.net.Socket;
import javax.swing.JOptionPane;
public class Login extends JFrame
{
	JLabel l1,l2,l3,l4;
	JTextField tf1,tf2;
	JButton b1,b2,b3;
	Font f1,f2;
	JPanel p1,p2,p3,p4,p5,p6;
	ImageIcon icon;
public Login(){
	super("Login Screen");
	p1 = new JPanel();
	p1.setBackground(Color.black);
	f1 = new Font("Monospaced",Font.BOLD,22);
	l1 = new JLabel("<html><body><center>User Login
Screen</center></body></html>");
	l1.setForeground(new Color(125,54,2));
	l1.setFont(f1);
	p1.add(l1);
	p1.setBackground(new Color(140,150,180));

	p2 = new JPanel();
	p2.setBackground(Color.black);
	icon = new ImageIcon("img/vista.jpg");
	JLabel label = new JLabel(icon);
	p2.add(label);

	p3 = new JPanel();
	p3.setLayout(new BorderLayout());

	p4 = new JPanel();
	f2 = new Font("Verdana",Font.PLAIN,14);
	l3 = new JLabel("Username");
	l3.setFont(f2);
	p4.add(l3);

	tf1 = new JTextField(15);
	tf1.setFont(f2);
	p4.add(tf1);
```

```
    p5 = new JPanel();
    l4 = new JLabel("Password");
    l4.setFont(f2);
    p5.add(l4);

    tf2 = new JPasswordField(15);
    tf2.setFont(f2);
    p5.add(tf2);

    p6 = new JPanel();
    b1 = new JButton("Login");
    b1.setFont(f2);
    b1.setBackground(new Color(51, 51, 51));
    p6.add(b1);
    b1.addActionListener(new ActionListener(){
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                    login();
}
    });

    b2 = new JButton("Reset");
    b2.setFont(f2);
    b2.setBackground(new java.awt.Color(51, 51, 51));
    p6.add(b2);
    b2.addActionListener(new ActionListener(){
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                    reset();
}
    });
    b3 = new JButton("New User");
    b3.setFont(f2);
    b3.setBackground(new java.awt.Color(51, 51, 51));
    p6.add(b3);
    b3.addActionListener(new ActionListener(){
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                    Register register = new Register();
                    register.pack();
                    register.setVisible(true);
                    register.setLocationRelativeTo(null);
}
    });
    p3.add(p4,BorderLayout.NORTH);
    p3.add(p5,BorderLayout.CENTER);
    p3.add(p6,BorderLayout.SOUTH);
    getContentPane().add(p1,BorderLayout.NORTH);
    getContentPane().add(p2,BorderLayout.CENTER);
    getContentPane().add(p3,BorderLayout.SOUTH);
```

```
}
public static void main(String a[])throws Exception{
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        Login screen = new Login();
        screen.setVisible(true);
        screen.pack();
        screen.setLocationRelativeTo(null);
}
public void reset(){
        tf1.setText("");
        tf2.setText("");
}
public void login(){
        try{
                String user = tf1.getText();
                String pass = tf2.getText();
                if(user == null || user.trim().length() <= 0){
                        JOptionPane.showMessageDialog(this,"Username must be enter");
                        tf1.requestFocus();
                        return;
                }
                if(pass == null || pass.trim().length() <= 0){
                        JOptionPane.showMessageDialog(this,"Password must be enter");
                        tf2.requestFocus();
                        return;
                }
                Socket socket=new Socket("localhost",1200);
      ObjectOutputStream out=new ObjectOutputStream(socket.getOutputStream());
      ObjectInputStream in=new ObjectInputStream(socket.getInputStream());
      Object req[]={"login",user,pass};
      out.writeObject(req);
      out.flush();
      Object res[]=(Object[])in.readObject();
                String msg = res[0].toString();
                if(msg.equals("success")){
                        setVisible(false);
                        UserScreen us = new UserScreen(this,user);
                        us.setVisible(true);
                        us.setExtendedState(JFrame.MAXIMIZED_BOTH);
                }else{
                        JOptionPane.showMessageDialog(this,"invalid login");
                }
        }catch(Exception e){
      e.printStackTrace();
   }

}
```

}

UserScreen.java : This code is going to deal with the view of the user once he logins.

```
package BFC;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.BorderLayout;
import java.awt.Color;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;
import javax.swing.JOptionPane;
import javax.swing.JFileChooser;
import java.net.Socket;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.io.RandomAccessFile;
import java.util.Random;
import java.io.File;
import java.io.FileInputStream;
public class UserScreen extends JFrame{
	JButton b1,b2,b3;
	JPanel p1,p2;
	Font f1;
	JTextArea area;
	JScrollPane jsp;
	Login login;
	String user;
	JFileChooser chooser;
	RandomAccessFile random;
	int tot_blocks;
public UserScreen(Login log,String usr){
	super("User Screen");
	login = log;
	user = usr;
	p1 = new JPanel();
```

```
f1 = new Font("Monospaced",Font.BOLD,16);
chooser = new JFileChooser();
b1 = new JButton("Upload File");
b1.setFont(f1);
p1.add(b1);
b1.addActionListener(new ActionListener(){
	public void actionPerformed(java.awt.event.ActionEvent evt) {
		int option = chooser.showOpenDialog(UserScreen.this);
		if(option == JFileChooser.APPROVE_OPTION){
			File file = chooser.getSelectedFile();
			upload(file);
		}
	}
});

b2 = new JButton("Download File");
b2.setFont(f1);
p1.add(b2);
b2.addActionListener(new ActionListener(){
	public void actionPerformed(java.awt.event.ActionEvent evt) {
		DownloadFile df = new DownloadFile(area);
		df.setUser(user);
		df.setSize(300,100);
		df.setVisible(true);
		df.setLocationRelativeTo(null);
		df.getFileName();
	}
});

b3 = new JButton("Logout");
b3.setFont(f1);
p1.add(b3);
b3.addActionListener(new ActionListener(){
	public void actionPerformed(java.awt.event.ActionEvent evt) {
		setVisible(false);
		login.setVisible(true);
	}
});

p2 = new JPanel();
p2.setLayout(new BorderLayout());
area = new JTextArea();
area.setFont(f1);
area.setEditable(false);
jsp = new JScrollPane(area);
p2.add(jsp,BorderLayout.CENTER);
```

```
        getContentPane().add(p1,BorderLayout.NORTH);
        getContentPane().add(p2,BorderLayout.CENTER);

}
public long getChunkSize(File file){
        long length = file.length();
        tot_blocks=0;
        long size = 0;
        if(length >= 1000){
                size = length/10;
                tot_blocks = 10;
        }
        if(length < 1000 && length > 500){
                size = length/5;
                tot_blocks = 5;
        }
        if(length < 500 && length > 1){
                size = length/3;
                tot_blocks = 3;
        }
        return size;
}
public void upload(File file){
        try{
                FileInputStream fin = new FileInputStream(file);
                byte file_data[] = new byte[fin.available()];
                fin.read(file_data,0,file_data.length);
                fin.close();
                byte encrypt[] = AES.encrypt(file_data);
                String sha = SHA.ShaSignature(encrypt);
                long chunk_size = getChunkSize(file);
                random = new RandomAccessFile(file,"r");
                Object row[][] = createChunks(chunk_size,file.getName());
                random.close();
                Socket socket=new Socket("localhost",1200);
      ObjectOutputStream out=new ObjectOutputStream(socket.getOutputStream());
                ObjectInputStream in=new ObjectInputStream(socket.getInputStream());
                Object
req[]={"upload",row,user,file.getName(),sha,Long.toString(file.length())};
                out.writeObject(req);
                out.flush();
                Object res[]=(Object[])in.readObject();
                String server_res = res[0].toString();
                area.append(server_res+"\n");
                out.close();
                in.close();
                socket.close();
```

```
        }catch(Exception e){
                e.printStackTrace();
        }
}
public Object[][] createChunks(long chunks_size,String name){
        area.append("Total chunks "+tot_blocks+" With chunk size "+chunks_size+"\n");
        Object row[][]=new Object[tot_blocks][2];
        try{
                String ext = name.substring(name.lastIndexOf(".")+1,name.length());
                for(int i=0;i<tot_blocks;i++){
                        byte b[]=new byte[(int)chunks_size];
                        random.read(b);
                        random.seek(random.getFilePointer());
                        row[i][0]=b;
                        row[i][1]=name+"_chunk"+i+"."+ext;
                }
        }catch(Exception e){
                e.printStackTrace();
        }
        return row;
}
}
```

DBCon.java : In this code all database related operations will be done like, storing and retrieving data.

```
package BFC;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
public class DBCon{
private static Connection con;
public static Connection getCon()throws Exception {
        if(con == null){
                Class.forName("com.mysql.jdbc.Driver");
                con =
DriverManager.getConnection("jdbc:mysql://localhost/BFC","root","root");
        }
        return con;
}
public static String register(String[] input)throws Exception{
        String msg="Error in registration";
   boolean flag=false;
```

```
   con = getCon();
   Statement stmt=con.createStatement();
   ResultSet rs=stmt.executeQuery("select username from bfc_users where
username='"+input[0]+"'");
   if(rs.next()){
      flag=true;
      msg = "Username already exist";
   }
   if(!flag){
                  PreparedStatement stat=con.prepareStatement("insert into bfc_users
values(?,?,?,?,?)");
                  stat.setString(1,input[0]);
                  stat.setString(2,input[1]);
                  stat.setString(3,input[2]);
                  stat.setString(4,input[3]);
                  stat.setString(5,input[4]);
                  int i=stat.executeUpdate();
                  if(i > 0){
                        msg = "Registration process completed";
                  }
                  stat.close();
   }
         rs.close();stmt.close();
   return msg;
}
public static String login(String input[])throws Exception{
   String msg="fail";
   con = getCon();
   Statement stmt=con.createStatement();
   ResultSet rs=stmt.executeQuery("select username from bfc_users where
username='"+input[0]+"' && password='"+input[1]+"'");
   if(rs.next()){
      msg = "success";
   }
         rs.close();stmt.close();
   return msg;
}
}
```

Code for File Compression in Java

```
package BFC; import
java.io.*; import
java.util.zip.*;
public class Compress {
        static final int BUFFER = 1048;
public static void main(String args[]){
        System.out.println(compress(new File("screens.docx")));
}

public static long compress(File file){
        long length = 0;
        try{
                BufferedInputStream  origin  =
                null;
                FileOutputStream dest = new FileOutputStream(file+".compress");
                ZipOutputStream out = new ZipOutputStream(new
                BufferedOutputStream(dest));
                //out.setMethod(ZipOutputStream.DEFLATE
                D);
      byte data[] = new byte[BUFFER];
      // get a list of files from current
                directory File list[] =
                file.listFiles(); for(int
                i=0;i<list.length;i++){
      FileInputStream fi = new FileInputStream(list[i]);
                origin=newBufferedInputStream(fi,BUFFER
                );
                ZipEntryentry=newZipEntry(list[i].getPath();
                out.putNextEntry(entry);
                int
                count;
                while((count  =  origin.read(data,0,BUFFER))  !=
                -1){
                        out.write(data,0,count);
                }
                }

                File fname = new File(file+".compress");
                length = fname.length();
```

```
        }catch(Exception e) {
                e.printStackTrace();
        }
        return length;
}
public static byte[] decompress(String file){
        byte b[] = null;
        try{
                final int BUFFER = 2048;
                BufferedOutputStream dest = null;
                FileInputStream fis = new
                FileInputStream(file);
                CheckedInputStream checksum = new CheckedInputStream(fis, new
      Adler32()); ZipInputStream zis = new ZipInputStream(new
      BufferedInputStream(checksum));
                ZipEntry entry;
                while((entry = zis.getNextEntry()) != null){
                        System.out.println("Extracting: " +entry);
        int count;
        byte data[] = new byte[BUFFER];
                        String name = entry.getName();
        FileOutputStream fos = new FileOutputStream(name);
                        dest = new BufferedOutputStream(fos,BUFFER);
                        while((count = zis.read(data,0,BUFFER)) != -1) {
                                dest.write(data, 0, count);
        }
        dest.flush();
        dest.close();
                        FileInputStream fin = new FileInputStream(name);
                        b = new byte[fin.available()];
                        fin.read(b,0,b.length);
                        fin.close();
                        File fname = new File(name);
                        fname.delete();
      }
                 zis.close();
       System.out.println("Checksum:"+checksum.getChecksum().getValue());
        }catch(Exception e){
                e.printStackTrace();
        }
```

```
        return b;
}
}
```

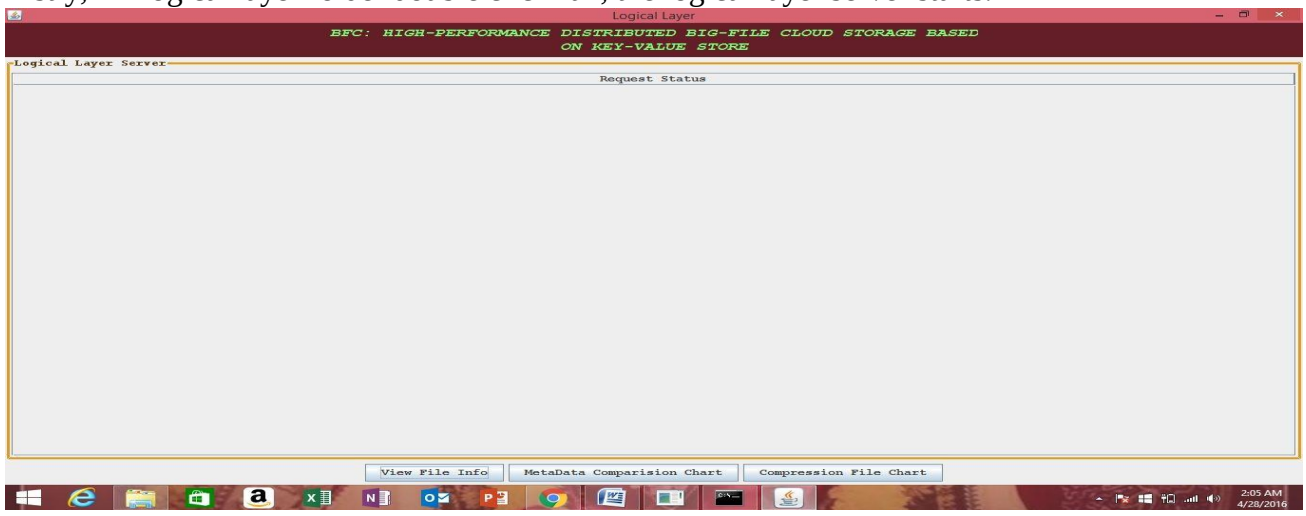
Chapter 6

Results

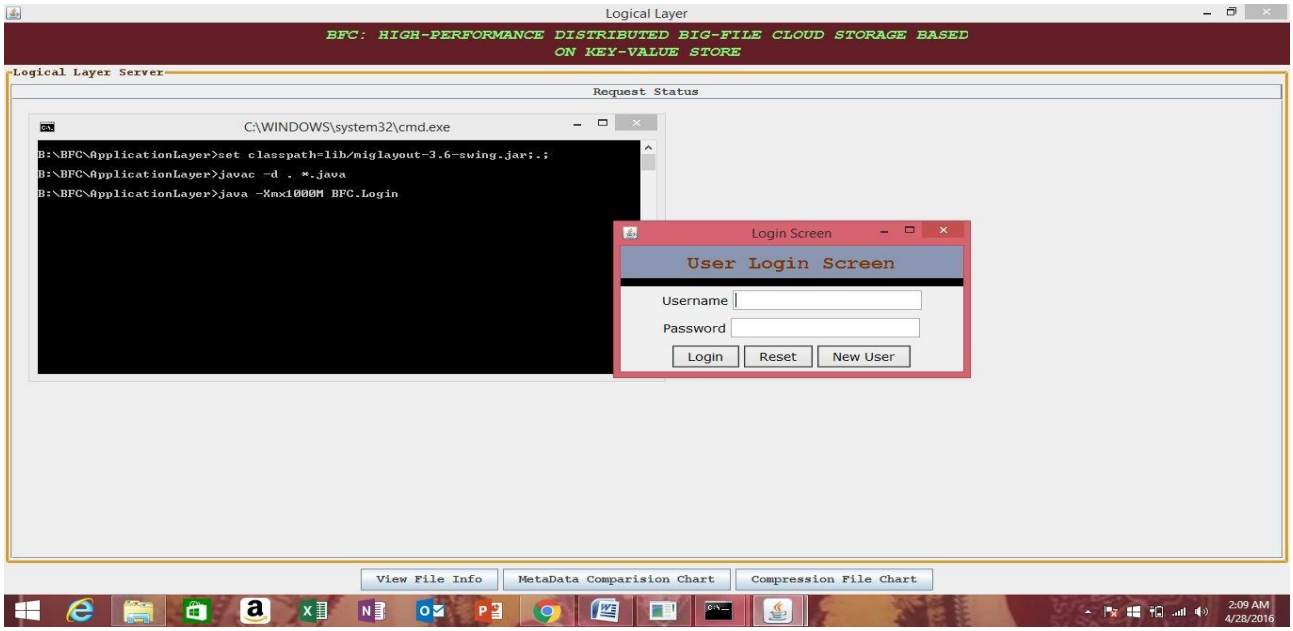
6.1 Screenshots

We can observe screen shots which were taken during the execution of the implemented programs.

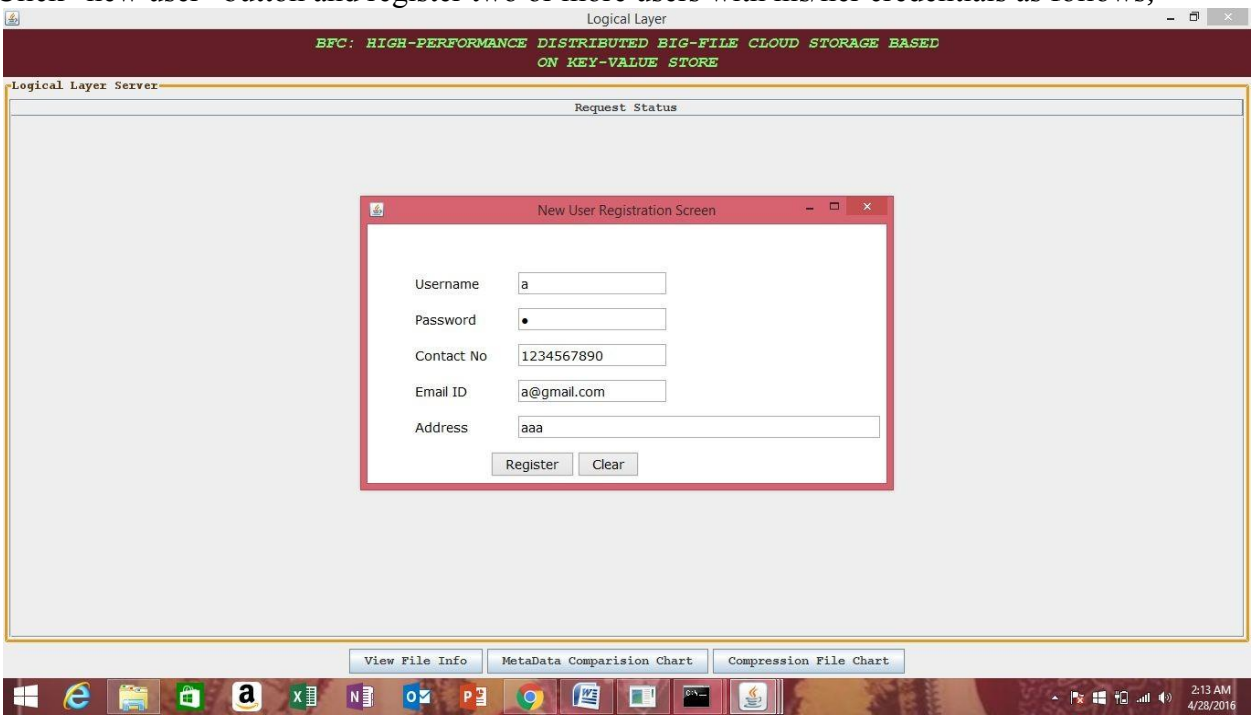
Firstly, In Logical layer folder double click run, the logical layer server starts.

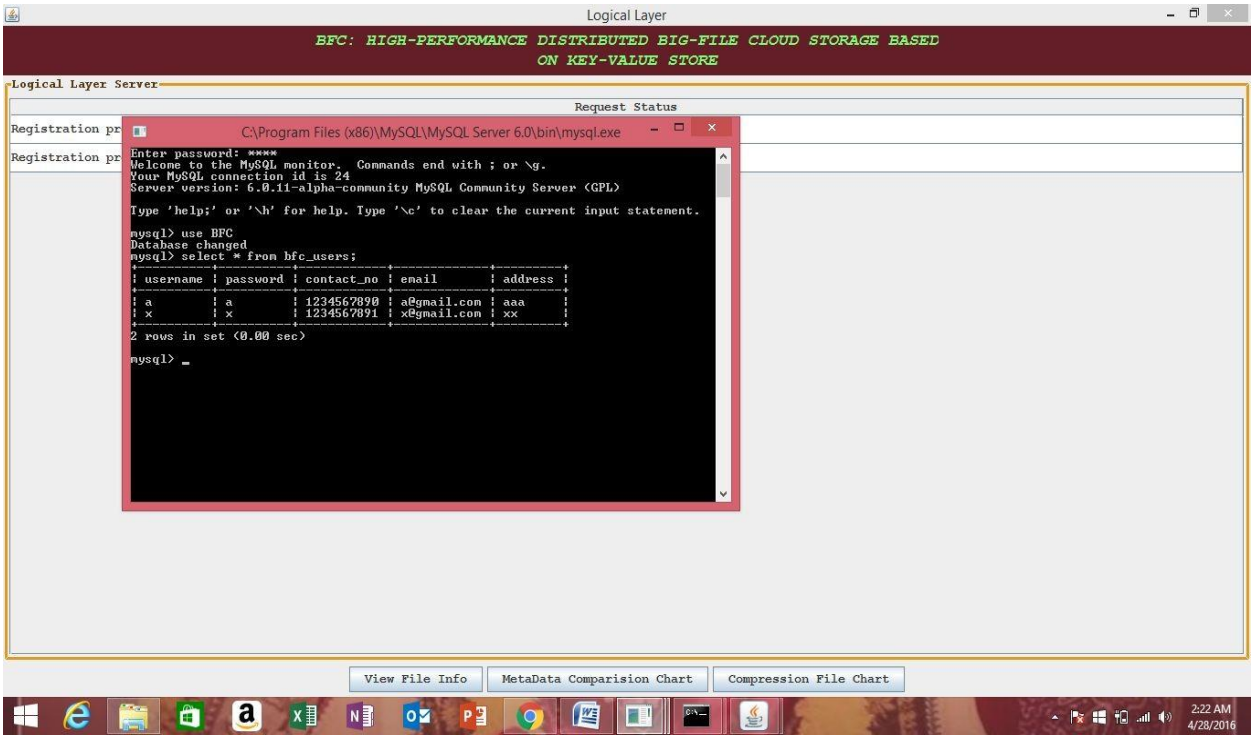


Now, go to application layer and double click run to start the user screen.

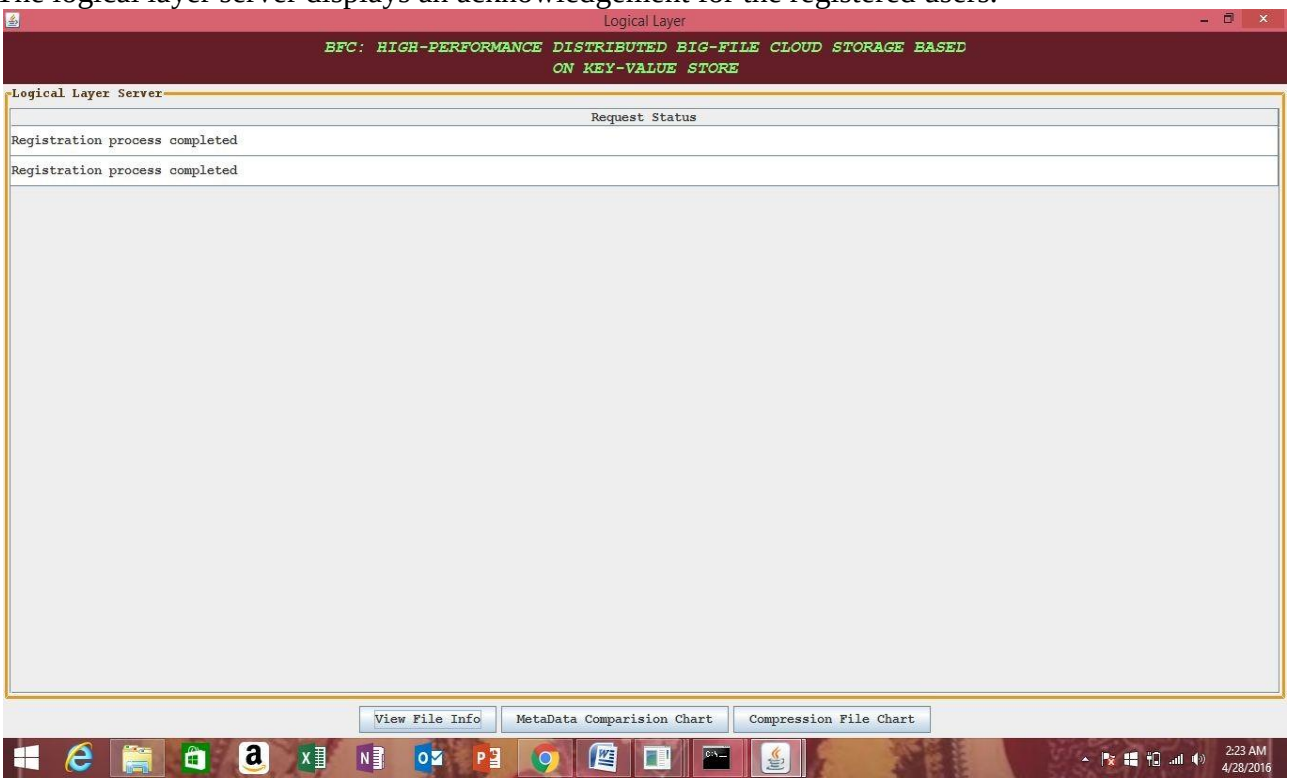


Click “new user” button and register two or more users with his/her credentials as follows,

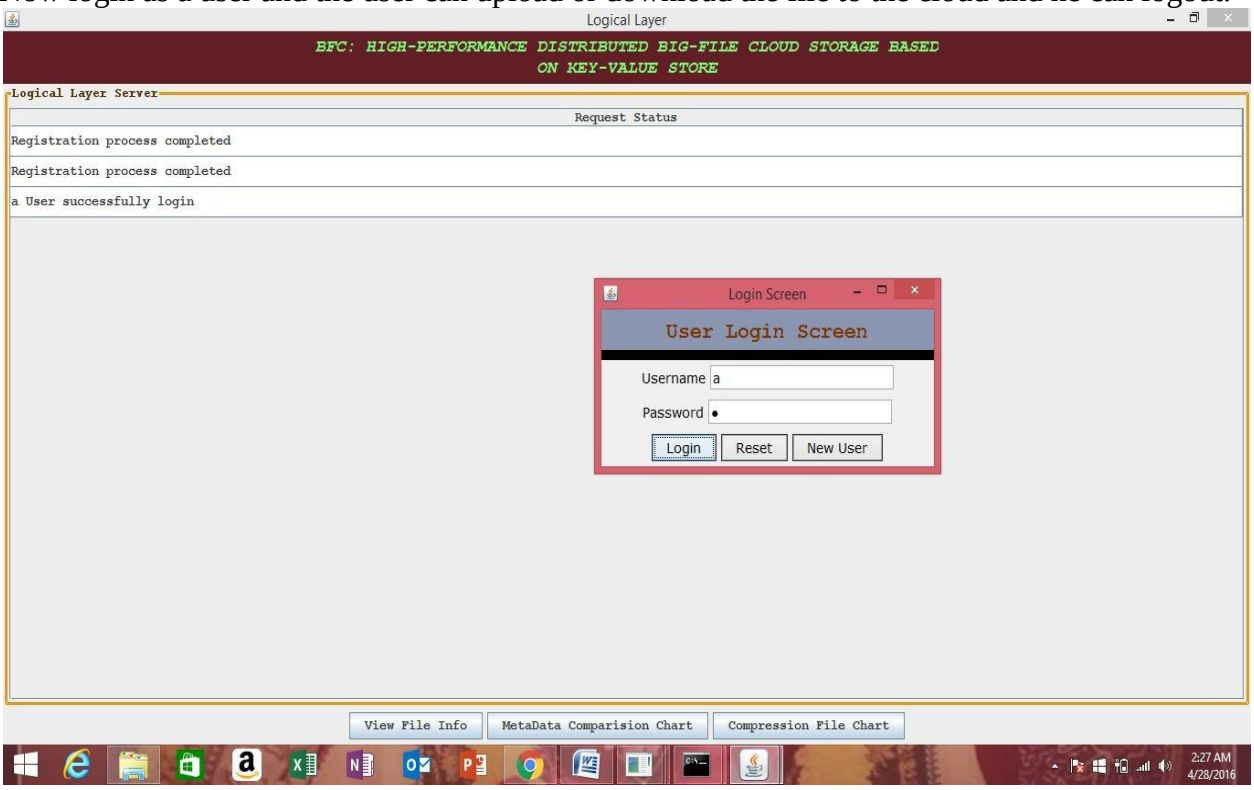


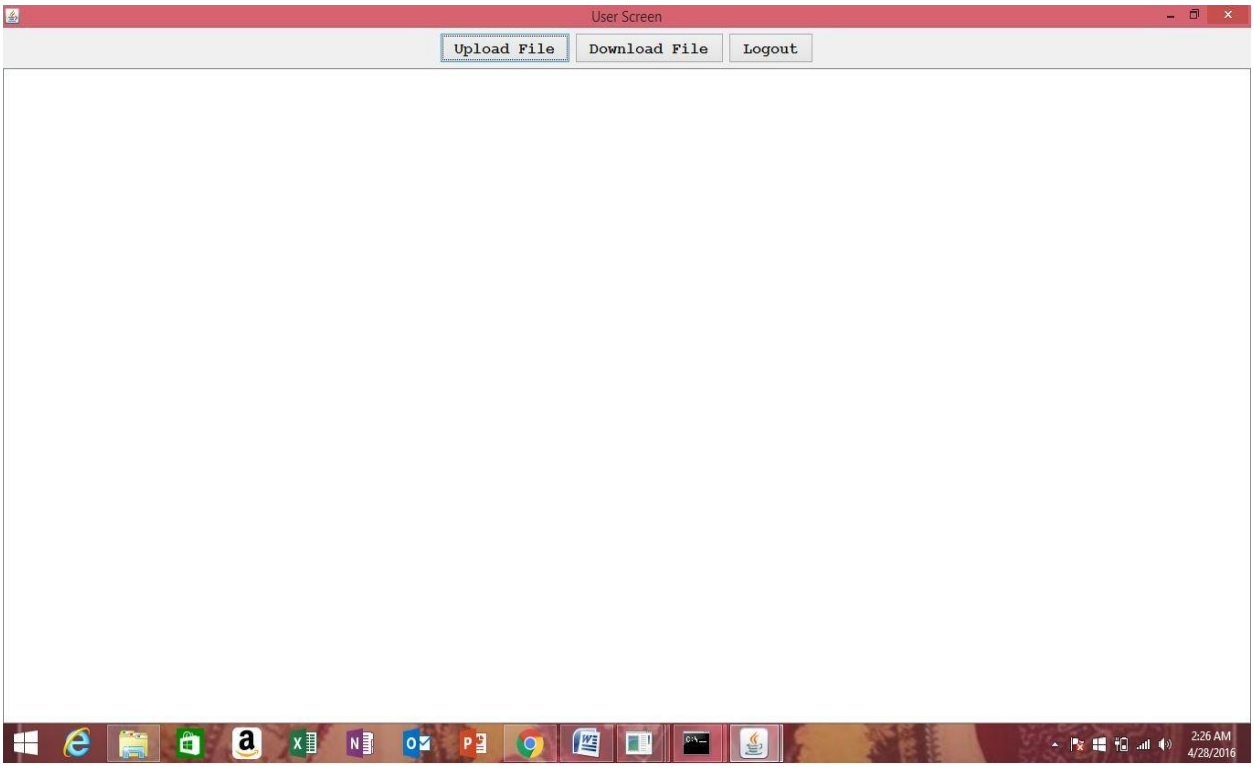


The logical layer server displays an acknowledgement for the registered users.

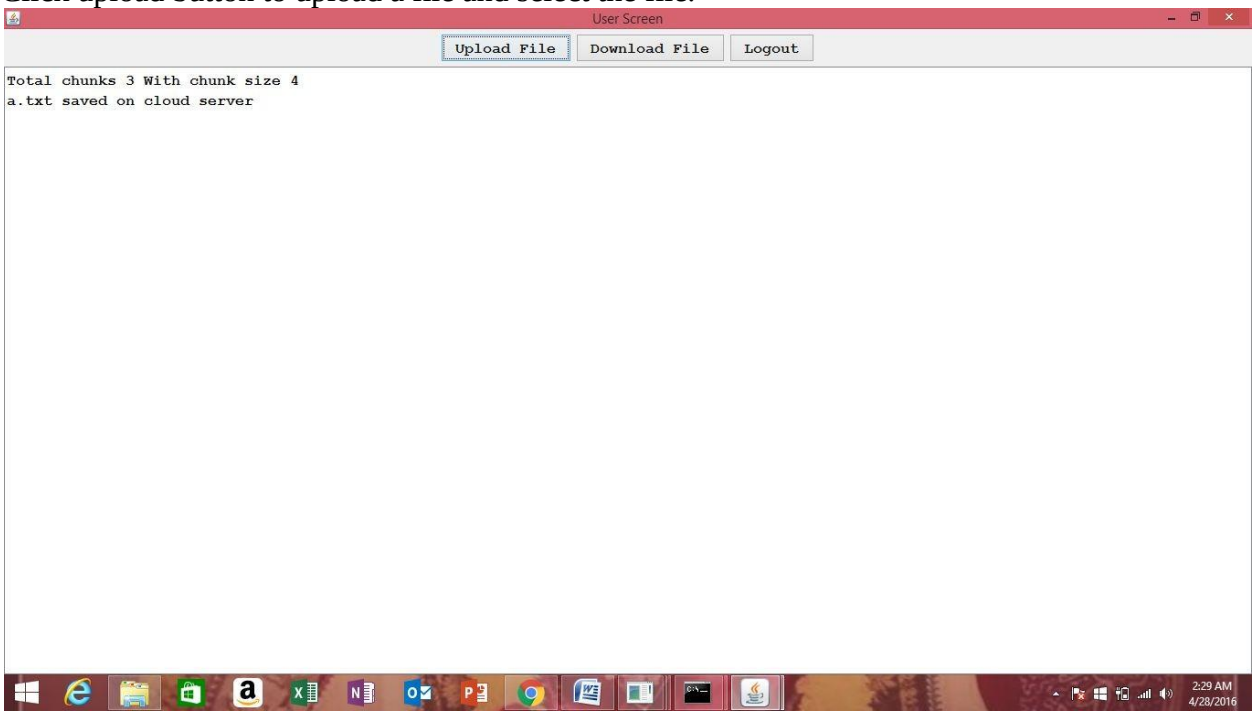


Now login as a user and the user can upload or download the file to the cloud and he can logout.

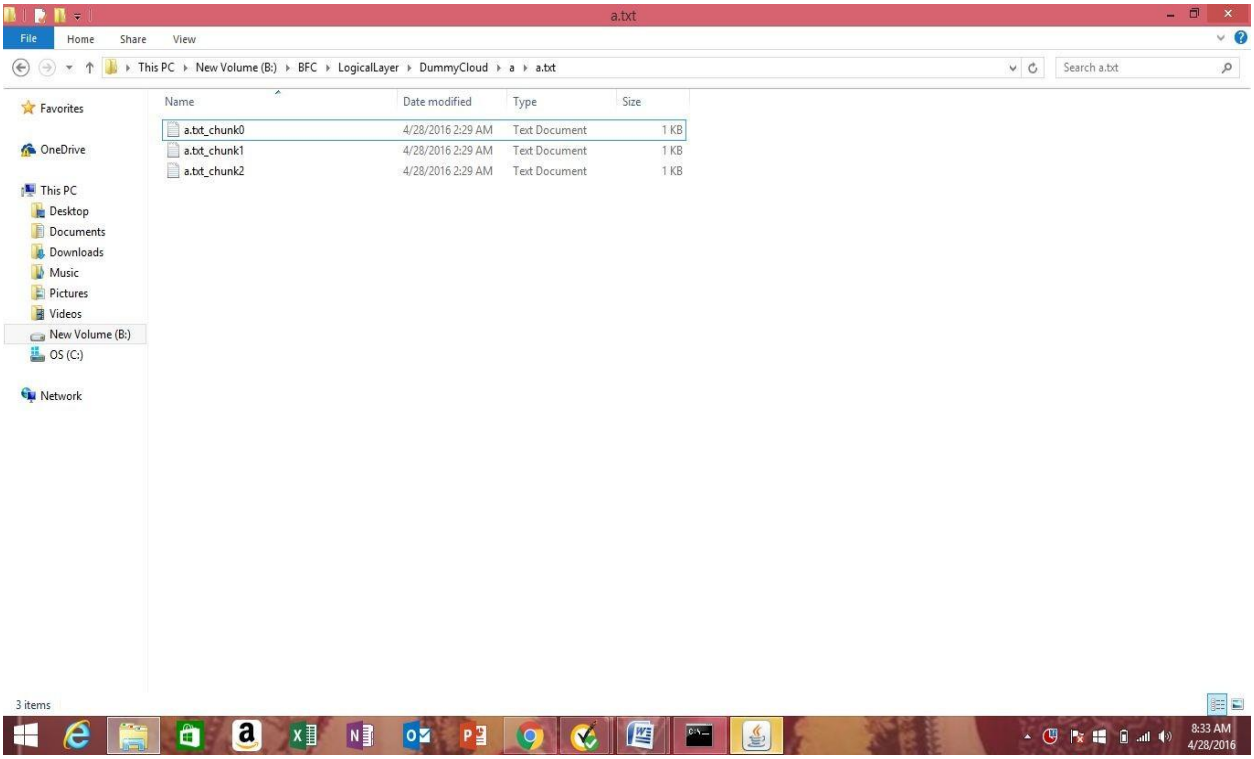




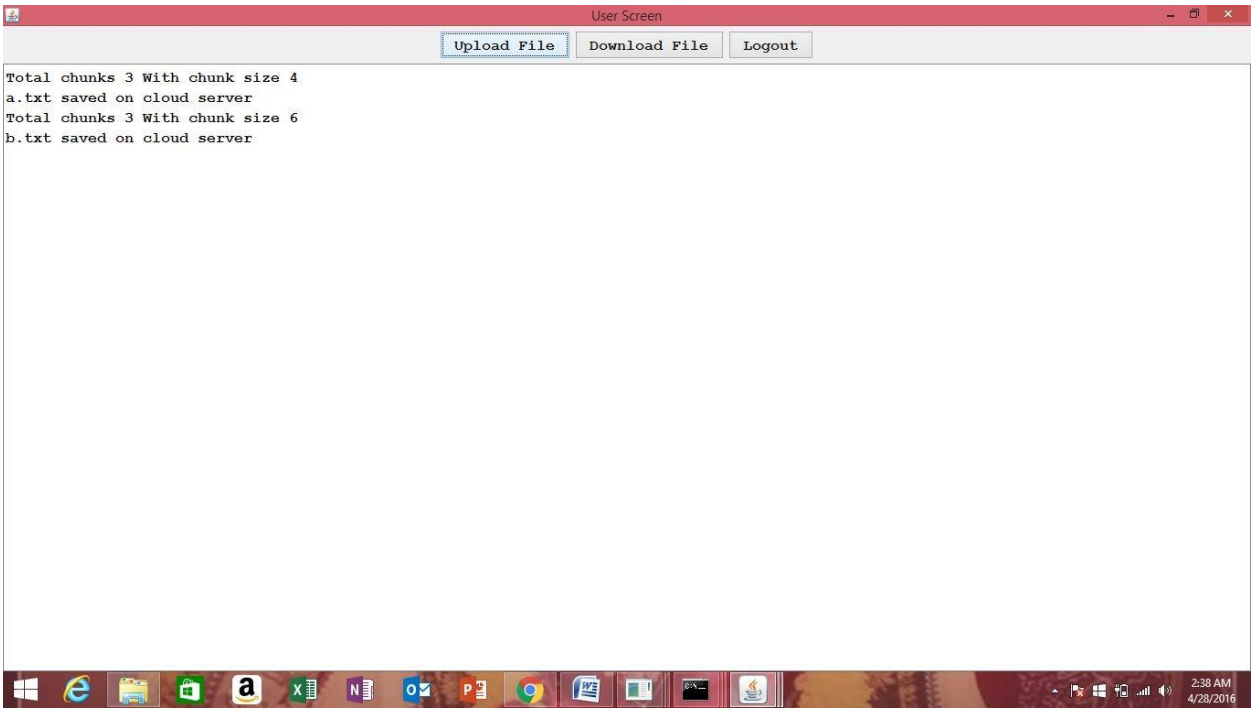
Click upload button to upload a file and select the file.



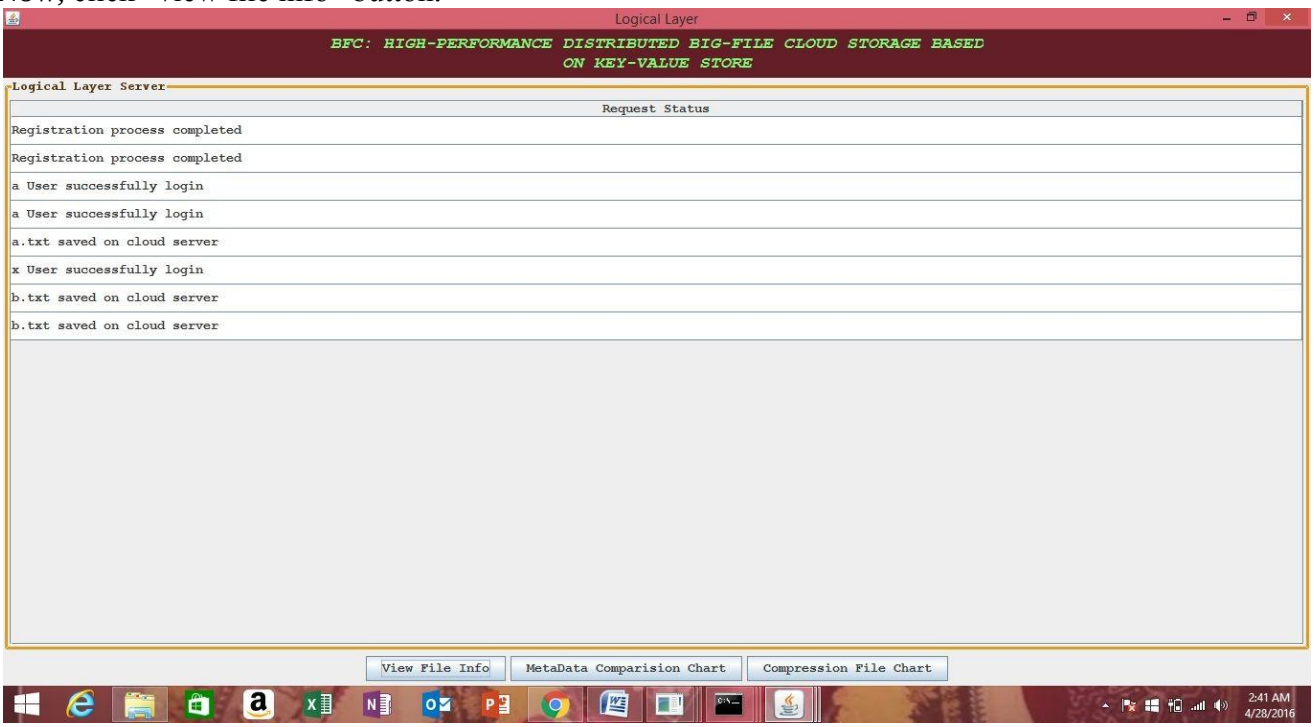
The uploaded file has been split into chunks and is saved on the cloud server in the encrypted format. In a similar way, we upload same and a different file from other user account.



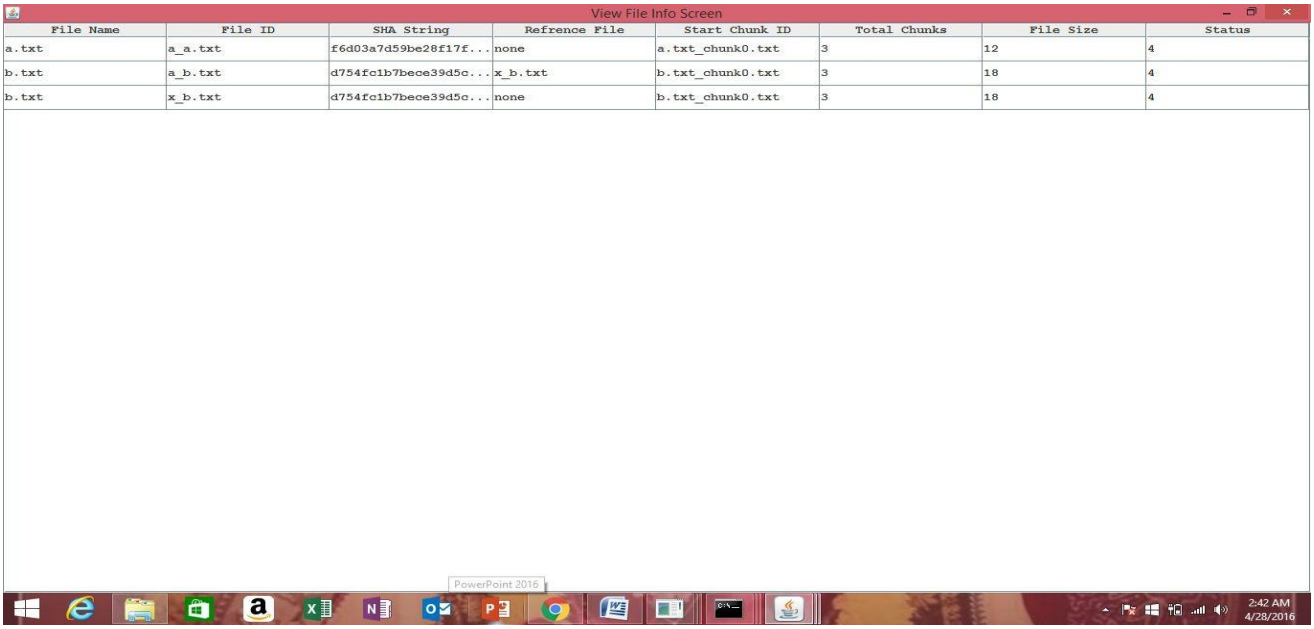
Login as the second user and upload the file.



In the Logical layer window, we can see user's activities, such as login and operations. Now, click “view file info” button.



Click on “View File Info”.

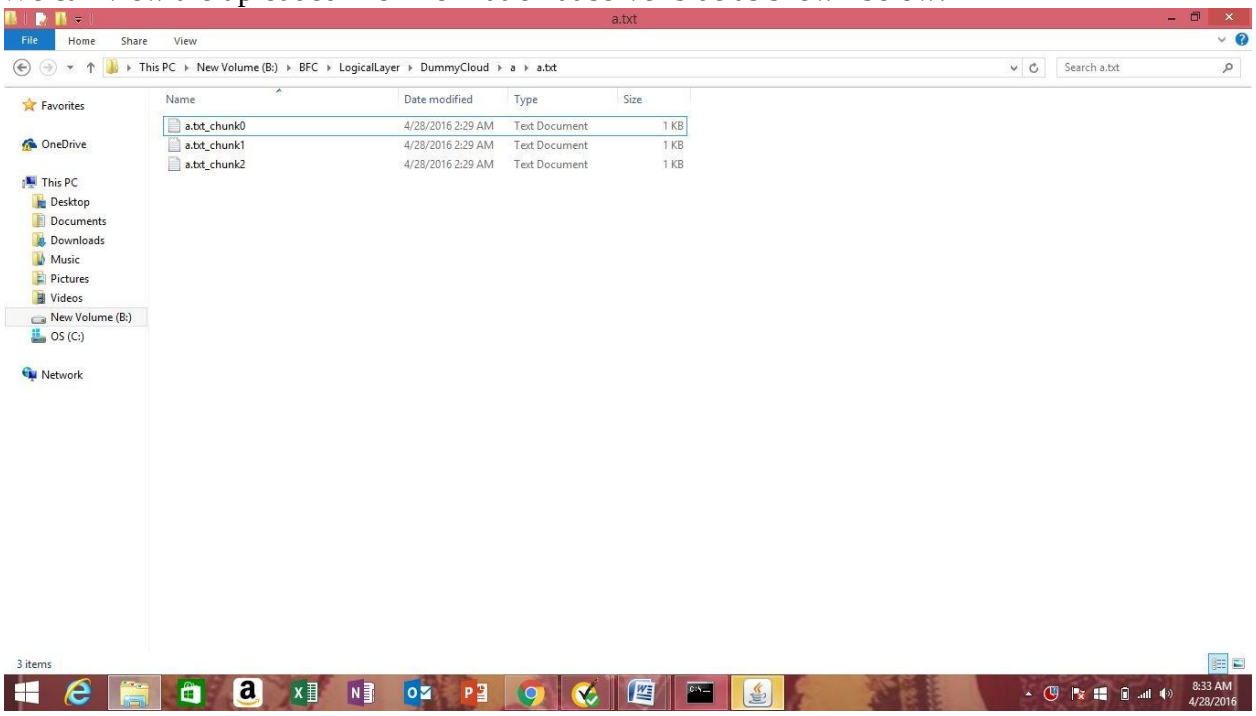


In the above, we can see the uploaded files and their information. It has details like the file name and id, its “SHA” value, reference file, start chunk id, total chunks, file size and status of the file uploaded.

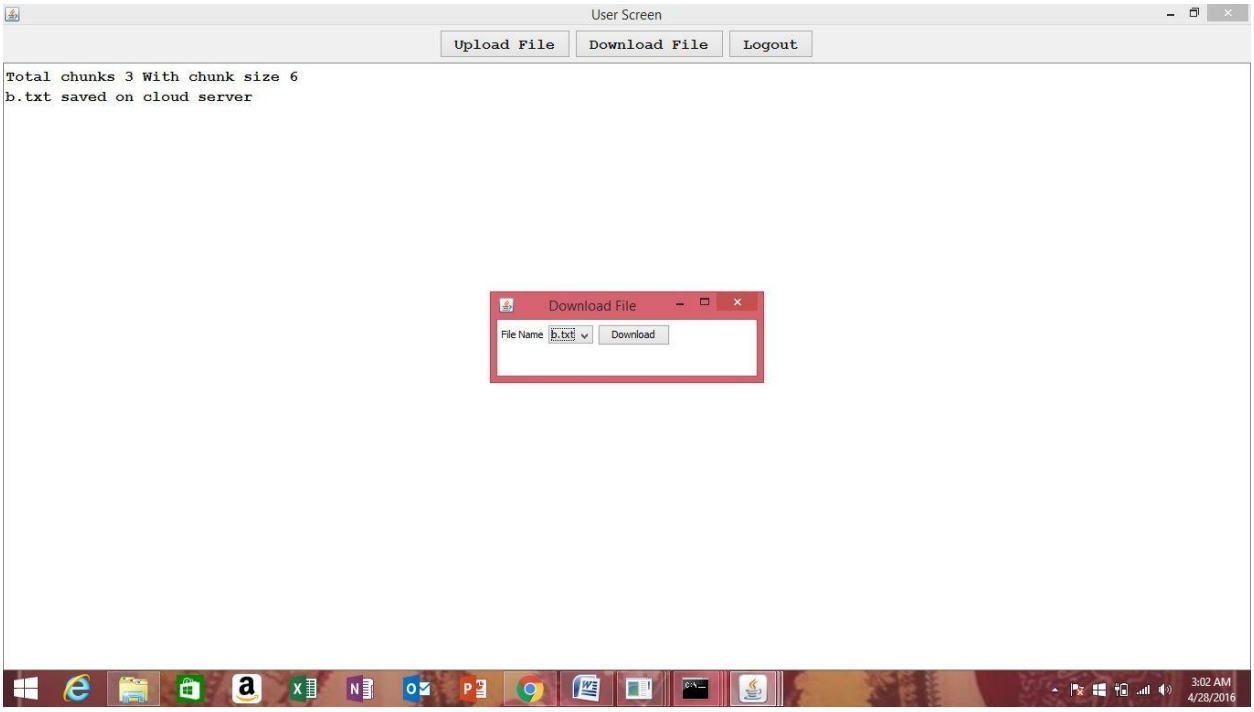
In the above window, we can see that the user “x “ has uploaded a file “b.txt” and user “a” has uploaded two files they are: “a.txt and b.txt”. The file "b.txt" was already uploaded by user “x”, as the same file was already available at server side, so it won’t upload for the next time (when

user "a" tries to upload same "b.txt" file) instead, it will tag to the already available file. By this, the deduplication is performed. “SHA” values and the no. of chunks for the file and their initial chunk id and the references are saved.

We can view the uploaded file information at server side as shown below:

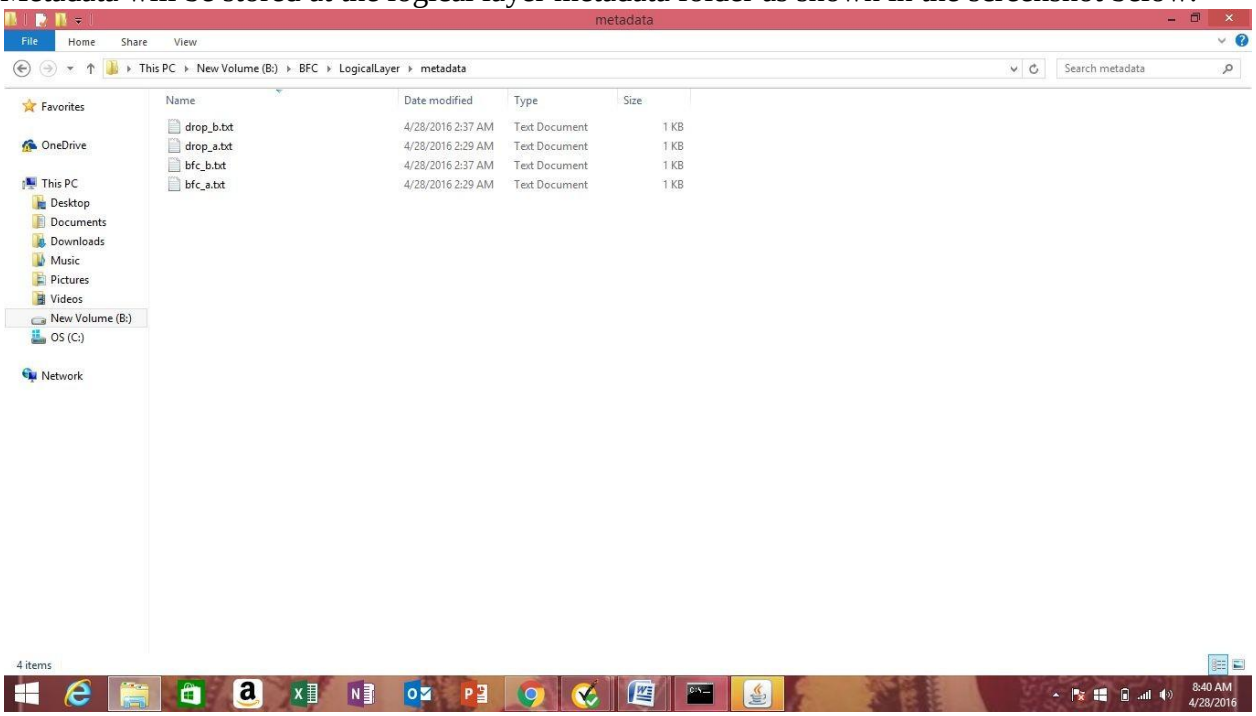


The user can download the uploaded file. To download open user screen and click download button a window will pop-up asking for the name of file to download , select name and click download.

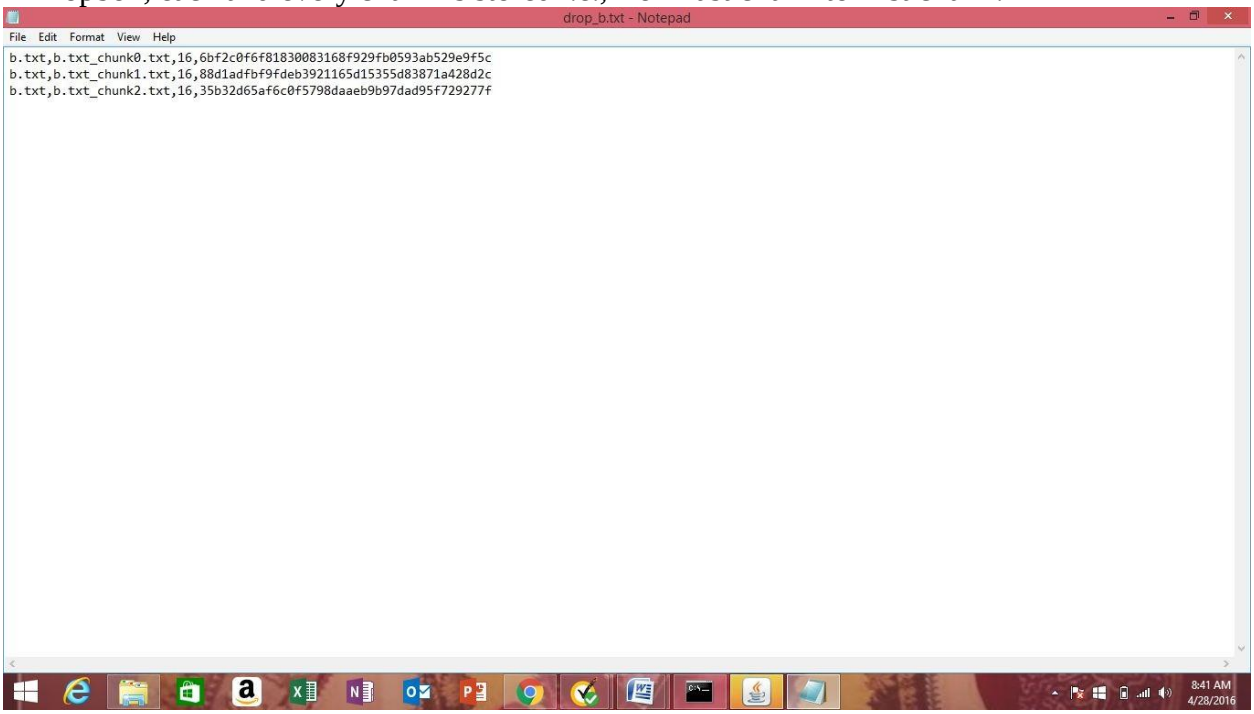


The file will be downloaded in D drive in the decrypted format.

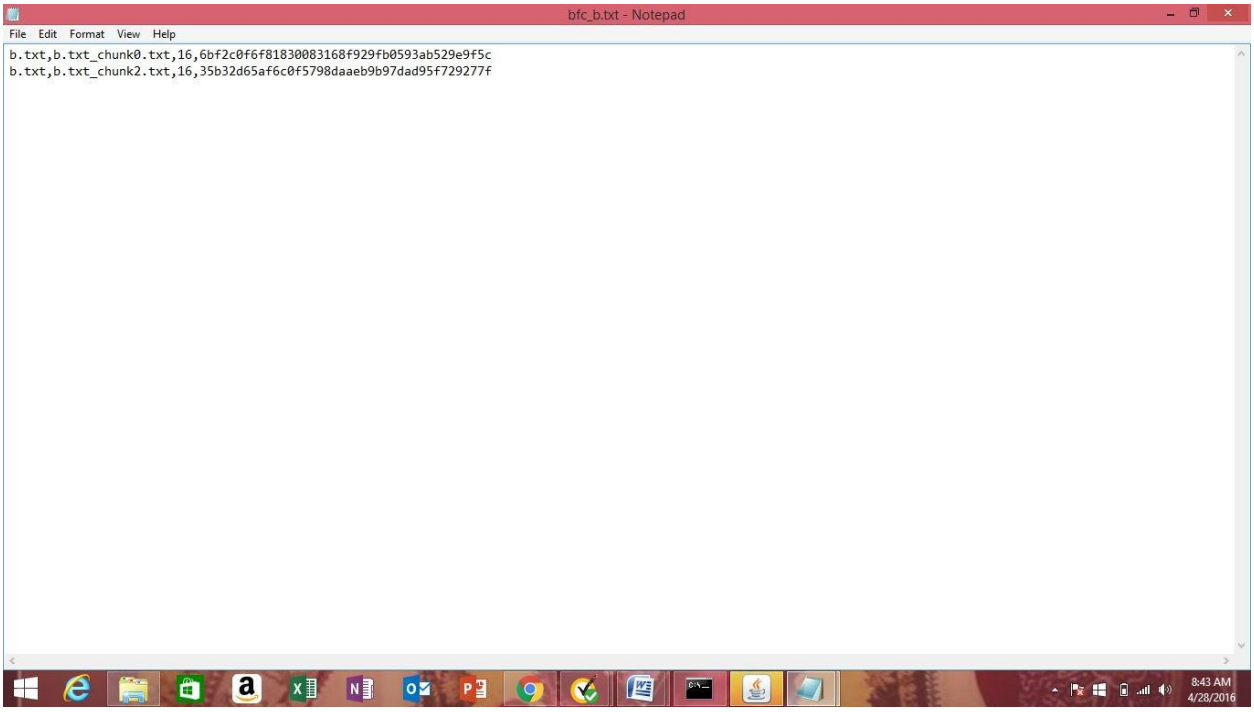
Metadata will be stored at the logical layer metadata folder as shown in the screenshot below.



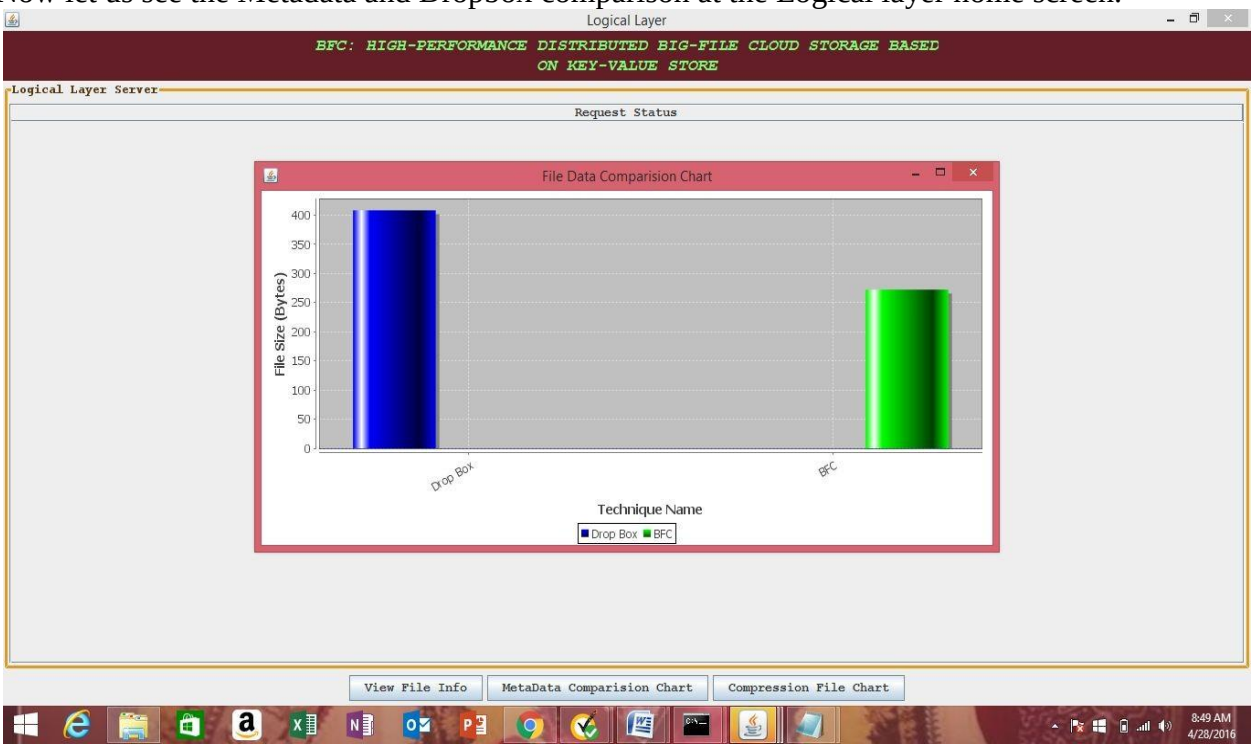
In Dropbox, each and every chunk is stored i.e., from last chunk to first chunk.



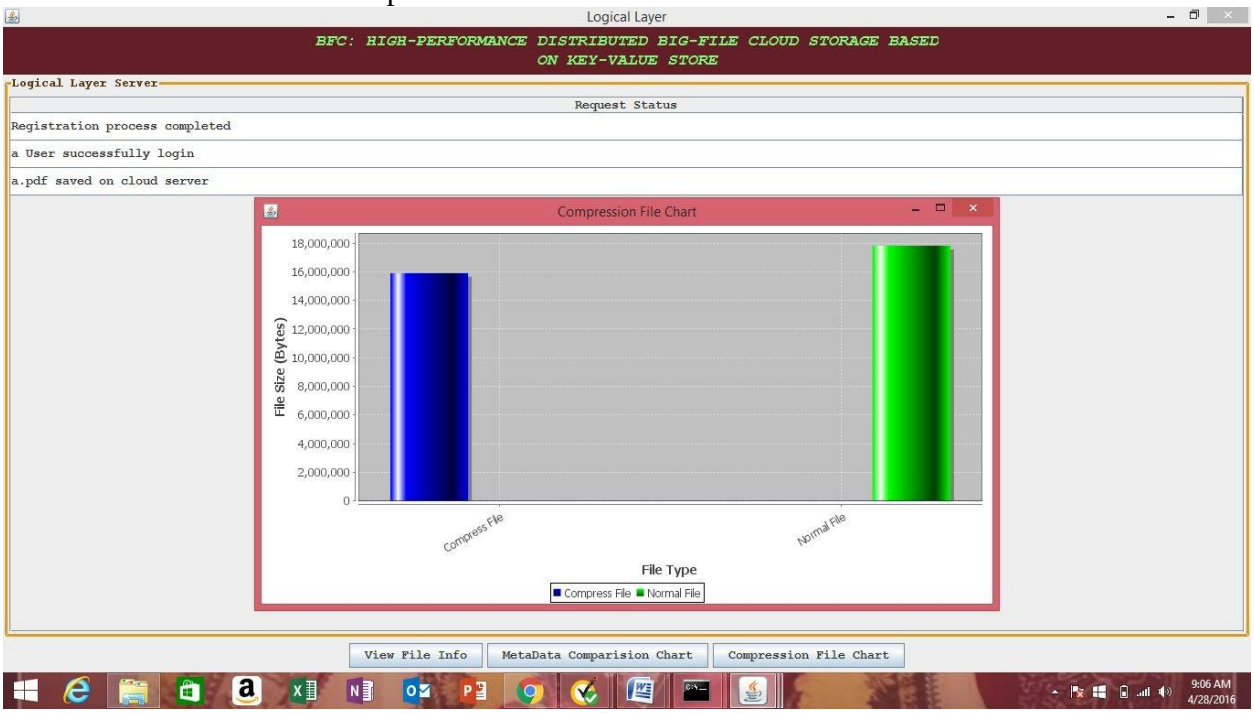
In Metadata, only the first chunk and the last chunk which are generated by the uploaded file is stored.



Now let us see the Metadata and Dropbox comparison at the Logical layer home screen.



When we are trying to upload large files, in order to reduce space we first compress and store in the cloud. Now let us see Compression File chart.



6.2 Test Cases

Test Case	Test Case Name	Test Case Desc	Test Steps			Test Case Status	Test Priority
			Step	Expected	Actual		
Logical layer 01	Logical layer Server	Start the Logical layer server	If the server is not started	Updated details will not be viewed here	Server is started	High	High

Register 02	Registration	New user has to enter all details to register	If the user is not registered	User can't perform further operation	Registration process is completed	High	High
Upload 03	Upload file	User has to upload file into server	If we do not upload any file at server	No file will be there at server	File is uploaded and it is splitted into chunks based on their size & stored at server	High	High
View File 04	View File Info	At logical layer window we can view the file	If files are not uploaded	We can't view the file	We can't view all the file information	High	High
Download 05	Download a file	Select a file in cloud & download it	If we can't download any file	Nothing will happen	File is downloaded and saved in D-drive		

Table 1: Test Cases

Chapter 7

Conclusion and Future Predictions

7.1 Security in BFC

One of the crucial requirements for our system is Data confidentiality. In order to achieve that, in this project we use the AES algorithms as well as SHA1 algorithm.

AES Algorithm

Advanced Encryption Standard (AES) is one of the most regularly used and most secure encryption algorithms available. It is openly available, and it is the cipher which NSA uses in order to secure documents with the classification "top secret". This algorithm depends on many substitutions, permutations, and linear transformations, each executed on data blocks of 16 byte - accordingly the term blockcipher. These operations are repeated many times called "rounds". A novel round key is calculated out of the encryption key, and incorporated in the calculations, during each round. The distinction between AES-128, AES-192 and AES-256 is the length of the key: 128, 192 or 256 bit- all drastic improvements compared to the 56 bit key of DES. By breaking a 128 bit AES key with a supercomputer could take longer than the assumed age of the universe. There exists no practicable attack against AES up to this point. Thereby, AES remains the favored encryption standard for governments, banks and high security systems around the world.[4]

SHA-1

SHA-1 [32] is ordinarily utilized as a part of cryptographic applications where there is need for data integrity and is high. It will likewise be utilized to index hash functions and in identifying data corruption and checksum errors. It will produce a 160-bit hash value or message digests from the input (data that will require encryption) which will resemble the hash value of the MD5 [32] algorithm. It will use 80 rounds of cryptographic operations in order to encrypt and secure data object. Some of the protocols that will use SHA-1 include:

- Transport Layer Security (TLS)
- Secure Sockets Layer (SSL)
- Pretty Good Privacy (PGP)
- Secure Shell (SSH)
- Secure/Multipurpose Internet Mail Extensions (S/MIME)
- Internet Protocol Security (IPSec)

7.2 Conclusion

BFC has designed a simple metadata in order to create a high performance Cloud Storag.Everyfile in the system has a same size of metadata independent of file size . In BFC every big-file is split into numerous fixed size chunks. The chunks of a file have a contiguous ID range, thereby it is easy to distribute data and scale out storage system. The data deduplication method of BFC uses SHA-2 hash function to speed up in order to detect data-deduplication on server side. It is important to save storage space and network bandwidth when various users upload same static data. Compression is implemented as the extension of the project. The project is extendable to accommodate decompression.

7.3 Future Predictions

The predictions of cloud in the future are as follows:

Developers are trying to build many new softwares for the cloud , by this it is predicted that there will be no differentiation between public and private clouds. We can also predict that in the coming years. Almost a quarter of all applications will be accessible on the cloud. This can be estimated when you look at the fact that approximately 60 percent of enterprises will spend more than 10 percent of their budget on cloud services (predicted by IBM).

Nowadays, cloud based services are rapidly developing and turned into a rising pattern in information technology field. Due to this, enterprises are relying more and more on cloud service in order to develop, market and sales of products, supply chain management. It is also estimated that Software as a Service (SAAS) would grow at an outstanding rate with yearly growth rate of 20.2 percent. This implies that it will be growing from $18.2 billion in 2012 to $45.6 billion in 2017[14].

In the future, there will be increase in implementation of hybrid clouds. It is estimated that 50 percent of the enterprise will implement hybrid clouds by 2017. The hybrid cloud which is a combination of on and off premises will provide the best of both worlds: integration of strengths permitting organizations so as to attain the performance of on premises solutions further also the management assistance of the cloud business model [12]. One such example of dynamic cloud is connection of new apps to already existing systems like operation new mobile social apps on clouds[15].

There will be increase in the development for the cloud going forward. According to Evans Data Corporation [13] at present there are more than 18 million software developers globally so far less than 25 percent are developers for cloud. It is estimated that as cloud continues to develop rapidly, more developers will develop for cloud - it becomes evident when you look at the fact that 85 percent of the new software being built today is for cloud computing. Going forward, it is recommended that there will be growth in third party, commercial and enterprise developers and contributions to cloud application systems.

It is estimated that with rise in competition in the cloud storage systems will lead to better products, services and innovation. When developers work collectively as a group, focusing on the quality of work they should deliver, innovative cloud services will come into existence because of code collaboration and dynamic developer ecosystems [30].

End users are trying to drift utilizing separate physical servers which are connected to storage area networks to more composed reference architectures which would include every component essential in order to run their applications. Previously, servers and storage are sold as independent IT infrastructure elements. The shift to the cloud has changed all of that discussed above. Since everything is converged into one single solution and virtualized on a converged network, there might not be any need for a conventional storage only network [29].

References

[1] Dropbox tech blog. https://tech.dropbox.com/. Accessed October 28, 2014.

[2] Zing me. http://me.zing.vn. Accessed October 28,2014

[3] Facebook. http://facebook.com, 2014

[4] Federal Information Processing Standards Publication 197 ADVANCED ENCRYPTION STANDARD (AES) November 26, 2001

[5] I. Drago, E. Bocchi, M. Mellia, H. Slatman, and A. Pras. Benchmarking personal cloud storage. In Proceedings of the 2013 conference on Internet measurement conference, pages 205–212. ACM, 2013.

[6] I.Drago, M.Mellia, M. M Munafo, A.Sperotto, R.Sadre, and A. Pras. Inside dropbox: understanding personal cloud storage services. In Proceedings of the 2012 ACM conference on Internet measurement conference, pages 481-494.ACM, 2012.

[7] S. Ghemawat, H. Gobioff, and S-T. Leung. The google file system. In ACM SIGOPS Operating Systems Review, volume 37, pages 29-43.ACM, 2003.

[8] Thanh Trung Nguyen, Tin Khac Vu, Minh Hieu Nguyen ,Le Quy Don Technical University, Ha Noi, Distributed Big File Cloud Storage based on key value store Viet Nam †VNG Research, Viet Nam

[9] International Journal of Advanced Research in Computer Science and Software Engineering Volume 5, Issue 12, December 2015.

[10] Yan Kit Li, Xialofeng Chen, Patrick P.C.Lee Secured Authorized De-duplication Based Hybrid Cloud Approach.

[11] Oraclehttp://www.oracle.com/technetwork/articles/java/compress-1565076.html

[12] SDLC http://www.veracode.com/security/software-development-lifecycle

[13] http://archive.thoughtsoncloud.com/2014/05/explained-dynamic-cloud-kids/

[14] http://evansdata.com/

[15] http://www.thoughtsoncloud.com/2014/03/what-is-software-as-a-service-saas/

[16] http://archive.thoughtsoncloud.com/2014/05/building-hybrid-cloud-3-ways-dynamic-cloud-powers-innovation

[17] J. Stanek, A. Sorniotti, E. Androulaki, and L. Kencl. A secure data deduplication scheme for cloud storage. 2014.

[18] M. Placek and R. Buyya. A taxonomy of distributed storage systems. Reporte tecnico, Universidad de Melbourne, Laboratorio de sistemas ´ distribuidos y computo grid ´ , 2006.

[19] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. In Proceedings of the 7th Symposium on Operating Systems Design and Implementation, OSDI '06, pages 307–320, Berkeley, CA, USA, 2006. USENIX Association.

[20] D. Borthakur. Hdfs architecture guide. HADOOP APACHE PROJECT http://hadoop. apache. org/common/docs/current/hdfs design. pdf, 2008.

[21] https://storageservers.wordpress.com/2016/01/26/who-uses-data-deduplication-and-why/

[22] SHA2hash.https://w2.eff.org/Privacy/Digital_signature/?f=fips_sha_shs.standard.txt

[23] NISOhttp://www.niso.org/publications/press/UnderstandingMetadata.pdf

[24] WinZip Computing Announces WinZip 16: The Easiest Way to Send Large Files Fast - New ZipSend and ZipShare services mark WinZip's expansion into file sharing through email and Facebook

[25] http://www.freebsd.org/cgi/man.cgi?gzip
[26] Oraclehttp://docs.oracle.com/javase/6/docs/technotes/guides/jar/index.html
[27] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi. Web caching with consistent hashing. Computer Networks, 31(11):1203–1213, 1999.
[28] http://www.androidauthority.com/best-cloud-storage-apps-for-android-657338/
[29] http://www.computerworld.com/article/2473557/data-center/will-cloud-computing-kill-the-storage-area-network-.html
[30] http://www.thoughtsoncloud.com/2014/05/future-cloud-computing-5-predictions/
[31] http://dev.mysql.com/doc/refman/5.6/en/mysql.html
[32] Marc Stevens Attacks on Hash Functions and Applications. 19 June 2012.