

Department of Computer and Information Science

University of Massachusetts Dartmouth

**BFC: High Performance Distributed Big Cloud Storage**

Masters of Science

A Project in

Computer and Information Science

By

***Apoorva Ravikant Mane***

***Date:05/12/2017***

I grant the University of Massachusetts Dartmouth the non-exclusive right to use the work for making single copies of the work available to the public on a not-for-profit basis if the University's circulating copy is lost or destroyed.

**Student Name:** Apoorva Mane

Date: 5/12/2017

TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| SR. No. | TITLE | PAGE NO. |
|  |  |  |
|  | ABSTRACT | 4 |
|  | INTRODUCTION | 5 |
|  | SYSTEM DESCRIPTION | 8 |
|  | DESIGN | 10 |
|  | IMPLEMENTATION & TESTING | 12 |
|  | SCREENSHOTS | 14 |
|  | CONCLUSION & FUTURE WORK | 20 |
|  | REFERENCES | 21 |

# ABSTRACT

These days, cloud-based capacity administrations are quickly developing and turning into a rising pattern in information stockpiling field. There are numerous issues when planning a proficient stockpiling motor for cloud-based frameworks with a few prerequisites, for example, enormous document handling, lightweight meta-information, low inactivity, parallel I/O, deduplication, circulated, high adaptability.

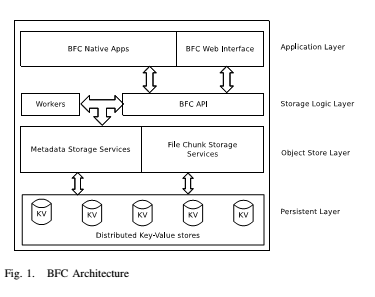
Key-worth stores assumed an essential part and indicated numerous points of interest when taking care of those issues. This paper presents about Big File Cloud (BFC) with its calculations and construction modeling to handle most of issues in a major document distributed storage framework in view of key value store. It is finished by proposing low-confused, settled size meta-information outline, which backings quick and exceedingly simultaneous, dispersed record I/O, a few calculations for resumable transfer, download and basic information deduplication technique for static information.

This examination connected the upsides of ZDB - an in-house key value store which was upgraded with auto-increase whole number keys for taking care of enormous document stockpiling issues proficiently. The outcomes can be utilized for building versatile appropriated information distributed storage that bolster huge document with size up to a few terabytes.

**INTRODUCTION**

**Cloud-based storage** services commonly serves millions of users with storage capacity for each user can reach to several gigabytes to terabytes of data. People use cloud storage for the daily demands, for example backing-up data, sharing file to their friends via social networks such as Face book, Zing Me. Users also probably upload data from many different types of devices such as computer, mobile phone or tablet. After that, they can download or share them to others. System load in cloud storage is usually heavy.

Thus, to guarantee an excellent quality of service for users, the system has to face many difficult problems and requirements: Serving intensity data service for a large number of users without bottle-neck; Storing, retrieving and managing big-files in the system efficiently; Parallel and resumable uploading and downloading; Data deduplication to reduce the waste of storage space caused by storing the same static data from different users. In traditional file systems, there are many challenges for service builder when managing a vast number of big file: How to scale system for the incredible growth of data, how to distribute data in a large number of nodes; How to replicate data for load-balancing and fault-tolerance; How to cache frequently accessed data for fast I/O, etc. A common method for solving these problems which is used in many Distributed File Systems and Cloud Storages is splitting big file to multiple smaller chunks, storing them on disks or distributed nodes and then managing them using a meta-data system. Storing chunks and meta-data efficiently and designing a lightweight meta-data are significant problems that cloud storage providers have to face. After a long time of investigating, we realized that current cloud storage services have a complex meta-data system; at least the size of metadata is linear to the file size for every file. Therefore, the space complexity of these meta-data system is *O*(*n*) and it is not well scalable for big-file. In this research, we propose new big-file cloud storage architecture and a better solution to reduce the space complexity of meta-data.

****

**Key-Value** stores have many advantages for storing data in data-intensity services. They often outperform traditional relational databases in the ability of heavy load and large-scale systems. Key -Value pairs have grown immensely both academically & industrially. Small and medium key-value pair size have given low-latency response time & better scalability. Present key-value stores are insufficient for directly storing big-values, or big files. This project executed several experiments in which we put whole file-data to key-value store, the system did not have superior performance as usual for many reasons: firstly, the latency of put/get operation for big-values is high, thus it affects other concurrent operations of key-value store service and multiple parallel accesses to different value reach limited. Secondly, for a bigger value, there is no more space to cache other objects in main memory for faster operations access. This makes scaling of a system difficult where number of users and data increase. This project is implemented to solve those problems for storing big-values or big-file using key-value stores. It brings many advantages of key-value store in data management to design a cloud-storage system called Big File Cloud (BFC). These are our contributions in this research: – Propose a light-weight meta-data design for big file. very file has nearly the same size of meta-data. BFC has *O* (1) space complexity of meta-data of a file, while size of meta-data of a file in Dropbox, HDFS has space complexity of *O*(*n*) where *n* is size of original file. Those make it easier to distribute data and scale-out the storage system. – Bring the advantages of key-value store into big-file data store which is not default supported for big-value. ZDB is used for supporting sequential write, small memory-index overhead. These contributions are implemented and evaluated in Big File Cloud (BFC) that serve storage for Zing Me Users. Disk Image files of VNG’s CSM Boot diskless system are stored in Big File Cloud.

**SYSTEM DESCRIPTION**

* **Existing System**

People use cloud storage for the daily demands, for example backing-up data, sharing file to their friends via social networks such as Face book [3], Zing Me [2]. Users also probably upload data from many diverse types of devices such as computer, mobile phone or tablet. After that, they can download or share them to others. System load in cloud storage is usually heavy. Thus, to guarantee an excellent quality of service for users, the system has to face many difficult problems and requirements.

**Disadvantages:**

* Storing, retrieving and managing big-files in the system efficiently.
* Parallel and resumable uploading and downloading.
* Data deduplication to reduce the waste of storage space caused by storing the same static data from different users.
* **Proposed System**

A common method for solving these problems which is used in many Distributed File Systems and Cloud Storages is splitting big file to multiple smaller chunks, storing them on disks or distributed nodes and then managing them using a meta-data system. Storing chunks and meta-data efficiently and designing a lightweight meta-data are significant problems that cloud storage providers must face.

After a long time of investigating, we realized that current cloud storage services have a complex meta-data system; at least the size of metadata is linear to the file size for every file. Therefore, the space complexity of these meta-data system is O (n) and it is not well scalable for big-file. In this research, we propose new big-file cloud storage architecture and a better solution to reduce the space complexity of meta-data.

**Advantages:**

* Propose a light-weight meta-data design for big file. Very file has nearly the same size of meta-data.
* Propose a logical contiguous chunk-id of chunk collection of files. That makes it easier to distribute data and scale-out the storage system.
* Bring the advantages of key-value store into big-file data store which is not default supported for big-value. ZDB is used for supporting sequential write, small memory-index overhead.

**SPECIFICATIONS**

**User Interface**

The user interface of this system is a user-friendly Java Graphical User Interface.

**Software Interfaces**

The required software is JAVA1.6.

**Operating Environment**

Windows 10

**HARDWARE REQUIREMENTS:**

* System : I-5 Processor
* Hard Disk : 500 GB.
* Floppy Drive : 1.44 Mb.
* Monitor : 15 VGA Colour.
* Mouse : Logitech.
* Ram : 512 Mb.

**SOFTWARE REQUIREMENTS:**

* Operating system : Windows 10.
* Coding Language : JAVA
* Front end : AWT, Swings
* Database : MySQL

**DESIGN**



Fig: Class Diagram



Fig: Use Case Diagram



Fig: Sequence Diagram



Fig: Collaborative Diagram

**IMPLEMENTATION & TESTING**

**Modules**

1. **Application Layer**
2. **Storage Logical Layer**
3. **Object Store Layer**
4. **Persistent Layer**

**Module description:**

**Application Layer**: It consists of native software on desktop computers, mobile devices and web-interface, which allow user to upload, download and share their own files.

**Storage Logical Layer**: it consisted of many queuing services and worker services, ID-Generator services and all logical API for Cloud Storage System. This layer implements business logic part in BFC.

**Object Store Layer**: It contains many distributed backend services. Two important services of Object Store Layer are FileInfoService and ChunkStoreService. FileInfoService stores information of files. Y-value store mapping data from fileID to FileInfo structure. ChunkStoreService stores data chunks which are created by splitting from the original files that user uploaded.

**Persistent Layer**: it based on ZDB key-value store. There are many ZDB instances  
which are deployed as a distributed service and can be scaled when data growing.

**TESTING**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case Id** | **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 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 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 | High | High |

**SCREENSHOTS**

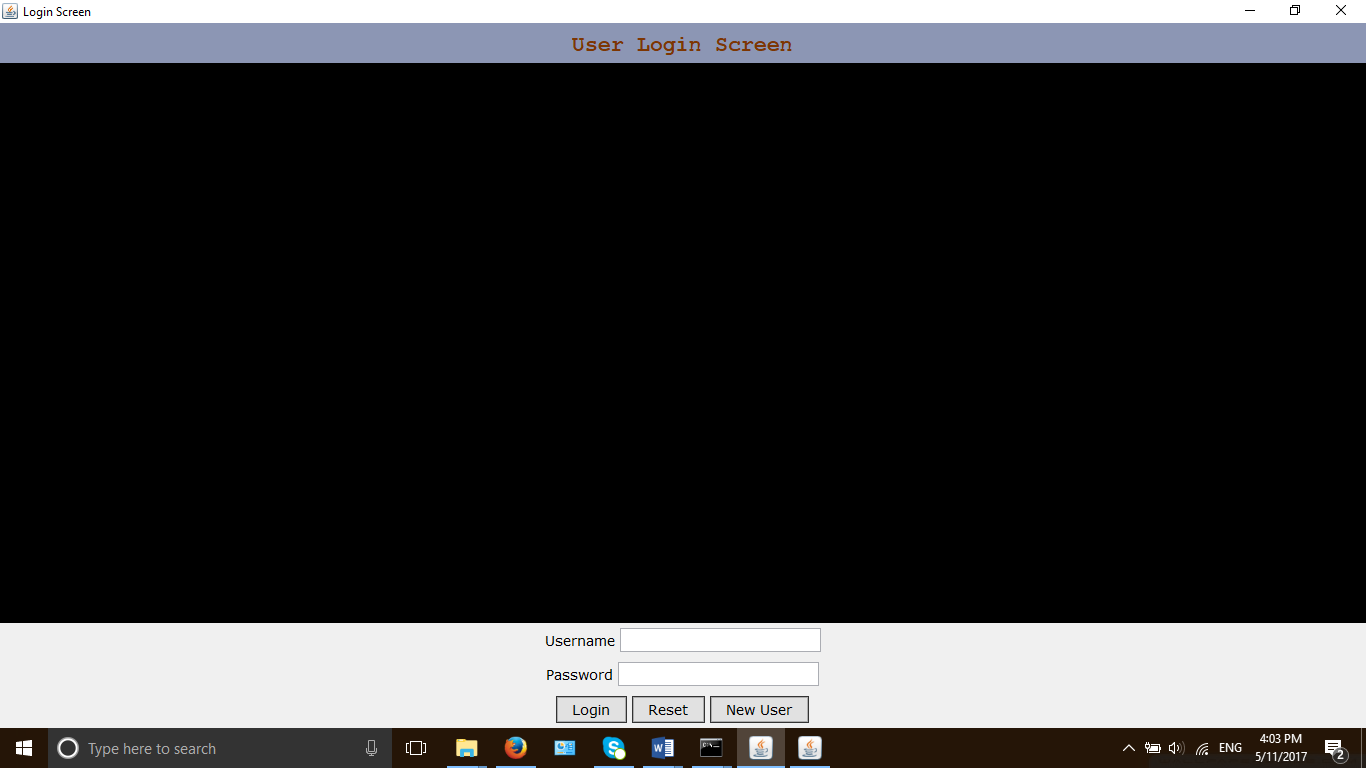


Fig 1: User Login Screen [Application Layer]

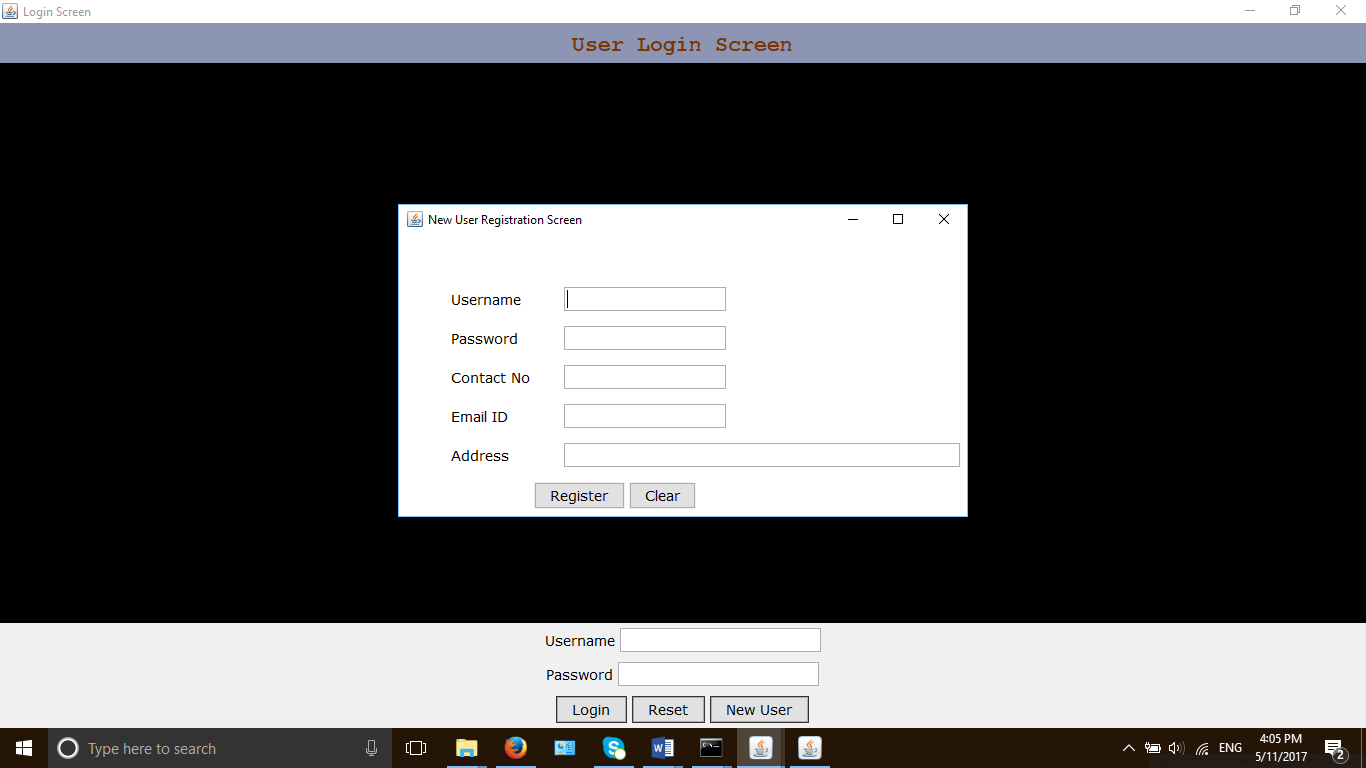


Fig 2: User Registration Screen [Application Layer]

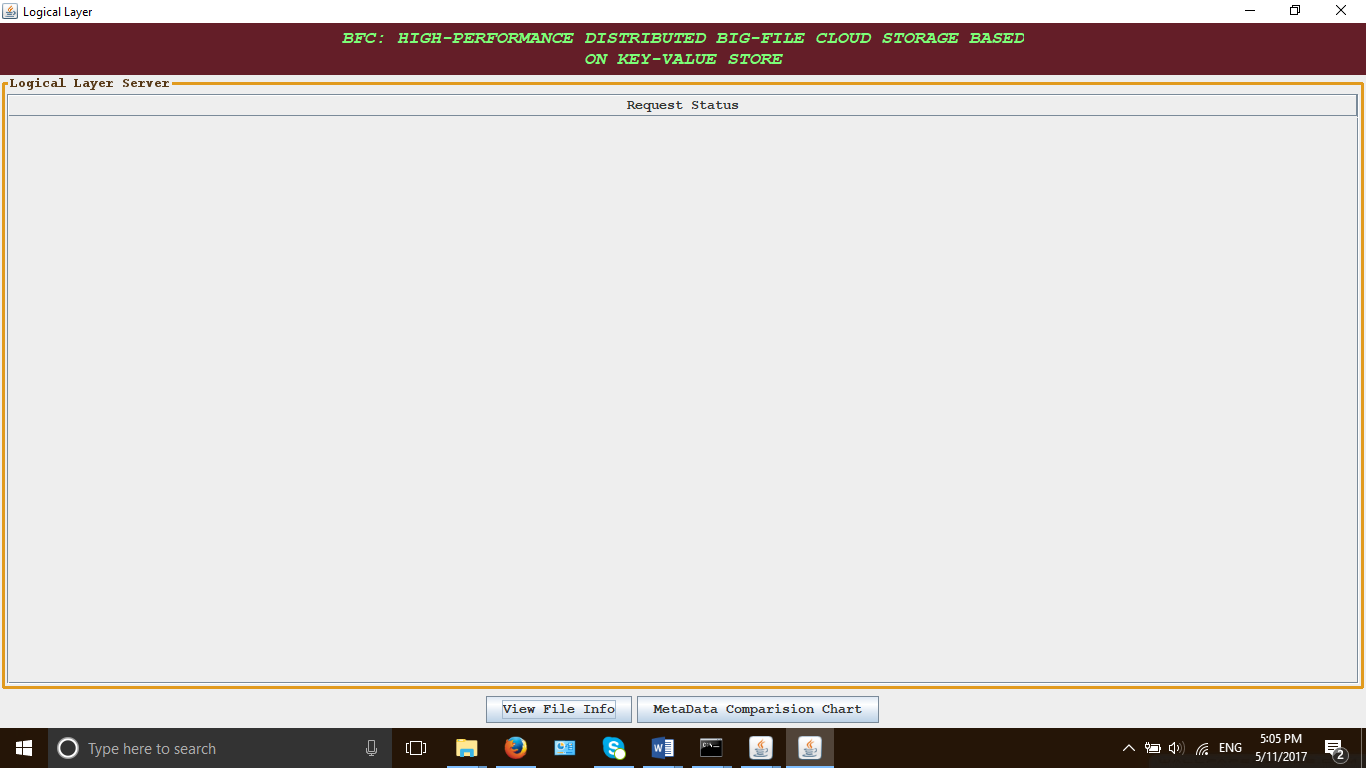


Fig 3: Logical Layer [Server end]

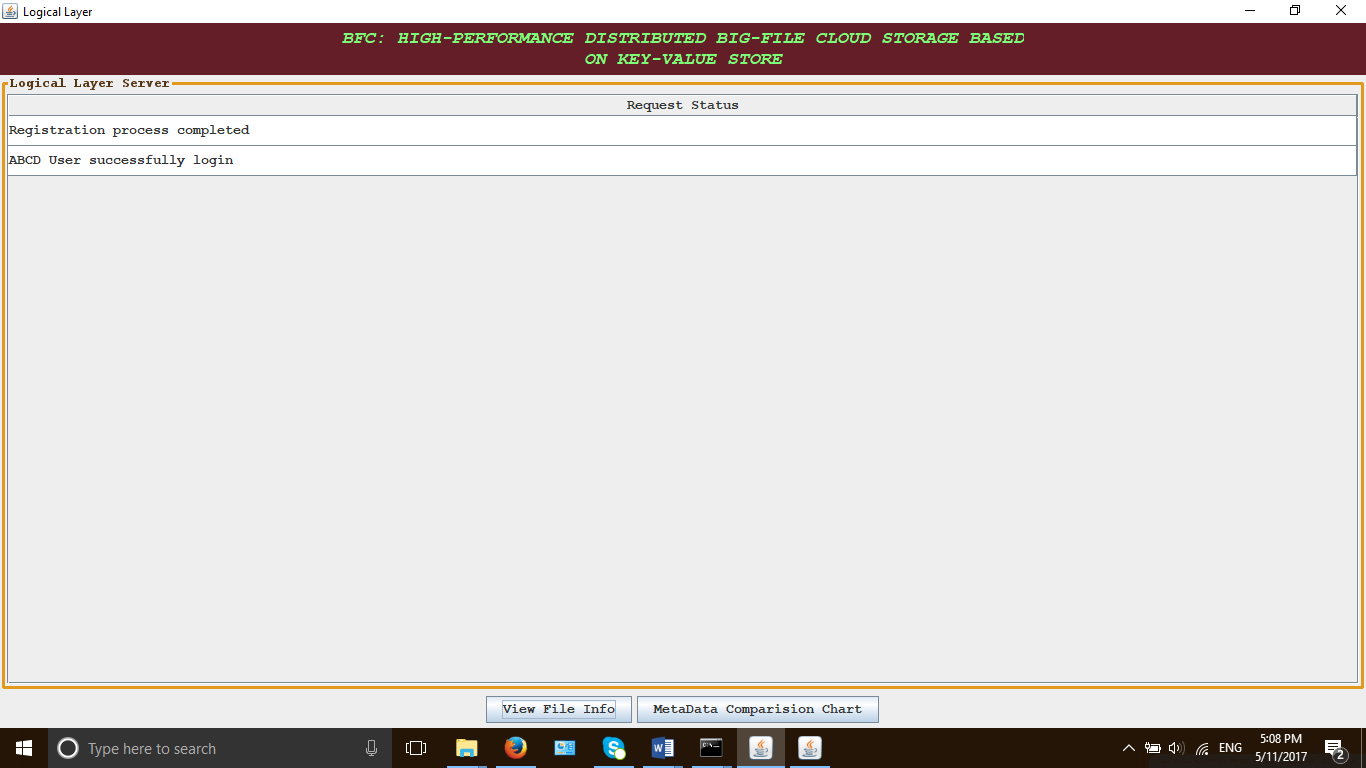


Fig 4: Request Status at server

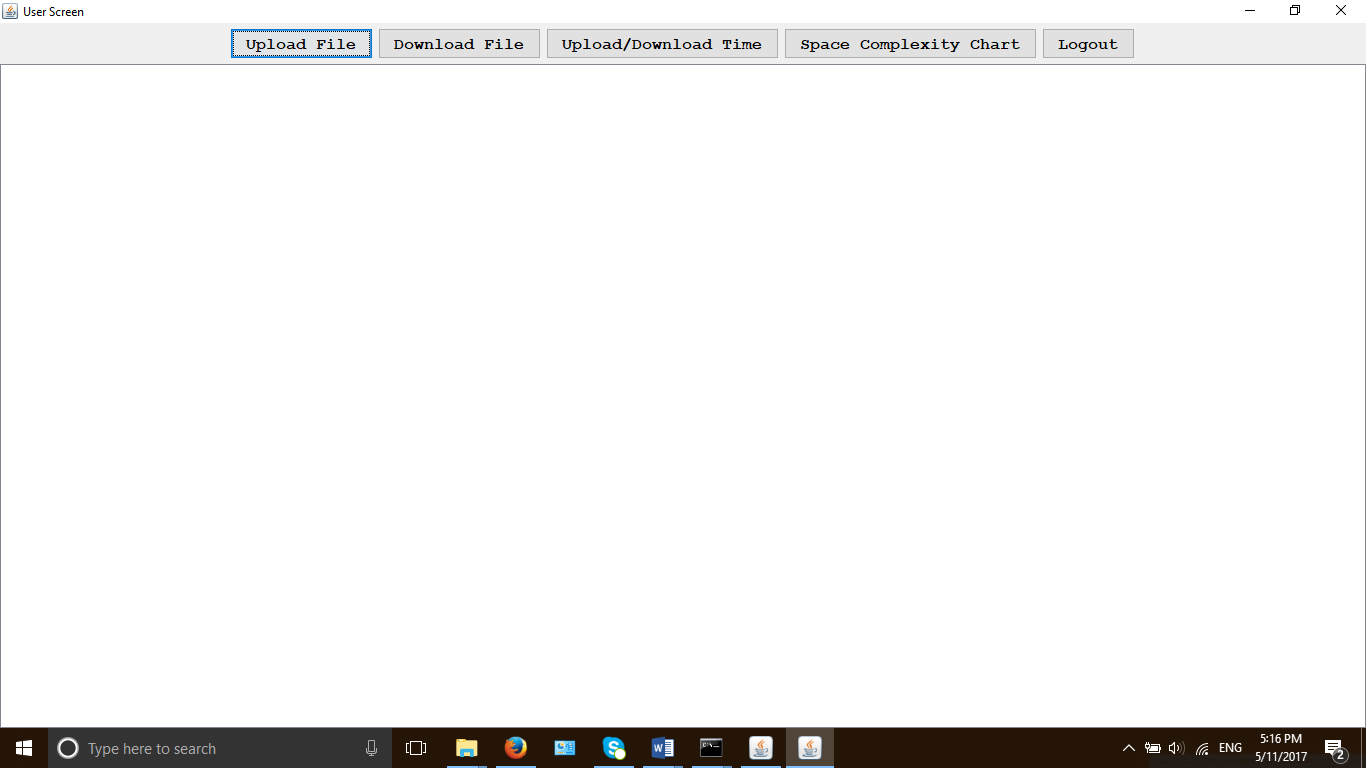


Fig 5: User Screen [Application Layer]

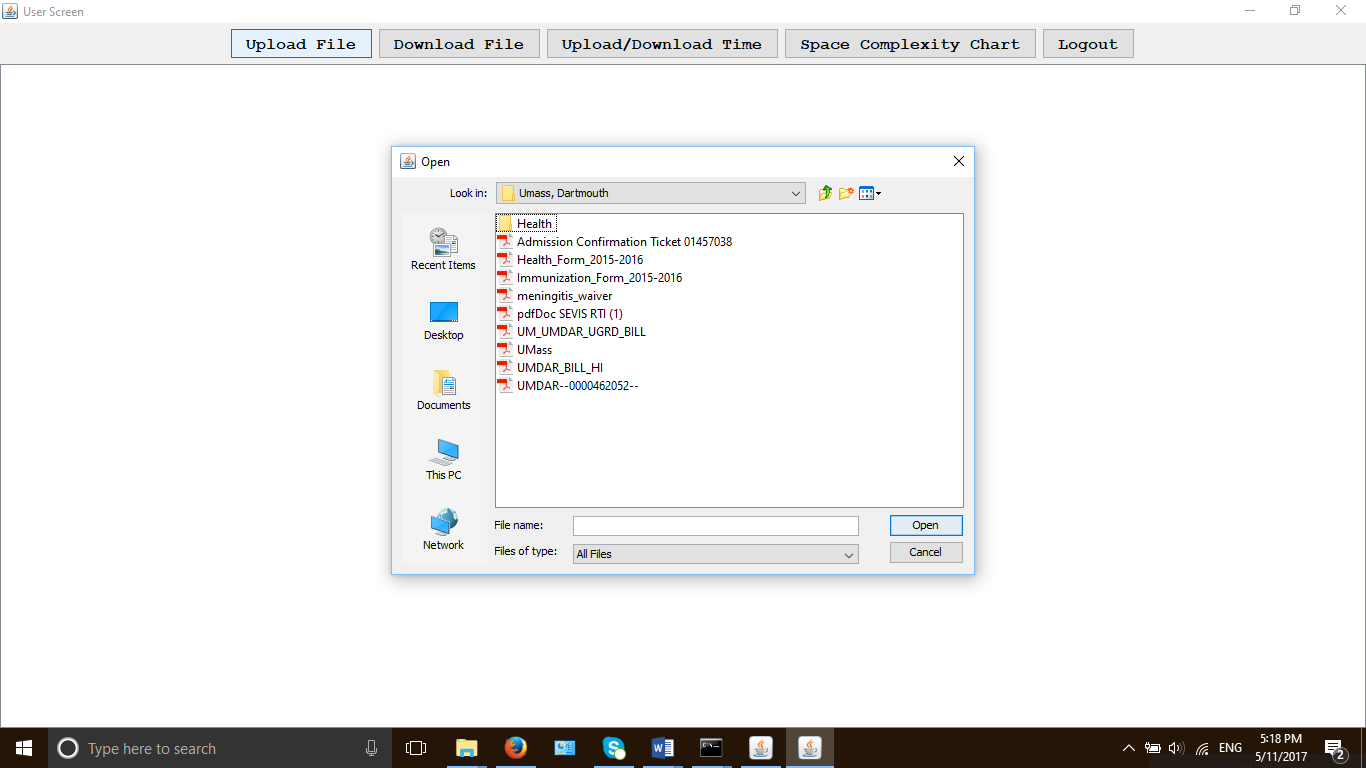


Fig 6: File Upload [Application Layer]

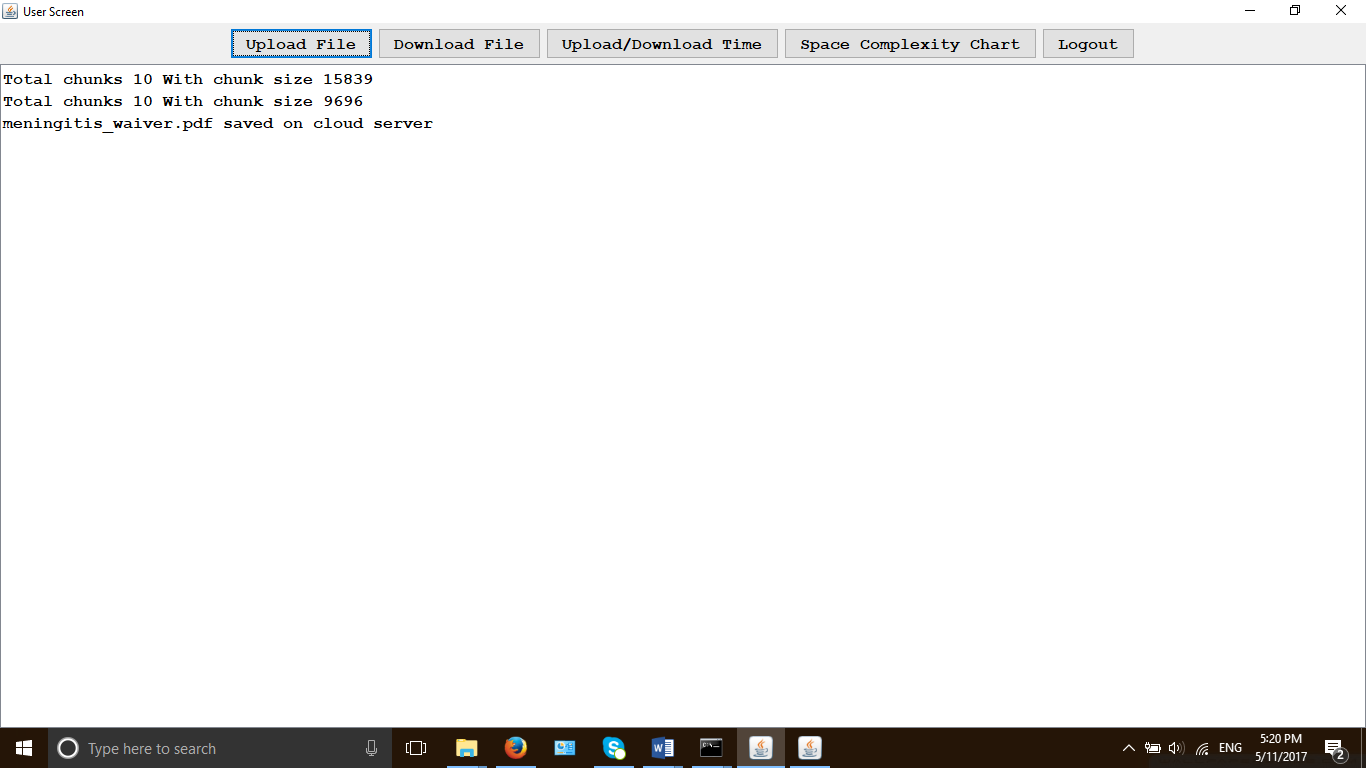


Fig 6: File Upload [Application Layer]

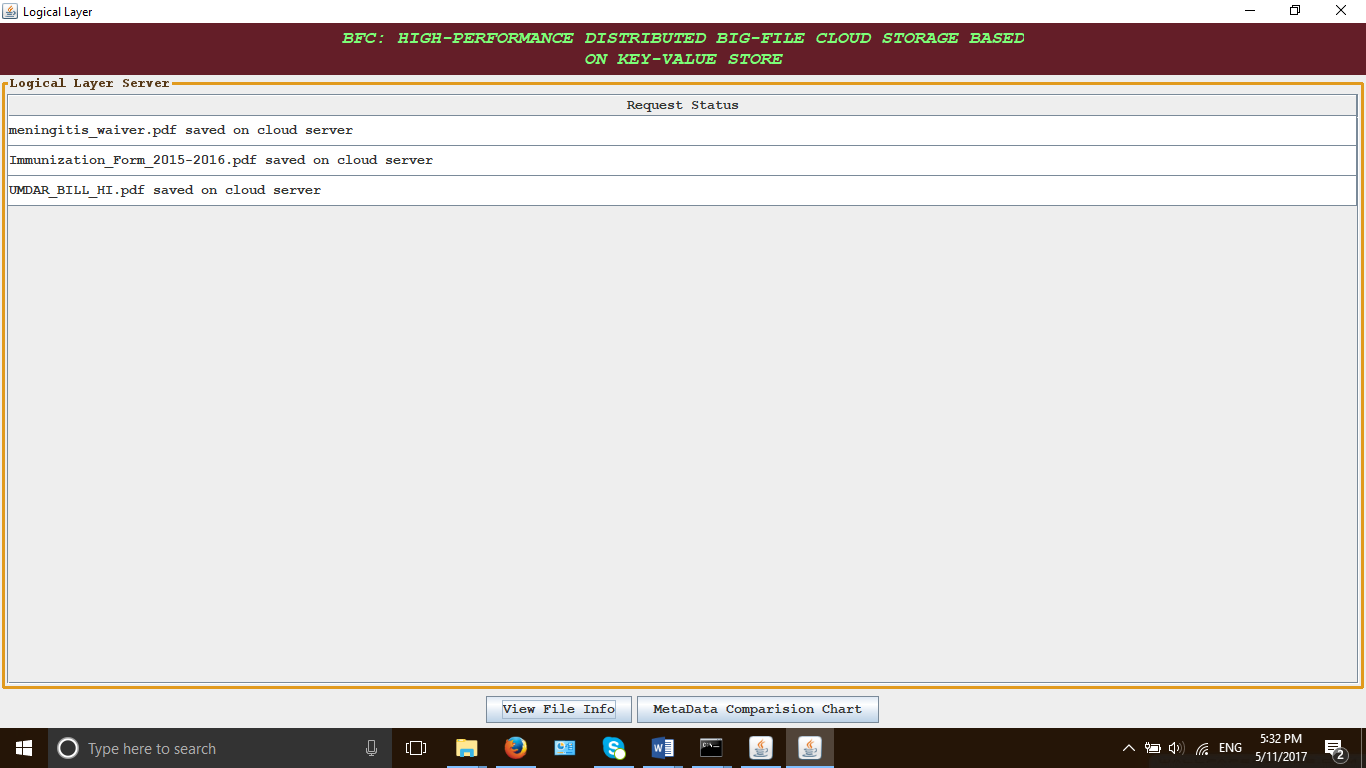


Fig 7: Logical Layer status after file uploading

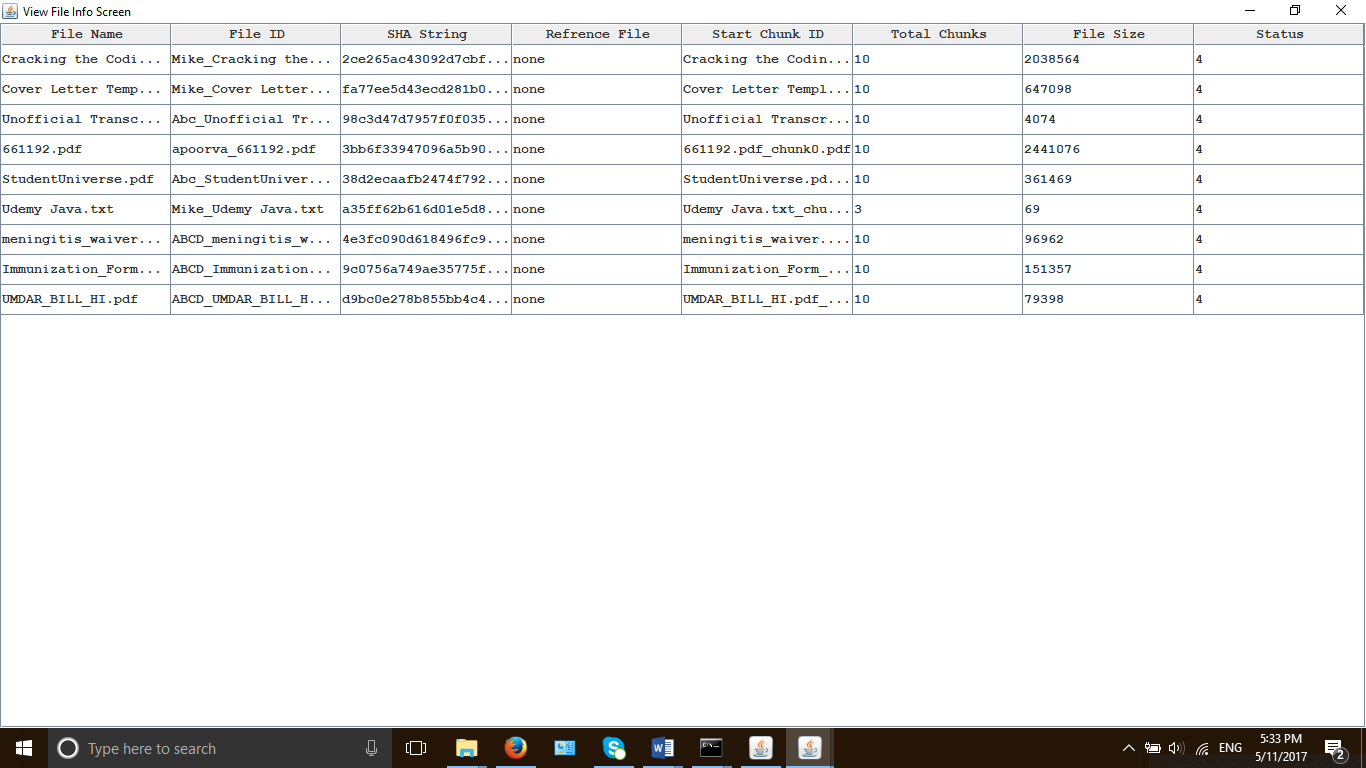


Fig 8: File Info Screen at Server Side [Logical Layer]

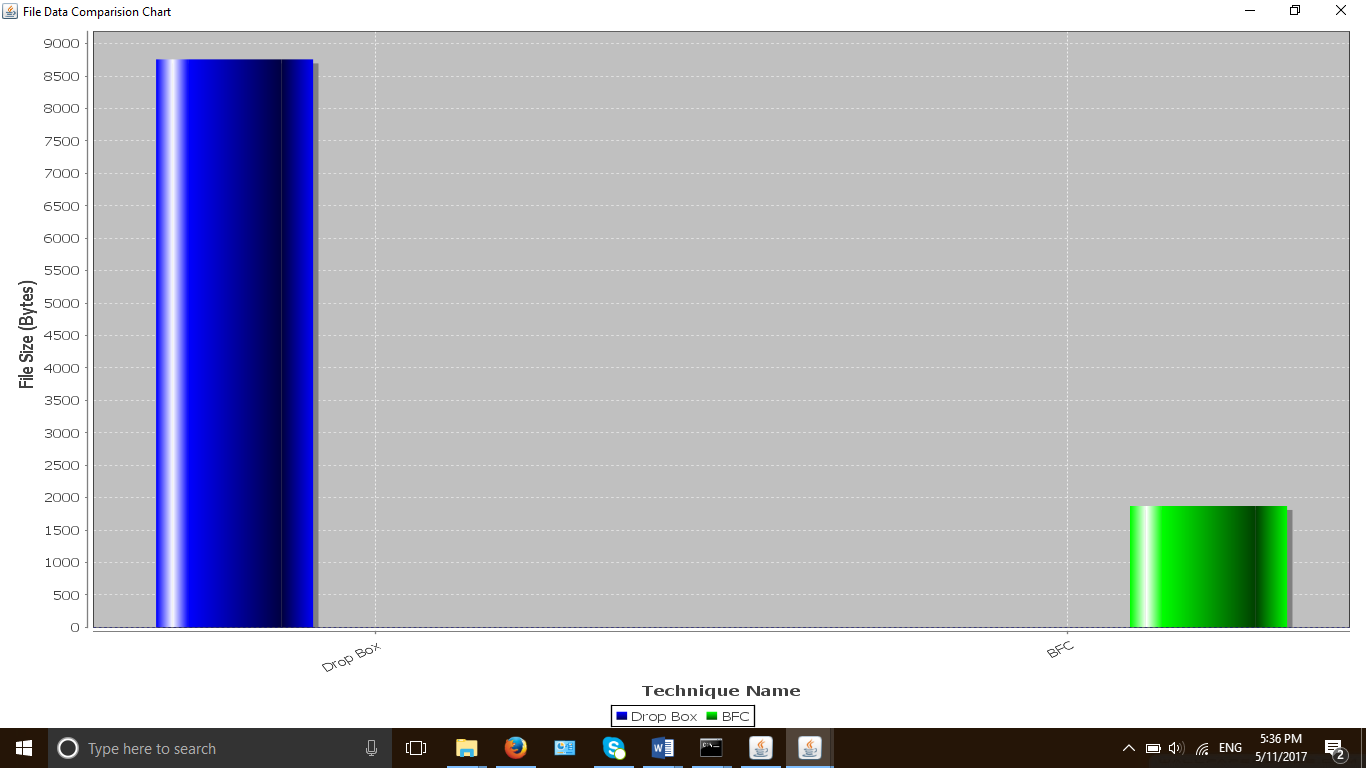


Fig 8: BFC & Dropbox Storage Comparison [Logical Layer]

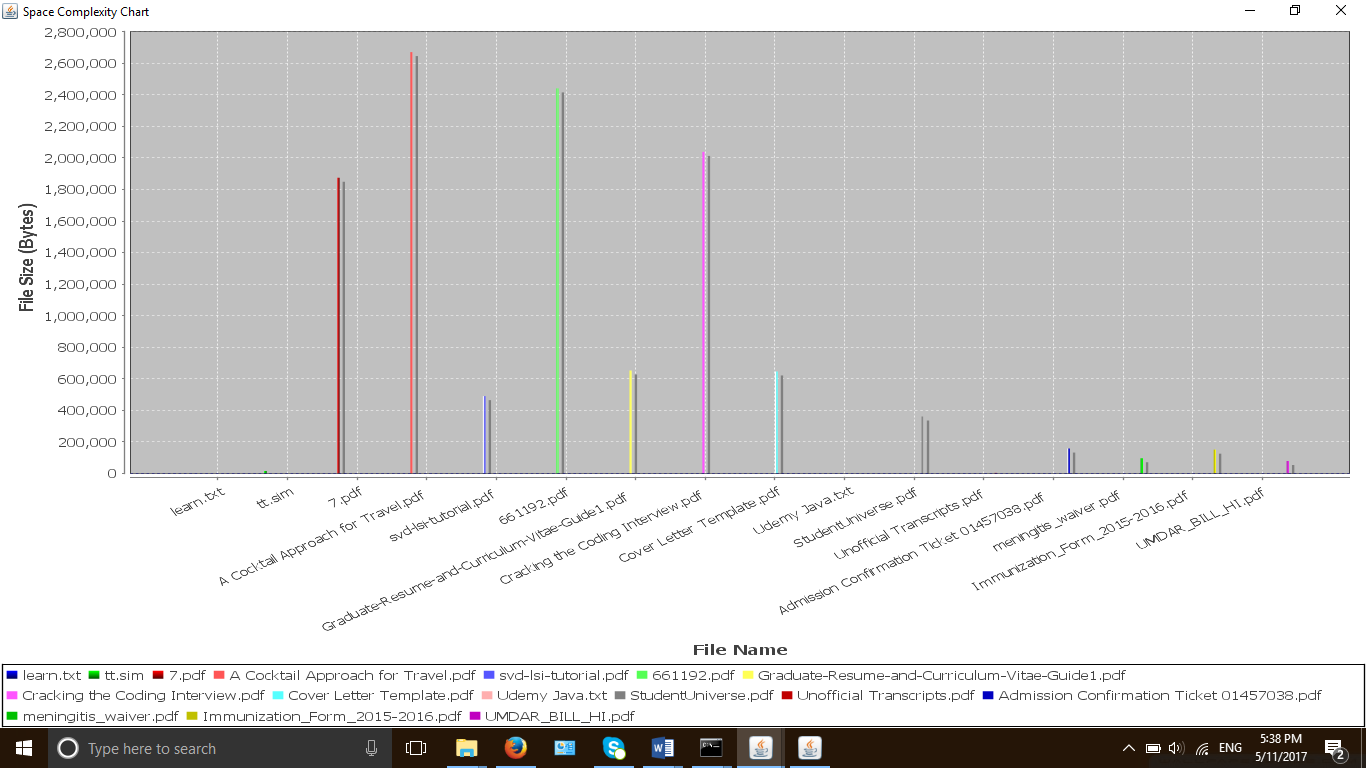


Fig 9: Space Complexity Chart

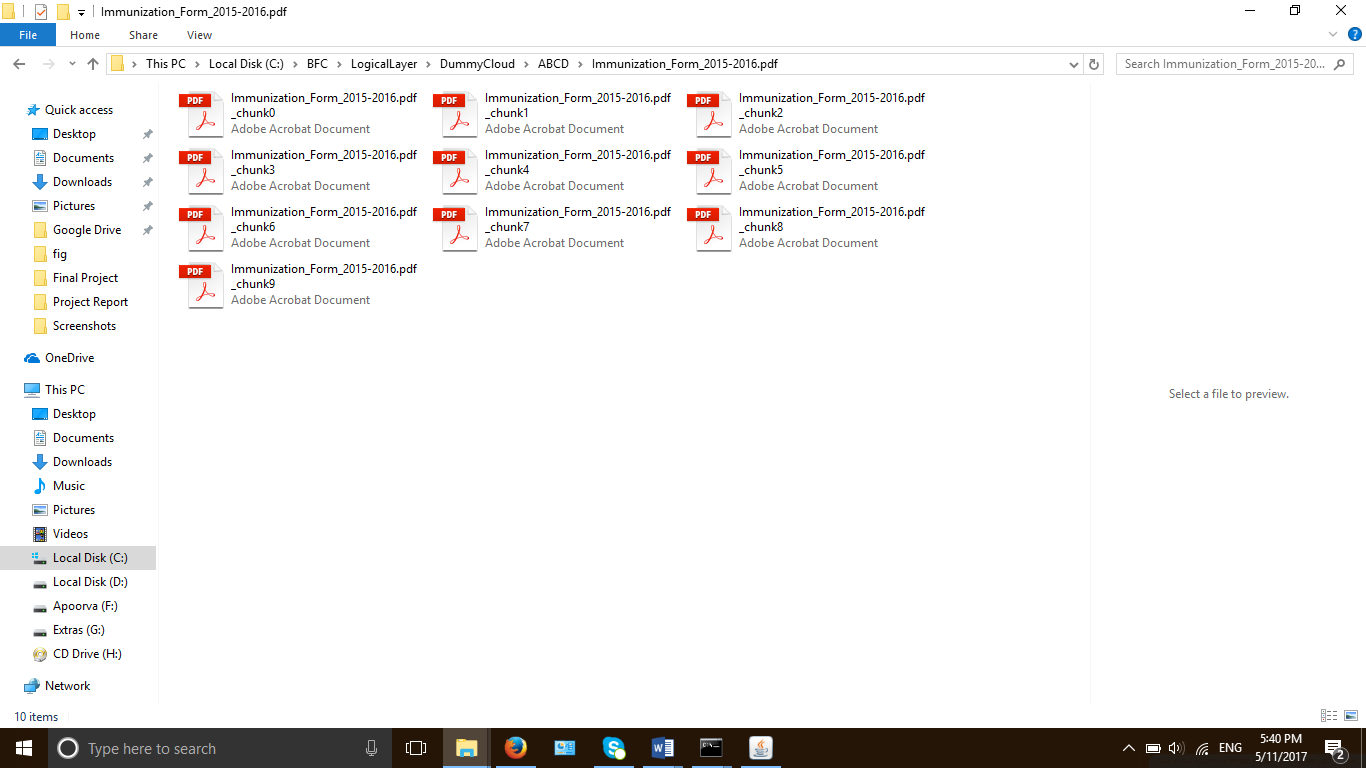


Fig 10: File Chunks at File System

**CONCLUSION**

BFC designed a simple meta-data to create a high-performance Cloud Storage based on ZDB key-value store. Every file in the system has a same size of meta-data regardless of file size. Every big-file stored in BFC is split into multiple fixed size chunks (may except the last chunk of file). The chunks of a file have a contiguous ID range; thus, it is easy to distribute data and scale-out storage system, especially when using ZDB.

This research also brings the advantages of key-value store into big-file data store which is not default supported for big-value. ZDB [16] is used for supporting sequential write, small memory-index overhead. The data deduplication method of BFC uses SHA-2 hash function and a key-value store to fast detect data-duplication on server-side. It is useful to save storage space and network bandwidth when many users upload the same static data. In the future, we will continue to research and improve our ideas for storing big data structure in larger domain of applications, especially in the “Internet of things” trend.

**BIBLIOGRAPHY**

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

[2] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS), 26(2):4, 2008.

[3] L. Chappell and G. Combs. Wireshark network analysis: the official Wireshark certified network analyst study guide. Protocol Analysis Institute, Chappell University, 2010.

[4] 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.

[5] 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.

[6] P. FIPS. 197: the official aes standard. Figure2: Working scheme with four LFSRs and their IV generation LFSR1 LFSR, 2, 2001.

[7] S. Ghemawat and J. Dean. Leveldb is a fast key-value storage library written at google that provides an ordered mapping from string keys to string values. https://github.com/google/leveldb. Accessed November 2, 2014.

[8] 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.

[9] Y. Gu and R. L. Grossman. Udt: Udp-based data transfer for high-speed wide area networks. Computer Networks, 51(7):1777–1799, 2007.

[10] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. Zookeeper: wait-free coordination for internet-scale systems. In Proceedings of the 2010 USENIX conference on USENIX annual technical conference, volume 8, pages 11–11, 2010.