CLOUDED UP : DEDUPLICATION WITH SECURE

STORAGE

A PROJECT REPORT

Submitted by

VIKRAM ARIKATH [RA1411003010701]

THONDA LAKSHMI HARSHITHA [RA1411003010703]

Under the guidance of

Mrs. S. SHARANYA, M.E.

(Assistant Professor, Department of Computer Sciene & Engineering)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY



S.R.M. Nagar, Kattankulathur, Kancheepuram District

MAY 2018

SRM UNIVERSITY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report titled “CLOUDED UP : DEDU-PLICATION WITH SECURE STORAGE” is the bonafide work of “VIKRAM ARIKATH [RA1411003010701], THONDA LAKSHMI HARSHITHA [RA1411003010703], , , ”, who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project re-port or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Mrs. S. SHARANYA, M.E.

GUIDE

Assistant Professor

Dept. of Computer Sciene & Engi-neering

SIGNATURE

Dr. B. AMUTHA

HEAD OF THE DEPARTMENT

Dept. of COMPUTER SCIENCE

AND ENGINEERING

Signature of the Internal Examiner Signature of the External Examiner

ABSTRACT

This project deals with a hybrid cloud system where the private cloud is responsible for promoting deduplication of files and the public cloud is used to store the encoded files for file sharing. Deduplication is the process of preventing duplicates of the same file being uploaded into a cloud thus ensuring better storage management. The duplicates of the files are checked by comparing the file name and the MD5 hash value that is being generated by each file when it is getting uploaded and this checking process happens in the private cloud system. To promote a more secure storage, the file that is being uploaded is encoded using UTF-8 and Base64 methods to ensure that the file is accessed by the user only with a key which generated during the upload by decoding. The encoded file is stored in the Private Cloud which is a Trusted Commodity as well as the Public Cloud which is a Commercial Commodity. The file sharing is done between members only via the Public Cloud and all the MD5 hash values are stored safely within the Private Cloud for deduplication checking. The cost of each file that is being uploaded is also calculated and represented.

ACKNOWLEDGEMENTS

We owe a great many thanks to the following people who helped and supported us in the preparation of this project. We would like to thank Dr. C. MUTHAMIZHCHELVAN, Director (ET), of SRM Institute of Science and Technology, Chennai.

We would like to express our deep sense of gratitude to Dr. B. AMUTHA, Professor and Head, Department of Computer Science Engineering, SRM Institute of Science and Technology for giving us the opportunity to take up this project. We would also like to thank our year co-ordinator, Dr. M. PUSHPALATHA, Professor, Department of Computer Science Engineering, SRM Institute of Science and Technology.

We take immense pleasure in conveying our thanks to our panel head, Dr. REVATHI VENKATARAMAN, Professor, Department of Computer Science Engineering, SRM Institute of Science and Technology for accepting the project and providing her valu-able suggestions. We would also like to thank our faculty advisers, Mrs.TYJ.NAGA MALLESWARI and Mr. J. SELVIN PAUL PETER, Assistant Professors (O.G), De-partment of Computer Science Engineering, SRM Institute of Science and Technology for their continuous support and cooperation.

We would like to express our deepest gratitude to our project guide, Mrs. S. SHA-RANYA, Assistant Professor (O.G), Department of Computer Science Engineering, SRM Institute of Science and Technology for her generous support and co-operation throughout every aspect of the project.

Several other people have also contributed to this project. We sincerely acknowl-edge all their efforts, which were indispensable to our project.

VIKRAM ARIKATH

THONDA LAKSHMI HARSHITHA

iv

TABLE OF CONTENTS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ABSTRACT | | |  | iii |
| ACKNOWLEDGEMENTS | | | | iv |
| LIST OF FIGURES | | | | vii |
| ABBREVIATIONS | | | | viii |
| 1 | INTRODUCTION | | | 1 |
|  | 1.1 | Cloud Storage and Security . . . . . . . . . . . . . . . . . . . . . . | | 1 |
|  |  | 1.1.1 | Encryption/Encoding . . . . . . . . . . . . . . . . . . . . | 2 |
|  |  | 1.1.2 | Deduplication . . . . . . . . . . . . . . . . . . . . . . . . | 2 |
| 2 | LITERATURE SURVEY | | | 3 |
|  | 2.1 | Existing Systems . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 3 |
|  |  | 2.1.1 | Primary Systems . . . . . . . . . . . . . . . . . . . . . . . | 3 |
|  |  | 2.1.2 | Hashing . . . . . . . . . . . . . . . . . . . . . . . . . . . | 3 |
|  |  | 2.1.3 | Convergent Key . . . . . . . . . . . . . . . . . . . . . . . | 4 |
|  |  | 2.1.4 | Chunking . . . . . . . . . . . . . . . . . . . . . . . . . . . | 5 |
|  |  | 2.1.5 | Other Systems . . . . . . . . . . . . . . . . . . . . . . . . | 5 |
| 3 | SYSTEM ANALYSIS | | | 7 |
|  | 3.1 | Deduplication . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 7 |
|  | 3.2 | Security . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 8 |
|  | 3.3 | Two Clouds . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 8 |
|  | 3.4 | Admin . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 8 |
|  | 3.5 | Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 9 |
| 4 | SYSTEM DESIGN | | | 10 |
|  | 4.1 | User Interface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 10 |

v

|  |  |  |  |
| --- | --- | --- | --- |
|  | 4.2 | Private Cloud . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 11 |
|  | 4.3 | Public Cloud . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 12 |
|  | 4.4 | Deduplication . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 13 |
|  | 4.5 | Encoding with Key . . . . . . . . . . . . . . . . . . . . . . . . . . | 14 |
| 5 | CODING | | 15 |
| 6 | SCREENSHOTS | | 43 |
| 7 | SOFTWARE AND HARDWARE REQUIREMENTS | | 47 |
|  | 7.1 | Software Requirements . . . . . . . . . . . . . . . . . . . . . . . . | 47 |
|  | 7.2 | Hardware Requirements . . . . . . . . . . . . . . . . . . . . . . . | 47 |
|  | 7.3 | Cloud Storages . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 47 |
| 8 | RESULTS AND DISCUSSIONS | | 48 |
| 9 | CONCLUSION AND FUTURE ENHANCEMENT | | 50 |

LIST OF FIGURES

|  |  |  |
| --- | --- | --- |
| 3.1 | System Architecture . . . . . . . . . . . . . . . . . . . . . . . . . | 9 |
| 4.1 | User Interface . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 11 |
| 4.2 | Private Cloud Workflow . . . . . . . . . . . . . . . . . . . . . . . | 12 |
| 4.3 | Public Cloud Workflow . . . . . . . . . . . . . . . . . . . . . . . | 13 |
| 4.4 | Deduplication Workflow . . . . . . . . . . . . . . . . . . . . . . . | 14 |
| 6.1 | Homepage . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 43 |
| 6.2 | User Registration . . . . . . . . . . . . . . . . . . . . . . . . . . | 43 |
| 6.3 | Private Cloud . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 44 |
| 6.4 | Private Cloud (Back end) . . . . . . . . . . . . . . . . . . . . . . | 44 |
| 6.5 | Private Cloud (Backup) . . . . . . . . . . . . . . . . . . . . . . . | 45 |
| 6.6 | Public Cloud . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 45 |
| 6.7 | Public Cloud (Back end) . . . . . . . . . . . . . . . . . . . . . . | 46 |
| 6.8 | Admin . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 46 |
| 8.1 | Uploading Costs without Deduplication . . . . . . . . . . . . . . | 48 |
| 8.2 | Uploading Costs with Deduplication . . . . . . . . . . . . . . . . | 49 |
| 8.3 | Comparing Uploading Costs . . . . . . . . . . . . . . . . . . . . | 49 |

vii

ABBREVIATIONS

CPT Ciphertext

DUP Deduplication

ECO Encoding

ENC Encryption

MD4 Message Digest 4

MD5 Message Digest 5

UI User Interface

viii

CHAPTER 1

INTRODUCTION

1.1 Cloud Storage and Security

The impact cloud computing has had over the last few decades are immense. Earlier times a single database was used by an organization to store all the data and unless the physical drives were transported from one location to the other there was no other form of sharing data between members in different locations. Later when the concept of internet became commercial and widespread it served as a platform and tool for various processes.

One such concept that arouse from this was the concept of cloud computing. Cloud computing offered several different means to transfer data between members present in different locations around the globe in a fast and effective manner by offering a virtual storage solution. This thus helped in reducing the costs to maintain a single database to store all the information and the need to transport the data physically from one end to the other. Cloud computing was offered as an infrastructure, service or tool based on the needs of the user.

The increased usage of cloud computing led to the necessity of a smart and secure storage system. This was needed since the storage was virtual and almost accessible from anywhere around the globe, malicious users or members can try to hack the data or get hold of sensitive information. To prevent such problems different Encryption (ENC) schemes were used to prevent the data from being hacked into and leaked. Different ENC schemes are still being developed to this day. After the need for security the storage management was the next issue that was addressed. Cloud computing offered a mobile storage and efficient sharing, however the storage space allocated was not infinite. Hence, the concept of Deduplication (DUP) became a necessity to prevent du-plicates of the data from being uploaded onto the cloud storage so that precious storage

can be managed correctly and not be wasted. There are several methods to promote DUP of files within a cloud system, and this project concentrates on one such method.

1.1.1 Encryption/Encoding

ENC can be simply defined as converting plaintext into Ciphertext (CPT) with the use of key using different schemes, whereas Encoding (ECO) is defined as converting the format of a file from one form to another using pre-defined schemes. The primary difference between ECO and ENC remains that one uses a key whereas the other does not. Both are implemented to store the data and later re-use it. The concept of ENC and DUP in a single cloud system has been problematic from the very beginning.

For example, a file when uploaded into the system gets encrypted with a key result-ing in a CPT which gets stored into the system. The same file when uploaded with a different key when uploaded results in a different CPT being generated and hence the process of duplicate checking is not possible. There are several methods that are imple-mented to circumvent this problem. The reason why ECO and ENC process in a cloud computing system is really important because it deals with data confidentiality.

1.1.2 Deduplication

De-duplication as the word suggests is the method of preventing duplication of files in the system. The most common type of de-duplication check is done by comparing the filenames of the files being uploaded into the system. Once, the name is changed and uploaded the file gets saved but the content is actually a duplicate of the file that has been uploaded and is hence a wastage of precious storage space. The need for checking the content within the file as well promoted the need for methods of DUP.

The earliest methods included chunking and simple hashing algorithms to check the content. Where hashing helped in checking many different types of files that are being uploaded chunking helped to introduce a threshold value and check as to how much actual data was similar to the file being uploaded into the cloud system. Both the methods have faced limitations and several alternatives are being implemented in its place.

2

CHAPTER 2

LITERATURE SURVEY

2.1 Existing Systems

2.1.1 Primary Systems

As mentioned earlier the most primal system of DUP checking involved only checking the filenames of the files being uploaded into the system. This did take care of the problem initially but proved to be a shorthand solution for a large problem. Moreover DUP went hand in hand with ENC. So just checking the content was only possible initially by comparing CPT.

In a simple cloud system the technique to check DUP and encrypt the file was not possible. ENC consists of 3 parts in particular. The plaintext, key and the CPT gener-ated form the key. The CPT is the final result of the plaintext and the key, it is dependent on both these factors. So for each file a different CPT was being generated since the key for each file type was different.

The logical solution around this problem was then to keep the key constant. This ensured that similar CPT were generated for the same file and hence helped in checking the DUP for each file being uploaded into the system. This however, prompted another problem of security. Since there was only one key being used, if the key got compro-mised the whole system and each and every file of the user became compromised thus prompting the need for a different type of DUP checking.

2.1.2 Hashing

A circumventing method for this problem was to implement the hashing function. The hashing function is an irreversible process where the content of the file undergoes a

specific algorithm where the file is split and rearranged to generate a unique ID based solely on the content of the file, irrespective of the format as well. So, each file had a unique hashing ID. Hash checking was then implemented to check for the files being uploaded. The hash value of the file was being stored along with the encrypted file to ensure that the DUP check was happening.

Message Digest 4 (MD4), Message Digest 5 (MD5) were some of the key hashing algorithms used in this process. The problem that came with hashing was that the weaker hashing algorithms led to hash collisions. Hash collisions could be avoided by using stronger hashing algorithms. Hence the emergence of the SHA-1,SHA-2 and SHA-3 algorithms came up. The problem then with the stronger hashing algorithms was that it increased the stress on the system and caused computation complexity. But still weaker algorithms were used as the hash collision problem was not that much of an issue to deal with.

2.1.3 Convergent Key

Another solutions to circumvent the problems caused by the primary systems was the method of using convergent key ENC. In the previous system the hash values of the files were to be stored along with the keys for each file. The hash value to check for DUP and the key to decrypt the files in the system and reuse them. Thus another method to decrease the content stored was to use the hash value of each file as the key to encrypt the files in the system. This ensured that separate keys did not have to be maintained to check the files in the system.

Using the hash value as the key also ensured that the same files that were being encrypted in a way that generated similar CPT. Thus the CPT comparison could be done directly to check for DUP. The weaker hashing algorithms still caused hash problems that were overlooked and this system was pretty successful and helped in assisting the co-existence of DUP and ENC in a cloud system.

4

2.1.4 Chunking

All the above DUP methods were used to check for the file-content as a whole. So when even a single word or letter or a pixel in a file was changed, a whole new hash value was being generated for the file as such. This created a need for the chunking algorithm. The chunking algorithm helped store files as chunks and encrypt them. The DUP check was done by comparing the chunks already stored in the system. Chunks basically defined as file divided and stored in parts. The chunks could also be encrypted and stored thus helping in the security part. A hash value could also be generated for the chunks to help in quick DUP check.

The basic form of chunking was Static Chunking. The process of static chunking involved in splitting the file into equal sized chunks. The problem with static chunking was that it caused the boundary value shifting problem. The boundary value shifting problem was such that if a single byte in a chunk was changed or deleted, it would cause a ripple effect and DUP check would be rendered useless. To go around this problem the Content Defined Chunking was used to solve the boundary value shifting problem. This chunking method split the file into variable sized chunks based off different variables

˘ ´

within the system. Even though a single byte was modified it didnâAZt cause a ripple effect and hence assisted in DUP check. The boundary shifting problem increased the load on the system causing time delays in the system.

2.1.5 Other Systems

It was Mark W. Storer et al. who first suggested the need for DUP in a cloud computing system and suggested the Authenticated and Anonymous models for cloud systems. Authenticated systems made sure that the users who registered were monitored by the admin and every file that was being uploaded was checked carefuly. Thus there is no specific privacy for the users since the third entity, which is the admin monitored the systems in place and kept a note of other information. This is why the anonymous model was suggested where the users registered were all anonymous so maximum privacy was ensured. This also meant that malicious users registered within the system and uploaded files that were harmful. The anonymous model ensured maximum privacy but

5

also resulted in a big flaw in security. Malicious content was easily uploaded and cause all kinds of problem in the system.

The Twin Cloud architecture suggested by Sven Bugiel et al. helped in setting the base for this project. It involved using two cloud systems together in a single environ-ment. The two clouds included a trusted cloud and a commodity cloud. The trusted cloud was responsible for the security critical operations and the commodity cloud was responsible for performance critical operations. Security operations always require a more sterile and safe environment to be performed on and hence was given to be done on the trusted clouds. Commodity cloud offered less security and more usage and hence the performance critical operations that required more processing were dumped on the commodity cloud and made to run.

For DUP as such the DupLESS system suggested by M.Bellare et al. gained wide popularity. The system involved using a key server. The main form of DUP check was done by message locked ENC which is otherwise a form of the convergent key ENC. The system was developed such that it was resilient to brute force attacks. The files that were being generated had a hash value that was being generated unique to it. The hash values were then being sent to the key server that used a specific algorithm to generate a key based off the hash value and sent it back to the user who then encrypted and stored the file. Even if the key server was compromised the system was such that the key server never revealed details of the user and remained private. The system was extremely successful.

Another alternative to DUP check was the usage of the DeKey system by Jin Li et al. The initial system was such that for each file a key was being generated and stored at the user side. Then a master key was generated to encrypt all the keys and store them. This master key however if compromised could result in the whole system being at a state of risk. The DeKey system helped to manage all these convergent keys. The DeKey system was such that the keys that the users possessed were managed by a secret sharing scheme. The secret sharing scheme was such that each user possessed part of the secret and a minimum amount of users and their secret schemes were required to decipher the secret sharing system to uncover the keys. The system was highly efficient in a real time application but also increased the complexity of the whole process.

6

CHAPTER 3

SYSTEM ANALYSIS

DUP and cloud security are key factors in cloud computing system nowdays. The major need of implementing a smart storage system is essential because of the growing amount of data generated per day. Almost 2.1 billion people use the internet per day and the all the information they share and upload in one form or the other is uploaded and stored in a cloud system. Be it social media sites like Facebook, Twitter or messaging apps such as Whatsapp, Snapchat, it is only with the help of cloud computing that data is shared from one user to the other.

Our project focuses mainly of providing an architecture with a twin cloud based system of having both a private cloud and a constant public cloud. This architecture is such that it promotes DUP and secure storage in a single environment.

3.1 Deduplication

The DUP process is checked with the help of generating an MD5 hash value for each of the file that is uploaded. The MD5 hashing algorithm generates a 128 bit hash value for a file based solely on the content of the file alone. The reason for using the MD5 hash is that it reduces computational complexity as compared to other algorithms and can be used for a number of type of files. The project supports the MD5 hashing for the extension such as .txt, .doc, .docx, .ppt, .pptx, .xls, .xslx, .xml, .mp3, .flv, .img and many more types. In summary it supports for text, audio and image file types.

The hash value and the file name for each file being uploaded is stored. The system displays an error message if a new file of the same content is being uploaded into the system or the same file name is used. All this happens in the Private Cloud that is generated for each user. The Private Cloud is implemented in SQL and backed up into the Dropbox cloud.

3.2 Security

The security of the system is guaranteed by generating a key for each file that is being uploaded and the only possible way to download this file back is with the help of the key. Each key is linked with the specific file with which it is being uploaded. The confidentiality of the file uploaded is maintained by ECO the file content with UTF-8 and Base64 methods. Only with the use of the key the reverse process of the ECO is done to obtain the file back.

3.3 Two Clouds

The system of two clouds is implemented so that each person can securely store their hash values in the private cloud for DUP checking and a copy of the encoded file is stored in the public cloud for file sharing purposes. The private cloud is represented as the Trusted Cloud and the public cloud is the Commodity Cloud. All file sharing between users occurs within the public cloud and no user can access the other users pri-vate cloud or stored hash values. The public cloud is represented by the online DriveHQ website. The keys can be sent from one user to the other to download the file from the public cloud.

3.4 Admin

An admin acts as a third party entity who monitors the users registered and governs the files that are being uploaded and downloaded by the users in the system. When each file is being uploaded the cost is simultaneously calculated by the size of the file being uploaded and displayed to the user. This is to introduce a metric system to represent real time environments where data uploads and downloads are accounted for and costs computed.

8

3.5 Architecture

The given diagram 3.1 represents the components in the project with a brief architecture.

It provides a summary of the system design with the working of they key modules.

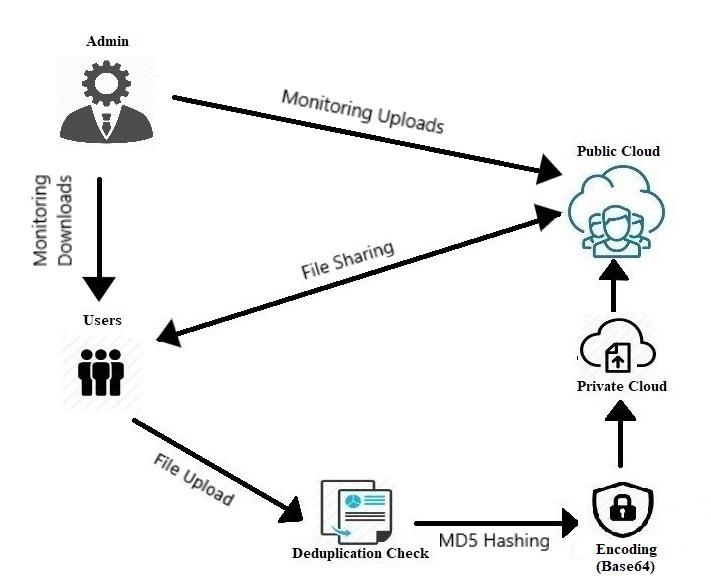


Figure 3.1: System Architecture

9

CHAPTER 4

SYSTEM DESIGN

4.1 User Interface

The User Interface (UI) forms the major front end of the system. It is the interacting layer between the user and the system. The graphical interface is made very interactive to ease the user-system compatibility. The main coding is done with the help of .jsp files to design and allot functionality of each page and the system is made to run on a browser with the help of the Apache Tomcat Server. The UIe displays the home page and the relevant buttons to navigate between pages.

Key components are the user login, registration and the admin login. The user reg-istration takes the required details from the user such as the name, email ID, password, credit card number and also gives the user the option to either have a private cloud of his own or be part of a public and private hybrid cloud system to share files between other users. The user logs into the system with the relevant credentials to reach the welcome page and displays an error if the credentials are wrong.

After logging in the users can upload, download files from their cloud or from the public cloud. If they are a member of the group scheme they also have the provision of viewing other members of the group and deciding to share keys with them. They can also view the usage history of their account, that is the number files uploaded and downloaded and the cost incurred.

The cost is computed based on the size of the file alone. The size of the file is taken in bytes and the following formula is applied to compute the cost :

SIZE X (0.02/100)

The cost is given in rupees and is represented to every user, the one who uploads the file as well as the one who downloads it. It helps maintain a metric system to monitor

the finances required while uploading and downloading files from the cloud system.

The diagram 4.1 gives the structure of the UI.

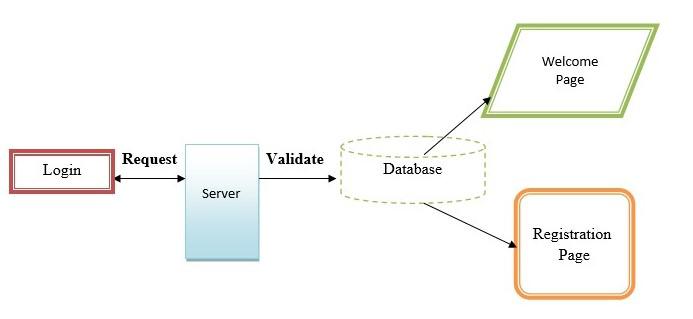


Figure 4.1: User Interface

4.2 Private Cloud

Private Cloud is one of the chief components and forms one of the clouds of our hybrid cloud architecture. During the process of registration a user can choose to have private cloud alone by choosing the individual scheme or have both the private and the public cloud syste by choosing the group scheme. If the individual scheme is chosen then the files stored in the private cloud of a user cannot be shared.

The major DUP checking process occurs in the private cloud system. A file when uploaded from the user side generates 3 things mainy. First is the key required to recover the file. Then is the MD5 hash value that is generated by the file on upload and then the final part is the file itself that gets encoded with the different schemes. The private cloud stores the key values, hash values and the encoded file.

A file when uploaded and hash value generates checks with the hash values already present in the private cloud. Here, the private cloud is the representation of the trusted cloud that is more secure than the public commodity cloud. Each user has his own private cloud and only the user alone can access his private cloud. The cloud is designed

11

such that it is kept away from outside interference. In this project, the private is set up by HeidiSQL and is backed up into the Dropbox cloud to represent.

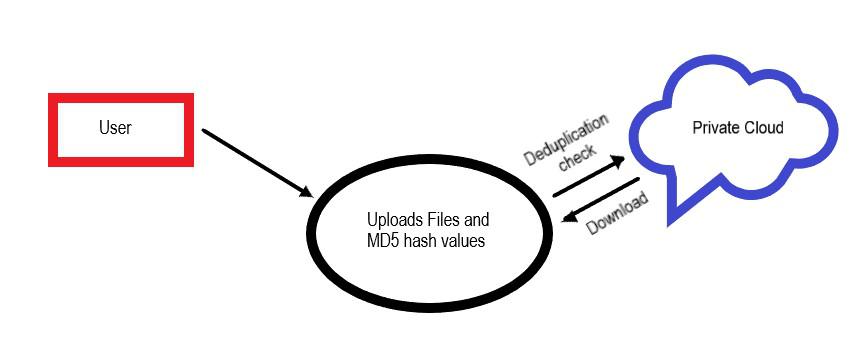


Figure 4.2: Private Cloud Workflow

4.3 Public Cloud

The public cloud is the second most important or forms the remaining part of the hybrid cloud system. This is a representation of the commodity cloud which is less secure but offers enough storage space for large number of files to be shared with other users.

Unlike the private cloud the public cloud does not show the keys or the hash values of the files being uploaded. The private cloud sends a copy of the encoded file into the public cloud and hence it stores the files in its encoded form. A user may have access to a public cloud if and only if during the registration process he has chosen the group scheme and the selected the group he wants to be a part of.

The public cloud is mainly present only for other users to access the files stored in it. However, the files cannot be directly accessed and requires the keys for the decoding processes to run on them. The keys can be sent from one user to the other belonging to the same group.

As the files get uploaded into the private cloud , a copy of the encoded file is written onto the public cloud as well. DriveHQ is the cloud storage platform used to represent the public commodity cloud. It is accessible by multiple users, where it lacks in security

12

by compensates in storage and sharing.

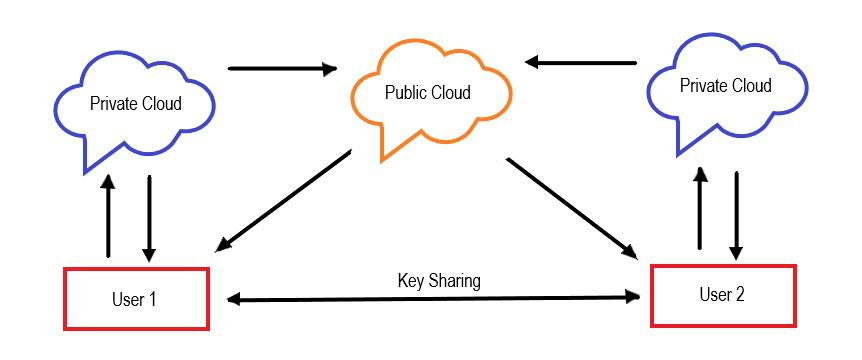


Figure 4.3: Public Cloud Workflow

4.4 Deduplication

The process of DUP forms the backbone of the architecture. It is the system that ensure smart storage management in the cloud architecture. DUP is the process of promoting the prevention of duplicate copies of files in the system. There are several methods of checking for duplicates in the system. One such method that is employed in this project is the process of hashing.

When each file is uploaded from the content an MD5 hash value is generated based on the content of the file alone. The reason for choosing this hashing algorithm is due to it capability of supporting multiple file types and generating hash values quickly without contributing to the computational complexity of the system.

The process of DUP runs only on the private cloud of a user. There are 2 steps to checking DUP in this project. First the filename is checked to see if there is a similar file with the same filename already existing withing the system, if yes it returns an error message stating that filename already exists else the file gets uploaded. The second is the hash value that gets checked with the files already exisiting in the private cloud, if the hash value matches then an error message stating that the file content already exists is displayed to the user else the file gets uploaded. The diagram 4.4 represnts the simple

13

workflow of the DUP technique occuring within the system.

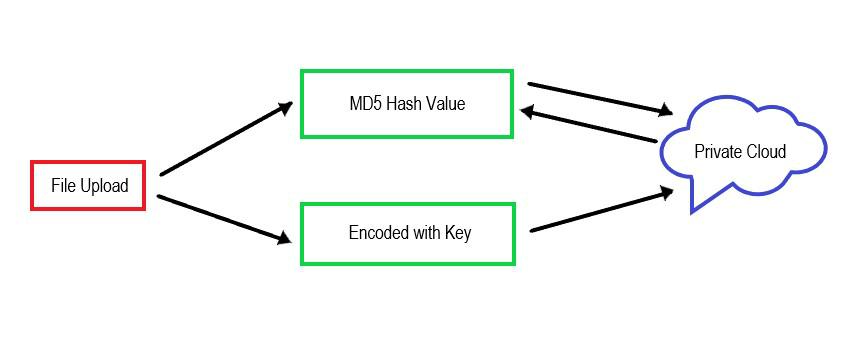


Figure 4.4: Deduplication Workflow

4.5 Encoding with Key

Unlike the usual method of encrypting files in the system we have used the method of data ECO to preserve the confidentiality of the data and also alloting a key based system to it.

ECO is the process of converting one form of data into another and storing it to later retrieve and use it by following the same process backwards,i.e, decoding. The ECO process forms the security backbone of the system. The key of a file is required to decode the encoded files and retrieve them later.

UTF-8 and Base64 are the two processes of ECO used to preserve the integrity of the file that is being uploaded. A process of double ECO is done to ensure better security. The UTF-8 algorithm first converts the content into Unicode and then the Base64 algorithm encodes this unicode to give the final content. ECO is the process chosen here since it gives support for a number of different file types. It differs from ENC in such a way that the key plays no part in the operations happening on the content. However, we have designed a key base ECO and decoding system to ensure security of the files stored in the cloud.

14

CHAPTER 5

CODING

The following codes represent the functionality of the system.

1. Decrypt.java

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

@WebServlet(urlPatterns = "/Decrypt")

public class Decrypt extends HttpServlet {

protected void processRequest(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

HttpSession session=request.getSession(true);

try {

String key1=request.getParameter("key");

Helper encrypter = new Helper("");

String decrypted = encrypter.decrypt(key1);

out.println("decrypted String:" + decrypted);

session.setAttribute("key1", decrypted);

response.sendRedirect("Decry.jsp");

} finally { out.close(); } }

2. Download.java

@WebServlet(urlPatterns = {"/Download"})

public class Download extends HttpServlet {

protected void processRequest(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

HttpSession session=request.getSession(true);

try {

String fkey = null;

String k=request.getParameter("k");

String id=request.getParameter("id");

String un=(String)session.getAttribute("un");

DB Db=new DB();

DB Db1=new DB();

ResultSet rs=Db.Select("select \* from upload where Fid="+id+"");

16

ResultSet rs1=Db1.Select("select \* from file\_share where

Fid="+id+" and User=’"+un+"’" );

if(rs1.next()){

fkey=rs1.getString("key"); }

System.out.println("Coming"+k);

if(rs.next()) {

try {

if(rs.getString("Keyvalue").compareTo(k)==0

||fkey.compareTo(k)==0 )

{

byte Content[] = {};

String filename = "";

double cost=0.0;

if(String.valueOf(session.getAttribute("Sn")).compareToIgnoreCase ("Not")!=0)

cost=(Double.valueOf(rs.getString("Size"))\*(0.03/100));

else cost=(Double.valueOf(rs.getString("Size"))\*(0.05/100));

Content = rs.getBytes("Content");

filename=rs.getString("Filename");

java.util.Date now = new java.util.Date();

String DATE\_FORMAT = "yyyy-MM-dd hh:mm:ss";

SimpleDateFormat sdf = new SimpleDateFormat(DATE\_FORMAT);

String strDateNew = sdf.format(now);

System.out.println(strDateNew);

System.out.println("\*\*\*\*" + Content.toString());

17

byte requestBytes[] = Content;

ByteArrayInputStream bis = new ByteArrayInputStream(requestBytes);

response.reset();

response.setContentType("application/octet-stream");

response.setHeader("Content-disposition", "attachment; filename="

* new File(filename).getName()); byte[] buf = new byte[1024]; int len;

while ((len = bis.read(buf)) > 0) { response.getOutputStream().write(buf, 0, len); } bis.close(); response.getOutputStream().flush();

int i=Db.Insert("insert into download values (’"+session.getAttribute("un")+"’,"+id+",’" +filename+"’,’"+rs.getString("Size")+"’,"

+cost+",’"+strDateNew+"’,’"+session.getAttribute("Sn")+"’,0)");

if(i>0) {response.sendRedirect("close.jsp"); } } else

{session.setAttribute("msg", "Invalid Key !!!");

session.setAttribute("color", "red"); response.sendRedirect("close.jsp"); } } catch (Exception e) { e.printStackTrace();}}

18

} catch (Exception ex) { ex.printStackTrace(); }} 3. Ftpcon.java

import java.io.File;

import java.io.FileInputStream;

import java.io.FileWriter;

import java.sql.Blob;

import org.apache.commons.net.ftp.FTPClient;

public class Ftpcon {

FTPClient client = new FTPClient();

FileInputStream fis = null;

boolean status;

public boolean upload(File file){

try{ client.connect("ftp.drivehq.com");

client.login("manoj0802", "manoj0802");

client.enterLocalPassiveMode();

fis = new FileInputStream(file);

status= client.storeFile(" /kk/"+file.getName(), fis);

client.logout();

fis.close();}

catch(Exception e){

System.out.println(e); }

if(status){

System.out.println("success");

19

return true; }

else{

System.out.println("failed");

return false; }}}

1. Helper.java

public class Helper {

public static Cipher dcipher, ecipher;

public Helper(String x,String y) { }

Helper(String passPhrase) {

byte[] salt = { (byte) 0xA9, (byte) 0x9B, (byte) 0xC8,

(byte) 0x32,(byte) 0x56, (byte) 0x34, (byte) 0xE3,

(byte) 0x03 };

int iterationCount = 19;

try {

KeySpec keySpec = new

PBEKeySpec(passPhrase.toCharArray(), salt, iterationCount);

SecretKey key =

SecretKeyFactory.getInstance("PBEWithMD5AndDES") .generateSecret(keySpec);

ecipher = Cipher.getInstance(key.getAlgorithm());

dcipher = Cipher.getInstance(key.getAlgorithm());

AlgorithmParameterSpec paramSpec = new

PBEParameterSpec(salt, iterationCount);

ecipher.init(Cipher.ENCRYPT\_MODE, key, paramSpec);

dcipher.init(Cipher.DECRYPT\_MODE, key, paramSpec);

20

} catch (InvalidAlgorithmParameterException e) { System.out.println("EXCEPTION: InvalidAlgorithmParameterException"); } catch (InvalidKeySpecException e) {

System.out.println("EXCEPTION: InvalidKeySpecException");

} catch (NoSuchPaddingException e) { System.out.println("EXCEPTION: NoSuchPaddingException"); } catch (NoSuchAlgorithmException e) { System.out.println("EXCEPTION: NoSuchAlgorithmException"); } catch (InvalidKeyException e) { System.out.println("EXCEPTION: InvalidKeyException"); }}

// Encrpt Password @SuppressWarnings("unused") protected String encrypt(String str) { try {

// Encode string to bytes

byte[] utf8 = str.getBytes("UTF8"); // Encrypt

byte[] enc = ecipher.doFinal(utf8); // Encode bytes to a string

return new sun.misc.BASE64Encoder().encode(enc); } catch (BadPaddingException e) {

}catch (IllegalBlockSizeException e) {

} catch (UnsupportedEncodingException e) { }

return null; }

21

* Decrpt password
* To decrypt the encryted password protected String decrypt(String str) { Cipher dcipher = null;

try { byte[] salt = { (byte) 0xA9, (byte) 0x9B, (byte) 0xC8, (byte) 0x32,(byte) 0x56, (byte) 0x34, (byte) 0xE3, (byte) 0x03 };

int iterationCount = 19;

try {

String passPhrase = "";

KeySpec keySpec = new

PBEKeySpec(passPhrase.toCharArray(),

salt, iterationCount);

SecretKey key = SecretKeyFactory .getInstance("PBEWithMD5AndDES") .gener-ateSecret(keySpec);

dcipher = Cipher.getInstance(key.getAlgorithm());

// Prepare the parameters to the cipthers

AlgorithmParameterSpec paramSpec = new PBEParameterSpec(salt, iterationCount);

dcipher.init(Cipher.DECRYPT\_MODE, key, paramSpec);

} catch (InvalidAlgorithmParameterException e) {

System.out .println("EXCEPTION: InvalidAlgorithmParameterException");

} catch (InvalidKeySpecException e) { System.out.println("EXCEPTION: InvalidKeySpecException"); } catch (NoSuchPaddingException e) { System.out.println("EXCEPTION: NoSuchPaddingException");

22

} catch (NoSuchAlgorithmException e) { System.out.println("EXCEPTION: NoSuchAlgorithmException"); } catch (InvalidKeyException e) { System.out.println("EXCEPTION: InvalidKeyException");

} // Decode to get bytes

byte[] dec = new

sun.misc.BASE64Decoder().decodeBuffer(str);

// Decrypt

byte[] utf8 = dcipher.doFinal(dec);

// Decode using utf-8

return new String(utf8, "UTF8");

} catch (BadPaddingException e) {

} catch (IllegalBlockSizeException e) {

} catch (UnsupportedEncodingException e) { } catch (IOException e) {

}

return null; }

public static void main(String[] args) {

try {

System.out.println("Inside Helper");

Helper encrypter = new Helper("");

System.out.println("encrypt the String: SimplePassword");

String encrypted = encrypter.encrypt("uYWQ");

System.out.println("encrypted String:" + encrypted);

23

String decrypted = encrypter.decrypt("WXhL8tsg72Y=");

System.out.println("decrypted String:" + decrypted);

} catch (Exception e) {

} } }

1. LoginCheck.java

@WebServlet(urlPatterns = {"/LoginCheck"})

public class LoginCheck extends HttpServlet {

protected void processRequest(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

HttpSession session = request.getSession(true);

try {

String Un=request.getParameter("Un");

String Pwd=request.getParameter("Pwd");

if(Un.compareToIgnoreCase("admin")==0

Pwd.compareToIgnoreCase("admin")==0)

{ session.setAttribute("un",Un); response.sendRedirect("AdminHome.jsp"); } else

{ DB Db=new DB();

ResultSet rs= Db.Select("select \* from reg where Un=’"

+Un+"’ and Pwd=’"+Pwd+"’");

24

if(rs.next())

{ session.setAttribute("un", Un); session.setAttribute("Sn", rs.getString("S\_Name")); if(rs.getString("Scheme").compareToIgnoreCase("Individual")==0 response.sendRedirect("Userhome.jsp");

else { response.sendRedirect("GroupUserhome.jsp"); } } else

{ session.setAttribute("msg", "Invalid Username or Password !!!");

response.sendRedirect("Alert.jsp"); } rs.close(); }

} catch(Exception e){ e.printStackTrace(); } } 6. SignUp.java

import Connection.DB;

import java.io.IOException;

import java.io.PrintWriter;

import java.sql.ResultSet;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import javax.swing.JOptionPane;

25

@WebServlet(urlPatterns = {"/SignUp"})

public class SignUp extends HttpServlet {

protected void processRequest(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

HttpSession session = request.getSession(true);

try {

String Name=request.getParameter("Name");

String Un=request.getParameter("Un");

String Pwd=request.getParameter("Pwd");

String Email=request.getParameter("Email");

String Cn=request.getParameter("Cn");

String Scheme=request.getParameter("Scheme");

String S\_Name=request.getParameter("S\_Name");

String G\_Name=request.getParameter("G\_Name");

String Key=request.getParameter("key");

DB Db=new DB();boolean st=false;boolean st1=false;

ResultSet rs=Db.Select("Select \* from reg where Un=’"+Un+"’");

if(rs.next())

{ session.setAttribute("msg", "This Username is Already Exist !!!");

session.setAttribute("color", "red"); response.sendRedirect("User.jsp!/page\_MORE"); }

26

else

{ st1=true; } if(st1==true)

{ if(S\_Name==null) { S\_Name="Not"; st=true; }

else if(S\_Name.compareToIgnoreCase("Create")==0 S\_Name!=null) { S\_Name=G\_Name;

st=true; }

else if(S\_Name.compareToIgnoreCase("Create")!=0 S\_Name!=null)

{ System.out.println("hi:Select \* from reg where S\_Name=’"+S\_Name+"’ and Security=’"+Key+"’");

rs=Db.Select("Select \* from reg where S\_Name=’"+S\_Name+"’"); if(rs.next())

{ if(rs.getString("Security").compareTo(Key)!=0) { System.out.println("hello");

session.setAttribute("msg", "Sorry,Incorrect Security Key !!");

session.setAttribute("color", "red");

response.sendRedirect("User.jsp!/page\_MORE"); }

else

st=true; }

rs.close(); }

if(st==true)

{ int i=Db.Insert("insert into reg values(’"+Name+"’,’"+Un+"’,’"+Pwd+"’,’" 27

+Email+"’,’"+Scheme+"’,’"+S\_Name+"’,’"+Cn+"’,’"+Key+"’)");

if(i>0)

{ session.setAttribute("msg", "Congratulations,Your Account has been Activated. You can Enjoy Our cloud service as ’"+Scheme+"’"); response.sendRedirect("Alert.jsp"); }

else

{ session.setAttribute("msg", "Registration Failed !!!");

session.setAttribute("color", "red"); response.sendRedirect("User.jsp!/page\_MORE"); } } rs.close(); }

} catch(Exception e) { e.printStackTrace(); } } 7. Upload.java

import Connection.DB;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.io.InputStream;

import java.io.PrintWriter;

import java.math.BigInteger;

28

import java.net.InetAddress;

import java.security.MessageDigest;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.Statement;

import java.text.SimpleDateFormat;

import java.util.Iterator;

import java.util.List;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import org.apache.commons.fileupload.FileItem;

import org.apache.commons.fileupload.FileUploadException;

import org.apache.commons.fileupload.disk.DiskFileItemFactory;

import org.apache.commons.fileupload.servlet.ServletFileUpload;

@WebServlet(urlPatterns = {"/Upload"})

public class Upload extends HttpServlet {

protected void processRequest(HttpServletRequest request,

29

HttpServletResponse response)

throws ServletException, IOException, FileUploadException {

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

HttpSession session=request.getSession(true);

try {

StringBuffer sb1 = new StringBuffer();

String saveFile="",sn="",un="";

int fileidnum=0,downloadcount=0,vc=0;

sn=(String)session.getAttribute("Sn");

un=(String)session.getAttribute("un");

String contentType = request.getContentType();

* Create a factory for disk-based file items DiskFileItemFactory factory = new DiskFileItemFactory();
* Set factory constraints factory.setSizeThreshold(4012); //factory.setRepository("c:");
* Create a new file upload handler

ServletFileUpload upload = new ServletFileUpload(factory);

* Set overall request size constraint //upload.setSizeMax(10024);
* Parse the request

List items = null;

try {

items = upload.parseRequest(request);

30

} catch (FileUploadException e) { e.printStackTrace(); }

byte[] data = null; String fileName = null;

// Process the uploaded items Iterator iter = items.iterator(); while (iter.hasNext()) {

FileItem item = (FileItem) iter.next(); if (item.isFormField()) { //processFormField(item);

String name = item.getFieldName(); String value = item.getString();

} else {

data = item.get();

fileName = item.getName();

System.out.println("fn:" + fileName); } }

String extension = "";

int i = fileName.lastIndexOf(’.’);

if (i > 0) {

extension = fileName.substring(i+1); }

System.out.println("–"+extension);

saveFile = fileName;

String path = request.getSession().getServletContext().getRealPath("/");

System.out.println(path);

31

String strPath = path+"//"+saveFile;

System.out.println(strPath);

File ff = new File(strPath);

FileOutputStream fileOut = new FileOutputStream(ff);

fileOut.write(data, 0, data.length);

fileOut.flush();

fileOut.close();

System.out.println(strPath);

System.out.println("Thrid");

Connection con = null;

PreparedStatement psmnt = null;

FileInputStream fis;

InputStream sImage;

FileInputStream inFile;

FileOutputStream outFile;

//inFile = new FileInputStream(patt+saveFile);

//outFile = new FileOutputStream(strPath);

BufferedReader br = new BufferedReader(new FileReader(strPath));

try {

String line = br.readLine();

while (line != null) {

sb1.append(line);

sb1.append("");

line = br.readLine();

32

* out.println(sb1); }}catch(Exception e){

} java.util.Date now = new java.util.Date();

String DATE\_FORMAT = "yyyy-MM-dd hh:mm:ss"; SimpleDateFormat sdf = new SimpleDateFormat(DATE\_FORMAT); String strDateNew = sdf.format(now); System.out.println(strDateNew);

String ip1=InetAddress.getLocalHost().getHostAddress(); try {

DB Db=new DB();

File f = new File(strPath); double cost=0.0;

long length = f.length(); if(String.valueOf(session.getAttribute("Sn"))

.compareToIgnoreCase("Not")!=0)

cost=(length\*(0.02/100)); else cost=(length\*(0.04/100)); System.out.println("length " + length);

String ip=request.getRemoteAddr();int id=0;

ResultSet rs=Db.Select("select max(Fid) from upload"); if(rs.next()) {

id=rs.getInt(1); } id=id+1; con=Db.con; int id1=id;

33

String dd="select \* from upload where Filename=’"+saveFile+"’";

Statement st=con.createStatement();

ResultSet rs2=st.executeQuery(dd);

String dd1="select \* from upload where enc=’"+hash(sb1)+"’";

Statement st1=con.createStatement();

ResultSet rs3=st1.executeQuery(dd1);

if(rs2.next()) { session.setAttribute("msg", "File Name Already Exists!");

session.setAttribute("color", "red");

response.sendRedirect("Upload.jsp!/page\_SERVICES"); }

else if(rs3.next())

{ session.setAttribute("msg", "File Content Already Exists!"); session.setAttribute("color", "red"); response.sendRedirect("Upload.jsp!/page\_SERVICES"); }

else { String queryString = "insert into upload(Un,S\_Name, Fid, Filename, Extension,Size,Content,Cost,Date,Keyvalue,req,enc) values (?,?,?,?,?,?,?,?,?,?,?,?)"; System.out.println("four" + un);

psmnt = con.prepareStatement(queryString); fis = new FileInputStream(f); psmnt.setString(1, un); psmnt.setString(2, sn);

psmnt.setInt(3, id1); psmnt.setString(4, saveFile); psmnt.setString(5, extension); psmnt.setLong(6,length);

34

psmnt.setBinaryStream(7, (InputStream) fis, (int) (f.length()));

psmnt.setDouble(8,cost);

psmnt.setString(9, strDateNew);

psmnt.setString(10, "No");

psmnt.setString(11, "");

psmnt.setString(12,hash(sb1)); int s = psmnt.executeUpdate();

int Flength=0,Dcount=0;

FileWriter fw=new FileWriter

(new File("C://XML/"+saveFile+""));

fw.write(extension);

fw.close();

boolean status=new Ftpcon().upload

(new File("C://XML/"+saveFile+""));

if(s>0) { if(String.valueOf(session.getAttribute("Sn")).compareToIgnoreCase("Not")!=0)

session.setAttribute("msg", "’"+saveFile+"’ has been stored.");

else session.setAttribute("msg", "’"+saveFile+"’ has been stored.");

session.setAttribute("color", "white");

response.sendRedirect("mail?fid="+id+"fn="+saveFile+""); }

else

{ session.setAttribute("msg", "Failed !"); session.setAttribute("color", "red"); response.sendRedirect("Upload.jsp!/page\_SERVICES"); } } } catch (Exception ex) {

out.println(ex); }

35

} catch (Exception e) { out.println(e); } }

private static String hash(StringBuffer sb) { // Implement hash (MD5)

String md5 = null; if(null == sb) return null; try {

//Create MessageDigest object for MD5

MessageDigest digest = MessageDigest.getInstance("MD5"); //Update input string in message digest digest.update(sb.toString().getBytes(), 0, sb.length()); //Converts message digest value in base 16 (hex)

md5 = new BigInteger(1, digest.digest()).toString(16); } catch (Exception e) {

e.printStackTrace(); }

return md5.toString(); }

1. mail.java

import Connection.DB;

import java.io.IOException;

import java.io.PrintWriter;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

36

import java.util.Properties;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.mail.\*;

import javax.mail.internet.\*;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpSession;

import org.apache.commons.lang.RandomStringUtils;

@WebServlet(name = "mail", urlPatterns = {"/mail"})

public class mail extends HttpServlet {

String decrypted="";

String encrypted="";

public String userName;

public synchronized boolean sendMail(String userName, String

passWord, String host, String port, String starttls, String auth,

boolean debug, String socketFactoryClass, String fallback, String[] to,

String[] cc, String[] bcc, String subject, String text)

{ this.userName=userName; Properties props = new Properties(); //Properties props=System.getProperties(); props.put("mail.smtp.user", userName); props.put("mail.smtp.host", host);

37

if (!"".equals(port)) {

props.put("mail.smtp.port", port); }

if (!"".equals(starttls)) {

props.put("mail.smtp.starttls.enable", starttls); }

props.put("mail.smtp.auth", auth);

if (debug) {

props.put("mail.smtp.debug", "true");

} else {

props.put("mail.smtp.debug", "false"); }

if (!"".equals(port)) {

props.put("mail.smtp.socketFactory.port", port); }

if (!"".equals(socketFactoryClass)) {

props.put("mail.smtp.socketFactory.class", socketFactoryClass); }

if (!"".equals(fallback)) {

props.put("mail.smtp.socketFactory.fallback", fallback); }

try {

Session session = Session.getDefaultInstance(props, null);

session.setDebug(debug);

MimeMessage msg = new MimeMessage(session);

msg.setText(text);

msg.setSubject(subject);

msg.setFrom(new InternetAddress(userName));

for (int i = 0; i < to.length; i++) {

msg.addRecipient(Message.RecipientType.TO, new InternetAddress(to[i])); }

38

for (int i = 0; i < cc.length; i++) {

msg.addRecipient(Message.RecipientType.CC, new InternetAddress(cc[i])); }

for (int i = 0; i < bcc.length; i++) {

msg.addRecipient(Message.RecipientType.BCC, new InternetAddress(bcc[i])); }

msg.saveChanges();

Transport transport = session.getTransport("smtp");

transport.connect(host, userName, passWord);

transport.sendMessage(msg, msg.getAllRecipients());

transport.close();

return true;

} catch (Exception mex) { mex.printStackTrace(); return false; } }

protected void processRequest(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException { response.setContentType("text/html;charset=UTF-8"); PrintWriter out = response.getWriter(); HttpSession session=request.getSession(true); String sn=(String)session.getAttribute("Sn");

String fn=request.getParameter("fn"); String fid=request.getParameter("fid"); System.out.println(sn);

String usrid="mvinomsc@gmail.com";

39

String passwd="1991vinoth";

try {

String otp=RandomStringUtils.randomAlphanumeric(4);

System.out.println(otp);

try {

System.out.println("Inside Helper");

Helper encrypter = new Helper("");

encrypted = encrypter.encrypt(otp);

out.println("encrypted String:" + encrypted);

decrypted = encrypter.decrypt(encrypted);

out.println("decrypted String:" + decrypted);

} catch (Exception e) { }

DB Db=new DB();

int i=0,j=0;

if(String.valueOf(session.getAttribute("Sn")).compareToIgnoreCase("Not")!=0)

{ j=Db.Insert("Update upload set Keyvalue=’"+encrypted+"’ where Fid=’"+fid+"’"); String text="Your Key for the File ’"+fn+"’ is: "+otp;}

else { String t="";

ResultSet rs=Db.Select("select \* from reg where Un=’"+session.getAttribute("Un")+"’"); System.out.println("select \* from reg where Un=’"+sn+"’"); if(rs.next())

{ t=rs.getString("Email");

System.out.println("\*\*"+t); }

rs.close();

40

String[] to = {t};

String[] cc = {usrid};

String[] bcc = {usrid};

j=Db.Insert("Update upload set Keyvalue=’"+encrypted+"’ where Fid=’"+fid+"’");

String text="Your Key for the File ’"+fn+"’ is: "+encrypted;

sendMail(usrid, passwd, "smtp.gmail.com", "465", "true",

"true", true, "javax.net.ssl.SSLSocketFactory", "false", to, cc, bcc, "mail", text); }

//response.sendRedirect("OtpAuthentication.jsp");

if(j>0){

response.sendRedirect("Upload.jsp!/page\_SERVICES"); }

} catch(Exception e) { e.printStackTrace(); } } 9. MailKey.java

@WebServlet(urlPatterns = "/MailKey1")

public class MailKey1 extends HttpServlet

protected void processRequest(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

HttpSession session=request.getSession(true);

try

String otp=RandomStringUtils.randomAlphanumeric(4);

System.out.println(otp);

41

Integer FileId=(Integer)session.getAttribute("fid");

String fname=(String)session.getAttribute("fname");

String size=(String)session.getAttribute("size");

String un = (String)session.getAttribute("un");

String namee=request.getParameter("namee");

DB db=new DB();

String[] values=request.getParameterValues("keys");

for(String str: values)

String name=str;

System.out.println(str);

ResultSet rs= db.Select("select \* from upload where Fid=’"+str+"’");

if(rs.next())

fname=rs.getString("Filename");

size=rs.getString("Size");

String d="no";

String sss="Insert into FileSharevalues(0 " + str + "0 ;0 " + fname + "0 ;0 "

+size+"’,’"+un+"’,’"+namee+"’,’"+d+"’,’"+otp+"’)"; db.Insert(sss);

response.sendRedirect("Groupmembers.jsp!/pageSERV ICES");

catch(Exception e)

System.out.println(e);

finally

out.close();

42

CHAPTER 6

SCREENSHOTS

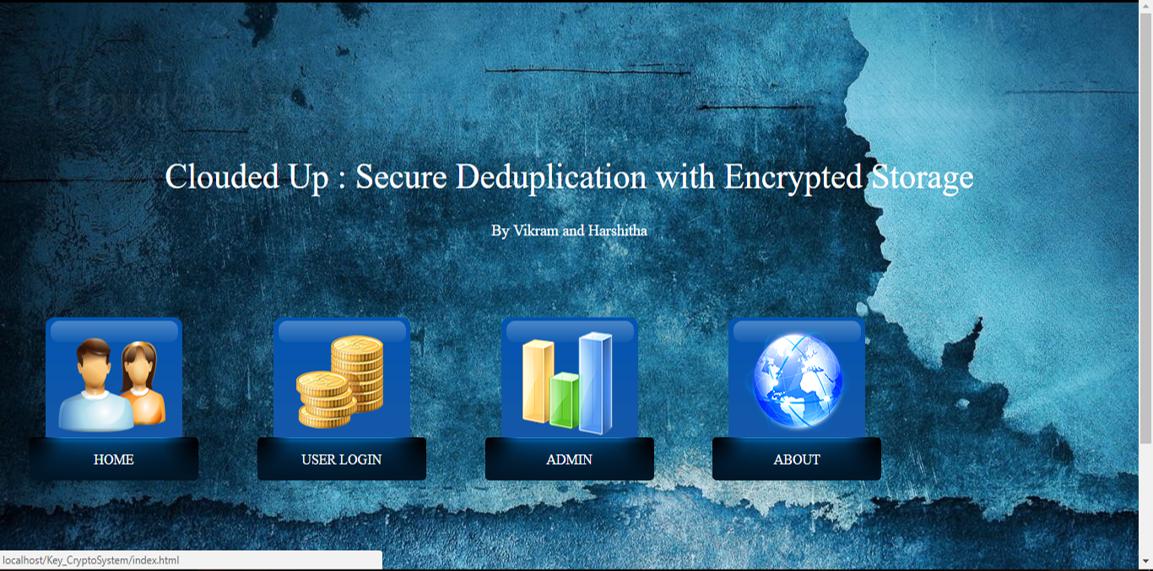


Figure 6.1: Homepage

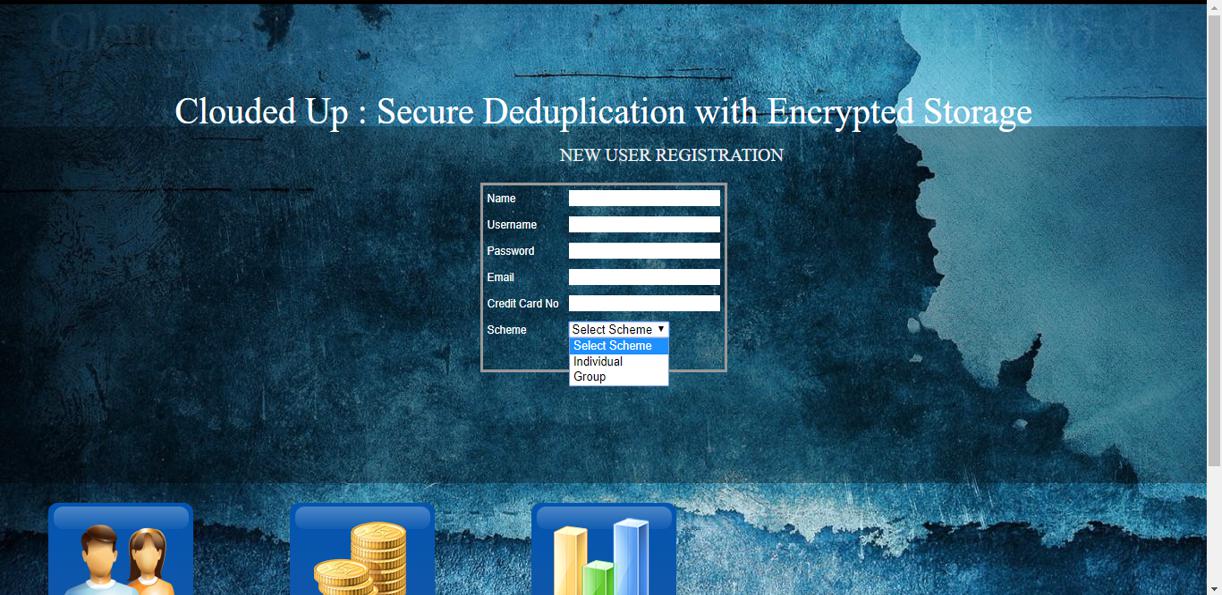


Figure 6.2: User Registration



Figure 6.3: Private Cloud

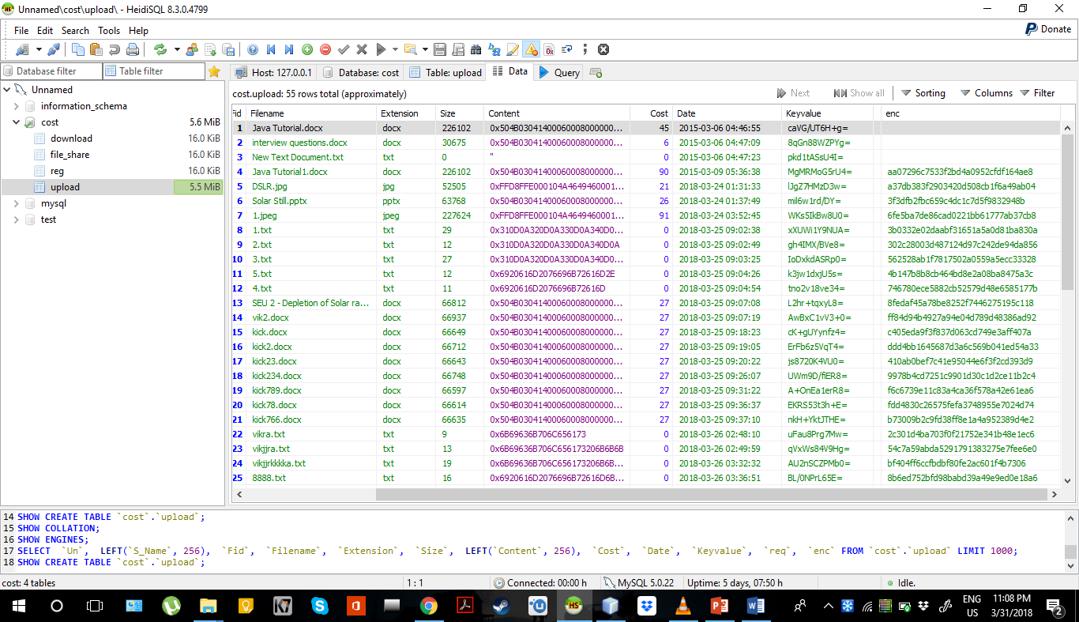


Figure 6.4: Private Cloud (Back end)

44

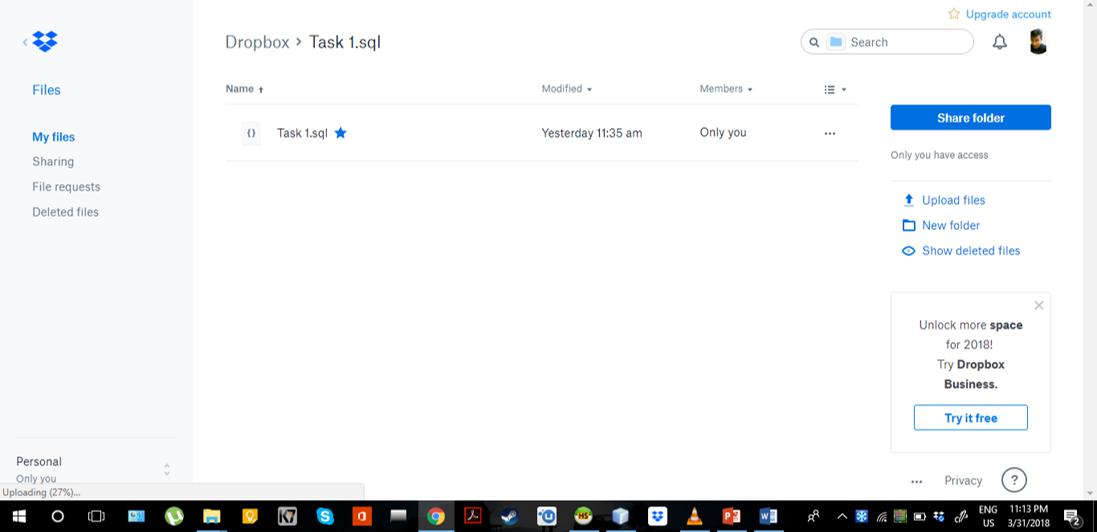


Figure 6.5: Private Cloud (Backup)



Figure 6.6: Public Cloud

45

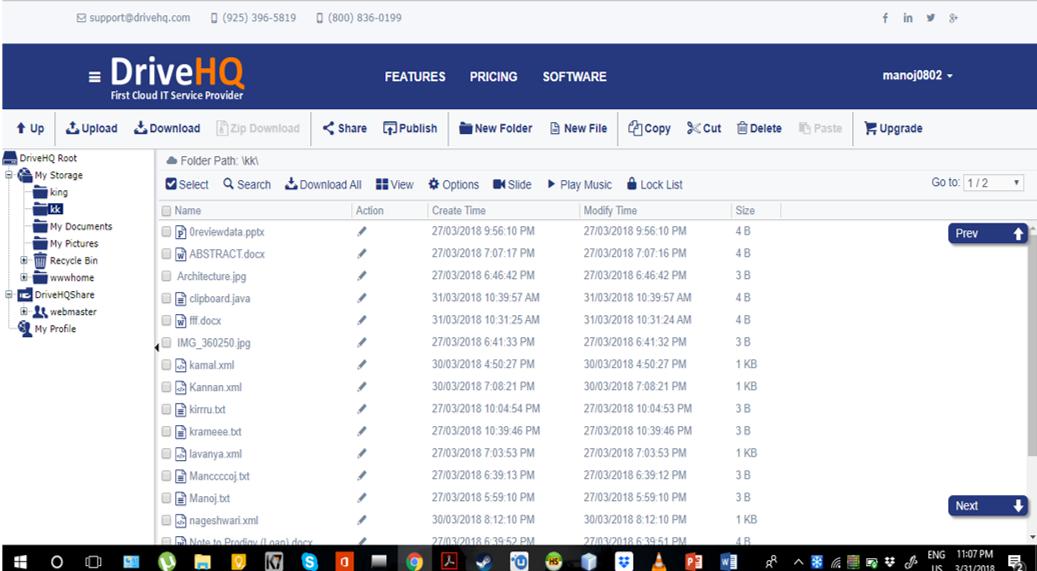


Figure 6.7: Public Cloud (Back end)



Figure 6.8: Admin

46

CHAPTER 7

SOFTWARE AND HARDWARE REQUIREMENTS

7.1 Software Requirements

\ Operating System - Windows 7/8/10 \ Heidi SQL 8.3.0 (32 bit)

\ NetBeans IDE 7.3.1

\ Browser - Google Chrome/Mozilla Firefox/Microsoft Edge

7.2 Hardware Requirements

\ Processors - Intel Core i3/i5/i7 \ Speed 1.6 GHz

\ RAM 512MB \ 1 TB HDD

7.3 Cloud Storages

\ www.dropbox.com \ www.drivehq.com

CHAPTER 8

RESULTS AND DISCUSSIONS

With the help of the cost that is being calculated for each file being uploaded into the cloud system we have tabulated and represented the graph for a system where DUP was not implemented and the same system with the implementation of DUP.

The graph 8.1 represents the system without DUP where the system consists of original files as well as duplicates and the cost is computed for all the files that were being uploaded including the duplicates.

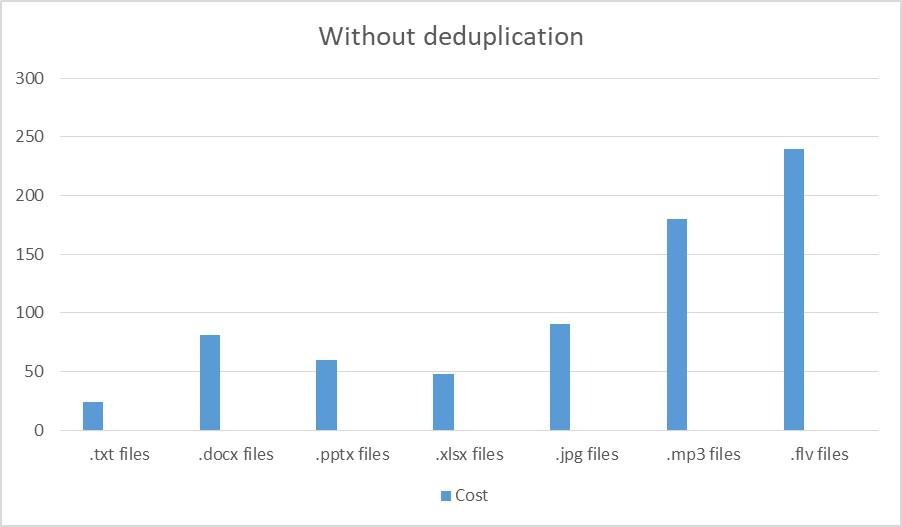


Figure 8.1: Uploading Costs without Deduplication

The graph 8.2 represents the same system with DUP implemented, hence the cost has considerably reduced for each type of file since the system prevents duplicates from being uploaded. It is also true that the hash values are being uploaded this time along with the file, but the size of the hash values are negligible and hence their cost is not taken into account.

The final graph 8.3 represents the comparison between the two cases where DUP was not implemented and where it was. At the least case, for .docx files there is a

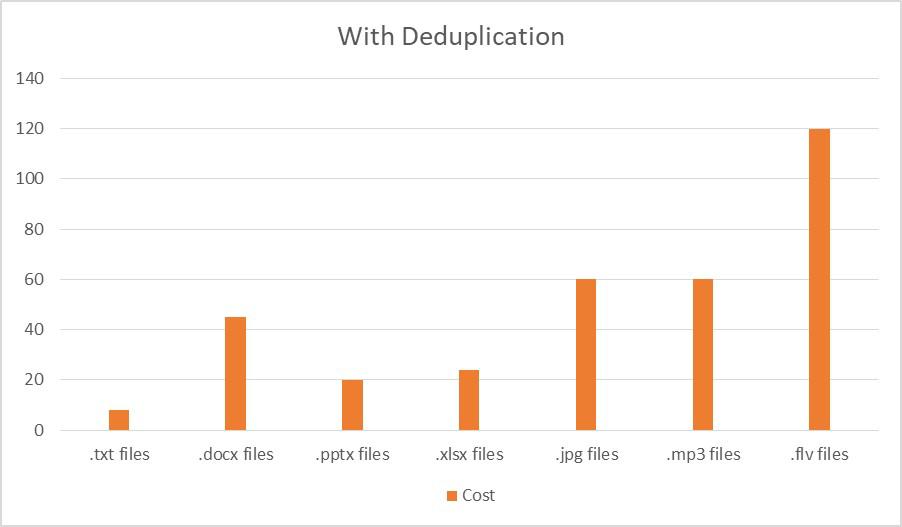


Figure 8.2: Uploading Costs with Deduplication

significant reduction of around 25% of the cost and in most of the other cases there is almost a 75% reduction in cost. This shows the implementing DUP into a system can not only provide better storage management but also helps to reduce the overall costs of network usage to upload files into the system.

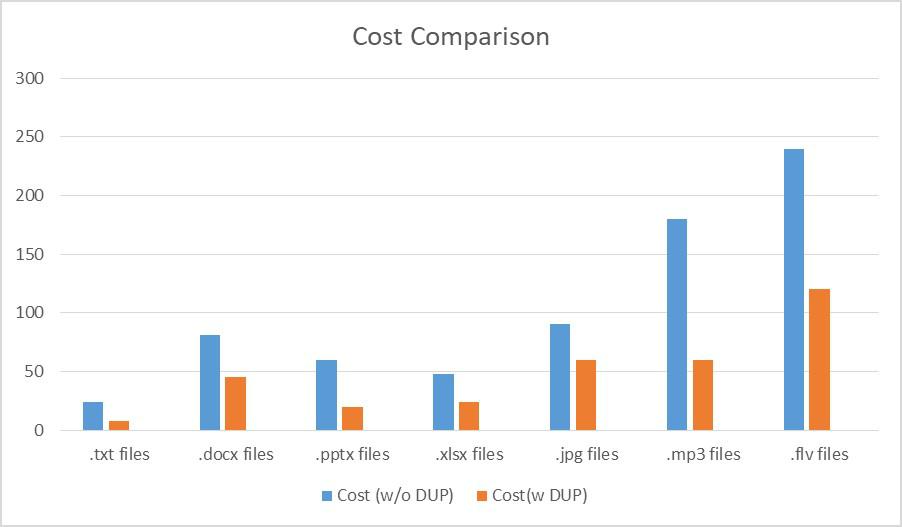


Figure 8.3: Comparing Uploading Costs

49

CHAPTER 9

CONCLUSION AND FUTURE ENHANCEMENT

The project successfully implements a hybrid cloud system where the DUP process by checking the MD5 hash values is securely occuring in the private cloud and the public cloud is used for file sharing by sending the keys from one user to the other. The ECO process ensures the security, confidentiality and integrity of data stored in the cloud systems. The metric system of cost also gives a clear understanding as to how much data is uploaded and downloaded with respect to the rupee currency system as well. DUP almost avoids causing at the minimum twice the amount that is being spent already.

The process of DUP can be subject to changes in the future with the implementation of different algorithms and methods. Text plaigarism methods can be introduced by implementing a threshold value so that even if a single word is changed the system can still identify that the majority of the file is a duplicate. Perceptual hashing is an upcoming hashing method that helps generate the hashing of muiltimedia files with the provision of having a similarity quotient between them. Implementation of different map reduce algorithms to check for DUP in big data can also be implementd within the system in an efficient manner. More ENC schemes with stronger CPT generation can also be introduced to improve the security and data confidentiality factor of the system.

**REFERENCES**

1. W. Xia, Y. Zhou, H. Jiang, D. Feng, Y. Hua, Y. Hu, Q. Liu and Y. Zhang, "Fast CDC: A Fast and Efficient Content-defined Chunking Approach for Data Deduplication," in 2106 *USENIX Conference on USENIX Annual Technical Conference*, Berkeley, CA, USA,2016.
2. Y. Zhang, D. Feng, H. Jiang, W. Xia, M. Fu, F. Huang and Y. Zhou, "A Fast Asymmetric Extremum Content Defined Chunking Algorithm for Data Deduplication in Backup Storage Systems," *IEEE Transactions on Computers*, vol. 66, no. 2, pp. 199-211, February 2017.
3. J. Wei, J. Zhu and Y. Li, "Multimodal Content Defined Chunking for Data Deduplication," *Huawei Technologies*, 2014.
4. Mark W. Storer Kevin Greenan Darrell D. E. Long Ethan L. Miller. “Secure Data Deduplication.” In *StorageSS 2008*, Fairfax, Virginia, USA.
5. S. Bugiel, S. Nurnberger, A. Sadeghi, and T. Schneider. “Twin clouds: An architecture for secure cloud computing.” In *Workshop on Cryptography and Security in Clouds* *(WCSC 2011)*, 2011.
6. J. Li, X. Chen, M. Li, J. Li, P. Lee, and W. Lou. “Secure deduplication with efficient and reliable convergent key management.” In *IEEE Transactions on Parallel and Distributed* *Systems*, 2013.
7. M. Bellare, S. Keelveedhi, and T. Ristenpart. “Message locked encryption and secure deduplication.” In *EUROCRYPT*, pages 296– 312, 2013.
8. M. Bellare, S. Keelveedhi, and T. Ristenpart. “Dupless: Server aided encryption for deduplicated storage.” In *USENIX Security Symposium*, 2013.
9. D. Meyer and W. Bolosky, "A Study of Practical deduplication," in 9th USENIX conference *on File and Storage Technologies (FAST’ 11)*, SAN JOSE, California, 2011.
10. Kameswari Bhaskar, Jayashree R, R. Sathiyavathi, L. Mary Gladence, V. Maria Anu. “A Novel Approach for Securing Data-Deduplication Methodology in Hybrid Cloud Storage.” In *International Conference on Innovations in Information, Embedded and* *Communication Systems (ICIIECS)*, 2017.
11. S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg. Proofs of ownership in remote storage systems. In Y. Chen, G. Danezis, and V. Shmatikov, editors, *ACM Conference on Computer and* *Communications Security*, pages 491–500. ACM, 2011.
12. J. Malhotra and J. Bakal, "A survey and comparative study of data deduplication techniques," in *International Conference on Pervasive Computing (ICPC)*, Pune, 2015.
13. B. Cai, Z. F. Li and W. Can, "Research on Chunking Algorithms of Data Deduplication," in *International Conference on Communication, Electronics and Automation Engineering*, Berlin,Heidelberg, 2012.

1. A. Li, S. Jiwu and L. Mingqiang, "Data Deduplication Techniques," *Journal of Software*, vol. 2, no. 9, pp. 916-929, 2010.
2. G. Raj, "Deduplication Internals – Hash based deduplication: Part-2," [Online]. Available: https://pibytes.wordpress.com/2013/02/09/deduplication-internalshash-based-part-2/.