Synchronized Video Surveillance System with Cloud Storage Services Using OpenStack SAIO

by

Md. Samiul Basher
16101090
Jarif Ahmed Soumik
16101118
Nowreen Zaman
16101167
Niloy Julious Rozario
16301162
Sakib Sadman
16101128

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
Brac University
October 2020

Declaration

It is hereby declared that

1. The thesis submitted is our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Samiul

Md. Samiul Basher
16101090

Jarif

Jarif Ahmed Soumik
16101118

Nowreen

Nowreen Zaman
16101167

Niloy

Niloy Julious Rozario
16301162

Sakib

Sakib Sadman
16101128

Approval

The thesis titled "Synchronized Video Surveillance System with Cloud Storage Services Using OpenStack SAIO" submitted by

1. Md. Samiul Basher (16101090)
2. Jarif Ahmed Soumik (16101118)
3. Nowreen Zaman (16101167)
4. Niloy Julious Rozario (16301162)
5. Sakib Sadman (16101128)

Of Summer, 2020 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on October 05, 2020.

Examining Committee:

Supervisor:
(Member)

Jannatun Noor

Jannatun Noor
Lecturer
Department of Computer Science and Engineering
BRAC University

Program Coordinator:
(Member)

DR.Md.Golam Rabiul Alam
Associate Professor
Department of Computer Science and Engineering
BRAC University

Head of Department:
(Chair)

Prof. Mahbub Majumdar
Chairperson
Dept. of Computer Science & Engineering
Brac University

DR.Mahbubul Alam Majumdar
Professor
Department of Computer Science and Engineering
BRAC University

Ethics Statement

The thesis is carried out in complete compliance with research ethics norms, and the codes and practices set by BRAC University. In our thesis we use the data from primary sources. We collect data from different participants and we use our own data-set for our thesis. we are ensuring we use references and in text citations properly. We the four co-authors take full responsibility for the thesis code violations. For solving problems, we read different websites, YouTube tutorials, and Questionnaire Free tools. We also took help from our university faculty members. Finally, we declare that we give credit to every person from whom we took help. We did not make any fraud able means for completing the thesis. Our work is in compliance with the ethics standard set by BRAC university.

Abstract

Cloud computing is one of the latest technologies we can use to save and secure our data on the internet. In past years people used to save their important data in computer hard drive, memory card, mobile devices but those devices have limited storage space and do not have any backup. If someone's device is destroyed or data is deleted, recovering those data is very much difficult. That's why we prefer cloud storage for storing any kind of data as it provides backup of our data and we can access our data from anywhere and with any device. Then comes a video surveillance system. We use our own cloud storage as the storage device for video surveillance systems. Using proper knowledge about cloud computing and the idea of object storage Swift we tried to build a proper storage system where we can store our big video surveillance data and also find a way to get those data even after a few years

Keywords: Cloud Computing, Object Storage Swift, Video Surveillance System, Big Data Store.

Dedication

We would like to dedicate this thesis to our parents. Without their support we may not be able to do our studies. They also give us mental support during this Covid-19 pandemic. We also want to dedicate the research to our supervisor Jannatun Noor Mukta. Our supervisor helped us a lot throughout the whole three semesters. Without her dedication, instructions we would not be able to complete our project. We also want to dedicate this paper to our friends who helped us a lot to improve ourselves. Mostly the participants who keep patient and stay strong and dedicated until we finish our work. We also want to dedicate our paper to those who died in this Covid-19 pandemic.

Acknowledgement

Firstly, we like to thank our beloved family members whom we always will be grateful to. Secondly, we thank our Supervisor Jannatun Noor Mukta for giving us constant guidelines. We also thank Ruhul Amin Sujon for helping us. Finally, we thank all our faculty members and staff of the CSE department for providing us such a wonderful, peaceful study environment.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Technology is growing and improving day by day and to cope with this, we have to create new things and recreate old things in a new way. In our daily life, we do record things whether it is video, audio or any data because it is very important to have a record of everything to overview our performance, investigation, memory. Everyone in this world thinks to have the best output from technology and we are not different from them. As we have said, this research paper is about storing video footage not only locally but also in a cloud system so that we can access our necessary data whenever we want.

Firstly, we thought about storing small video footages from smartphones or cameras to the cloud and then we realized google and other companies have already made an automated cloud system which synchronizes data from smartphones and stores it in cloud. For this, we casually targeted our aim to store CCTV or IP camera footage in the cloud but when it's about the cloud, we must have to roam the field or industries of existing things. So, we decided to go through the previous research paper on cloud storage systems and then we got some new ideas to recreate the old research with our very own cloud system.

All of us know storage is one of the biggest problems of today's world. According to Forbes.com, we are creating around 2.5 quintillion bytes of data every day [1]. This is actually a huge amount of data and to store this much data we need storage. Our normally used physical storage devices have a very limited size and also the backup plan of those devices are very poor. If the device is destroyed or data is deleted most of the time those data are gone forever. For these reasons nowadays people use cloud storage systems as it has an unlimited storage system and data backup is very easy. That's why we also planned to make our own storage system to implement some techniques to make the storage system better. We also focus on storing big video data. We connect our own build storage to CCTV and store those video data to our storage. We use OpenStack as our cloud platform as it is open source. We build our storage system using OpenStack SAIO (Swift All In One). As object storage Swift is sufficient for us to implement all our modules and also it has

various building techniques which help us a lot during our project. We also focus on the duration of data in our storage. We wanted to make our service easy to use for our customers or users. That's why we tried to build an interface by which we can directly upload data to cloud storage. Finally, we tried to keep track of every information of our stored data through a database.

1.2 Motivation

Cloud is the most significant system for this modern world as every company is looking forward to keeping their data in their own cloud system which is the strongest revolution in this era. So, we have decided to make a proposal of a new cloud system which can store the CCTV/IP camera footage from the existing models which are used already but with some additional features.

Present technology is changing rapidly. What is new today will be old very soon. If we do not upgrade yourself with the new technologies we will lag behind. So, we need to take the blessings of new technology in our everyday life to make our lives easier. Cloud computing is one of the newest inventions in the technical field. Cloud is the safest storage to store data so we got interested to do some research in this field. Then we started reading research papers, related websites, blogs to learn more about it. Then we found OpenStack which is one of the most widely used cloud platforms and it is also free. So, with the help of our faculty we started exploring the object storage of OpenStack swift. Within a few months we were able to set up our very first cloud storage system. Then we started our next step, a video surveillance system. We started gathering knowledge about CCTV cameras. Then it comes to our mind that instead of CCTV's physical storage we can use our own object storage to store CCTV video data. There comes our biggest struggle, storing big video data in cloud storage as CCTV video data is relatively bigger that normal video data. From a few related articles we came to know that big data storing systems in a cloud server have never been so easy. Most of the world's renowned cloud service providers had a limitation to store video data. [2] [3] [4] [5] [6] [7]

Table 1.1: Video size limitation of different cloud platform

Cloud Provider	Video Size Limit
iCloud	2GB
Google Drive	750 MB per day
Azure	2GB
Dropbox	2GB
Zip Cloud	5GB
AWS	2GB

As we can see each and every company has a limitation for uploading video files. If we want to upload large videos either we need to buy extra storage or need to compress a file within those limits. We have handled these problems by segmenting

our large file into multiple parts. Then It comes to our mind to make our plan user friendly so we made an interface by which we can easily upload and download video files to our cloud system. Finally, we made a database where we will store every information about our stored data so that we can keep track of our data.

1.3 Problem Statement

We know data is important for every individual and companies. Using a proper storage media with proper backup plan and ability to download or use data anytime anywhere is the most important thing. As we are researching cloud computing, at first it was very much difficult for us to understand. We had to set up our cloud system on Linux. Most of us had totally zero idea about the Linux operating system, how it works, how the terminal command works. So, learning all those new things together was quite tough for all of us.

Then we combined the storage system and video surveillance system in our project and finding the proper harmony between both was the most difficult part. We have to focus most on what people can expect from these kinds of systems. according to a survey 50% of cloud service users are not happy because it is not user friendly [8]. That's why we tried to make our service as user friendly as possible. At very first we were segmenting our big video data using the swift building tool of swift but the problem was when downloading we have to download the full video again. If somehow, we are able to download a single segment using other tools it will not be playable as the header will be missing. That's why we have to use outside tools called FFmpeg. By using it we can segment our videos and also download as a playable segment. Then it comes how long the video will be stored in our storage. Then we come with an idea of archiving our video data. After a few days of staying in main storage we will compress those videos and store it in a different storage called archive storage. We also set expirer which indicates after how many days/times the video will be permanently deleted.

1.4 Objectives and Contributions

Firstly, our main objective was to make a proper cloud storage with all the properties of cloud. Because without a proper storage for our data we will not be able to implement our system. so, we focus first to set up SAIO (Swift All In One) on our own computer. To achieve the properties of cloud we had to make our storage easily accessible from any device from anywhere. Then we tried to achieve the objectives from a user perspective. Making it user friendly and secure. As we are the very first team who are working on combining both surveillance systems and cloud storage systems so we had to handle each and every topic very carefully and had to find harmony within every part of our project.

OpenStack helps us a lot regarding making our own cloud storage. It is a free platform and most of the information/command of SAIO setup was given in the website. We didn't have to pay any fees so it makes our project easier to achieve all the objectives.

1.5 Thesis Structure

To complete our total thesis, we will first set up and will try to segment them through FFmpeg tool in a playable segment. Then will test our storage system by uploading those segmented video files to our storage and later downloading it. We will also test our user satisfaction part by archiving our video to gain those videos even after a long period of time. We will also test our user interface to check if we can upload video from the interface or not. We will use Python programming language for that part. Then we will also use our database module to keep track of our stored data

1.6 Workflow

To gain success in any type of work we need a proper work plan. Without proper planning research like this will never be successful. We had to read lots of related research papers as it was a totally new topic to us. He had to analyze lots of things through our research. We had to take into account people's demand, problems and struggles in this field. We had to change our plan several times as working with an unknown topic was not so easy. Though we have lots of difficulties in our path, we succeeded just because of our work plan and the dedication of all the fellow mates and supervisor.

The figure 1.1 shows the workflow diagram of the complete system. This system will follow the steps mentioned above to fulfil our objective of making a cloud based proper storage system along with a video synchronize system. First, we have to study lots of research papers to gain knowledge about cloud computing as we are new in this sector. Then we will set up our very first cloud storage system using OpenStack SAIO. Instructions are given on the OpenStack website. After that we have to check if our service is properly working or not. We will check it by uploading a file to our system and later downloading that file from our system.

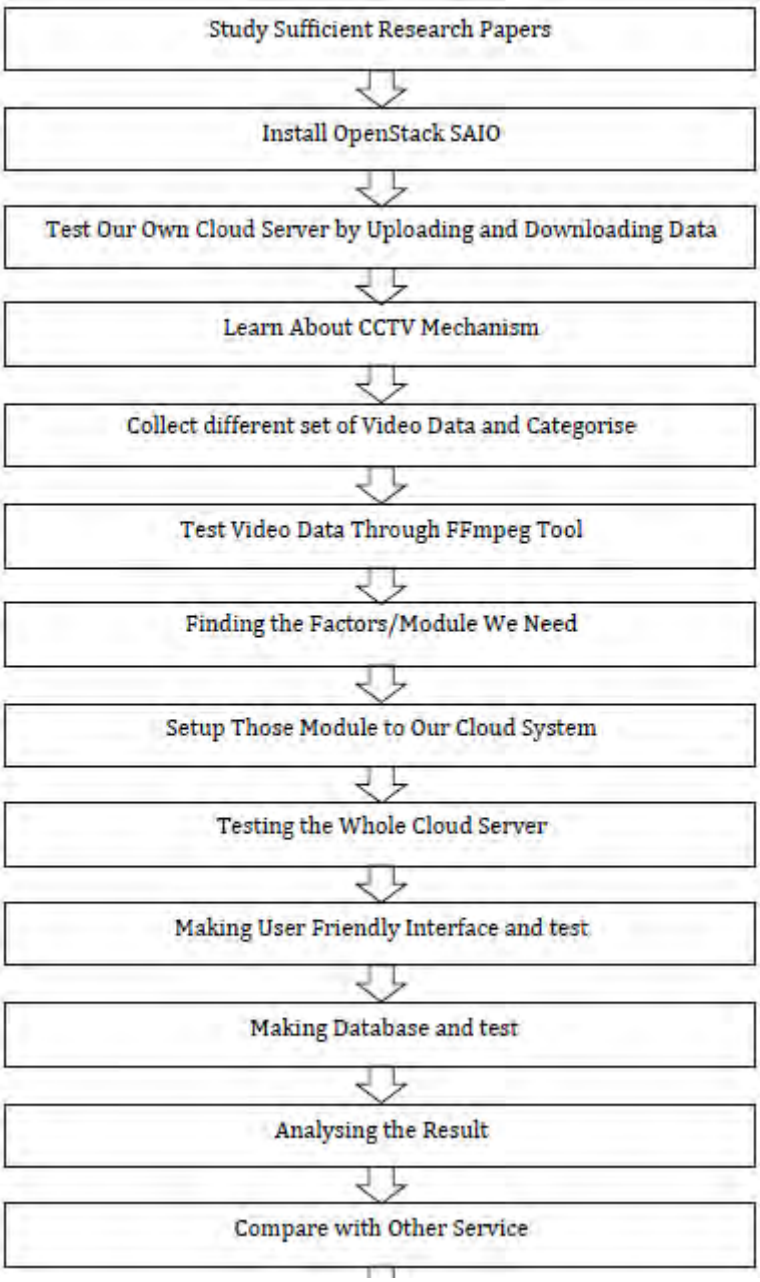

Figure 1.1: Workflow of the thesis

As we are also working on surveillance systems, we will learn about CCTV cameras, how it works and details about its physical storage system and details about CCTV video footage. Then for converting our big video data into playable segments we will test FFmpeg tool. After that we will try to find what more factor or module, we need to make our setup more realistic or useful to the users. After finding those modules we will set up/implement those modules to our system.

Then we will build the user interface to make it user friendly and test it. Then we will make a database to keep track of all the data and will also test it. And finally, we will analyze all the results we will get throughout the whole research and will compare our result with other different tools and systems. After doing all this we will be able to say how successful our system will be.

Chapter 2

Related Work

2.1 Literature Review

Cloud computing is an on-demand internet based remote computing system that allows users to perform any computation through storage, network, database, application, servers with minimum interaction with the service provider [9]. Cloud computing provided services can be accessed from anywhere anytime and it makes computing much more flexible. Multiple users or clients can access and use cloud platforms at a time with pay as you go system so it is more cost effective. Cloud services basically divided into Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS), included by ITU (International Telecommunication Union) and also cloud computing can be divided into five layers. Cloud computing can also be divided into five other layers. The key characteristics of cloud computing are agility, cost efficiency, can be accessed from any location anytime, multi-tenancy, high reliability, high scalability, user friendliness, virtualization, Internet centric, variety of resources, automatic adaptation, pay-per-use service and so on but the most discussed problem of cloud computing platform is privacy and security.

As the advancement in technology, cloud computing technology has grown in a dramatic way in the last decade. And as a result of this, cloud computing has become an inseparable part of our life. But the thing to be concerned about in this context is that cloud computing consumes a huge amount of energy. A research done on this show CC consumed around 3% of the electric energy around the globe [10]. The energy is mainly used and consumed in 3 sectors on a cloud computing system. First comes IT load, which handles the CPU usage, disk storage, memory and network connectivity. It consumes around 59% of the energy. Furthermore, the cooling system use around one third of the total electrical energy of the cloud system. Rest 8% is consumed during distribution. The data center servers tend to emit a lot of heat when they are in use. And this overheating is a main cause of more energy wastage and more hardware failure.

As in [11] mentioned, one of cloud computing's five essential characteristics is on demand self-service which ensures cost efficiency as the user is only paying for the services he or she is using. Furthermore, in [12] this paper, the authors proposed an on-demand cost efficient method for storing video streams in the hierarchical

storage of the cloud which enables the method to decide the video streams that should be pre-transcoded as it requires a large storage space which can be costly and this method can minimize the cost up to 40%. This method talks about video streams that can be divided into many sequences which are created by Group of Pictures (GOP) and both of them have headers that allows both sequential and GOP to contain meta-data. This method requires cloud services such as computational services and storage services that can be provided by any cloud service providers in a pay-as-you-go system. In addition to mobile based video streaming, [13] proposed an efficient framework for smooth and good quality video transmitting based on DASH protocol for personal cloud such as Dropbox, One Drive. This framework is named VSync. mobile devices are not much suitable for large video streaming as battery, memory storage, network bandwidth are some issues. In this paper, contrary to DASH-based systems, VSync can provide real-time videos of high quality as well as smooth transitions that is able to make the storage facility more efficient as they will store only one copy of the video with high resolution and it will keep the memory of personal cloud free from overload.

In this generation of Smart TV, Video on Demand (VOD) are more popular among end-users as it allows the receivers to choose the shows or videos they want to watch through television and use different kind of internet-based applications in the same platform and as the demand grows, the load and storage management become difficult. In [14], the authors proposed a cloud-based VOD system that can ensure concurrency and scalability that is much needed for this sort of environment that can be used as computers or smartphones, to continue providing services around the globe. And they used Hadoop Distributed File System (HDFS) based cloud storage that helps to process and accumulate large amounts of data.

Cloud based Video Storage System (CVSS) is one of the extensively used services of cloud systems which brings the concern of security and privacy policies of cloud-based systems. In [15], the authors provided a method which will be able to provide security for CVSS. Here the video footage such as CCTV footage are encrypted with a key and when the file is requested from the user, it can be also decrypted by a key and this way the data will be protected even when there is a hacking going on in the cloud storage. This way, not only video data but also all sort of data stored in the cloud can be protected.

As cameras and mobile phones are widely used and available, it enables the users to collect a large number of video data which can hold important resources. This type of video data analysis is quite costly, needs a significant amount of time and mostly needs human monitoring but at the same time they are mostly precise. With the increased use of video data, most of the organizations prefer CCTV footage or the video data monitoring for security purposes and it raises the necessity of constant monitoring which can be quite challenging. In [16], the authors proposed a three-layered cloud-based video monitoring and analysis system that addresses these problems such as processing time, human monitoring, GPU space and so on and seeks to find a solution. In the first layer, scalability of the cloud platform is checked based on the cloud provided services. In the second layer, each cloud node is evaluated in order to determine efficiency. Finally, in the third layer, the

object recognition algorithms are tested through the Local Binary Pattern (LBP) algorithm. Together it makes a cloud based high performance system that can process huge amounts of video data in short time which can use the GPU efficiently without overloading it. The decoding of video frames and processing of each video happens in GPU of each cloud node. The increased number of videos can also be scaled in order to process them. By doing this, we can shift from CPU reliable cloud servers to GPU reliable servers. Their work relies heavily on GPU but they chose to run their benchmarks on a gtx 780 which was dated even when they released their paper. The card however did have 2304 CUDA cores which is a lot even by today's standards so an argument could be made against the first issue. Secondly, they tested 704X528 resolution videos but in the present day most phones come equipped with cameras capable of shooting at least 1080p videos so how cost effective it is to run much larger video files moving from CPU based systems to GPU based platform remains to be seen.

CCTV footage can be troublesome to deal with for face recognition as it can be affected by the footage quality or rain or expression, clothing, light and so on. [17] this paper proposes an access control method based on video surveillance. It also includes a face recognition system based on CCTV machine learning system using Radio Frequency Identification (RFID). This proposed method can be applied where facial recognition is difficult to acquire for certain difficulty. It can be the lack of feature factor or unclear video footage. Individual mobile devices can be used for RFID features. After that, multi-channel authentication can be done using RFID. Even if RFID authentication tags are not needed, the dual-channel authentication can still conduct the facial recognition process and thus the individual information security can be ensured. But the recent CCTV systems don't have proper security or privacy preservation and yet cloud based security surveillance systems need even more protection because public cloud is shared by many devices causing many security issues. Hackers can easily hack cloud systems containing valuable data. This proposed system has few layers as a terminal layer, a transport layer, a surveillance layer, and a monitoring layer. Assembling video data, sensing information is mainly done by a terminal layer which uses CCTV systems and IoT terminals. The transport layer executes using E2EE. Its main function is to transmit data to the next layer which is the surveillance layer. Surveillance layer mainly executes archiving of the provided data and securing storage using de-identification of the video data of the CCTV environment. It also masks the meta information. Lastly the monitoring layer mainly focuses on client authentication and access control for proper privacy check of the client. Bio-metric authentication is also very popular now a day because it's quite safe and reliable in case of accuracy. As [18] mentioned, facial and object detection both can be done by the same authentication system. But at the same time, it's not very accurate. But bio-metric authentication and face recognition can be quite challenging in order to secure individual privacy and identification [19].

Most of the CCTV surveillance systems deal with large amounts of video data that requires huge storage space. These large video data are being saved in a separate storage device for a certain period. But storing these data in a cloud system is more efficient as storing data is more efficient and always accessible from any location and also available anytime. But it has a disadvantage as often the same data can be

stored multiple times resulting in a waste of storage. [20] proposed a new method named deduplication that can be able to solve this issue. Data deduplication is a method which can prevent any type of storing the same data multiple times in a cloud environment. In this way a vast amount of space will be free. But there is a challenge of security as cloud is a public platform shared by many people. In this paper, the authors also proposed an encryption framework to support their method. Deduplication is categorized as server-side deduplication and client-side deduplication. In server-side deduplication, all the video files are sent to the server, so that the server can perform the procedure. But this type of deduplication is not effective because of its high possibility of being resulted in a bottleneck as people can upload the same data simultaneously in the cloud. The client-side deduplication is more efficient as the server checks the deduplicated data first and sends the list to the client and the client performs the deduplication. And after that, all the deduplicated data is being uploaded in the cloud. This way, storage can be free without the repetition of the same data. To ensure security for video data, deduplication uses hashing to verify if the data is deduplicated. Regular encryption cannot be done as it is not secured for deduplication. Therefore, convergent encryption (CR) is the best way to solve this issue [21] but also it has few disadvantages such as poison attack and dictionary attack. As convergent encryption uses hashing for encrypting data with key, it can be attacked by a hacker if he is able to guess the source and this type of attack is known as dictionary attack. On the other hand poison attacks happen when meta data cannot be matched with the stored data. It can cause damages like loss of data source and damage of malware and modified data. This proposed method is based on dynamic ownership management of Hur et al in [22].

[22] Introduces convergent encryption using proof of ownership (POW). To randomized convergent encryption, the initial up-loader encrypts the message with randomly chosen key, then encrypts the message encryption key and generates key encrypting key (KEK). Message tags are generated from KEK and cipher-text utilizes public parameters. Through this method of deduplication, only authorized access will be accepted as it is the most crucial issue of cloud storage systems. Furthermore, a dynamic ownership management system is here to ensure the forward and backward security of a deduplication system even if there is a possibility of ownership change of data. It ensures data reliability to the valid owner of the data in a cloud environment.

As CCTV surveillance systems mostly work with 24/7 video streaming, it's difficult to monitor as human monitoring loses concentration and misses most of the details after 22 minutes. That is why there are a large number of algorithms for tracking, monitoring, moving object detection, face detection and so on. In this paper [23], the proposed method includes large scale video retrieval using a layered architecture and deep learning and semantic approaches. This process is named IntelliBVR. IntelliBVRs architecture is made of four layers such as Big Data Curation Layer (BDCL), Video Data Processing Layer (VDPL), Deep Video Annotation Layer (DVAL), and Knowledge Curation Layer (KCL). The first layer is the base layer which is used for a section of large video data. The second and third layer is responsible for processing videos using deep-learning. The fourth layer is the knowledge curation layer where all the low- and high-level features are mapped. In this paper, the proposed system

is using a customized cloud platform named SIAT.

From [24] we can learn how to process a video on a cloud platform. Which includes uploading and downloading footage and converting in different resolutions, segmenting and retrieving the whole file.

In [25], it shows how to create media storage in the cloud using OpenStack. They created a Secure Processing-Aware Media Storage then evaluated different types of video quality, Bit rate, resolution, duration and size by comparing in two different ways (Subjective and Objective evaluation).

Chapter 3

Our Proposed Approach

3.1 System Model

Figure 3.1 shows the methodology of our proposed system. Here first we will collect video data from CCTV camera/ IP camera. These videos are stored at local storage first. From that local storage through FFmpeg tools we will pass the video data to our Swift object storage by segmenting those videos to playable parts. Then we will compress those videos. After passing through Swift security tools and auto archive tools it will be stored into archive storage. While archiving we will set expirer to each video. Expirer defines after how long the video will be deleted. Finally, after storing videos successfully the information will be updated to our database module.

Along with these we also have an interface for user purposes. Users will be able to upload and download videos through the interface.

This part will focus on how we can store the video files into our servers and archiving them as part of the process. For this we will use two servers: the first server will be a storage server which will store the raw video files without any compression and after a certain period like one month, the video file will be automatically deleted. The second server will be an archive server where the files will be stored indefinitely in compressed form. The reason for selecting two different servers is consumers might require some files immediately after a certain incident had occurred. They can access the storage server with ease and retrieve uncompressed and untampered data. But they might wish to access past videos for varied reasons, then they can retrieve their data from the archive server.

Here, we designed two models for this task. We will discuss both of the models advantages and drawbacks after talking about the models themselves. The first model will work as follows: First, video files from cctv's own NVR storage will be segmented using ffmpeg. Then it will be uploaded to the storage server at a fixed time at 12:00 each day and uploaded to the storage server with an object expiration header ¡object expiration link¿ set to "X-Delete-After:2592000". 2592000 is one month in seconds. This will delete the file in the storage server after a month. Before deleting, it will be compressed with a scheduled Cron job and then moved to the archive server with no expiration header.

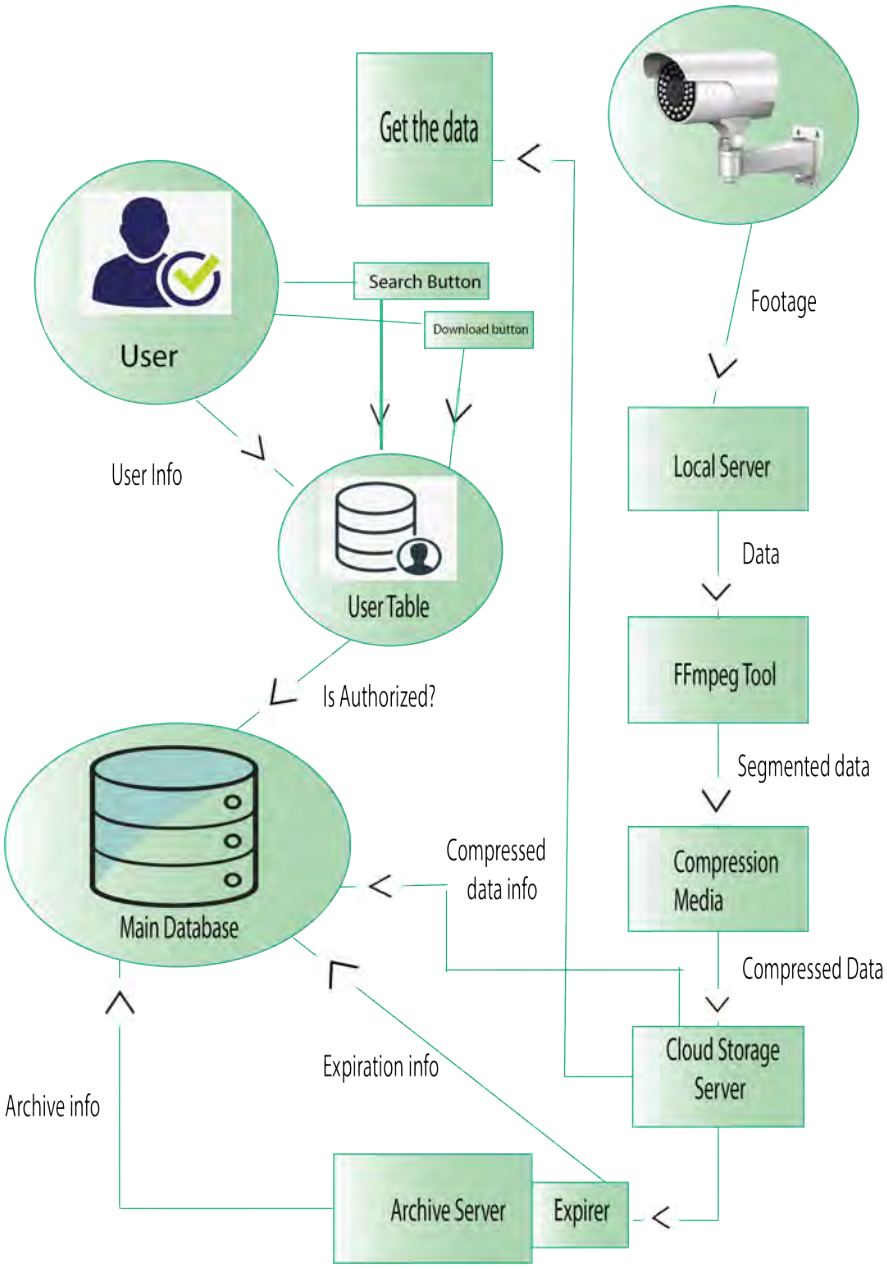

Figure 3.1: System Model of Our Thesis

In the second model, The video files will be uploaded parallelly to both the storage server and archive server. At first, the file will be segmented using ffmpeg. After that, in the storage server it will be uploaded with the expiration header set to ""X-Delete-After:2592000". Which means it will be deleted automatically after a month. In the archive server, we will compress the segmented ffmpeg files and upload them with no expiration headers set. So files uploaded here will not be deleted.

The advantage of the first model is, it will not waste space like the second model. In the second model, identical data will sit in both storage server and archive server. Data kept in Archive Server will sit idly with no use because if someone needs access to that data, they could access it in the storage server instead. But the drawback is, sending data from a container to a different server is much harder. It also proved

hard to keep track of and unnecessarily raised the complexity of our database. But the biggest drawback we found is, swift does not have any compression module built in. If it had an optional middleware for compression, perhaps this model would make the most sense. But because of this, we cannot directly send the data from our storage server to Archive server. First we would have to download the data and compress it. After compressing and logging it into our database, we could reupload it to the archive server. This would cost more bandwidth and make the process lengthier than it needs to be.

The second model is intuitively simpler, does not need the download and reupload process. So what we lose in storage, we make up in bandwidth saving, time and efficiency. For these reasons we chose to implement the second model in our project.

3.2 Surveillance System

3.2.1 CCTV/IP Cameras

Before going to depth of this model, we need to understand which components are being used for CCTV/IP cameras because we need to know the whole system of the CCTV system how it works and what will be the tools in terms of connection and data storage [Figure 3.2].

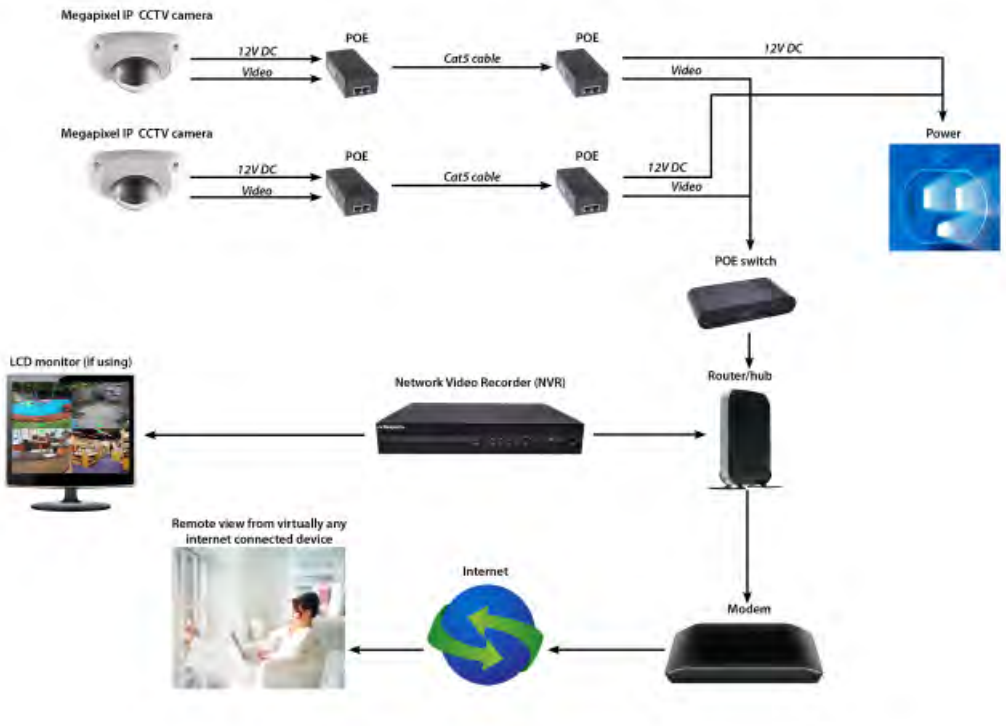

Figure 3.2: Components of CCTV [26]

There are many companies who provide CCTV cameras of good quality but camera is not only the thing here. Moreover, we also need to know the components which will be used during the installation and connection and initially it will be in our university area for primary test. We have proposed to use a CCTV/IP camera system which will have NVR for storing the data and other components like cat5 cable, optical fiber, power cable, PoE (Power over Ethernet) switch, PVC (Polyvinyl Chloride Channel) for the installation. Additionally, CCTV uses coax and a power cable but IP camera uses standard network cable (Cat 5e or Cat 6) and after some time, we will talk about this more why we want to use IP camera over CCTV camera. From the figure below we will be able to see components of CCTV/IP cameras [26].

We have surveyed CCTV and IP cameras of the leading companies in Bangladesh and they are Dahua, Jovision, and Bosch etc. as these are leading the market. Most of the companies import these cameras from China though Bosch is imported from Germany.When it's about cameras whether it is smartphone, DSLR or any other cameras, we firstly think about the camera quality of the camera. In medium and small organizations and houses, the most used camera is of 2 megapixels. Every larger company or organization uses 3MP, 5 MP, 8MP, 13MP etc. All these cameras are used depending on user and industry demand.

As we have talked about the camera, resolution is the main factor of it. Our main aim is to create a model about cloud storage. So, resolution is not the main factor of this research. But, when it's about the user, we have to think about it and we have gone through the previous research of CCTV camera resolution to get an idea of it so that users cannot claim anything about it. From the figure below we see the resolution lines of CCTV cameras [27]. To get the resolution of great quality, we will be using a good IP/CCTV camera because we must need to know that resolution depends upon the number of pixels in the CCD (Charge-coupled device) chip. In other words, the resolution is directly proportional to the number of pixels in the CCD chip [27].

Additionally, CCTV is much lower in terms of resolution than IP cameras. The resolution of analogue cameras results very small field of view when compared to IP cameras from an old evaluation.

Table 3.1: Analogue and Digital Cameras [28]

Designation	HxV
CIF	352x240
2CIF	704x240
4CIF	704x480
D1	720x480
720p HDTV	1280x720
1.3MP	1280x1028
2MP	1600x1200/1920x1080

This figure 3.3 is an old history of CCTV cameras but modern science has made it vaster [29]. Now-a-days, we can get 4K Video cameras for the best quality of resolution and footage. But, in this research, we have focused on 2MP so that we can evaluate an average result for our proposed model because we have proposed the model for all range of users.

Figure 3.3: Quality and Wide Dynamic Range [30]

Figure 3.3 shows that if the camera does not have good quality of resolution and dynamic range, the output will be unacceptable [31]. To take care of this, we need to know the "WDR" term and in market there is many kinds of WDR (Wide Dynamic Range) with different labels. WDR is mainly used in cameras for getting the good result of images because many cameras cannot provide the good daylight and night pictures and it makes the quality of the image too exposed or dark respectively.

3.2.2 Depth of Local Storage

Till now, we already have known the basics of CCTV/IP camera's components and all other things. Now, let's get into the storage. Basically, when it's about local storage, we will use here NVR or DVR for storing the footages and it depends on user demand though NVR is the most powerful storage system for storing the data. If the cameras are used for long distance areas, then NVR is recommended for network stability. Before going to the cloud storage part, we have to know calculate more few things so that we can settle a simple cloud operating storage and efficient and cheap system.

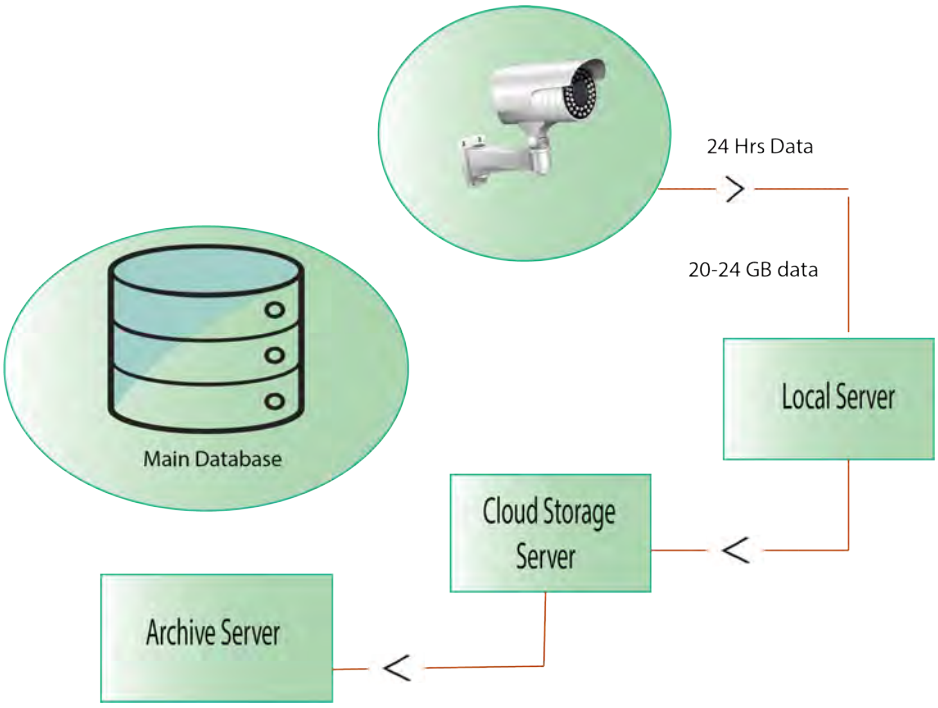

Figure 3.4: Surveillance System to Storage

First of all, we have to choose a storage for NVR like HDD, flash drive, SD card, etc. But we are working on CCTV footage, we need HDD for storing the footage as following figure shows the expected storage, we need to store the footage is too big and it will not be a good idea to use here flash drive or SD card as a storage though we always think about user demand. We have prepared this equation in Seagate and SuperCircuits website for better calculation.

From the figure 3.5, we can be assured that NVR is obviously much efficient and quality full for storing the footage though many users wants DVR because they do not know the inside calculation of it [32] and [33]. NVR always provides the best quality of the footage in terms of frame size. Though high bandwidth is required, good quality always requires some expenses. Moreover, we have proposed to use H.264 in terms of 24hrs data though future models can be used by H.265 (HEVC). And for the ease of calculation we have evaluate desired days of storage is 1 day. Actually, everything depends here on user demand because a user may want to store his data for 20 days and then may want to delete that for making space in the storage and store new footage of camera. In our whole model we have focused on NVR based storage so that we can get the best result and our model will be more reliable to every type of user.

Storage Type	DVR	NVR
Camera Stream	H.264	H.264 (Good Quality)
Camera Resolution	2 MP (1920*1080)	2 MP (1920*1080)
Video Quality	Medium	Medium
Average Frame size	13.714 KB	21.2 KB
Frame Rate	15	15
Hours/Day	24	24
Bandwidth Required	2.4 Mbps/per Camera	2.6 Mbps/per Camera
Desired Days of Storage	1	1
Estimated Storage (Depends on camera)	16-37 GB (Approx.)	18-28 GB (Approx.)

Figure 3.5: Calculation of NVR and DVR [34]

3.3 Storage Server

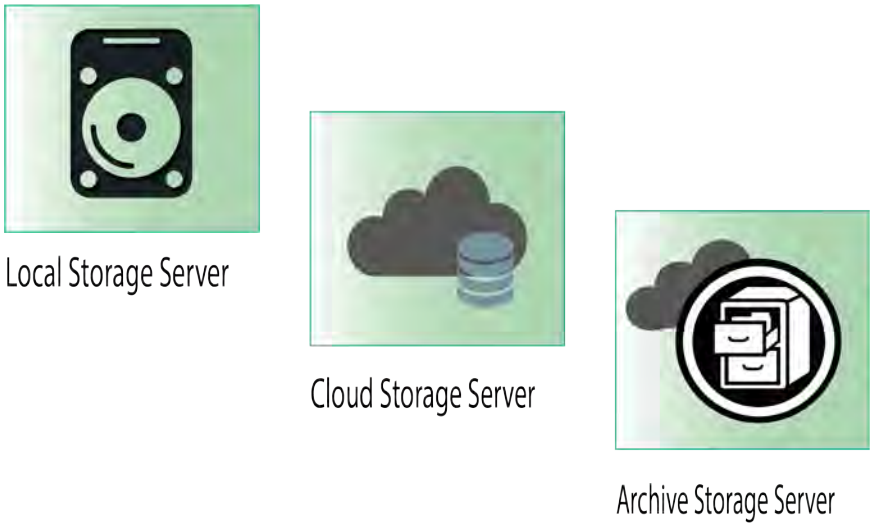

Figure 3.6: Storage Server

From small to large, every device needs storage to keep the data for the user. So, let's get into the storage part of our proposed model. In this model, we have used three types of storage for the best storing management [Figure 3.6]. And the storages are local storage, swift cloud storage, and archive storage of cloud.

Video optimization is the main important thing in terms of cloud because when we store the footage on cloud, we must take care of the video file size. The main reason for it is the bandwidth and it makes our expenses too high. Already there is a paper based on storage optimization where they have implemented a method to reduce the size of the video clips [35]. In that research, they have actually made a model and integrated that into a program so that they can perform every stage of the model individually and they also prepared their model by the help of Seagate's technical research paper. On the other hand, in our proposed model, we already have made a calculation in the Seagate website and we have thought of compression and segmentation so that users do not fall into hassle for the stored data. In the next discussion, we will expand about it.

3.4 FFmpeg Media Converter

In terms of video footage, we have to take care of many things because when we send the data to the compression media, the internet connection can be lost or the file can be damaged. So, if we make the file segmented, it will help us to video more synchronously and data will never be damaged and the user experience side will be smoother and clearer. For instance, if we make 100MB files each from 10GB data, there will be no problem, if we have a slow connection or low bandwidth.

Converting video:
ffmpeg -i “lala.mp4” -c:v libx264 wildlife.mp4
Splitting video:
ffmpeg -i ”lala.mp4” -map 0 -c copy -acodec copy -ss 0:02:00 -t 0:04:00 ”lala-1.mp4”

splitting into 8second chunks:
ffmpeg -i ”lala.mp4” -ss 164 -f segment -segment_time 120 -vcodec copy -reset timestamps 1 -map 0:0 -an output_video%d.mp4

splitting into 120 second chunks:
ffmpeg -i ”lala.mp4” -c copy -map 0 -segment_time 120 -f segment output%03d.mp4

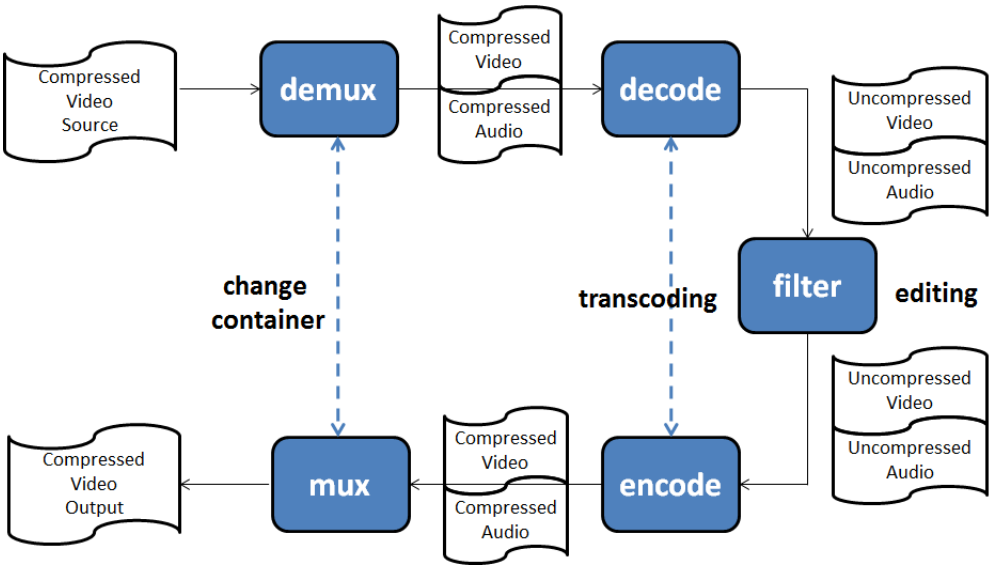

Figure 3.7: FFmpeg Architecture

From figure 3.7 we can see the internal architecture of FFmpeg and how it works [36]. FFmpeg is built with a number of self-contained libraries which provide discreet functionality that can be included into other applications. These features include codec encoding and decoding, compression, image scaling, resampling, and format conversion.

3.5 Object Storage

When we think about the cloud system, Amazon cloud services come first in our mind but for the cost of these services, we don't step forward to make a cloud storage and we decide to use only local storage services and delete data after a certain time and it is like watching the video surveillance moving around and using as a showpiece which is the same thing of not using the camera. So, we have started to dig this thing deeper and lastly, we got an older research which delivers our desired model a great boost though there are many ISPs in Bangladesh who provide cloud systems for the users if they want. From that research [37], we become clearer that Windows Azure or other services like Amazon are so much more costly.

Because surveillance systems need a huge amount of storage and it increases the expenses more than anyone expects as when it's about money, no one wants to harm their businesses by expensing more without having their profit and revenue. Figure 7 shows that the estimated price for a VMS is too much expensive. And the cloud is not the only thing when we think about a full model because there are also many other components for the setup of a video surveillance system such as cameras, small/medium computers, NVR, bandwidth, database of higher storage, etc.

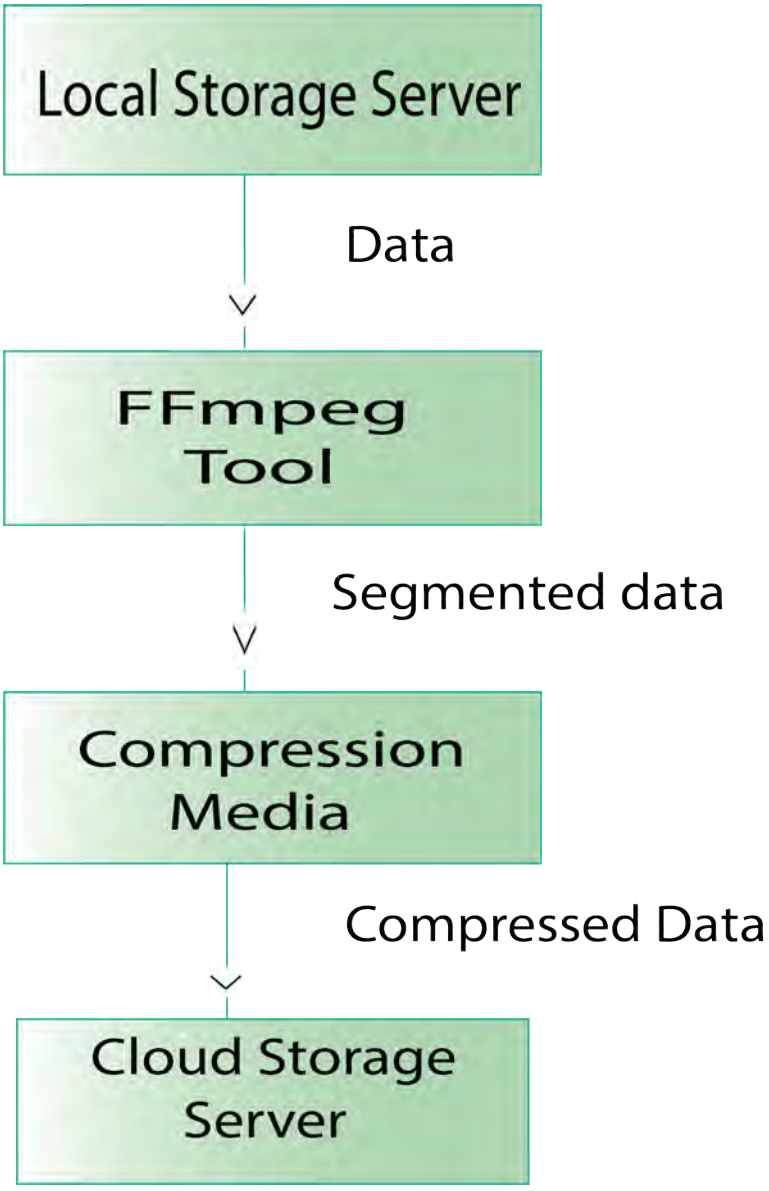

Figure 3.8: Local Storage to Cloud Storage

So, we have started to think about how it can be more flexible to everyone and then the journey of going with OpenStack SAIO comes in our mind and we started to work on it.

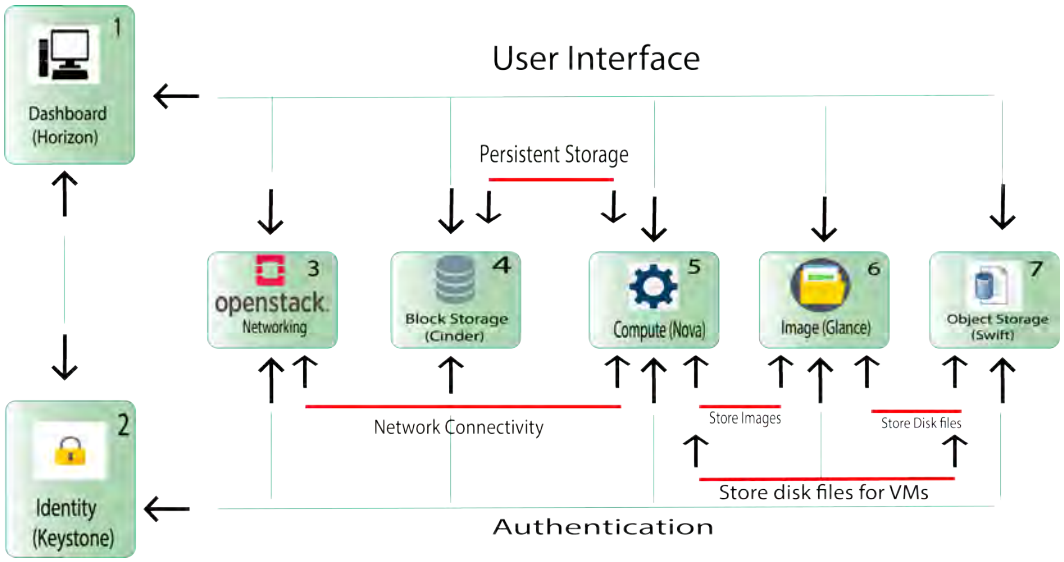

Figure 3.9: OpenStack Internal Diagram

3.5.1 OpenStack Swift

OpenStack swift is an Object Storage software, which is known for its capability to store huge amounts of data with ease. The data can be stored for a very long period of time with minimal financial cost. And OpenStack Swift/ OpenStack Object storage being open source, it gives developers an advantage. It can be customized to any one's needs and can be scaled up or down accordingly. More importantly it can be used to make clusters of servers using traditional server hardware.

OpenStack Swift was developed by Rack space Hosting Inc. It is based on Cloud Files technology invented by the same company mentioned above. Along with NASA, Rack space started this project in 2010. Also, both of these companies made the community for OpenStack swift, which is responsible for maintaining and developing the software as of now. OpenStack swift provides many services in this one software, among these computing services, storage services are most used to create a cloud computing service experience. The main use of these are to store/upload, create replications or backups and archiving huge amounts of data [38].

3.5.2 Characteristics of Swift

Swift is an open source software designed to be used as an Object Storage system. It is part of OpenStack and is completely free to use. Swift can is very versatile and can be used alongside an existing cloud computing environment or it can also be used solo as an Object storage system. Because of its efficiency and scalability, many major companies use swift as their go to cloud storage. In recent times, Swift is being run in many big companies like IBM, HP and Rack space Cloud files. Apart from these, Swift is also used by many individuals as their own private storage clusters. Swift is meant to be run on Linux distributions. And it can also run on any x86 hardware setup. Swift has an architecture, which is called "Eventual Consistency Architecture". This allows Swift to create very enormous cloud infrastructures, which can store tons of unstructured data. Objects in Object Storage has a unique identifier. In Swift, each object has an URL given to them. Objects can be accessed by going to this URL. Data can be stored and retrieved from Swift server in a specific way. Globally popular RESTful HTTP API is the most popular way to do so. More on these is discussed in the Swift API section.

While being stored in Swift Object storage, each object will have some metadata attached to it. These might be identifiers or unique tags, which will be used to organise the objects in a particular way, or might be used to search the objects to find an object. Every single object in the storage has multiple copies of it in different regions. This is called replication. Swift is hugely modular, so it can be scaler up or down easily. By adding nodes, one can increase the size of the server. This is a very less costly and fast solution in terms of increasing capacity. Anytime there is a failure in nodes, the entire down node can be swapped without hampering the functionalities of the OpenStack server and cluster. There is literally no down time in this system. Similarly, we can add new nodes and drives and also remove faulty ones with the system running smoothly.

There are many companies which are providing Object Storage systems in the cloud. Amazon S3 and Swift being on top of these companies in terms of popularity. Both of these systems provide huge scalability, which means the ability to increase the storage size and capacity. And also ensures that the data is always available, no matter whenever it is requested. Because of this consistent design, no efficiency gap is being created here. The client must keep up with the need to upload or store, organize, manage and access a huge number of data.

The way Swift archives its availability is by making and storing multiple copies of data which are stored. At first glance it might seem unnecessary to make multiple copies of a file. But if we think about it, if a node fails or a specific server containing the data file crashes; then the data is lost forever. But with multiple copies in different nodes, even if a node fails, we can still retrieve the data from other nodes. When it needs very high performance and the power to scale the storage, this system shines the best. It is particularly useful for very massive and scattered distributed infrastructures. And works best with data which are being stored into multiple different regions all over the world.

3.5.3 Different Types of Storage System

In most of our daily lives, we are using some kind of storage to store our files. Among these, File storage is the most popular one. File storage is also known as file-level storage or file-based storage. In file storage, data is treated as a single information, and it is placed inside a folder. Each folder might contain more folders inside of them. When someone needs to excess a specific file, they will need the path of the file to find it. In this file system, data is stored in an organized fashion and can be retrieved by using a metadata , which contains information about exactly where a file is in the storage or computer .

Another popular storage system is called block storage. The concept of block storage goes as follows; here data is cut into blocks. And these pieces can be stored separately. The advantage is that a small block of data can be stored anywhere it finds a small space for that block. And each block can be identified by the unique identifier they are given when they are being chopped into a block. Which means that some part of the same data can be stored into different operating system environments.

Finally, the third type of storage that is being used more and more these days is called Object Storage. Swift is an Object Storage system. It is also called Object-based storage. This is like a flat structure and files are broken into small parts and are distributed among the flat surface of the hardware. The broken parts are called objects. All the objects are stored into a repository. As we know of Object Storage, OpenStack Swift also keeps data as binary objects. It is stored in the storage server's underlying file system. Every object in the storage is attached to a metadata, and this metadata contains other details about that specific object.

3.5.4 Swift Architecture

Cloud computing works by virtualizing the physical device to a remote server. All the things from processing to storage are being done there. The system which is used to do all the middle work is OpenStack. OpenStack is an open source cloud computing software package that can handle a huge amount of data, can process the data, has security features and a lot more. Different parts of the software handle different tasks. The dashboard program which is called Horizon controls the monitoring features of the entire system and is being used as a dashboard. Nova provides the computation service. The 2 different types of storage available at OpenStack are Swift, which can be used for Object Storage. In addition to that, there is Cinder, which is responsible for Block Storage operations. Connectivity to other networks are handled by Neutron, while Keystone works on the security and authentication side of the entire system. All these packages and many more, together they make the OpenStack cloud service software.

The first step is understanding how a file is being saved on an object storage system. The object storage system is divided into several parts or components. In the very first layer, there is the proxy server. All the data that goes in and out of the

storage has to go through the HTTP file transfer protocol. The requests for data are done by API requests. The task of the proxy server is to capture the requests and work accordingly. The requests can be understood by looking at the URL of that request. Proxy server determines the location of the data or it's storage node by the URL. Proxy server also handles any failed transmission or failed data packets and maintains a timestamp. After this, there are Rings, which keep the address of the information like names and entries which are stored on the cluster and also keep track of the actual physical location of the data. Ring maps these two together. The way Rings keep the mapping work is by introducing zones, devices, partitions, and replicas. Zones might be any storage devices like a hard drive to a full server. Zones are generally located differently. The main purpose of this is to minimize any kind of data failure. Data is broken into small parts and different parts are stored in different zones. So even if a part of the data gets damaged in a particular zone, the rest of the file from other zones will be fine. After that there are containers and accounts. These are SQLite databases. The list of containers in a particular account is being stored in that account's database. Similarly, a list of all the objects inside a container is being stored in the container's database [39].

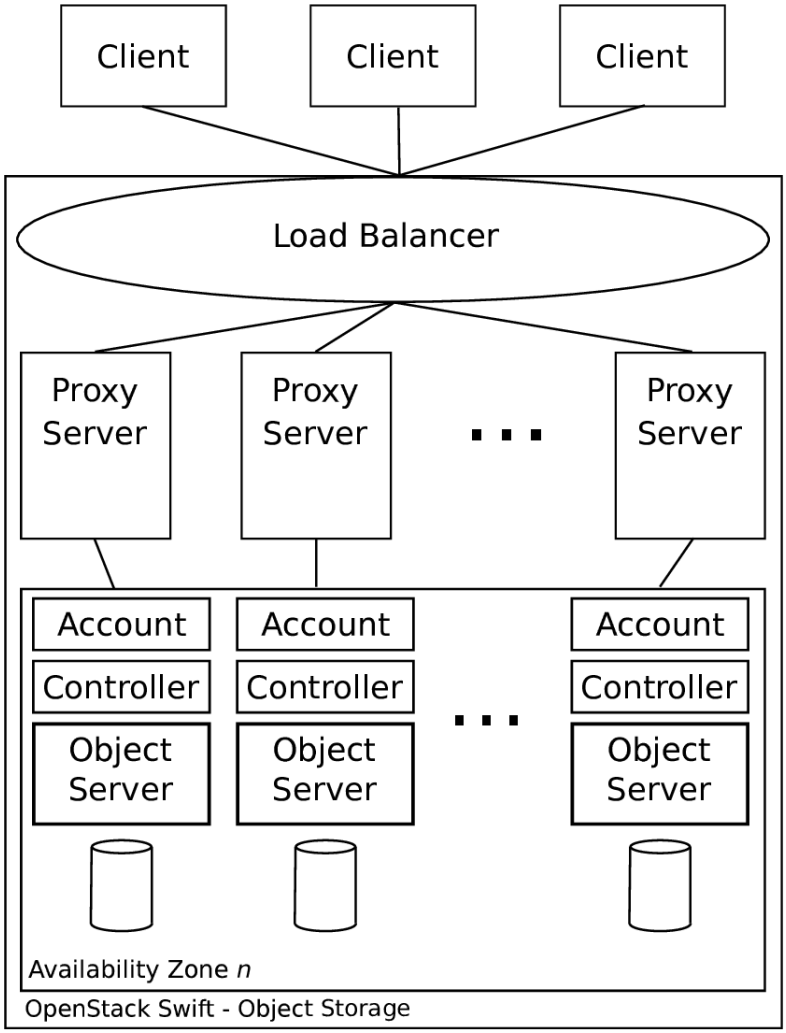

Figure 3.10: OpenStack Swift Architecture [40]

The system that processes OpenStack object storage SWIFT data is called nodes which is a physical server and multiple nodes make a cluster. Clusters are able to run all the nodes containing processes and services that need to be run for executing swift. There are zones and regions in cluster [41]. When multiple physical nodes are separated by geography are considered as zones. Clusters contain a minimum one region but there can be more than one region in a cluster. When the cluster has multiple regions it's called Multi-region cluster (MRC). These regions have read and write affinity. Now, regions can have more than one zone as they can have separate hardware facilities and failure of one facility won't affect another zone.

Swift server processes contain mainly four types of processes such as proxy, container, account and object. These processes are referred as swift process layers when these processes are executing on nodes as in proxy layer, container layer, account layer and object layer. In this layer, the proxy server process mainly deals with the read and write request of the client as all the massage processes with HTTP verbs like GET, PUT, POST, DELETE [41]. This layer works with all the API requests. It also deals with the failures and timestamps. Account server processes mainly deal with the metadata of individual accounts and the container that is listed in the account and all the information regarding this process is contained in SQLite database.

Container layer: In this layer, Container server processes deals with the meta data contained in the container and the object lists of this particular container. This information is also stored in SQLite database. Object layer: object server process layer is accountable for storage services such as store, delete or retrieving objects from the drives of its node. Objects are stored as binary files. These binary files are made by partitions and timestamps. In this layer, all the data and metadata is considered as a single unit.

Consistency services mainly work when account, container or object server processes are working and there can be data corruption or hardware failure. Consistency services work in the background to detect any failure. Auditor and replicator are two main consistency services. But there are other specialized services such as account reaper, container and object updaters, object expirer.

Auditor runs in the background of the account, container and object server process to ensure the stored data is not having a file-system corruption. If there is an error, auditors send those data into the quarantine area. Replicator runs alongside the account, container and object server process and works with object and container detection and also compares the accounts examining the local nodes and if it finds a missing file, a copy of the local data file will be sent to the node.

When an account is deleted, the account reaper terminates all the association of that account with objects and containers. It can also be configured in a way so that there is a bit of time before removing the account. Container updates mainly keep the container up to date and updates container count, metadata and bytes used in the account. And object updates keep the object listing updated. Object expirer is responsible for the configuration that will delete the object within the designed time.

In Swift, ring is a collection of tables and they are assigned into different nodes of a cluster for mapping data. Account, object and container have separate rings for mapping the location of any operation. Ring specially works with zones, devices and replicas for mapping purposes [42]. Ring has different partitions and the location of the partitions are stored in a mapping. Every ring has a consistent hashing ring. Ring data structure deals with a device list of cluster, device id list referring to the portions of device assignment and an integer to indicate number of bits to shift MD5 hash for calculation partition of the hash [43].

Partition power is the value that is used for calculation partition in a ring and it should remain unchanged after initialization keeping the total number of partitions unchanged. Furthermore, the partitions need to be assigned into the exact locations from the starting point so that, due to re-balancing, partitions are not affected. Replica count, replica lock, weight unique placement is responsible for the placement of the partitions [44].

3.5.5 OpenStack Swift Installation

OpenStack being one of the top tier cloud computing platforms has seen a growth in its user base in the recent years. In our project, we developed our entire system based on OpenStack Swift All in One. In addition to being very powerful and flexible, the main reason for choosing this was that it is open source. So that we used and tinkered OpenStack according to our needs. In our project, we did all our setup in a virtual environment. As it is easier and more convenient to do work on a Virtual machine then an actual machine. A Virtual machine can be configured and readjusted according to our needs.

The very first step while installing is to make sure that the system is up to date. After that we installed the dependencies. The code goes as follows [45]:

sudo apt-get update
sudo apt-get install curl gcc memcached rsync sqlite3 xfsprogs $\backslash git - corelibffi - devpython - setuptools\backslash liberasurecode - devlibssl - dev$

sudo apt-get install python-coverage python-dev python-nose $\backslash python{-}xattrpython{-}eventlet\backslash python{-}greenletpython{-}pastedeploy\backslash python{-}netifacespython{-}pippython{-}dnspython\backslash python - mock$

Here we update the Linux system and install all the packages that we need to run swift on our Virtual machine.

As swift setup requires an initial storage, we used a loopback device for our storage. It is recommended to use at least 30 GB of storage. But it can increase or decrease depending on your needs. One thing that has to be ensured about is that the system has a XFS file system running. If there is no XFS file system available, then we ran the following code to fix it.

sudo apt-get install xfsprogs

After that, we had to edit the “fstab” file located at /etc/fstab.

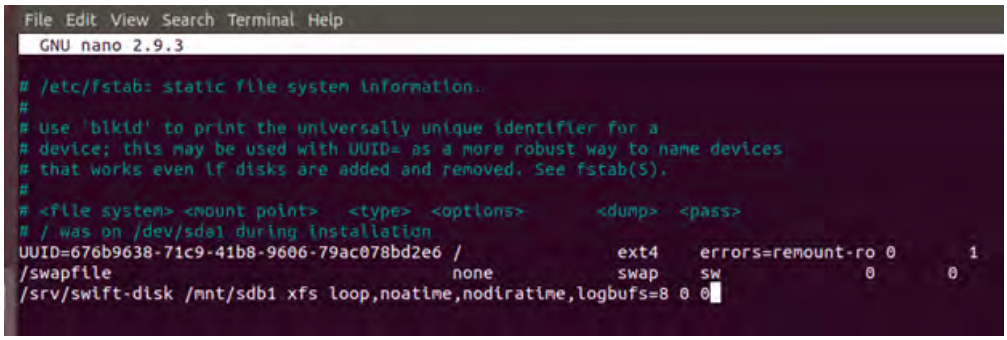

Figure 3.11: Edit ”fstab” file

In some cases, xfs file systems can be created with some commands. After that the device has to be mounted so that it can be used. Then some directories were created to prepare the system for the swift setup.

The next part was installing Swift. The straightforward way is to clone it from GitHub. There might be several existing versions. But one should always try to install the most recent stable version. Newer versions might have some bugs and inconvenience, which the stable versions do not have. We checked from GitHub and downloaded the stable one.

The following code was executed to install swift:

cd $HOME; git clone https://github.com/openstack/python-swiftclient.git

cd $HOME/python-swiftclient; sudo python setup.py develop; cd –

cd $HOME/python-swiftclient; sudo pip install -r requirements.txt; sudo python setup.py develop; cd –

git clone https://github.com/openstack/swift.git

Setting up the Rsync is the next part. We edited the Rsync.conf file and enabled Rsync to Enable. Then we used the enable commanded and started the rsync. At this time, we verified that Rsync is accepting connections for all servers. And as with Rsync, we also have to start and enable memcached. We used 2 lines of code to enable and start memcached which is an essential part of our setup. Configuring

each node is also an important task as it will ensure that each node is working as it’s supposed to. Then we had to construct rings for our swift setup. We wrote a ringmaker script with the following value:

```
#!/bin/bash

set -e

cd /etc/swift

rm -f *.builder *.ring.gz backups/*.builder backups/*.ring.gz

swift-ring-builder object.builder create 3 3 1

swift-ring-builder object.builder add r1z1-127.0.0.1:6210/sdb1 1

swift-ring-builder object.builder add r1z2-127.0.0.2:6220/sdb2 1

swift-ring-builder object.builder add r1z3-127.0.0.3:6230/sdb3 1

swift-ring-builder object.builder add r1z4-127.0.0.4:6240/sdb4 1

swift-ring-builder object.builder rebalance

swift-ring-builder object-1.builder create 3 1 1

swift-ring-builder object-1.builder add r1z1-127.0.0.1:6210/sdb1 1

swift-ring-builder object-1.builder add r1z2-127.0.0.2:6220/sdb2 1

swift-ring-builder object-1.builder add r1z3-127.0.0.3:6230/sdb3 1

swift-ring-builder object-1.builder add r1z4-127.0.0.4:6240/sdb4 1

swift-ring-builder object-1.builder rebalance

swift-ring-builder object-2.builder create 3 3 1

swift-ring-builder object-2.builder add r1z1-127.0.0.1:6210/sdb1 1

swift-ring-builder object-2.builder add r1z1-127.0.0.1:6210/sdb5 1

swift-ring-builder object-2.builder add r1z2-127.0.0.2:6220/sdb2 1

swift-ring-builder object-2.builder add r1z2-127.0.0.2:6220/sdb6 1

swift-ring-builder object-2.builder add r1z3-127.0.0.3:6230/sdb3 1

swift-ring-builder object-2.builder add r1z3-127.0.0.3:6230/sdb7 1

swift-ring-builder object-2.builder add r1z4-127.0.0.4:6240/sdb4 1

swift-ring-builder object-2.builder add r1z4-127.0.0.4:6240/sdb8 1
```

swift-ring-builder object-2.builder rebalance

swift-ring-builder container.builder create 3 3 1

swift-ring-builder container.builder add r1z1-127.0.0.1:6211/sdb1 1

swift-ring-builder container.builder add r1z2-127.0.0.2:6221/sdb2 1

swift-ring-builder container.builder add r1z3-127.0.0.3:6231/sdb3 1

swift-ring-builder container.builder add r1z4-127.0.0.4:6241/sdb4 1

swift-ring-builder container.builder rebalance

swift-ring-builder account.builder create 3 3 1

swift-ring-builder account.builder add r1z1-127.0.0.1:6212/sdb1 1

swift-ring-builder account.builder add r1z2-127.0.0.2:6222/sdb2 1

swift-ring-builder account.builder add r1z3-127.0.0.3:6232/sdb3 1

swift-ring-builder account.builder add r1z4-127.0.0.4:6242/sdb4 1

swift-ring-builder account.builder rebalance

Here, Object builder will act as the storage server. So, when we created it, we gave it 3 3 1 values. The first value is ¡part-power¿ which creates 2n̂ partitions. Since we gave it the value 3, it will create 8 partitions. The Next value is replication count. Since we want extra security in our storage server, we gave it the value 3 which will create 3 replications or copies of the same file and store it in 3 different locations. The last value is part-hour which does not have any effect in our system so we gave it the default value 1. Object-1 will function as our archive server. The values are identical to our storage server except for replication count. Since we are archiving old files, we felt we do not require a safety net so to save space we gave it a replication count of 1 which means it will store only one instance of the video file.

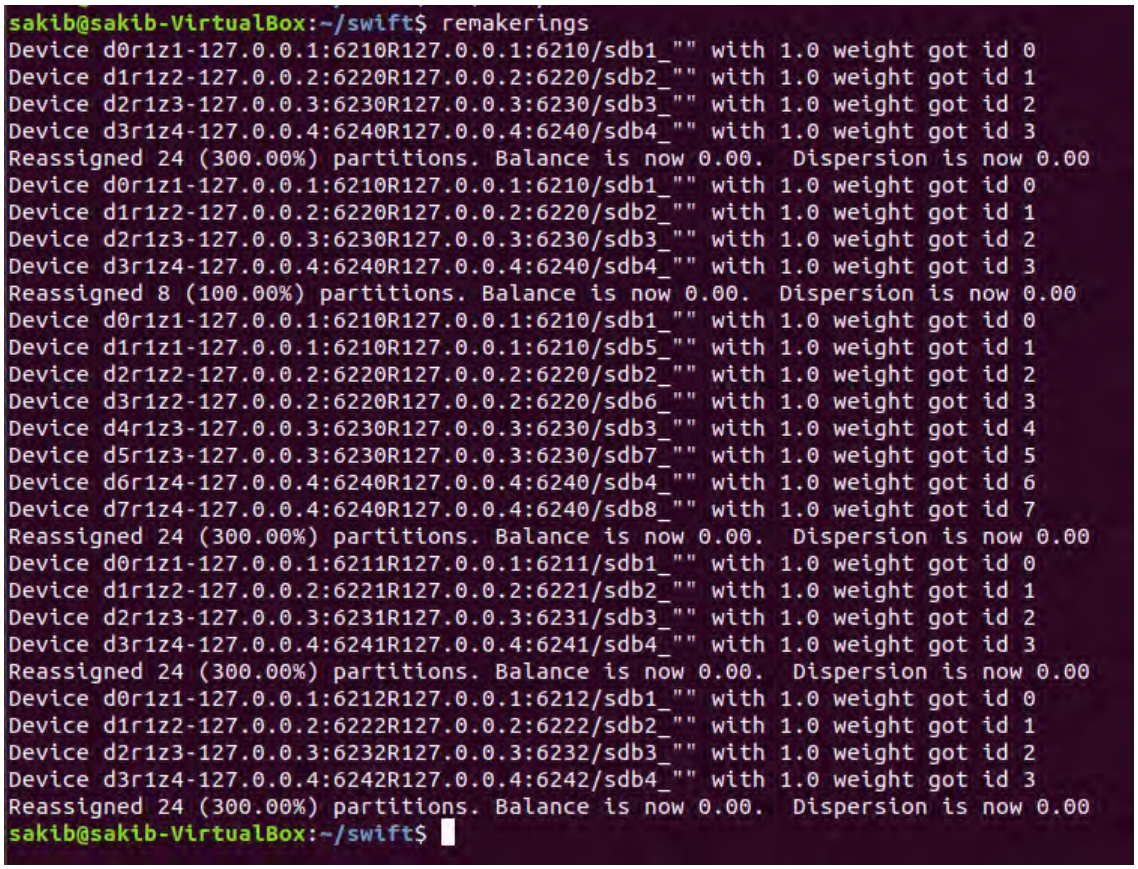

Figure 3.12: Result of Ring Build

Later we also used some additional scripts. Start main script is among them. These scripts contain codes which will automatically be used to start the swift storage system. This completes our initial setup phase. Then comes Account Authentication and connecting to the swift server. To connect to a swift server, we need to generate an X-Storage-URL and X-Auth-Token. Which we find using a line of code and giving users as users. Output gives us an X-Storage-URL and X-Auth-Token. These tokens are valid for a certain amount of time. When used with a get command, we can retrieve data and information of our swift setup and do many more things.

The next phase is to create a container. The container will store the files which we will upload to the container and we can download our file from the container if we need to. Downloading and uploading a file is very easy and can be done with one line of code, provided that we give the right X-Storage-URL and X-Auth-Token with the upload or download request.

3.6 User Interface

Our workflow is designed for simplicity and high effectiveness. We wanted to give the client or end users an easy to use interface which will be both intuitive and provide fast results. Most of the task, which starts from getting the video data from CCTV to splitting to processing and compression; most of these will be handled automatically on the cloud by our system. There are only a few times when clients may want to try to connect to the storage manually. Because of that reason, we came up with a user interface to make things flow smoothly and that client can get the job done without knowing any technical understanding of the entire video storage system.

As the first step of creating a user interface, we did the design part. We tried to keep it simple and clean, while being fully functional and intuitive. We decided to use a Python GUI library called Tkinter.

Tkinter is very flexible and easy to use. It has a simple learning curve and very well documentation available online. For the basic step, there are two buttons for uploading and downloading a file. Whenever a button is pressed, Tkinter executes the HTTP Request command attached to it.

Sudo Code of User Interface:

Start

Define Upload button

Define Download button

On click: Upload button:

Connect to API

Authentication

Upload pre-selected file into container

Show Status of the container

On click: Download button:

Connect to API

Authentication

Download pre-selected file into container

Show Status of the container

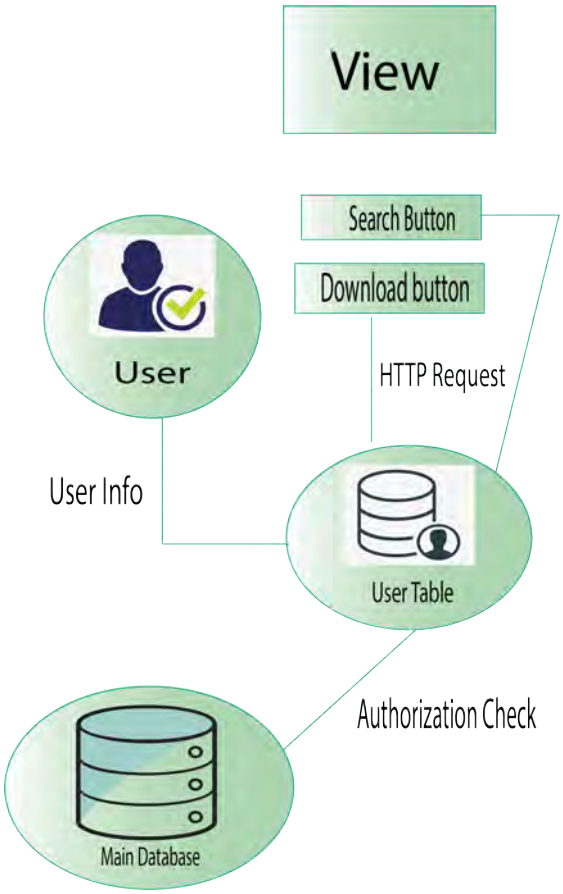

Figure 3.13: User model

Here we can see that there are two buttons for uploading a file and downloading a file. When a file is uploaded, we don't actually see any confirmation on the terminal but we get no error message.

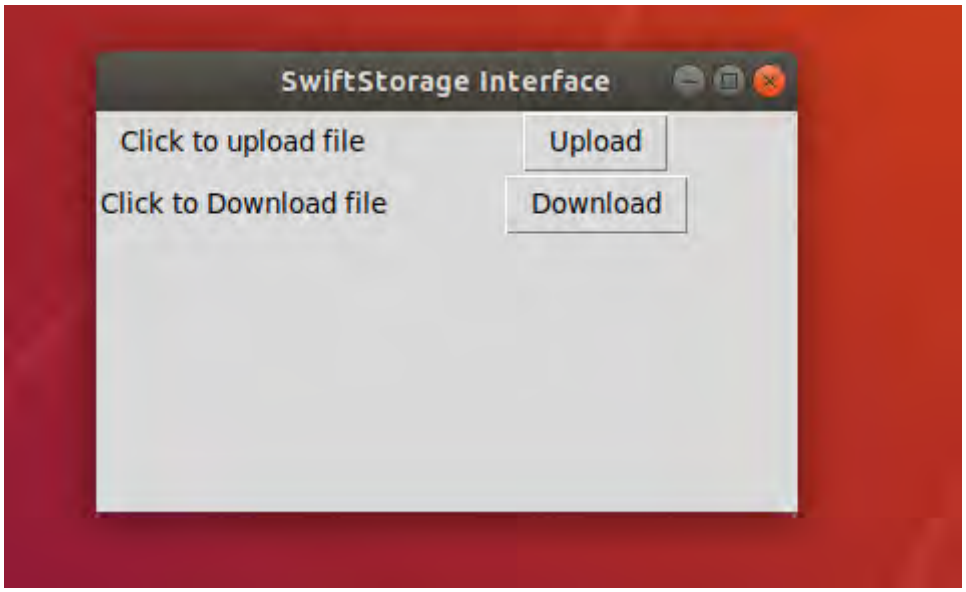

Figure 3.14: User Interface GUI

```
juli@juli-VirtualBox:~$ swift -A http://127.0.0.1:8080/auth/v1.
0/ -U test:tester -K testing upload mycontainer julifile.docx
julifile.docx
```

Figure 3.15: Uploading a File

```
juli@juli-VirtualBox:~$ swift -A http://127.0.0.1:8080/auth/v1.
0/ -U test:tester -K testing download mycontainer julifile.docx
julifile.docx [auth 0.004s, headers 0.039s, total 0.040s, 11.17
8 MB/s]
```

Figure 3.16: Downloading a File

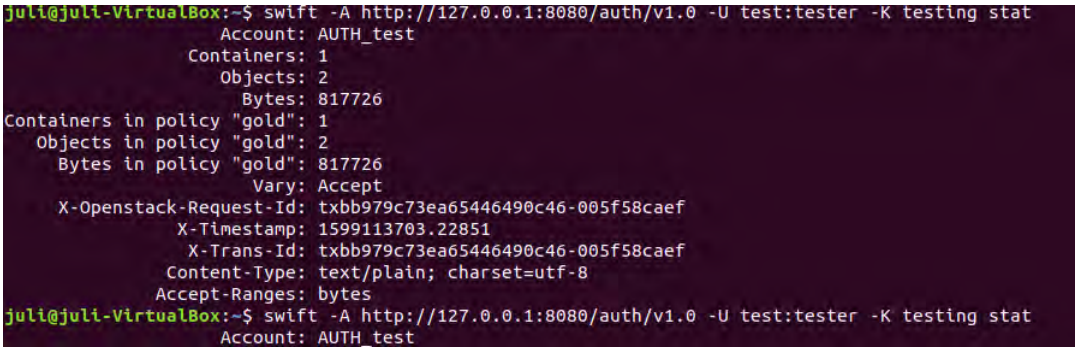

Figure 3.17: Container Stats After Upload and Download

3.7 Request

All objects and data stored in the swift server can be accessed with HTTP Requests. In Python, there is the Python Requests library, which allows us to use HTTP requests directly from Python Scripts. Here we utilized that to get our expected results. Hypertext Transfer Protocol /HTTP acts as a communication medium between client and server. In our case, the client is our customer who wants to store and get data from CCTV. And the server is the OpenStack Swift server.

All the requests made by Swift are made by using the RESTful API, which uses HTTP. Any requests that are made must consist of 3-4 mandatory parts. These are as follows: Name of the request like GET POST PUT etc, Token of Authentication, Object Storage URL and finally any other things that might be necessary.

There are many kinds of requests which can be made via HTTP. These commands tell the server what to do. Like GET request will retrieve information about a data from the server, While PUT will upload a file or object into the server. Authentication tokens are needed to verify that one has access to the server. Tokens can be generated from the terminal when given the user, password and URL address. Tokens are valid for a limited time. After this time, the token gets expired and will not work. A new token has to be generated to work with the HTTP requests again.

In our case, the storage URL looked like this:

http://127.0.0.1:8080/v1/tester/mycontainer/file

Here, we can see that there are few parts of this storage URL. First part shows the location or the IP of the cluster. And then the second part states the Username, container name, and the file name. This URL specifies exactly where the HTTP requests should be made.

From the given URL, we can find 3 things about the Storage URL.

First, we have our Account part. This part is where the information about the containers and the account itself are kept. The Account name must be unique. One account can have multiple containers inside of it.

Similarly, a container is a storage area for keeping information about files inside the container and also about the container itself. One container must belong to an account. But a container can be empty or it can have many files in it.

Finally, in the last part of the URL, we have the object. It is where the actual file is kept and metadata related to the file is stored.

Following are the code we used to connect and GET / PUT objects in our Swift server using HTTP Requests.

3.7.1 Getting Access to Server and Authentication

The very first step toward communicating with the server is to get it validated.

curl -v -H 'X-Storage-User: test:tester' -H 'X-Storage-Pass: testing'

http://127.0.0.1:8080/auth/v1.0

These above codes provide the server with username and password and the url. The curl request then communicates with the server, and retrieves a set of X-Storage URL and X-Auth Token.

```
sakib@sakib-VirtualBox:~/swift$ curl -v -H 'X-Auth-Token: AUTH_tk1d8bd901517a4a4
194e53e0ea7c501f1' http://127.0.0.1:8080/v1/AUTH_test
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET /v1/AUTH_test HTTP/1.1
> Host: 127.0.0.1:8080
> User-Agent: curl/7.58.0
> Accept: */*
> X-Auth-Token: AUTH_tk1d8bd901517a4a4194e53e0ea7c501f1
>
< HTTP/1.1 204 No Content
< Content-Type: text/plain; charset=utf-8
< X-Account-Object-Count: 0
< X-Timestamp: 1601847191.53619
< X-Account-Bytes-Used: 0
< X-Account-Container-Count: 0
< X-Put-Timestamp: 1601847191.53619
< Content-Length: 0
< Vary: Accept
< X-Trans-Id: tx54e99a556b784cd5837bf-005f7a3f97
< X-Openstack-Request-Id: tx54e99a556b784cd5837bf-005f7a3f97
< Date: Sun, 04 Oct 2020 21:33:11 GMT
<
* Connection #0 to host 127.0.0.1 left intact
```

Figure 3.18: Request Generating Token

Both of these are necessary for authentication purposes.

The output should be as follows. Here we can see that the server provides us with a X-Auth-Token and a X-Storage-Token and a X-Storage-url. Now using X-Auth-Token and url for storage we send a GET request through curl:

curl -v -H 'X-Auth-Token:

AUTH_tk1d8bd901517a4a4194e53e0ea7c501f1'

http://127.0.0.1:8080/v1/AUTH_test

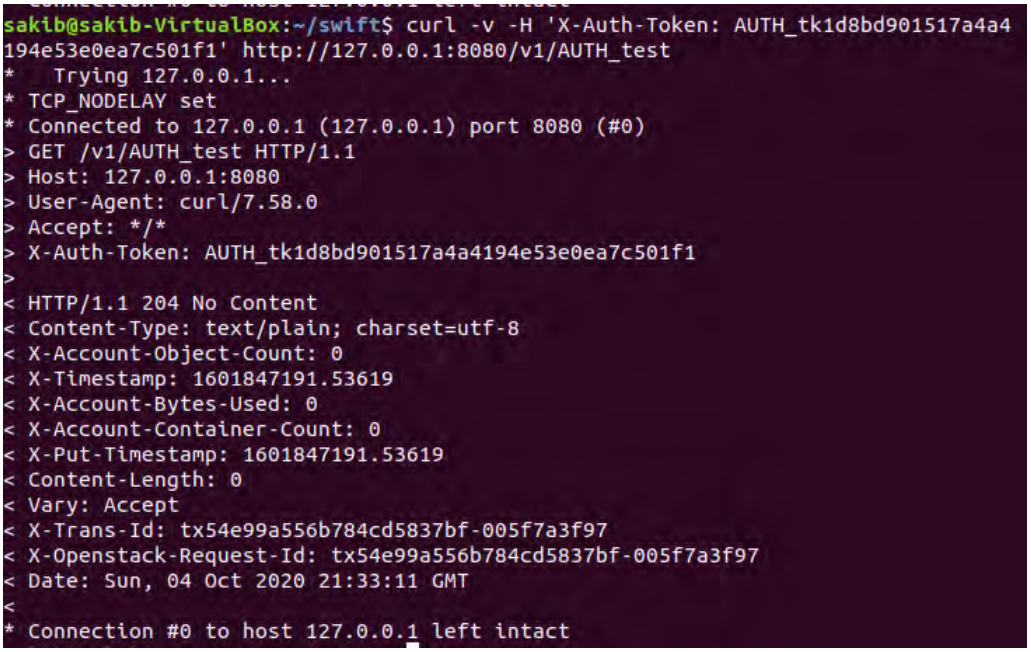

Figure 3.19: Get Account with Generated Token

This will show us that we gained access to that account.

3.7.2 Container Creation

Now to store data / files in the storage, we first have to create a container. Without a container, no objects can be stored. To create a container, we ran the following code:

curl -v -H 'X-Storage-Token:

AUTH_tk02e686223bab4a0fa959b211c5871a97' -X PUT

http://127.0.0.1:8080/v1/AUTH_test/mycontainer

This code will create a container called "mycontainer". Now inside this container, we can upload our files.

```
sakib@sakib-VirtualBox:~$ curl -v -H 'X-Storage-Token: AUTH_tk02e686223bab4a0fa9
59b211c5871a97' -X PUT http://127.0.0.1:8080/v1/AUTH_test/mycontainer
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> PUT /v1/AUTH_test/mycontainer HTTP/1.1
> Host: 127.0.0.1:8080
> User-Agent: curl/7.58.0
> Accept: */*
> X-Storage-Token: AUTH_tk02e686223bab4a0fa959b211c5871a97
>
< HTTP/1.1 201 Created
< Content-Length: 0
< Content-Type: text/html; charset=UTF-8
< X-Trans-Id: txa57c60807c574b8cb1f47-005ea39429
< X-Openstack-Request-Id: txa57c60807c574b8cb1f47-005ea39429
< Date: Sat, 25 Apr 2020 01:36:41 GMT
<
* Connection #0 to host 127.0.0.1 left intact
```

Figure 3.20: Container Creation

A 201 created will be there confirming that the container is created.

```
juli@juli-VirtualBox:~$ swift -A http://127.0.0.1:8080/auth/v1.0 -U test:tester -K testing stat
                  Account: AUTH_test
               Containers: 1
                  Objects: 2
                    Bytes: 817726
Containers in policy "gold": 1
   Objects in policy "gold": 2
     Bytes in policy "gold": 817726
                     Vary: Accept
     X-Openstack-Request-Id: txbb979c73ea65446490c46-005f58caef
              X-Timestamp: 1599113703.22851
               X-Trans-Id: txbb979c73ea65446490c46-005f58caef
             Content-Type: text/plain; charset=utf-8
           Accept-Ranges: bytes
juli@juli-VirtualBox:~$ swift -A http://127.0.0.1:8080/auth/v1.0 -U test:tester -K testing stat
                  Account: AUTH_test
```

Figure 3.21: Server Stats after container creation and uploading 2 files

3.7.3 HTTP Request

Now that the setup phase is complete, we can upload files to our newly created container.

To upload a file, do run the following code:

swift -A http://127.0.0.1:8080/auth/v1.0/ -U test:tester -K testing upload mycontainer myfile

Here, one thing to be noted is that myfile is the filename of the file we want to upload.

This code will successfully upload the file name myfile to our container named mycontainer.

To download the file, what we have to do is run the above code, but swap the request verb.

So instead of uploading, we have to write a download.

swift -A http://127.0.0.1:8080/auth/v1.0/ -U test:tester -K testing download mycontainer myfile

This will download the file named myfile from the container.

3.8 Large File Upload

Swift has a limit on the size of a single uploaded object. However, the download the size of a single object is virtually unlimited with the concept of segmentation. Segments of the larger object are uploaded and a special manifest file is created that, when downloaded, sends all the segments concatenated as a single object. This also offers much greater upload speed with the possibility of parallel uploads of the segments. Swifts DLO (Dynamic Large Object) can be used to upload large files like that.

The command is:

swift -A ¡Storage-URL¿ -U ¡Account_name¿ -K ¡Password¿ upload

¡container-name¿ '¡file_name¿' –segment-size ¡segment_size¿

In this process, swift command uses a strict convention for its segmented object support. it will upload all the segments into a second container named ¡container¿_segments. The main benefit for using a separate container is that the main

container listings will not be polluted with all the segment names. The reason for using the segment name format of ¡name¿/¡timestamp¿/¡size¿/¡segment¿ is so that an upload of a new file with the same name won't overwrite the contents of the first until the last moment when the manifest file is updated.

We used the following command to use Swift's built-in segmentation feature:

swift -A http://127.0.0.1:8080/auth/v1.0 -U test:tester -K testing upload MyContainer 'Casino 1995.720p.BrRip.x264.YIFY.mp4' –segment-size 100000

In here, segment size was 10000 and the container name was MyContainer. This segmented the file into 10000 segments and uploaded each segment individually in the same container [46].

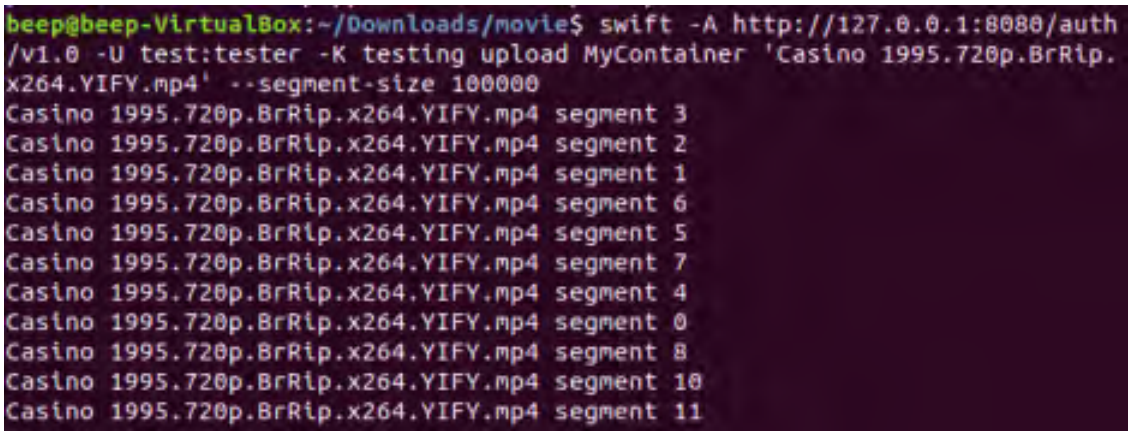

Figure 3.22: Large File Upload

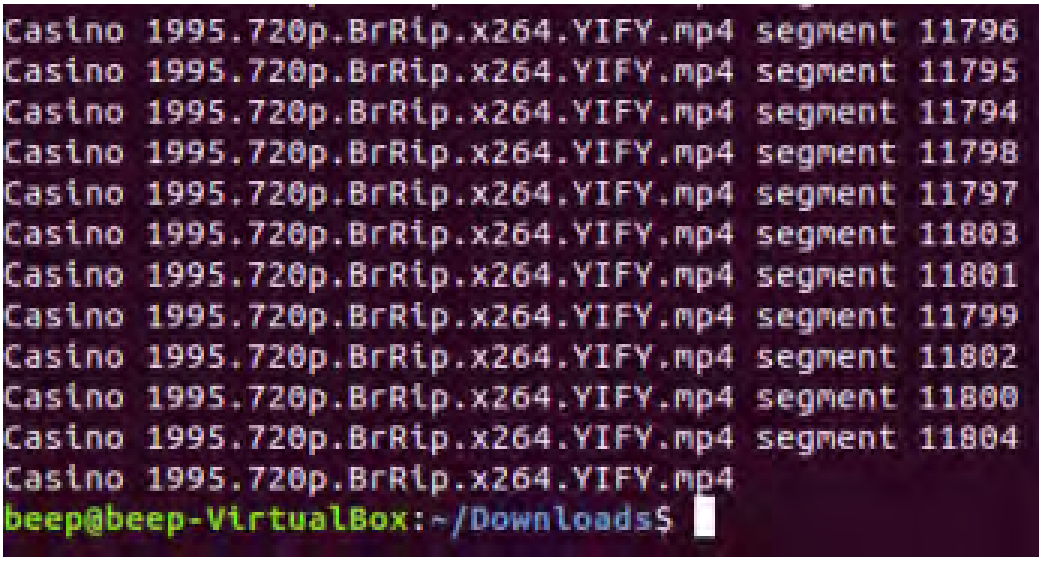

Figure 3.23: Large File Download

To download, this command was used:

swift -A http://127.0.0.1:8080/auth/v1.0 -U test:tester -K testing download MyContainer 'Casino 1995.720p.BrRip.x264.YIFY.mp4'

The same file is then downloaded as a single file.

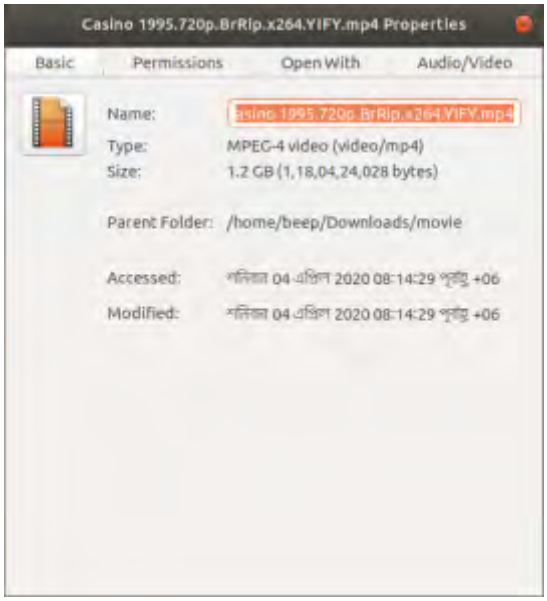

Figure 3.24: Downloaded as a Single File

3.9 Object Expiration

Object expiration is Swift's built-in feature, which as the name implies, expires object. It automatically objects after a given time that the user can set. How it works is after we upload an object to our server, we pass an expiration header X-Delete-At or X-Delete-After header with a POST request. After the time we send in the header passes, it automatically deletes that particular object. And once the object is deleted, swift will no longer serve the object and it will be deleted from the cluster shortly thereafter.

The X-Delete-At header takes a Unix Epoch timestamp, in integer form; for example: 1601877823 represents 10/05/2020 6:03 am (UTC) time.

The X-Delete-After header takes a positive integer number of seconds. The proxy server that receives the request will convert this header into an X-Delete-At header using the request timestamp plus the value given.

As expiring objects are added to the system, the object servers will record the expirations in a hidden .expiring_objects account for the swift-object-expirer to handle later.

To test this, We first started with an empty container.

```
sakib@sakib-VirtualBox:~/Desktop/thesis/test$ swift -A http://127.0.0.1:8080/auth/v
1.0/ -U test:tester -K testing stat mycontainer
               Account: AUTH_test
             Container: mycontainer
               Objects: 0
                 Bytes: 0
              Read ACL:
             Write ACL:
               Sync To:
              Sync Key:
         Accept-Ranges: bytes
                  Vary: Accept
      X-Storage-Policy: gold
         Last-Modified: Sun, 04 Oct 2020 22:57:32 GMT
           X-Timestamp: 1601848226.40218
            X-Trans-Id: tx5b370e7d6e044ef2992f8-005f7a5aff
          Content-Type: text/plain; charset=utf-8
X-Openstack-Request-Id: tx5b370e7d6e044ef2992f8-005f7a5aff
sakib@sakib-VirtualBox:~/Desktop/thesis/test$
```

Figure 3.25: Empty Container

Then, we uploaded a sample video and gave the expiration header with POST middleware. We set the X-Delete_After value to "120" which is 2 minutes. Now, notice the time. It was uploaded at 05:31:23. So, it should not be available after 05:33:23 but should be there between that 2 minute time window. In our 2 minute window, at 05:34:07, roughly after 1 minute had elapsed we check if the object is still there, and sure enough it is.

```
sakib@sakib-VirtualBox:~/Desktop/thesis/test$ timedatectl
                      Local time: সোম 2020-10-05 05:34:07 +06
                  Universal time: রবি 2020-10-04 23:34:07 UTC
                        RTC time: রবি 2020-09-27 15:12:15
                       Time zone: Asia/Dhaka (+06, +0600)
       System clock synchronized: no
systemd-timesyncd.service active: yes
                 RTC in local TZ: no
sakib@sakib-VirtualBox:~/Desktop/thesis/test$ swift -A http://127.0.0.1:8080/auth/v
1.0/ -U test:tester -K testing stat mycontainer sample_video.mp4
               Account: AUTH_test
             Container: mycontainer
                Object: sample_video.mp4
          Content Type: video/mp4
        Content Length: 17839845
         Last Modified: Sun, 04 Oct 2020 23:32:20 GMT
                  ETag: d7a3decdb6280eb0ef3a059ac35ea0ee
           Meta Mtime: 1601848534.222017
         Accept-Ranges: bytes
           X-Timestamp: 1601854339.83441
            X-Trans-Id: tx3d068403959a4b009043e-005f7a5bf6
X-Openstack-Request-Id: tx3d068403959a4b009043e-005f7a5bf6
```

Figure 3.26: Expirer Set

But at 05:36:29, after the window had passed, the file was deleted automatically.

```
sakib@sakib-VirtualBox:~/Desktop/thesis/test$ timedatectl
                      Local time: সোম 2020-10-05 05:36:29 +06
                  Universal time: রবি 2020-10-04 23:36:29 UTC
                        RTC time: রবি 2020-10-04 23:36:29
                       Time zone: Asia/Dhaka (+06, +0600)
       System clock synchronized: no
systemd-timesyncd.service active: no
                 RTC in local TZ: no
sakib@sakib-VirtualBox:~/Desktop/thesis/test$ swift -A http://127.0.0.1:8080/auth/v
1.0/ -U test:tester -K testing stat mycontainer sample_video.mp4
Object HEAD failed: http://127.0.0.1:8080/v1/AUTH_test/mycontainer/sample_video.mp4
 404 Not Found
Failed Transaction ID: tx848b07de032f44a5b40ee-005f7a5c85
sakib@sakib-VirtualBox:~/Desktop/thesis/test$
```

Figure 3.27: Automatic File Deletion

This is the feature we utilized to archive our object which will be discussed later in the paper.

3.10 Data Archive

As we want to make the best experience for the user, we already mentioned before that users will get data anytime, they want. So, archiving is the only solution here to store the footage of our surveillance system.

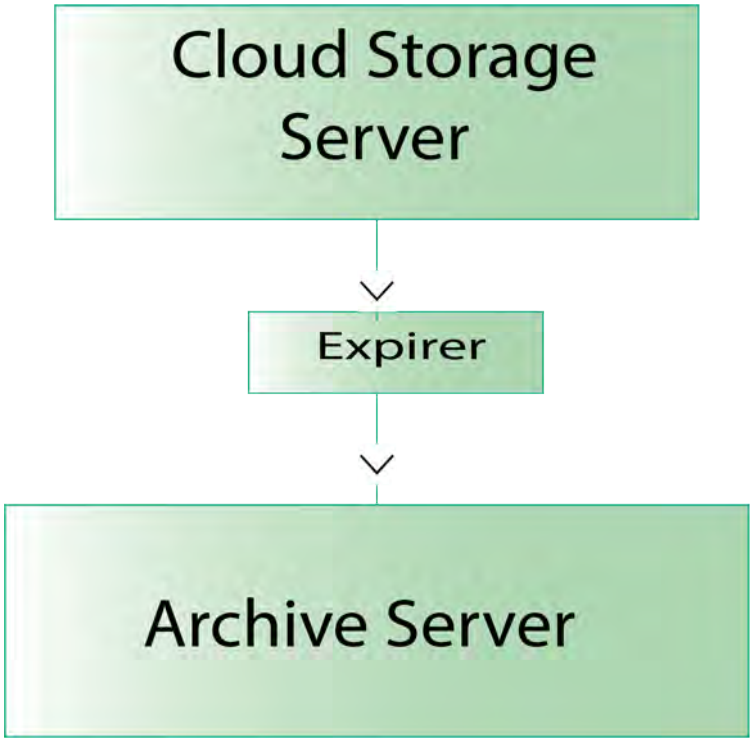

Figure 3.28: Cloud Server to Archive Server

The first model will work as follows: First, video files from CCTV's own NVR storage will be segmented using FFmpeg. Then it will be uploaded to the storage server at a fixed time at 12:00 each day and uploaded to the storage server with an object expiration header ¡object expiration link¿ set to "X-Delete-After:2592000". 2592000 is one month in seconds. This will delete the file in the storage server after a month. Before deleting, it will be compressed with a scheduled Cron job and then moved to the archive server with no expiration header.

In the second model, the video files will be uploaded parallelly to both the storage server and archive server. At first, the file will be segmented using FFmpeg. After that, in the storage server it will be uploaded with the expiration header set to ""X-Delete-After:2592000". Which means it will be deleted automatically after a month. In the archive server, we will compress the segmented FFmpeg files and upload them with no expiration headers set. So, files uploaded here will not be deleted .

The advantage of the first model is, it will not waste space like the second model. In the second model, identical data will sit in both storage server and archive server. Data kept in Archive Server will sit idly with no use because if someone needs access to that data, they could access it in the storage server instead. But the drawback is, sending data from a container to a different server is much harder. It also proved hard to keep track of and unnecessarily raised the complexity of our database. But the biggest drawback we found is, swift does not have any compression module built in. If it had an optional middleware for compression, perhaps this model would make the most sense. But because of this, we cannot directly send the data from our storage server to Archive server. First, we would have to download the data and compress it. After compressing and logging it into our database, we could reupload it to the archive server. This would cost more bandwidth and make the process lengthier than it needs to be.

The second model is intuitively simpler, does not need the download and reupload process. So, what we lose in storage, we make up in bandwidth saving, time and efficiency. For these reasons we chose to implement the second model in our project.

3.11 Database Module

To store the information of the data of a model, there must need a database so that every information can be stored in that data. This thing helps the admin panel to get every record of data. In our model, we have made a database which has 6 tables and they are for USER, footage (named as DATA), LOCAL STORAGE, CLOUD STORAGE, ARCHIVE STORAGE, and SECURITY (for future work). All of these tables are connected with each other that means it has one to one, many to many, many to one, etc. relationships. As we know, every database's tables have a primary key which are u.id, d.id, etc. respectively. Here the user table is used for the security of the user's personal footage information. Without a registered user, no one can see the information of that particular user's information. Now, let's get into the depth of the database. DATA (Surveillance footage) table has one id and information of

the particular size of the video and the name of that file. Moreover, we have done our segmentation and compression by our default local system. For this, we have the information of every segment so that we can provide a particular information to the registered user when their data is lost or other problems and how much time has passed for every segmentation and compression. Additionally, our cloud storage's information is the most important factor in this model. Finally, we have added the archive storage table for every information of the archive storage server's data processing. For our future work, we also have added a security table so that every security system's information can be restored or not get hacked because cyber-attack is the biggest disadvantage of this modern world.

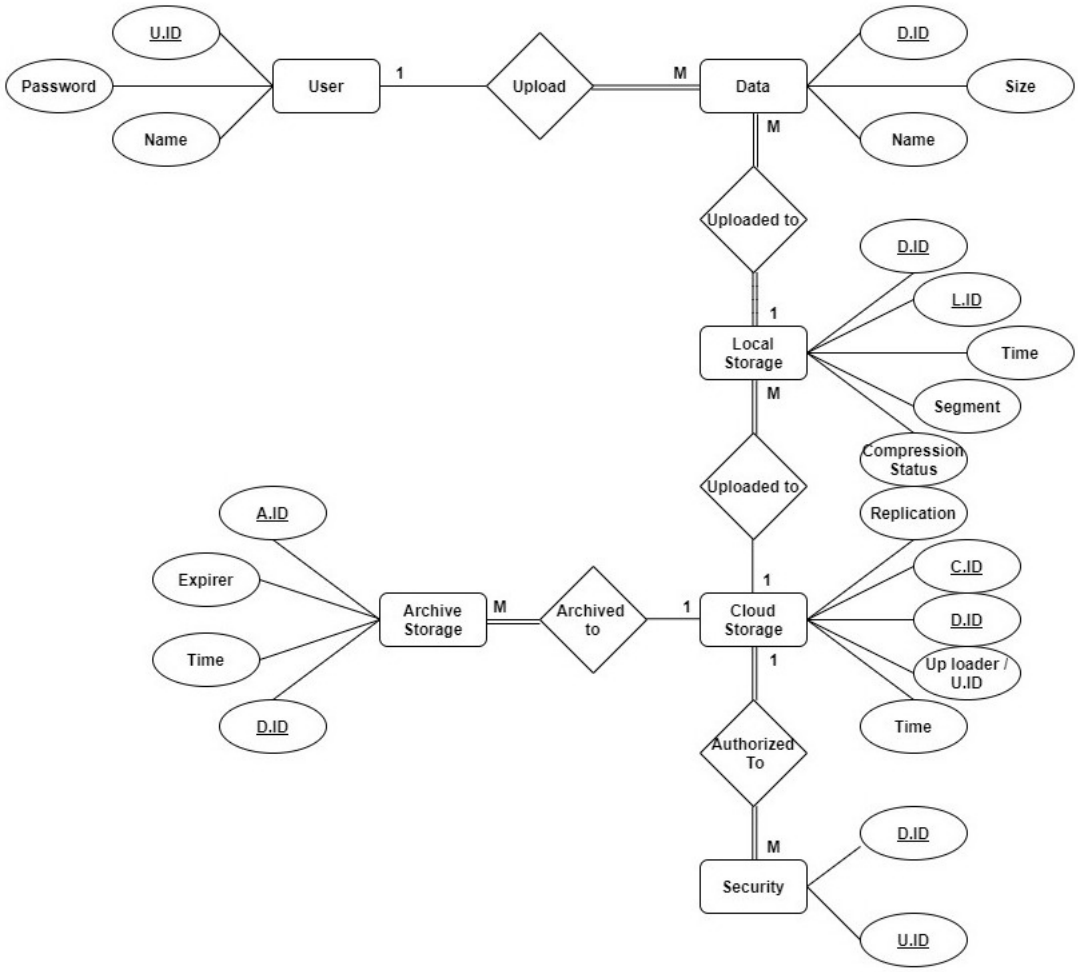

Figure 3.29: ER Diagram of Our Database Module

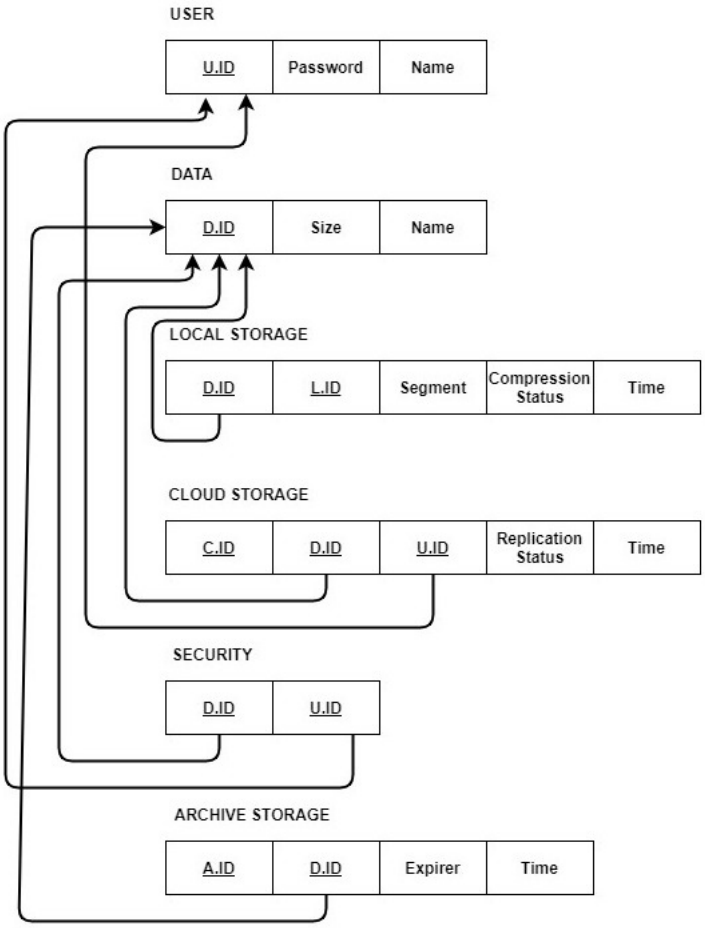

Figure 3.30: Relation Model of Our Database Module

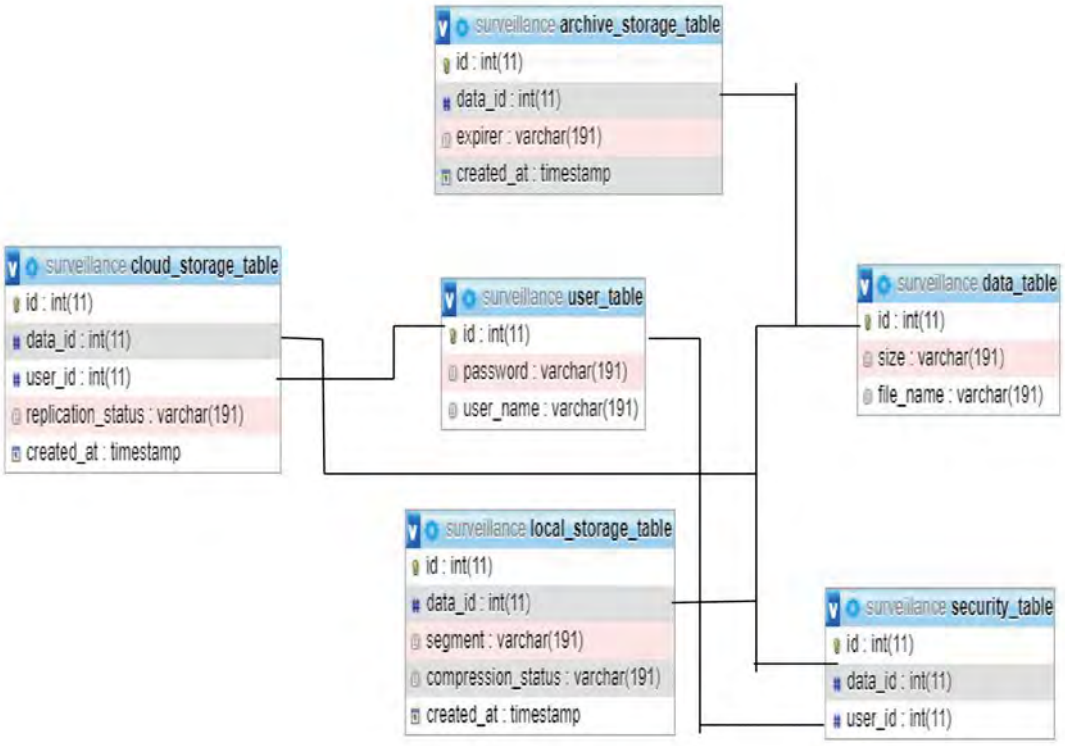

Figure 3.31: Mysql Database

Chapter 4

Experimental Evaluation

We built our cloud storage server and set up one media converter tool (FFmpeg) on our client machine to convert, segment different types of video data. We will record different categorical videos and compare between then to come to a final decision.

4.1 Experimental Settings

Our experimentation includes the comparison of different resolution video conversion with FFmpeg along with the timing later resolution and bit-rate with various presets of FFmpeg tools. Here, we do our video conversion and segmentation experiment in a client machine location in Dhaka, Bangladesh having a specification of Processor Ryzen 5 3600, CPU 3.6-4.2 GHz, Memory GB and GPU Colorful 1660 Super. We will use some chosen video files to compare video file conversion and changing of size, bit-rate and time to convert alongside. We will also compare different presets of FFmpeg tools.

We also set up Dropbox, google drive and iCloud to our local machine. We will also compare the upload time to different storage servers.

Table 4.1: Video Data Used in Experiment

Data	Size(MB)	Bit-Rate(kbps)	Resolution	Duration(Min)
Video1	41.8	735	854x480	6.23
Video2	28.22	1176	1280x720	2.51
Video3	46.9	879	1920x1080	6.11

We also did set up our own storage server in another local machine located in Dhaka, Bangladesh having a specification of Processor Intel core i7 7700 CPU 3.6-4.2 GHz Memory 16GB and GPU Zotac 1660 ti amp. We store those experimental videos to our storage server along with other popular storage servers.

4.2 Experimental Results

We use Linux based FFmpeg media converter tool to convert the three videos into different resolutions with 3 different preset settings of FFmpeg and collect those data.

Table 4.2: Video Conversion Result Using Default Preset

Data	Size(MB)	Bit-Rate(kbps)	Resolution	Time(Min)
Video1	128.4	2538	1280x720	7.33
Video2	30	1258	854x480	2.24
Video3	33.1	573	720x720	2.07

Table 4.3: Video Conversion Result Using Veryfast Preset

Data	Size(MB)	Bit-Rate(kbps)	Resolution	Time(Min)
Video1	119	2347	1280x720	2.35
Video2	26.5	1096	854x480	1.05
Video3	31.2	531	720x720	1.14

Table 4.4: Video Conversion Result Using Ultrafast Preset

Data	Size(MB)	Bit-Rate(kbps)	Resolution	Time(Min)
Video1	288.9	5888	1280x720	0.55
Video2	76.7	3438	854x480	0.32
Video3	88.8	1175	720x720	0.58

Table 4.5: Video Storing Time of Different Server

Data	iCloud	Google Drive	Dropbox	Storage Server
Video1	142s	12s	35s	10s
Video2	88s	7s	12s	3s
Video3	132s	8s	34s	7s

We also archive our data in archive storage and measures the time of storing to archive storage.

Table 4.6: Video Storing Time of Storage Server and Archive Server

Data	Storage Server	Archive Server
Video1	10s	15s
Video2	3s	9s
Video3	7s	12s

We compared the three types of converting data and came to the result that if we found that converting to higher resolution will increment the data size as well as the bit rate. The two presets mode veryfast and ultrafast both are more efficient than. Though ultrafast preset decreases the conversion time heavily but the size and bit-rate also increases.

Table 4.7: Rate of Change of Video size in Different Preset Mode

Data	Original to Default	Original to Veryfast	Original to Ultrafast
Video1	+207.18%	+184.61%	+519.15%
Video2	+4.26%	-5.67%	+171.99%
Video3	-29.42%	-33.46%	+89.34%

We can also find the improvement factor of our server from a graph comparing with other servers-

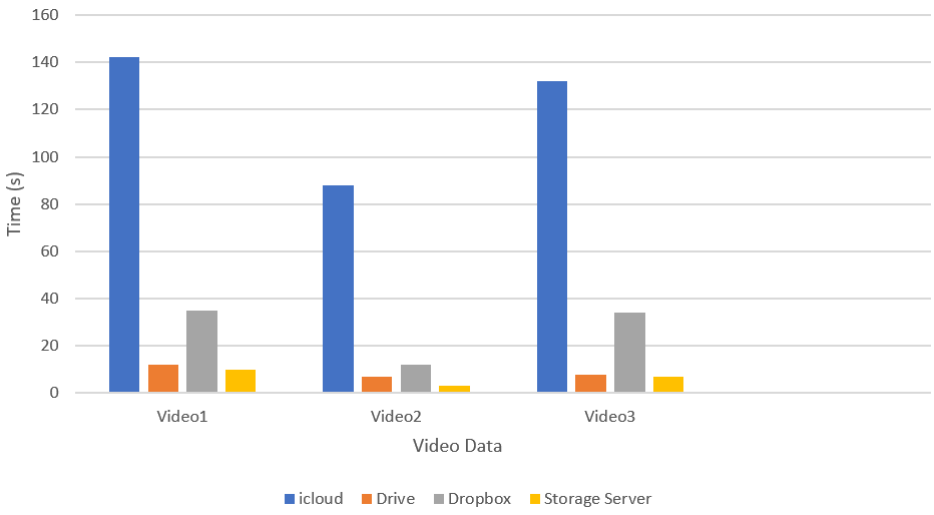

Figure 4.1: Video Storing Time of Different Server

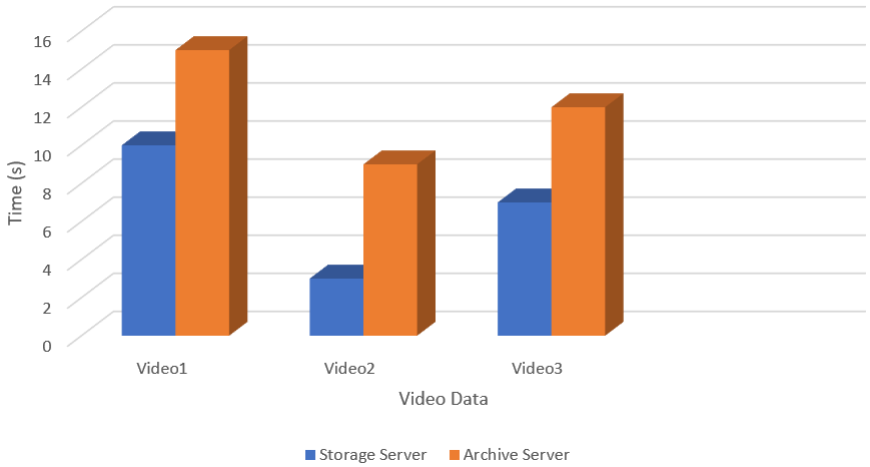

Figure 4.2: Video Storing Time of Storage Server and Archive Server

4.3 Experimental Findings

From the above experiment we can see default preset mode is slowest and ultrafast preset is fastest for converting a video. Moreover, from table 4.7 we find in ultrafast mode data size is changed the most. If data size and storage of servers does not impact on individuals one can choose ultrafast mode but if limited server storage it will be always wise to use veryfast preset mode of FFmpeg.

We can also find the improvement factor of data storing between different servers with our storage server. From table 4.5 we can already see the storing value of our storage server is relatively lower than other servers. We also want to mention that for different bandwidth it may vary. That's why we have checked 3 times and then took average value.

Table 4.8: Improvement of Storing Time

Data	iCloud	Google Drive	Dropbox
Video1	92.96%	16.7%	+71.43%
Video2	96.59%	57.14%	75%
Video3	94.7%	12.5%	79.41%
Average	94.75%	28.78%	75.28%

From table 4.8 we can see that we have improved the storing time of our server and it performs best in our experience among few worldwide used servers.

In [25] we find a similar category project where they build a SPMS server with OpenStack SAIO and also improved the storing time.

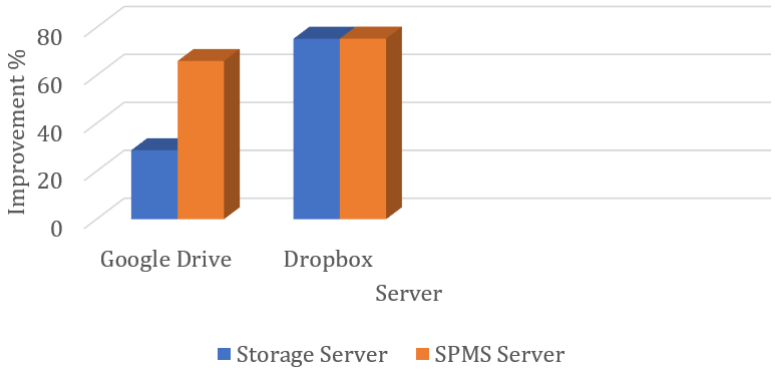

Figure 4.3: Comparison Between Proposed Storage Server and SPMS Server

From figure 4.3 we can see we did a great improvement on the Dropbox side compared to SPMS server but still lacks behind on Google Drive. Though it is not the best result compared to SPMS server but still it is better than most of the other servers.

As our main focus was not to improve the storing time but still, we made a great improvement. Our proposed model is basically to have a synchronous cloud system which will be accessible to the registered users to use it from anywhere and anytime they need. For instance, as the data will be stored in the cloud storage, they can use it anytime. Our model includes a long time archive system by which data will not be lost. Additionally, it is cost efficient because it does not require any extra hard drives for storing the footage of the video surveillance system. As we have told before, it will be much helpful for the organizations for any kind of investigation in terms of robbery and other consequences. To make it clear, we can say that it is flexible, reliable, easily accessible, secured file backup, user friendly and so on. When it is about more than one camera, this is the only flexible way to store the data. For example, if we have 10 cameras, we need to have many HDD storage for the NVR and it is too costly and a very bad idea of storing the footage.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Our proposed model is based on a cloud system which can be made by a cheap system design that we have already discussed. In this modern time, it is very important to have a cloud system in all prospects. For this, we have implemented this model with some useful tools and the OpenStack SAIO. All of these tools can be synchronized with the OpenStack SAIO. Firstly, we have taken some existing models that are already available in the market but those are so expensive and complicated. Secondly, the existing model in the market cannot be applicable for all types of people because of the expenses but our model is integrated for all kinds of users because we have used swift all in one and this system is not so expensive. With all kinds of plugins, we made it friendlier to the admin side as well as the user side. Though our model has some limitations, we can extend it broadly in future to make a better and user-friendly system. Our future plan can be more workable if we get deeper into OpenStack SAIO and all of its tools. To be more specific, we want to add some security layers in future for the user's security as well as the whole system of our cloud server. Moreover, in this research paper, we have proposed the archive storage server which can be working and synchronized as we do not have connected the Mysql database to our server yet. Without these lacking, our model is already more reliable than the existing models which are already researched in terms of surveillance and cloud management.

5.2 Future Work

We have added various modules in our proposed model. We wanted to connect all of them together so that they all would work automatically but could not manage to do so. All our modules are now working manually. We have to manually input data into the database, segmenting, converting video data, and storing data. Our nest plan is to make one single interface to manage all these works from a single platform.

One more thing that we wanted to do for this thesis project, but could not finish in time, is adding machine learning to this system. We wanted to add face recognition to our CCTV security and storage system, so that any person who is on the videos will get detected.

In order to make a human face detection system for our CCTV video files, we had to learn about object detection using machine learning. Machine learning is used in almost any kind of modern detection system. It can handle huge amounts of data and give real time results with very high accuracy. As our system will get its data from CCTV footage, manually viewing these big data is very much impractical. It will also be a waste of resource and time. Therefore, we are trying to implement a machine learning and deep learning algorithm to make our task easier and achievable. The first thing we did was choosing between the two most used tools for machine learning. TensorFlow and OpenCV were the main options for us. As OpenCV is mainly used for images and videos, we chose this. OpenCV comes with many great tools and functions which helped us a lot during our processes. In OpenCV, we followed three different types of algorithms. First one being Haar cascade, then You only look once algorithm and finally LBPH algorithm [47]. Haar cascade is a machine learning algorithm which can detect objects from photos or videos based on features. This cascade function is trained with a lot of positive and negative images and the accuracy increases with the amount of labeled data. This process is done with 4 steps. First step is to select the Haar features. This consists of some mathematical calculations of the pixels on a given window, and distinguishing them based on the difference of the number of pixels. Next step is that an integral image is created based on this Haar function. And it's often very hard to match the image with over 160000+ features, so a concept called Adaboost comes to play here, which dynamically selects the best features which are similar to the case image and trains the classifier accordingly. The outcome of the entire process is stored in a cascade file, which we used in our task in OpenCV.

The next algorithm we followed is called YOLO (You Only Look Once) real-time object detection algorithm. This is a very fast object detection algorithm which identifies objects from a live stream of video or pictures faster than Cascade function, and it can be processed with CPU or GPU. It has proved to be one of the most effective and also the most innovative object detection algorithms out there. The concept of the Yolo algorithm is not very complicated.

YOLO is an artificially intelligent convolutional neural network (CNN). The way it works is that a single neural network is attached to the full image. There the image is divided into smaller parts called boundary boxes. The algorithm tries to predict the region of the boxes and the small boxes are given weights based on the predictions. Because of its simplicity, You Only Look Once algorithm is extremely reliable than its competitors and can give your real time results which is crucial in this modern era. The reason we choose to work with this is because it gave us faster results than the other object detection techniques we tried to use. Moreover, Yolo has access to the entire image while it is training. So, it's very easy for the algorithm to encode contextual information where necessary. And finally, as we put more data on the system, it can get a generalizable representation of objects. Which

means that it will perform better in this context and will give us higher chances of accuracy. In our work, first we put some images on python with Haar cascade, where we used eye and face cascade to detect face and eyes from an image. It resulted with visual boxes on the faces it found and also on the eyes. It successfully works on the images, so we also applied it to videos. It gave very highly accurate results. It tracked through the videos and identified faces and eyes which were visible on the screen. Then we moved to You only look once algorithm for faster and wider results. The problem with Haar cascade is it takes a lot of images and datasets to train it, while training a You only look once algorithm is easy. And it comes with a lot of pre-existing trained data. By using this algorithm, we successfully detected multiple objects from a still image and videos and got very accurate results. It made a box around every object it detected and labeled them separately with different colors so it can be identified with ease.

We will work on this to make it work in sync with our CCTV storage. So, when a client needs to check the CCTV video data, they can also see how many faces were there in the video. This will make any type of security check so much easier. Even though we could not make it work in this paper because of a shortage of time, it is something we will definitely work on to implement with our system.

Bibliography

[1] *How much data do we create every day? the mind-blowing stats everyone should read*, https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/#718499e660ba, (Accessed on 12/21/2019).

[2] *Getting those giant video clips off your iphone*, https://www.nytimes.com/2018/01/25/technology/personaltech/giant-video-clips-iphone.html, (Accessed on 12/21/2019).

[3] *What is the maximum file size to upload?*, https://support.google.com/drive/thread/12027363?hl=en/, (Accessed on 12/21/2019).

[4] *Upload and index your videos*, https://support.google.com/drive/thread/12027363?hl=en/, (Accessed on 12/21/2019).

[5] *Dropbox file size limits and how to upload large files*, https://help.dropbox.com/installs-integrations/sync-uploads/upload-limitations/, (Accessed on 12/21/2019).

[6] *Amazon cloud drive photos app expands to support video upload playback on android*, https://techcrunch.com/2013/07/25/amazon-cloud-drive-photos-app-expands-to-support-video-upload-playback-on-android//, (Accessed on 12/21/2019).

[7] *File size limits for cloud storage explained*, https://www.cloudstoragereviews.co/cloud-storage-basics/file-size-limits-for-cloud-storage-explained.html/, (Accessed on 12/21/2019).

[8] *File size limits for cloud storage explained*, https://www.accenture.com/us-en/insights/cloud/cloud-outcomes-perspective/, (Accessed on 12/21/2019).

[9] R. Regaieg and M. Koubaa, "Review on cloud computing security challenges",

[10] M. D. Kenga, V. O. Omwenga O, and P. J. Ogao, "Energy consumption in cloud computing environments", 2017.

[11] S. Rahul *et al.*, "Cloud computing: Advantages and security challenges", *International Journal of Information and Computation Technology, ISSN 0974-2239*, vol. 3, no. 8, pp. 771–778, 2013.

[12] M. Darwich, Y. Ismail, T. Darwich, and M. Bayoumi, "Cost-efficient storage for on-demand video streaming on cloud", *arXiv preprint arXiv:2007.03410*, 2020.

[13] E. Baik, A. Pande, Z. Zheng, and P. Mohapatra, "Vsync: Cloud based video streaming service for mobile devices", in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, IEEE, 2016, pp. 1–9.

[14] Z. Liu, Q. Wang, J. Huang, Y. Wu, Y. Wang, X. Jia, and H. Chen, "Cloud-based video-on-demand services for smart tv", in *2017 Seventh International Conference on Information Science and Technology (ICIST)*, IEEE, 2017, pp. 81–84.

[15] K. I. Choi, J. H. Lee, and B. C. Lee, "Cloud based video storage system with privacy protection", in *2015 17th International Conference on Advanced Communication Technology (ICACT)*, IEEE, 2015, pp. 460–463.

[16] M. U. Yaseen, M. S. Zafar, A. Anjum, and R. Hill, "High performance video processing in cloud data centres", in *2016 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, IEEE, 2016, pp. 152–161.

[17] J. Kim, D. Lee, and N. Park, "Cctv-rfid enabled multifactor authentication model for secure differential level video access control",

[18] N. Park and H.-C. Bang, "Mobile middleware platform for secure vessel traffic system in iot service environment", *Security and Communication Networks*, vol. 9, no. 6, pp. 500–512, 2016.

[19] N. Park and N. Kang, "Mutual authentication scheme in secure internet of things technology for comfortable lifestyle", *Sensors*, vol. 16, no. 1, p. 20, 2016.

[20] W.-B. Kim and I.-Y. Lee, "Secure and efficient storage of video data in a cctv environment.", *TIIS*, vol. 13, no. 6, pp. 3238–3257, 2019.

[21] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-locked encryption and secure deduplication", in *Annual international conference on the theory and applications of cryptographic techniques*, Springer, 2013, pp. 296–312.

[22] J. Hur, D. Koo, Y. Shin, and K. Kang, "Secure data deduplication with dynamic ownership management in cloud storage", *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 11, pp. 3113–3125, 2016.

[23] A. Alam, M. N. Khan, J. Khan, and Y.-K. Lee, "Intellibvr-intelligent large-scale video retrieval for objects and events utilizing distributed deep-learning and semantic approaches", in *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*, IEEE, 2020, pp. 28–35.

[24] H. Wang, S. Mehrotra, M. Martini, D. Wu, and Q. Zhang, "Guest editorial: Cloud-based video processing and content sharing", *IEEE Transactions on Multimedia*, vol. 18, no. 5, pp. 805–806, 2016.

[25] J. Noor, H. I. Akbar, R. A. Sujon, and A. A. Al Islam, "Secure processing-aware media storage (spms)", in *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*, IEEE, 2017, pp. 1–8.

[26] *What are the components of a megapixel ip cctv system?*, https://trinitycctv.co.nz/cctv-and-security-cameras/learn-about-cctv/ip-cctv-faq//, (Accessed on 12/21/2019).

[27] *Understanding camera specifications*, https://clearview-communications.com/wp-content/uploads/2017/10/Understanding-the-CCTV-specifications.pdf/, (Accessed on 12/21/2019).

[28] *Cctv vs. ip cameras: Which is best suited for your business?*, https://www.taylored.com/blog/cctv-vs-ip-cameras-which-is-best-suited-for-your-business, (Accessed on 09/03/2020).

[29] *Cctv vs. ip cameras: Which is best suited for your business?*, https://www.taylored.com/blog/cctv-vs-ip-cameras-which-is-best-suited-for-your-business//, (Accessed on 12/21/2019).

[30] *What wdr does on cctv cameras?*, https://securitycamcenter.com/wdr-cctv-cameras/, (Accessed on 09/30/2020).

[31] *What wdr does on cctv cameras?*, https://securitycamcenter.com/wdr-cctv-cameras//.

[32] *Nvr storage calculator for network ip security cameras*, https://www.supercircuits.com/security-nvr-storage-calculator/.

[33] *Surveillance storage calculator*, https://www.seagate.com/as/en/video-storage-calculator//.

[34] *Surveillance storage calculator*, https://www.seagate.com/as/en/video-storage-calculator/, (Accessed on 09/07/2020).

[35] K. B. Shikhar Arora, "Storage optimization of video surveillance from cctv camera", *2nd International Conference on Next Generation Computing Technologies*, pp. 710–711, 2016.

[36] *Transcoding : How ffmpeg works*, http://multimedia-howto.blogspot.com/2015/05/transcoding-how-ffmpeg-works.html/.

[37] D. Neal and S. S. Rahman, "Video surveillance in the cloud?", *International Journal on Cryptography and Information Security (IJCIS)*, vol. 2, no. 3, p. 16, 2012.

[38] *How openstack swift works*, https://searchstorage.techtarget.com/definition/OpenStack-Swift/.

[39] *Openstack swift architecture*, https://www.swiftstack.com/docs/introduction/openstack_swift.html/.

[40] *Openstack swift architecture*, https://www.researchgate.net/figure/OpenStack-Swift-architecture_fig9_330710550, (Accessed on 10/04/2020).

[41] *Openstack swift architecture - data storage is changing*, https://www.swiftstack.com/docs/introduction/openstack_swift.html, (Accessed on 10/04/2020).

[42] *Swift architectural overview*, https://docs.openstack.org/swift/latest/overview_architecture.html, (Accessed on 10/04/2020).

[43] *Openstack documentation - ring-builder*, https://docs.openstack.org/swift/pike/admin/objectstorage-ringbuilder.html, (Accessed on 10/04/2020).

[44] J. Arnold, *OpenStack Swift: Using, administering, and developing for swift object storage.* " O'Reilly Media, Inc.", 2014.

[45] *Object storage api overview*, https://docs.openstack.org/swift/latest/api/object_api_v1_overview.html/.

[46] *Openstack - large object support*, https://docs.openstack.org/swift/latest/overview_large_objects.html, (Accessed on 09/03/2020).

[47] *Face detection – opencv, dlib and deep learning (c++ / python)*, https://www.learnopencv.com/face-detection-opencv-dlib-and-deep-learning-c-python/.