

Designing a New Scalable Load Test System for Distributed Environment

by

Md. Azizul Haque Emu

18101075

Miraj Mahmood

18101090

Mohtasin Mehmod Asif

18101096

Abdur Rob Tanvir

18101211

Raunaq Sayiara Joyeeta

18101118

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 and Engineering

Department of Computer Science and Engineering
Brac University
January 2022

© 2022. Brac University
All rights reserved.

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:

Azizul

Md. Azizul Haque Emu
18101075

miraj

Miraj Mahmood
18101090

Asif

Mohtasin Mehmod Asif
18101096

Tanvir

Abdur Rob Tanvir
18101211

Joyeeta

Raunaq Sayiara Joyeeta
18101118

Approval

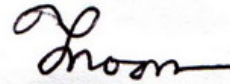
The thesis titled “Designing a New Scalable Load Test System for Distributed Environment ” submitted by

1. Md. Azizul Haque Emu (18101075)
2. Miraj Mahmood (18101090)
3. Mohtasin Mehmod Asif (18101096)
4. Abdur Rob Tanvir (18101211)
5. Raunaq Sayiara Joyeeta (18101118)

Of Fall, 2021 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on January 18, 2022.

Examining Committee:

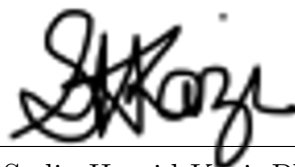
Supervisor:
(Member)



Jannatun Noor
Lecturer

Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)



Sadia Hamid Kazi, PhD

Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Ethics Statement

Our research is of the highest quality and integrity. The confidentiality and integrity of our research paper contributors are extremely important to us. We conducted our study in an unbiased manner in order to produce an analysis that is both independent and impartial. Perhaps, in the hereafter, this method of analysis will provide some further benefit to humankind in order for them to progress.

Abstract

Cloud computing is considered as a computer paradigm in order to retrieve data and store from users. It is also essential for efficient operations in a distributed system (a system that communicates and coordinates various components of different machines located in multiple places and appears as a single system to the end-user). With the rapid increment of users, it became challenging to maintain the capacity of incoming load and suitable resource allocation. Among existing all other cloud servers OpenStack Swift is one of the most widely used open-source cloud computing and storage management solutions. An object storage service provided by OpenStack Swift that is commonly utilized for cloud-based storage solutions. We have used OpenStack swift as our cloud server and performed the load test using Apache JMeter as our load testing tool and compared the average response time under different test cases such as large file and small file size, having same or different URL and servers with load and without load. The average response time gives a brief idea about the load handling ability of the server. An in-depth comparison between the average response times between the test cases helped us to figure out the best possible case.

Keywords: Load Testing; JMeter; OpenStack Swift; Cloud Computing; Geographical Locations; Latency; Average Response Time

Acknowledgement

With the blessings of almighty Allah (swt) we finally finished our thesis on the topic named Designing a New Scalable Load Test System for Distributed Environment. It was a beneficial experience for all of us.

We would like to express our gratitude towards our honorable supervisor Ms. Jan-natun Noor for giving us this opportunity. We have learnt and experienced many important and interesting things throughout this task which will definitely help us in near future.

Finally, we would like to thank all the people who have given us their precious time and helped us by providing all kinds of information that we needed for the report without any objection.

Table of Contents

Declaration	i
Approval	ii
Ethics Statement	iii
Abstract	iv
Acknowledgment	v
Table of Contents	vi
List of Figures	viii
List of Tables	ix
1 Introduction	1
1.1 Motivation	2
1.2 Aim and Objectives	2
1.3 History of Cloud Computing and Load Testing	3
1.4 Contribution	4
1.5 Thesis Orientation	4
2 Literature Review	5
3 Background Studies	14
3.1 Cloud Computing	14
3.2 IaaS	15
3.3 PaaS	15
3.4 SaaS	16
3.5 FaaS	17
3.6 Cloud Storage	18
3.7 OpenStack Swift	18
3.7.1 Characteristics	18
3.7.2 Components	19
3.7.3 Ring-builder	19
3.7.4 Object Storage Monitoring	19
3.8 Command line	20
3.9 Servers	20
3.10 Virtual Machine	20

3.11	Virtualization	21
3.12	Test Case Metrics	22
3.13	Test Case	22
3.14	Load Testing	23
3.15	Scalability	24
3.16	Testing Tools	24
3.17	JMeter	25
3.18	Latency	25
3.19	Response Time	26
4	Proposed Methodology	28
4.1	Work Flow	28
4.2	Test Case Metrics	29
4.2.1	Concurrent Request for Load Testing	30
4.2.2	Cloud Server	30
4.2.3	Client PC	31
4.3	Test Case Design	31
4.4	Geographical Location and System Configuration	32
4.5	Testing through JMeter	32
5	Experimental Evaluation	34
5.1	Experimental Setup	34
5.1.1	Swift Setup	34
5.1.2	Testing Tool Setup	36
5.2	Experimental Result	37
5.2.1	TCO	37
5.2.2	TC1	38
5.2.3	TC2	39
5.2.4	TC3	40
5.2.5	TC4	41
5.2.6	TC5	42
5.2.7	TC6	43
5.2.8	TC7	44
5.3	Experimental Findings	45
6	Future Work	47
7	Conclusion	48
	Bibliography	51

List of Figures

4.1	System Work Flow	29
4.2	Proposed Test Case Metrics	29
4.3	Concurrent Request Architecture	30
5.1	Ring Builder Creation	35
5.2	Ring builder configuration for the mounted drives	35
5.3	Swift Starting	36
5.4	Account Authentication	36
5.5	Container Creation Confirmation	36
5.6	Response Time Graph of TC0	38
5.7	Response Time Graph of TC1	39
5.8	Response Time Graph of TC2	40
5.9	Response Time Graph of TC3	41
5.10	Response Time Graph of TC4	42
5.11	Response Time Graph of TC5	43
5.12	Response Time Graph of TC6	44
5.13	Response Time Graph of TC7	45

List of Tables

4.1	Test Case Scenarios Under Different Parameters	31
4.2	Geographical Location and System Configuration	32
5.1	Result Table of TC0	37
5.2	Result Table of TC1	38
5.3	Result Table of TC2	39
5.4	Result Table of TC3	40
5.5	Result Table of TC4	41
5.6	Result Table of TC5	42
5.7	Result Table of TC6	43
5.8	Result Table of TC7	44
5.9	Average Response Time for Test Cases	46

Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

AWS Amazon Web Services

CL Command Line

CPU Central Processing Unit

FaaS Function as a Service

HPC High Performance Cloud

HTTP Hypertext Transfer Protocol

IaaS Infrastructure as a Service

IT Information Technology

OS Operating System

PaaS Platform as a Service

QoS Quality of Service

RTT Round Trip Time

SaaS Software as a Service

TC Test Case

URL Uniform Resource Locator

VM Virtual Machine

Chapter 1

Introduction

Cloud computing is one of the most widely used technologies nowadays. It provides a versatile and effective means to store and retrieve data files, and both industry and academia are enthusiastic about it. In modern network technology, cloud computing is showing significant growth because of its communication technology advancement and extensive problem-solving. It distributes data to the users at a low cost. Users only need to pay as much as they use the resources. There are a good number of existing issues in cloud computing. Load testing is an essential aspect of cloud computing. It is crucial to ensure the efficiency of a system. To establish an application, we need to test the load conditions, how much time it takes to produce work. These are very important to test before releasing an application. Load testing provides the maximum outcome from a system by proper utilization of cost-effective means. It is done by performing workload simulation from a set of multiple computer servers of the client host in order to test the service responses. We can determine an application's maximal operating capacity as well as any obstacles. Load testing processes need to be done in a large population of real workloads for the targeted application. Multiple autonomous computers communicate through a computer network and create a distributed environment. The goal of interacting with computers is the same. Cloud computing is considered the latest application of the classical distributed environment sometimes. It lowers the cost of computing, application hosting, content storage, and distribution. Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) are examples of cloud computing services. Processing, networks, storage, and other primary computer resources can all be provisioned using IaaS. It enables users to install and run any type of software, including operating systems and apps. It delegated certain IT functions to third firms. Secondly, PaaS can deploy applications written using programming languages and tools offered by the provider onto cloud infrastructure created by the user or bought applications. Finally, SaaS can operate providers cloud-based apps. Through a thin client interface, such as a web browser, these programs can be accessed from various client devices.

Load testing on the cloud is a type of Software as a Service (SaaS). In order to shift load testing to the cloud, it is necessary to implement some resource allocation and scheduling approaches. The resource issuance that issues client-side virtual machines for workload simulation is required to efficiently complete load testing activities and the operating costs of cloud testing providers. Multi-tenancy is a major aspect of SaaS, which allows a single application to satisfy users needs from

various organizations and corporations at the same time. However, it will not be an easy task because there will be numerous obstacles, such as isolation, conductivity, etc. Furthermore, resource management is a critical component of multi-tenant applications, particularly for resource-intensive applications like cloud-based load testing. In a multi-tenant setting, efficient resource management is required to bring out the most significant tenants possible.

1.1 Motivation

The traditional cloud computing server is very costly when we want to expand the storage. So, we wanted to come with a very cost efficient, easy to use and secure web server for the consumers. Firstly, Swift offers cloud storage software so that the user can use this to store and retrieve a huge amount of data with a very simple API. Also, OpenStack Swift is freely available for anyone to use. All though it has some technical issues for the users using it for the first time. But with time they will be familiar with the system by following the guidelines. OpenStack has a good security system and it can be reinforced by creating a replicator. So, the users need not to worry about losing any data or security breach. It doesn't take any storage access permission like the traditional cloud servers. As a result, the local stored data is always safe. Swift server can easily be expanded on demand so we can enlarge the server storage when it is necessary. The costing for the system can be reduced compared to the existing ones in the world. Keeping all these factors in consideration we have decided to work on this.

1.2 Aim and Objectives

This research aims to determine the parameters such as throughput and latency of an application when the number of users using the application concurrently and different load profiles as well as many other performance metrics. There are some fundamental matrices such as storage, bandwidth, processing, the number of users accessing it at a given time range. These parameters are vital for successful cloud implementation. The objective of this research are:

1. To deeply understand load testing and how load testing works.
2. To deeply understand how the system components perform under various loads.
3. To calculate the server response time.
4. To get the maximum performance from our cloud server.
5. To compare the results obtained from JMeter.
6. To determine how the performance is under various test cases.

1.3 History of Cloud Computing and Load Testing

The fast spreading of the Internet all over the world and current information technology in all aspects of the world and individual lives has resulted in a significant enhancement in the number web users over the last decade. People are becoming more and more attracted towards technology for its high availability, efficiency and ease of use. The term “cloud” means a set of Internet resources [29]. Cloud computing is one of technology’s trendiest buzzwords. It appears 48 million times on the internet.

In the early days, computers were big and expensive. So, sharing resources were time consuming and costly. But nowadays computers are cheaper as media capacities have increased and, therefore, costing of stored information of 1 MB has decreased. As a result, the cost of storage media services has dropped significantly in addition to a huge growth in the amount of data stored. The advancement of programming methods aided the efficient use of multiprocessor computing resources and the flexible deployment of cloud computing capacity. A rise in Internet data throughput resulted in faster data interchange, lower Internet traffic costs, and more cloud technology availability [8]. All these reasons have contributed in the enhancement of growth in cloud computing in several area of IT. The need for cloud-based data are continually being uncovered. as technology is evolving every day. We have witnessed the evolution of floppy discs to zip drives, CDs to USB storage devices, and many more throughout our lives.

If we want to know the birth of cloud computing we have to venture back to almost 70 years. The idea of cloud computing was first proposed by John McCarthy an American computer scientist in the 1960s. He explained that forthcoming computations will be carried out by public utilities. He is known as the inventor of the cloud computing idea [8]. But the people of that time did not appreciate the idea. They thought what they had at that time was enough efficient. But as the time fleet the idea was recognized and Salesforce.com implemented it in 1999. The company pioneered the delivery of an enterprise application via the internet. Then In 2002, Amazon started AWS where Amazon will offer online storage and computing over the internet. In 2006, they launched Elastic Compute Cloud Commercial Service and this is accessible for everyone. The ideology of cloud computing became popular in 2007 as a result of quick advancements in communication channels and a rise in the geometric progression of commercial and private users desires to grow their information systems [8]. Later in 2009, Google Play began offering a Cloud Computing Enterprise Application, and as other businesses saw the importance of cloud computing, they began offering their own cloud services. As a result, Microsoft launched Microsoft Azure in 2009, and other firms such as Alibaba, IBM, Oracle, and HP followed suit with their own Cloud Services [18]. Cloud computing has gained widespread acceptance and popularity in today’s globe.

1.4 Contribution

Based on our work, we make the following set of specific contributions in this paper:

1. We propose 8 different test cases based on diversified real scenario covering HTTP protocol types, file size (large or small) URL types (same or different URLs) and load types (with or without loads) for performing load test on cloud systems using several tools JMeter.
2. We send rigorous concurrent request on the server along with various test cases. We found the response time based on those cases designed.
3. We also find the best result and the worst result based on average response time.

1.5 Thesis Orientation

The whole part of our thesis is represented into six parts and they are mentioned below:

Chapter 1 contains the introduction to cloud computing, the history of cloud computing, different types load testing, and OpenStack Swift. Along with that our objectives and motivation to conduct this thesis is clearly explained in this part.

In Chapter 2, the detailed discussion about the existing work related to our field exists. Furthermore, the literature review related to cloud based load testing, OpenStack swift and how the server can be tested using JMeter under different parameters are also briefly explained.

In Chapter 3, we have discussed the particular topics that are related to our topic cloud computing, then open stack swift and testing tools and all the related topics that are directly connected to our thesis.

In Chapter 4, the detailed discussion about the system of our work flow and proposed methodology is given. In this part, we have explained our work, how the test cases were implemented, how we have used JMeter in order to perform the testing, and how different geographical locations were giving various outcomes as well as the system configurations. In the last part, the distinctiveness of our experiment compared to others is well explained.

In Chapter 5, we have described the experimental setup. Then we have explained the outcomes of each test case and made an analysis of our experimental findings.

In Chapter 6, we have figured out our limitations and mentioned the future works.

In Chapter 7, we have made a summary of our work and came up with a conclusion.

Chapter 2

Literature Review

Cloud computing utilizes resources on a need basis service. This allows the users to connect via the internet while giving them a hosted environment with required services. Along with the emergence of cloud technology, everyone is moving towards embracing a cloud-based framework due to the enormous benefit that we derive from cloud computing. Cloud computing has opened new vistas of opportunity for testing. The Covid-19 epidemic has dramatically increased the number of internet users, demonstrating the benefits of working from home [31]. Load testing is required to ensure a positive user experience. Besides, any malfunction in the system will create great havoc. The system's downtime will cost a significant amount of money in lost revenue. In addition, if the system's reaction time is slow, users would lose interest. Performance requirements, on the other hand, can pose a risk.

Cloud computing has opened up new possibilities for performance assessment. It is a recent computing method that includes resource pooling management, multi-tenant sharing, browser-based access, on-demand payment and other features [26]. The application testing procedure that has already shifted or planned will be shifted into the cloud is defined as cloud testing. This cloud testing is done to assure that the applications' performance, reliability, and security go with or surpass the expectations considering the change of delivery methods. Cloud testing can also be used to perform standard tests for on-premise applications, such as load, security, performance, compatibility, and stress testing, using cloud computing resources and cloud-based physical infrastructure [15]. While proceeding into the real running system, web-based load testing skip the real attributes in the case of web services. Moreover, the installation and configuration process is much more challenging as it is not easy to use for the tester, creating difficulty for the testers.

In this day of competitive marketing, the system's accuracy is critical; otherwise, they would fall behind in the market. As a result, load testing is required to address these issues. Load testing lowers the risk of downtime and identifies the system's performance bottlenecks. Load testing also improves user happiness and makes the system more scalable. To meet the demands of a rapidly developing market, maximize the marketing campaign funds. Load testing lowers the cost of failure as well. Load testing also aids in the system's continual improvement by assisting with

performance tweaking. Load-based application testing saves money in terms of both capital and operating costs and provides support for distributed development [17].

People are required to connect while at home or even on the go whilst using the Agile Method in teams. To see how scalable the application is when multiple users are connected at the same time when it's operating in the cloud. Taking the applications through its trials with just as many worldwide web users. A script file for deploying a Virtual Office application will be read by the Test Harness. In three types of scenarios, we have load tested with various numbers of users. For the initial phase with 10 users, we have a CPU consumption at server is 32 percent. For the middle phase with 25 users, the consumption of CPU is 67 percent. CPU consumption at the server is 98 percent for a handful of 45 users. When the consumption reached 80 percent the Billable Quota was triggered on the server. Which will give us a brief idea about the costing analysis in various situations. Finally, when the consumption reached a full 100 percent, the instance hours were automatically increased. The application continued to function with online users working effectively without a break/failure, irrespective of the number of instances created [9]. When it has been functionally evaluated and achieves the appropriate level of performance, it is actually helpful for web-based applications.

This study [18] presents a globally distributed web server design and examines several load balancing methods to estimate and assess their performance. A design is investigated and utilized to evaluate multiple ways for request transferring in a web cluster that simulates regular scenarios of internet and includes all phases of the HTTP request service. Moreover, the HTTP client requests are created using the JMeter testing program. The GDLB web cluster system's performance that was proposed was evaluated using variables such as the number of requests serviced, CPU usage, and average response time are the metrics investigated in the experiment and the findings were compared to those of other techniques and JMeter was used to evaluate client requests.

The goal of this design is to make it simple to measure a variety of web server cluster performance data, for example, average request time of response, CPU usage, rate of error, as well as throughput. This design was created with scalability, the capacity to manage a wide audience and servers, the ability to simulate real-world internet conditions, and the ability to easily build load balancing solutions in mind. Furthermore, the design should be generic, allowing for the implementation of various load balancing strategies. Lastly, it should be possible for dynamic schemes to make decisions based on information about the current state of the system.

In this design, all components were implemented on Linux-based PCs. JMeter is in charge of generating requests and collecting statistics. Based on the client's global address, up-gradation of DNS server to return one of the potential addresses of IP. At last, the dispatcher's software was used to allocate requests in a transparent and random manner throughout a cluster. Recent schemes may be applied as modules and easily incorporated into existing architecture thanks to the design of all of the components. In the paper, it can be observed through a comparison of DNS, dispatcher-based web server systems with the help of GDLB web cluster system that the proposed GDLB solution for dynamic load balancing has a superior time

of response, CPU usage, and throughput. This also offers a reliable and effective method for dynamic load balancing in a web cluster system.

The purpose of the paper [5] is to assess how long it takes a web server to respond to a client's request and compare the findings to those of the existing stress tool of the web, Pylot. To simulate demand on a web application, stress tools for web server are utilized. Simulating server load, concurrency difficulties, and understanding the server's responsiveness and how it acts under stress are all possible. Assessment of whether or not the site is capable of handling client loads will be provided by it.

From the experiments they understood that, there can be several reasons for getting different response time from servers. Websites with a larger size take longer to load than those with a smaller size is one among them. Aside from that, response time gets significantly influenced by client and server bandwidth availability.

In this research [28] they used JMeter and WAPT to determine the best quality mobile server web application, in order to make recommendations when a mobile web site is being utilized as this is yet uncertain which web server of mobile application offers the finest quality among the many paid or free accessible web server programs. As smartphones using the Android operating system still hold 94 percent of the smartphone market in Indonesia, this study focuses on the Android-based web server of mobile.

The smartphone web server utilized in this study was optimized for Android-based devices. The BitWeb, the penguin PHP/ MySQL, and the KickWeb, these servers were utilized as study items. On the other hand, they used the Apache JMeter application and WAPT to test the three web servers. Choose a mobile web server, Tools for Testing the Quality of Web Servers, choosing a measurement System of quality using the JMeter as well as WAPT are three stages of this research.

According to the findings of the tests conducted with JMeter, they concluded that BitWeb Server had a better quality overall than web servers of other mobile out of 5 sites that was tested with 45 test items. Also, they concluded from the tests run using the WAPT test tool on the servers such as Web Server Bit, Penguin Php / MySQL, and KickWeb that the BitWeb Server is better than the other Web Servers, as proved by more Successful hits of 6652 and a faster Average response time of 5.79 seconds. As per the finding of the testing from this research, Android-based mobile web server's quality which was purchased from Google Play Store was superior to that of other free mobile web servers.

OpenStack Swift is implemented in order to establish cloud storage so that the whole process can be done in an effective way. An article [14] about implementing cloud storage based on OpenStack swift, shows how this process is so quick and efficient. Here, the authors used Swauth in order to authenticate the whole system and establish such a cloud storage that is compatible with S3. However, they choose Cyberduck, which is an open source software client. With the vast development of technology, the tendency of buying new smart devices such as Android phone, Tablet, iPhone, iPad etc. also increased. These users are increasing the amount of data, specifically the unstructured data including images, videos, audios, etc. In

this way, every day the total amount of data in the storage is getting bigger and bigger. As the usage of these devices increased rapidly, our traditional storages and technologies are unable to deal with this. There are some certain reasons for the traditional system not to cope up with the increasing usage. The traditional servers are very costly, their operation process is complex, maintenance is very difficult and the scalability is very limited. Here, the author [15] suggested that cloud storages can easily meet the requirement of this increasing amount of data and that storages are very cheap and provide scalability and safety of data. People related to this field, used the FTP servers in order to manage the files and documents and it also can provide the security of the files that were unavailable in the traditional storages. But now, object based storage has created a new era of data storage that gives data security along with low cost, reliable, bigger capacity and scalability. However, cloud storages also provide the functionality of accessing their uploaded files from anywhere in the world. Rackspace and NASA wanted to build a service for their enterprises and service providers and so they developed a platform known as OpenStack which is similar to S3 and EC2 of Amazon. But, the major differences between them is that OpenStack is an open source project and the software is completely free.

In this paper [15] they have used Ubuntu which is a popular OS of OpenStack and synchronized it with the OpenStack. After that they developed the swift on a VM and then ran the Swift service on a central server. After that they have set up some scripts that help to run their swift daemons. They have stored their container and account database in some separate folders. The account and the containers directories were established in different nodes. Here, the object's metadata was stored in the object folder. Following that the authentication of the user was required before the server can grant the permission of giving any access to the resources and requests to the user. This authentication part is completely an external part of the swift server and enterprises can establish their own authentication subsystem with the help of the proxy servers. There are in total three different authentication methods and the default authentication method for the swift is TempAuth. In this method the username and password is stored in a plain text format and by default the system grants the permission of the read the access of credential file location. As this method is not used for production deployment, they have used the Swauth method for the authentication as this method is appropriate for the configuration of security of the password data. However, the Keystone can provide more secured and better configuration. Here, they have used Cyberduck as an open source storage browser so that they can connect the swift along with other API.

The first step is adding the user before logging into Swift. This process can be done easily using a simple cURL, which is a CL tool that supports the common internet protocols. Three extra functions were added in the register page so that the system can be fulfilled. This system can be more efficient through the deployment of the clusters in the real life scenarios.

The swift and its related technologies duplicate every single object over several storage nodes in order to provide greater reliability and ultimate consistency. The object synchronization method's efficiency is highly dependent on two critical criteria. The first one indicates the number of replicas associated with each item, while the second

one indicates how many objects are housed by every single storage node. In this paper [24], they have configured the scenarios where the set the value of r equal three and value of n is greater than 1000. They discovered that the sync procedure is greatly delayed and generates a large amount of network overhead, a phenomenon called the problem of sync bottleneck. The inspecting process of OpenStack Swift source code defines that their object sync protocol makes use of a rather basic and network-intensive way to ensure stability across object copies. So, a lightweight sync process was proposed and the hash values were exchanged in each node. Along with that they reduce the overhead of the sync and it is very reliable and consistent. This feature is derived from three innovative building components in LightSync. The authors [24] proposed that their open source path will result in reduction of sync delay by 879 times and overhead by 47.5 times.

There has been a lot of thesis work and research done based on OpenStack swift as cloud server from time to time. In this research paper [21] authors described that public cloud market has grown more than 17 percentage by the end of 2016. Market has increased a total over 208 billion dollars, which was 178 billion dollars in 2015, as per new Gartner estimates. Authors [21] also claimed that cloud computing motivates consumers to shift their apps to clouds. They also mentioned that OpenStack is one of the most widely used open-source cloud computing and storage management solutions, and it is used to develop and manage cloud computing, storage, and networking resources. An object storage service provided by OpenStack Swift that is commonly utilized for cloud-based storage solutions. Swift was mentioned as one of the primary components of the OpenStack software package, according to the authors. They also come up with a survey where they found more than 53 percentage of all OpenStack deployments use Swifts. Moreover, they added large Swift installations are becoming more popular, with 24 percentage having over 100 TB of storage and 32 percentage having over 10,000 items. Backup and storage of Docker/VM images, application data, and Big Data are some of the key use cases for swift. The authors found a drawback of Swift. They claimed Swift is uses traditional TCP/IP sockets-based communication which has some performance issues like context-switch and buffer copies for each message transfer [13], [20]. In this paper, authors tried to analyze the performance characteristics of OpenStack Swift design. After analyzing they found three major bottlenecks in Swift design, namely communication, I/O and hash-sum computation. They found OpenStack Swift operations mainly rely on proxy servers, limiting overall throughput and scalability. Based on their observation they suggested Swift-X, a high-performance architecture and implementation of OpenStack Swift for the purpose of developing efficient HPC clouds. Authors proposed two new designs for improving the performance and scalability of Swift applications. One of the designs is client-oblivious, without modifying the client library or requiring RDMA-capable networking devices on the client node users can use this. Another design is metadata server-based design, which entirely redesigns Swift's put and get functions. Rather than employing the proxy server to route requests, the authors highlighted repurposing it as a metadata server. To enable the fastest feasible object transfer, they presented high-performance implementations of network communication and I/O modules based on RDMA for these both two designs. To further increase the object verification efficiency in Swift, they also investigated other hashing methods that are being used in the community. The

authors wrote in their paper that most people use Swift to upload and download software, simulation input files, experimental results and large datasets. They also use it to download VM images, VM images, and configuration files. There are a number of other Swift requests that go beyond uploading and downloading objects. Since no objects are being sent, none of these activities require a substantial amount of network connectivity or I/O. According to their findings, such processes have a latency of just a few hundredths of a second. Uploading and downloading huge objects, on the other hand, results in significant network and I/O overhead. Using TCP sockets-based communication, the Swift code is written in Python, and network connectivity is implemented using the Swift code. As a result, it is critical to examine the performance of upload and download operations when using the default Swift implementation and to come up with ideas for how to improve the performance of these operations in the future. The authors [20] gave a figure where they showed the breakdown of get and put operations into distinct components for a 5 GB object. In this paper, they suggest designs to accelerate the network, I/O, and object verification (hashsum) components of get and put operations, as well as the overall performance of the system. The researchers proposed expansions and improvements to the Swift command-line client library, object server, and proxy server, as well as to the Swift object server. They introduce a low-latency communication module based on RDMA in the client, object server, and proxy server to facilitate communication between them. They also include an object file-specific I/O module in both the client and the object server to facilitate object file-related activities. The authors make no changes to the Swift client API that is already in place. In this way, existing applications can operate seamlessly over Swift-X without the need for any code modifications. For designing the Client-Oblivious researchers take the replication semantics and design from the default implementation and add a level of fault-tolerance. The working method of this design is the proxy server receives requests and data from the client over TCP. The client's default client implementation sends these requests and data. The proxy server sends requests and data to object servers at the same time through RDMA communication. It then waits for the object servers to send back their responses, before giving the client the final answer. Again, for the metadata server-based design the client will use the RDMA communication module to send a get or put request to the proxy server. To do this, the proxy server will look up what the object needs and where it needs to be sent or gotten from. Once the request and data have been sent to all object servers in parallel, the client terminates the process. This is accomplished through the use of RDMDA (Remote Direct Memory Access). The authors ensure that the semantics of replication are identical to those of the default design and that the fault-tolerance of the cluster is not affected. For object verification the writers describe that while uploading and downloading objects Swift computes the md5 hashsum of each object in the collection. While md5 is a widely used hashing method that produces high-quality hashes, it has a low performance due to its complexity. To take use of their proposed designs, they added new operating modes to Swift. They used microbenchmarks, ssbench, and synthetic application benchmarks to evaluate their suggested design in depth. Our ideas can boost speed by up to 2x for customer designs and up to 7.3x for metadata server-based solutions, according to their study. They want to change the S3 and HDFS Swift clients in the future to operate with their ideas. They also want to conduct more benchmarking and application scenarios as part of

the evaluation.

In another paper [19] they experienced performance testing of web applications using Jmeter where the applications are linked with UAP. They showed the challenges they faced during testing web application in this system. Before launching any application it is an important thing to test the performance of that application in pre stage. To increase the authentication and security web application is linked with UAP. As they faced problem with other tools when testing the performance so they used Jmeter among all the testing tools as it is user friendly and fully open source. Another problem they faced while configuring the master and slave server. In their setup when a user try to connect with web application a script recorder utilize that request and load generator continuously sends that request to web application to test scalability. After a certain time when the site reply to the request the load generator again sends multiple request. Here each search engine is called virtual user. For every web application test there have to be virtual user to request multiple time to the web. Though the virtual user behaves similar to real user. If the virtual user cannot copy the behavior of real user than the result will not be accurate. Jmeter thread attributes are used to run Jmeter scripts. The amount of virtual users to utilize the test may be specified using Jmeter thread settings. The thread group calls HTTP request samplers in order; the UAP gateway panel is launched initially, followed by HTTP request sampler. After that, the user is prompted to choose an authentication method, after which the user is sent to the Login page. When the user enters login credential, UAP verify him and directs him to the program launch screen. The driver script will run in a fixed order described in the Thread group. The Configure parts in the driver script are used to call specified functions. The contents of intermediary data obtained using the pattern extractor are checked using the debug sampler. The number of executing threads and the ramp-up duration are specified using thread groups. Each thread represents a user, and the load duration determines how long it takes to produce all of the threads. For example, if there are 5 threads and the ramp-up time is 10 seconds, each thread will be created in 2 seconds. The loop count determines how long a thread will run. You may also specify the start and finish times of the run using the scheduler. HTTP samplers are utilized, which are server queries that may be customized. This sampler makes a web server an HTTP/HTTPS query. It also determines whether JMeter debugs HTML files for pictures and other associated assets, whether or not HTTP queries are sent to get them. To conclude, they discussed the difficulties they encountered while utilizing JMeter to do performance testing on a web application that was linked with UAP on SSO. As stated in the Experimental Outcomes, they were successful in accomplishing for a limited number of users

To provide high level data dependability and longevity, systems like OpenStack Swift duplicate every single data item over several storage nodes, necessitating the maintenance of replica consistency. The ultimate consistency paradigm for OpenStack Swift-like systems is implemented by utilizing async protocol of an object to compare distinct replica copies of each item. While Swift-based OpenStack systems have been extensively adopted, we intend to gain a better understanding of how effectively they provide consistency in practice. Using OpenStack Swift they began by creating a lab-scale case study. They designed the quick with r equal to three and n fewer than one thousand. Here, r indicates the total unit of replicas asso-

ciated with every single object, where n indicates the number of items housed by every single storage node. In this configuration, the synchronization procedure of the object is severely delayed and generates a large amount of network overhead. Additionally, the issue is significantly exacerbated for the data updates and node failures cases. When a node fails, it requires numerous sync rounds to converge and again enter into a stable state. Additionally, the results demonstrate that parallelism approaches cannot be used to solve this problem fundamentally. As a result, the sync bottleneck problem might easily have a detrimental impact, as many of today's data-centric applications require back-end configurations with r larger than 3 and n more than 1000. In an actual object storage system, the total unit of items is often far more than 1000; second, and perhaps more critically, systems requiring quicker access to a large number of little objects frequently use a bigger value of r . Following that, they dug into OpenStack's Swift source code to gain a detailed understanding of why the sync slowdown occurs. To be more precise, partitioning is a common storage technique that divides the total object storage space into smaller portions, each of which is referred to as a (data) partition.

In comparison to the strong consistency model, openStack Swift provides eventual scalability for each data item, a popular consistency approach in the field of distributed environment. OpenStack Swift ensures the dependability of each object by duplicating it across several storage nodes. In the case of an OpenStack Swift cluster, there are two sorts of nodes. While storage nodes are in charge of storing things, proxy nodes associate links between clients and storage nodes. OpenStack Swift, on the other hand, creates a logical ring which is a representation of the full storage area. However, a logical ring has a large number of equivalent subspaces.

LightSync is intended to take the place of the original object sync protocols in the existing Swift-based OpenStack environment. It obtains the needed qualities by combining the three new building components described below. To begin, it makes use of the Hashing of Hashes (HoH) technique to accumulate all of the h hash values in every single partition into a single yet characteristic hash value using the tree data structure established by Merkle. The determination of the aggregated hash value of data partition results in inconsistencies, the decision will be made first by the local node, the incorrect suffix directory. After that it will figure out the more current version of the suffix directory. While a circular hash checking is ongoing, the storage nodes that host the r replicas of a particular partition P form a tiny logical ring known as the clone ring of P that already exists within the big object ring.

The authors [24] demonstrate in this study that the sync protocol of objects is critical to their performance, namely the important parameters r and n . Their measuring investigation demonstrates an excessively lengthy object sync latency and an unreasonably large network overhead. Situations like this can be described as the sync bottleneck, and it is impossible to resolve fundamentally by enlarging the total number of sync threads. As a result, they develop a revolutionary protocol called LightSync that effectively solves the sync bottleneck problem. The effectiveness is confirmed by both conceptual analysis and real-world trials.

Performance testing was carried out on two identical websites: Politeknik Negeri

Malang's admission website and Universitas Brawijaya's admission website [22] as admission websites are visited by a large number of users in order to gather details of admission. The performance of websites was measured with Apache JMeter by the authors.

The tests were repeated four times with a total of ten tests to discover the error and success on the websites. Both websites were subjected to stress testing to guarantee that the system could handle huge numbers of queries during the peak period.

With the results after testing the researchers concluded that the Selma web performance falls as the duration of tests performed with varied lengths of time increases in four trials undertaken by the researchers. In contrary to the first finding Of Selma UB, Polinema's admissions website was initially steady, but with passing time, the experiment began to deteriorate. This indicates that the website's performance is consistent at first, but with time, it begins to become unstable.

Chapter 3

Background Studies

Cloud computing utilizes resources on a need basis service. This allows the users to connect via the internet while giving them a hosted environment with required services. Along with the emergence of cloud technology, everyone is moving towards embracing a cloud-based framework due to the enormous benefit that we derive from cloud computing. Cloud computing has opened new vistas of opportunity for testing. The Covid-19 pandemic has dramatically increased the number of internet users, demonstrating the benefits of working from home. Load testing is required to ensure a positive user experience. Besides, any malfunction in the system will create great havoc. The system's downtime will cost a significant amount of money in lost revenue. In addition, if the system's reaction time is slow, users would lose interest. Performance requirements, on the other hand, can pose a risk.

3.1 Cloud Computing

Cloud computing is a web-based framework for storing data on remote computers. Cloud-based storage enables you to save files in a distant database rather than on a personal hard drive or local storage device. If the gadget is connected to the internet, it will have access to data as well as all the software programs required to run it. Cloud computing is an emerging alternative for people and businesses because it enables cost savings, increased productivity, effectiveness, and quality, as well as performance, speed, and security. Cloud computing may be classified as either public or private [12]. After accepting payment, public cloud service providers make their services available to users. On the other hand, providers of private cloud limit the amount of customers who may access their services. In cloud computing, there are three services. Examples include Software-as-a-Service (SaaS), Infrastructure-as-a-Service (IaaS), and Platform-as-a-Service (Platform-as-a-Service) (PaaS) [12]. It has two parts: the front-end and the back-end. The front end of a web browser or a cloud computing app is used by people to get to data that is stored in the cloud. This is called front end. The back end is the most important part because it stores information and data in a safe way. They all belong to the same thing: web servers, computers, and databases. We described the various services of cloud computing

services one by one.

3.2 IaaS

Infrastructure as a service (IaaS) is a form of cloud computing [30] that uses virtual computer capabilities to supply resources over internet. When using the Infrastructure as a Service (IaaS) model, the cloud provider manages IT infrastructure such as memory, servers, and web servers, and then delivers them to subscriber firms through virtual machines that are accessible through the internet. IaaS can give various benefits to businesses, including the ability to run workloads more quickly, easily, and cost-effectively, as well as the ability to scale workloads. Under the IaaS service architecture, a cloud provider maintains the infrastructure resources that are traditionally housed from an on-data center. This includes both the hardware elements of workstations, memory, and connectivity, and the simulation or particular material of the operating system. Users of IaaS access to resources and services using a wide area network (WAN), including the web, and can take advantage of a clouds company's services to finish the software stack installation on their servers. If the user logs into the IaaS platform, he or she can create virtual machines (VMs), install OS systems on each VM and install components such as databases; create memory containers for operations and restorations; and install an enterprise workload onto the VM. The services provided by the supplier can then be used to analyze expenses, manage progress, regulate traffic on the network, diagnose application issues, and manage disaster recovery, among other functions. IT organizations want to employ infrastructure as a service since it is often very efficient, speedier, and value to run a workload without the need to own, manage, and maintain the underlying system. Internet as a service can be come in handy for different types of applications. The resources it provides via a model can be used for many types of tasks. There are several critical considerations to make when implementing an IaaS product. Before considering different technological requirements and providers, it is critical to describe the IaaS use cases and infrastructure requirements precisely.

3.3 PaaS

Developers essentially lease anything they require, construct an application using the Platform-as-a-Service (PaaS) model [27], relying web technologies, architecture, and software platforms are all hosted by a cloud service provider. It's one of the three models of cloud computing that are currently available. Web application development is made substantially easier using PaaS, because all backend management is handled behind the scenes, rather than in front of the developer's eyes. While PaaS and serverless computing have a number of qualities in common, they also have some important differences. PaaS is available via any internet connection, allowing for the construction of a complete application in a web browser without the need for any other software. Due to the absence of a local development environment, developers can access the application from any location in the world and work on

it. This permits collaboration amongst teams that are geographically dispersed. Additionally, it implies that the development environment is less controlled by the developers, however at a far lower cost. PaaS is accessible via any internet connection, which enables the development of a whole program in a web browser. Due to the absence of a local development environment, developers can work on the project from any location they prefer. This permits collaboration amongst groups which are geographically dispersed. Additionally, it implies that they have less control over this, however at a far lower cost. With the help of this, developers only require to write code and check it; the vendor takes care of the rest. PaaS enables them to analyze, troubleshoot, distribute, serve, and upgrade their applications all at the same time. This helps developers to validate the application will run effectively as an application hosted prior to releasing it, and thus simplifies the application development lifecycle.

3.4 SaaS

SaaS is an acronym that stands for Software as a Service [23] is a cloud-based service in which, rather than installing on user desktop PC or workplace network to operate and update it, users access an application using a web browser. Software applications can be anything from productivity software and collaboration tools to voice over IP systems and collaboration tools. They can also be any of a number of other business apps. This has a variety of advantages as well as disadvantages. In addition to being easy to use and compatible with other systems, SaaS also offers the advantage of operational control. Additionally, in compare to conventional software downloading and installing, SaaS models are less expensive up front, enabling smaller enterprises to challenge established markets while simultaneously providing providers with greater flexibility and empowerment. One notable advantage of any Software as a Service application is it can be accessed via an internet browser, regardless of the Os that is being used to access it. In this way, the application remains available irrespective of if the users attempt to use it on a desktop OS like window, Mac, or Ubuntu (or on a smartphone running Android or iOS). As a result, SaaS applications are incredibly versatile and can be used in a variety of ways. Another key advantage of Software as a service is that, because they are housed in the web, the vendor can update them from a central location without interfering with the customers' day-to-day business activities. When compared to on-premise technology, which usually requires suitability and endpoint protection testing before even the most basic upgrades and fixes can be applied, this is a significant advantage. The cloud-based prototype avoids the pitfalls of test results, which can slow down the production loop and consumer exposure to technology features, while also making sure that security fixes are applied quickly, as opposed to on-premises software, which might also persist open to risks until the IT provider management staff has finished their checking. Then it brings us to another of SaaS's primary selling points: the lack of primary costings needed for using it. On-premise software requires more than just compatible software and hardware configurations on business PCs or other computers; extra servers and switches and routers may be needed as part of a broader investment in IT infrastructure services to ensure

that the software is supported throughout a company's operations. SaaS eliminates this problem, which means that even a small firm can now use this solution using SaaS-based cloud apps that were previously exclusively available to companies. Additionally, it's scalable in the sense that if the number of clients using your service needs to be increased or decreased, you simply adjust the pricing plan - rather than needing to spend in extra gear or put expensive gadgets on hold when demand drops. Overall, SaaS offers a wide range of advantages that should be beneficial to both providers and consumers. However, while some corporations may prefer to develop their own cloud management services and maintain control of the data via harmonies among devices and sites, SaaS provides unrivaled opportunities for the vast majority of small businesses to grow, expand, and provide greater value to both their employees and customers.

3.5 FaaS

Function as a Service is a sort of virtual reality that enables the user to apply coding skill in scenarios where the need for the extensive infrastructure is related with developing and starting small spaced applications. It is common for internet-based software applications to require provisioning and administration of a cloud or genuine real server, as well as several operating systems and cloud servers for the essential tasks. With function as a service, cloud service provider manages the desktops, on the operating system that are on VMs, and web-based software's automatically. This enables the user to concentrate entirely on specific functions within your application's code. Because it offers easy isolation and scaling of transactions, FaaS is well-suited for a huge amount and ridiculously workloads at a time. Additionally, this can be also come handy to develop the backend servers and doing the processing of data, format changing, debugging, and data manipulation. This is also an excellent medium for developing server-based applications, behind the servers, stream allowancing, as well as for developing online chatbots and backends for Internet of things-based devices. Function as a server can help users with managing as well as utilizing third-party services. If one user is considering developing a mobile application, for instance, a function as a server model is very cost efficient. Due to the fact that users are only charged when the app connects to the server for a conducting a particular task, such as group processing, prices are usually significantly cheaper compared with a more known method. Furthermore, it can significantly improve the performance. For instance, three students recently collaborated with experts to investigate ways to use Cloud Functions to conduct a proposed simulations (computing methods for estimating the feasible outcomes of particular difficult-to-predict situations) to calculate stock values. The proposed simulations are a revolutionary task performance enhancing computing. The combination of proposed simulations and cloud functions gave opportunity to the team to do computations on a vast scale while concentrating on business logic.

3.6 Cloud Storage

When using cloud storage, data is stored on the Internet by a cloud computing provider who manages and administers data storage as a service, which is a cloud computing technique. Cloud storage is a service [25] that stores data, manages and backups it remotely, and makes it available to customers through the internet (via internet). Cloud storage is obtained through the purchase of a service from another hosting company that owns and operates data storage capacity and makes it available via the Internet on a pay-as-you-go basis. These cloud storage companies handle capacity, security, and durability in order to make data globally available to your applications. There are numerous cloud storage companies. Most companies offer free storage space up to a specified number of gigabytes. DropBox, for example, offers free storage space of up to 2GB, while Google Drive space up to 15GB, Amazon, and Apple Cloud offer free storage space of up to 5GB, and Microsoft SkyDrive offers free storage space of up to 7GB. If a customer exceeds the allotted free space, they will be charged the appropriate amount according to the plan. Limitations on file size, automatic backups (if available), bandwidth, and upgrades for restricted space vary from provider to provider; for example, DropBox supports files up to 300MB in size, while Google Drive supports files up to 1TB in size. Customers who use cloud storage services save money since they do not have to purchase storage devices and do not require technical support for maintenance, backup, and disaster recovery [25]. There are many types of cloud storage. Among them we have worked with OpenStack swift.

3.7 OpenStack Swift

The object or blob storage available for distributed and eventually consistent environments is known as swift. Many institutions use swift in order to store their huge data cheaply and safely. Swift is a project that is known to store the OpenStack object and it is used in order to store and access huge amounts of data using a very simple API. Swift is a boundless storage that can store the unstructured data.

3.7.1 Characteristics

Any object that is located inside the swift should have a specific URL and its own metadata. All the fragments for an object are located in a unique position in order to get a longer durability and availability. The data doesn't require any type migration to a new storage system. Besides, the addition, swapping and removal of new nodes into a cluster can be done without any interruption. Developers can write in any existing popular programming language or in the swift API.

3.7.2 Components

There are some components of swift that ensure the availability and durability and we will discuss them.

Objects: Objects are the data that are stored in the swift.

Proxy servers: Proxy servers are used in order to handle all the incoming requests of the API.

Rings: Rings perform the task of mapping the logical data names into the location of a specific disks.

Accounts and Containers: Both the account and container are separate databases, where the list of containers are stored in accounts and the list of the object is stored into the containers.

Zones: Zones are used to separate the data from one another and any error in one zone doesn't have any impact on the rest of clusters.

Partitions: A partition is used in order to store the objects, databases of container and account as well as assist to control the location of data inside the cluster.

3.7.3 Ring-builder

The management and building tasks of the rings are done by swift ring-builder. Ring-builder stores its own builder file along with the ring information as well as the data that are required further in order to create a new ring in future. So, keeping multiple copies of builder files is highly suggested. This backup process can be done by copying the builder files outside the server and uploading them into the cluster. failure to do so will result in creating a new ring from starting.

3.7.4 Object Storage Monitoring

As the swift cluster contains a huge amount of daemons collaborating across various nodes. As the cluster contains so many components, it becomes important to have a clear knowledge of each task as well keep tracking them. Swift recon, swift-informant, statsdlog, swift stats-dlogging are used for this monitoring process. Beside them cluster architecture, replication, large object support, object auditor, erasure coding, account reaper and troubleshooting object storage are points that are a major part of OpenStack swift.

3.8 Command line

We have used the command line for setting the whole OpenStack swift setup. Command line is a user interface that can perform all types of tasks that can be done by GUI (Graphical User Interface). Though the GUI and command line perform the same task, the command line interface can perform some tasks very quickly and can be done remotely. The linux command line can be also known as terminal or console.

3.9 Servers

Servers are computer programs that provide functionality to other applications or devices called "clients". This design is called the client-server model. This design is called the client-server model. A server is a collection of web pages that responds to a user's request for information about a certain website. Servers are capable of performing a wide range of tasks that are collectively known as services, such as exchanging data or resources among a few clients or conducting a calculation for the benefit of a single client. A single server can serve several clients at the same time, and single client simultaneously can access several servers from the same device at the same time. It's possible that a client process is running on the same device as the server process, or it may connect to a server process running on a different device through a network. Simply typing the web address into a browser and pressing the return key constitutes this request. The server watches these requests through the use of ports, providing a response that is practically instantaneous in order to provide the web page requested. As soon as the server receives and validates the request in hand, it begins gathering the pieces that build up a website and communicating this gathered returns data to the user's internet browser. Client-server systems are most commonly used to implement the request-response model, a client requests a server, which does some action and returns a response back to the client, generally followed by a conclusion or an acknowledgement of the request and action. The request-response cycle is the foundation of the work a server performs on a daily basis.

3.10 Virtual Machine

Virtual machine is an image file that performs like an actual computer. It is a separate window inside a computer where the user can run another computing environment and even the operating system can be different. However, though a VM contains a different environment and OS, it doesn't have any impact on the computer interface and primary OS. Beside, inside a single physical hardware the user can create multiple OS with the help of VM. There are many types of virtual machines where the windows VM is used to run different versions of windows OS. It creates a partition and runs a separate windows OS with any overlapping

issue. Then the Mac virtual machines help to run the MacOS in the mac hardware builded interface. Android VM is used to run Google's open source androidOS virtually. Java VM is used to run any environment that is written in java, and it can be run in any hardware setting using the java platform VM known as JVM. The python virtual machines are similar to Java, it translates the program into bytecode and then bytecode converts into machine code and executes it. Lastly, the desktop and server version of Ubuntu can be run in VM and it offers a lot of benefits for the user [6]. The VM has some advantages that we are going to discuss here. Installing a new OS inside the VM is easier than installing it inside the computer. So, VM gives the user the faster and easier experience of different OS. Besides, the VM offers the testing of new operating and malware investigation. However, A VM also allows the user to test some older versions of softwares that are not available with the updated version anymore. Another most important feature of VM is security. The VM allows the user to visit any sites without the concern of any infection. VM is also a fundamental part of cloud computing [6]. In this work, we have completed the work on Virtual Machines of our own system.

3.11 Virtualization

Virtualization is the act of generating a virtual version of something in computing. This might include virtual computer hardware platforms, virtual storage devices, and virtual computer network resources, among other things. A means of logically partitioning the system resources given by mainframe computers across distinct programs, virtualization was first introduced in the 1960s and has been in use ever since. As a result, the meaning of the phrase has grown more expansive. Running multiple operating systems on one computer system at the same time is the most common application of virtualization. As a result, when applications run on top of a virtualized machine It seems as if they are running on a distinct dedicated machine; the operating system, libraries, and other programs are all specific to the guest virtualized machine and are not connected to the host operating system, which is running below it on the host computer's virtualization infrastructure. Virtualization has become a standard approach in organizational information technology design. It is also the technology that is responsible for the economics of cloud computing. When cloud providers use virtualization, they can continue to use their existing physical computer hardware to serve their customers; when cloud customers use virtualization, instead, they can acquire only the computing resources they demand at the moment of need and scale those resources as their workloads expand in a cost-effective manner [12]. A wide variety of virtualization technologies are available; desktop virtualization, network virtualization, storage virtualization, application virtualization, and data virtualization are just a few examples [11].

3.12 Test Case Metrics

Metrics can be both beneficial and damaging to the development and testing life cycle. It depends on how they are interpreted and applied [7]. People generally talk about metrics and how to do measurements correctly in any type of organization. Some of them employ potentially dangerous measures to evaluate team members' performance, while others use relevant, meaningful, and insightful metrics to improve their processes, efficiency, knowledge, productivity, communication, teamwork, and psychology [2]. A software testing metric is a numerical measure that aids in estimating progress and quality of a software testing process. A metric is a measure of how much a system or one of its components has a particular feature. Metrics for software testing are significant for a variety of reasons. It aids in the decision-making process for the next phase of activity. It is proof that the claim or prediction is correct. It assists us in determining the type of change that is required. It facilitates decision-making and technological progress. Process metrics, product metrics, and project metrics are different kinds of software testing metrics. In the software development life cycle, process metrics are used to increase the efficiency of the process. Product metrics are used to assess the software product's quality. Process metrics are used to assess the efficiency of the project team as well as the testing instruments employed. Determining software test metrics entails a number of stages. Identify the essential software testing processes that will be measured first. After that, the tester utilizes the data to create the metrics. Furthermore, the defined metrics must be calculated, managed, and interpreted correctly. Identify areas for improvement based on how the defined metrics are interpreted. The test cases we have designed in this work have been prepared following these metrics.

3.13 Test Case

The documentation in a testing process that contains the set of test data, preconditions for the test and what are the expectations from the test is known as test case. In order to run a successful test, the test case contains a major part in it. It offers the testers to check the validity of some specific requirements and parameters. The testers set some test cases that will meet those requirements and after running the test based on those test cases, the validity and success rate for each task can be identified very easily. Often, a test case is considered as the starting point of any testing as all the experimental setup and work flow is designed based on the test cases. Based on the test case the test values are declared and then the test has been executed. After running the test, the testers receive some outputs and outcomes. Then based on the post condition the testers try to evaluate the result.

Test cases are considered as a set of steps that defines the structure of executing a specific task and verifies the functionality of that specific function. As the test cases play an important role for a project, the specification of the test cases is also important so that all the functionalities can be checked. In order to do so, the test cases contain some parameters that are discussed below:

1. A test case should contain a test id so that we can easily identify it anytime.
2. A test case should contain the test scenario where the test should be conducted.
3. The test case can also include the description of the test case.
4. The test case should contain the steps to run the test case. When the testers have a good idea about the steps serially, it will be easy for them to implement the test and the process will be quicker.
5. The test case might contain the prerequisite of that specific task. It will be easy to set everything for the test before implementing the test steps.
6. The test case might contain the information about the test data.
7. As every task should have specific outcomes, every test case should contain the outcome or expectations from the case. Additionally, it will be helpful for the evaluation in the future
8. Test case can contain some comments too.

The test cases can be divided based on these parameters and the testers can select the necessary parameters based on their project type. However, when a test case contains too much information, the tester can divide the test case into smaller parts [3]. The test cases have some advantages that we are mentioning here.

1. Test cases ensure that the testing process is done efficiently and without any error.
2. It is beneficial for project quality improvement.
3. It helps a tester to design the plan keeping every aspect and possible angles.
4. It helps the tester to verify that all the requirements are met properly.
5. Test cases can be used in the future and so it can be reusable in further improvement of the project.

However, along with so many benefits, improper design of test cases may lead to some disadvantages. The number of the test cases should not be large as this will create the testing process tough for the tester and will make the process very long. So, it is very important to design the test cases according to the requirements [16].

3.14 Load Testing

Load testing aims to estimate how well an application performs when it is loaded with an increased number of users. This load is applied to the program for a certain period, and the obtained results show if the requirements of a particular application

are met with the expectations [4]. Load testing allows to assess site's QoS based on real-world customer behavior. Customers' requests are used by a script recorder to produce interaction scripts when they visit site. The scripts are then replayed against the Web site by a load generator, which may be adjusted by test settings. The load generator behaves in the same way that a browser would. It sends requests to the Web site on a regular basis, waits a certain amount of time until the site responds, and then sends another request. To test the scalability of a Web site, the load generator may simulate thousands of simultaneous users. A virtual user is a critical load-testing term that refers to each emulated browser. A load test is only valid if the behavior of virtual users is similar to that of real users. System component performance under various loads, database component performance under various loads, network interruption in between server and client, software design problems and server configuration errors like web server, application server, database server, and so on are all identified through load testing. Establishing a dedicated load testing environment, generating load test scenarios, conducting test scenarios, collecting various metrics, evaluating the findings, providing recommendations, fine-tuning the system, and re-testing are all steps in the load testing process.

3.15 Scalability

Scalability is the ability of an organization, system, model, or function to cope with and perform well under an increased or expanding workload. In cloud computing, scalability refers to the capacity to scale up or down IT resources as needed to meet fluctuating demands. Scalability is one of the distinguishing characteristics of the cloud, and it is the fundamental reason for the cloud's rising appeal among organizations. Generally, scalability refers to strategies that ensure that some level of service quality is maintained even when the number of users grows or the complexity of the world grows. While unique latency requirements justify the selection of a specific consistency method, the majority of these strategies make the assumption that a certain amount of bandwidth is readily available. Using the level of interest that clients have in other entities in the environment is the most essential way in which scalability is made possible [1]. It is common for stability and competitiveness to be accompanied by scalability, as it indicates that a given network, system, software, or organization is capable of dealing with an increase in demand as well as with increased productivity, trends, changing needs, and even the presence or introduction of new competitors [1].

3.16 Testing Tools

A tool that is used for software testing is a product that facilitates one or more test operations, such as planning, requirements, building a build, test execution, defect tracking, and test analysis. Several parameters may be used to categorize testing tools. Such as, the tool's objective, the activities that the tool facilitates, the technology that was utilized, supported types and levels of testing, and the

licensing type (open source, freeware, commercial) etc. Testing tools in cloud load testing plays a significant role in the Software Testing Technology. Few cloud-based testing tools are Apache JMeter, Blazemeter, Gatling, SOASTA, WebLOAD etc. These testing tools analyze the problem carefully to identify strengths, weaknesses and opportunities. They also take into account limits like as budgets, timelines, and other considerations. Shortlisting the solutions that satisfy the requirements and evaluating them is also a task of testing tools. In this work we have worked with Apache JMeter for load testing.

3.17 JMeter

The purpose of load testing is to check if the system under test can handle the needed number of concurrent user visits on the web server without failing. JMeter one of the greatest load testing tool available since it can test both static and dynamic resources. JMeter is a tool for detecting concurrent users on a website and providing graphical performance information. JMeter is a free, open source program that anybody may download. Another advantage of JMeter is that it is simple to use. There are no prerequisites for using this tool, which implies that anybody may use it regardless of their skill level or domain expertise. If anyone wants to use JMeter to test he has to know some elements of JMeter which are Thread Group, samplers, Listeners and Configuration. Thread represents one user using the application. From the thread group one has to add the action which will be taken if sampler error occurs. Then he has to set the number of request he wants to conduct. After setting up the thread group he have to add a sampler in the thread group to set what kind of request we will send to test. There are multiple option like http requests, flow control action, ftp requests and many more. If he choose HTTP request then that HTTP request let him send http requests to the server. In the HTTP request dialog box he has to give URL or IP of the server as well as port number and path. Then to see the results of the test one can select any option he wants to see. There are multiple option to see the output like table, graph and so on. We have used table and graph to show the result in the later chapter.

3.18 Latency

In the networking field, Latency simply defines the travel time of a packet from source to destination. To expand it, when a sender wants to know something, the sender sends a packet to the receiver and after receiving the packet, receiver sends another packet to the sender that contains the required information of the sender. So, the time required between a sender sending a packet to receiving the responding packet is known as RTT(Round Trip Time). Almost half of the RTT is known as latency. In other words, it can be said that latency is the time delay occurring between the users requesting for information and the receiver receiving the request. For instance, let us consider two different connections, the first connection is from BRAC University Mohakhali campus and the other one is from BRAC

University residential campus. Both of the connections have similar configurations and bandwidth. However, when the users try to access some information from the network, in both networks, users get the similar output. But, the users from BRAC University residential campus can see the blank pages for a longer time compared to the BRAC University Mohakhali campus. This delay happens due to the difference of latency between the networks. The connection of BRAC University residential campus is slower than the BRAC University Mohakhali campus as the latency for BRAC University Mohakhali campus is lower than the BRAC University residential campus.

Latency differs from one geographical location to another [10]. One of the major reasons for latency is the distance from the user to server. To illustrate, if we have a server situated in Dhaka city, then we will get the lowest latency in Dhaka city when the user is from Dhaka. On the other hand, when a user from Chattogram will try to access the same information, it will take more time for the users as the latency is higher there. Even though the data transmission through the internet is a very quick process, it will take some time to travel from one point to another. However, even if the data transmission rate is almost close to the speed of light, it will take longer to travel a longer distance. So, distance is a major issue to make an impact on latency. Furthermore, the internet traffic is another reason for latency difference. When the sender requests information to the server, if the traffic is busy on that path, the data will try to find a new path to transmit the data and it will result in a difference. When the HTTP response amount is higher, it will create a delay to pass through and the latency will increase as well. Besides, the routers need to process the data and divide them into smaller packets if needed that will add some extra time. So, this is also a reason for an increase in latency. However, in most of the cases, the sender sends the packet to a server and the establishment of connection with the server, the size and type of the requested file are other reasons that cause the difference in latency value. To conclude with, Data latency can be reduced by public clouds, where the user can easily access a file from their nearby cloud server, it will reduce their latency and offer them a good internet experience. Because of latency we get different types response time.

3.19 Response Time

The whole amount of time required to respond to a service request is referred to as response time. A memory fetch, a disk IO, a complex database query, or the time required to load an entire web page are all instances of services. For the sake of argument, consider that response time equals the sum of service time and wait time. The service time is the time required to do the task you've requested. The wait time indicates how long a request must wait in the queue before being served, and it can range from 0 (no waiting required) to a large multiple of the service time (many requests are already within the line and must be treated first). It's straightforward to calculate the increase in average wait time as the device supplying the service goes from zero to one hundred percent occupied using standard queuing theory calculations. As the device becomes busier, the average wait time increases non-

linearly. If we approach 100% capacity, the response time increases dramatically; this increase is entirely due to increased wait time, which is the result of all requests waiting in queue to be processed. We have run our test 4 times and taken the highest response time because it gives us the idea of time taken in the worst case for every test cases. We have run our test 4 times and taken the highest response time because it gives us the idea of time taken in the worst case for every test cases.

Chapter 4

Proposed Methodology

Throughout this chapter, we will provide a quick and extensive explanation of our overall system workflow, as well as all of the approaches we followed to perform our investigation. Because load testing of a distributed server needs to be done in a certain way, it is very important to have a plan in place. Despite the fact that there are numerous approaches for detecting cognitive load on an OpenStack swift server, the necessity to measure it objectively and validly has piqued the interest of researchers and prompted the development of numerous alternative methods. In our methodology, we have tried to give an idea of the cloud server which we have created using OpenStack swift, client PC and test cases.

4.1 Work Flow

For getting the desired output from our experiment we need to set up an environment where we need to give concurrent requests to the server and evaluate the outcome of the server. At first, we created the server using OpenStack swift and enabled all the permissions. After that, we have uploaded a large amount of files to the server. The files were categorized based on their type and size so that the results can vary from one another. Furthermore, we have selected some client PCs that will interact with the main server. Besides, we have installed the testing tool apache JMeter in each client PC. Unfortunately, due to the pandemic, we have used a VM along with the primary OS so that we can add some extra devices to run the test. Then, we started to send the request to the server from the client PCs. At first, we verified the process by checking one single device. After that we started to give concurrent requests to the server and checked for the validity of the server. While giving the concurrent requests, we have used our test cases that we have designed and then we have stored all of our test cases so that we can analyze them later. However, Apache JMeter was installed in every single PC so that we can check the performance of every client too.

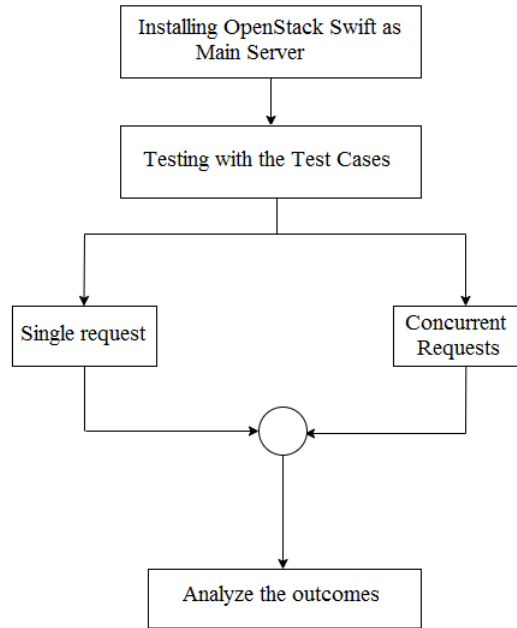


Figure 4.1: System Work Flow

4.2 Test Case Metrics

While conducting research, the first step is to figure out what our expectations are and based on them we need to set some test cases. Here, we have designed a total of 8 test cases that will cover our whole work findings.

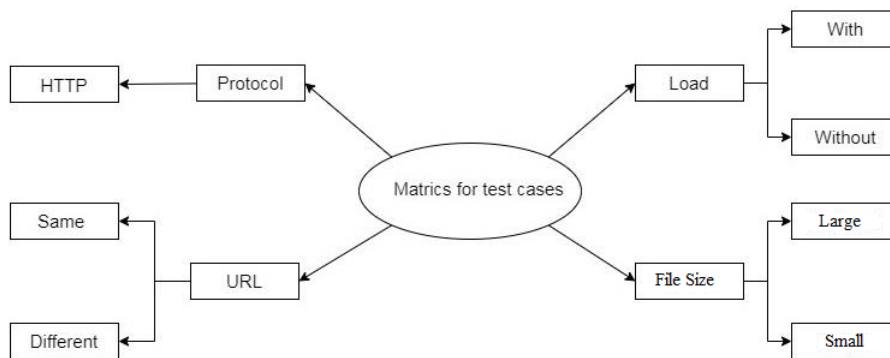


Figure 4.2: Proposed Test Case Metrics

Here, we have divided our test cases based on 4 parameters showed in figure 4.2. The first parameter is protocol. For our each test case we have used HTTP protocol which is a very common protocol that people use on a daily basis. It is an encrypted protocol. The second parameter is file size. We have decided to divide the unstructured files into two categories. The first category contains the large size file and the second one contains the smaller size file. The reason for this categorization is that

we want to check how our server will react when we try to access a file. The other two parameters are URL and load. Here we have planned to run our test case in case the same file contains the same URL with and without load. We will verify that if we request a small file with and without load, how our server will respond. Besides, the similar process will be run for the large files. Based on these parameters, TC0, TC1, TC2, TC3, TC4, TC5, TC6, and TC7 total 8 test cases were designed. We will use these test cases to verify our requirements and expectations.

4.2.1 Concurrent Request for Load Testing

To get a better understanding about the server response, we have sent a single request to the server in the initial stage and we verified the connectivity. Along with that we have collected all the data and stored it so that we can verify the result. After that we have started to request for different types of file concurrently. The reason for sending the concurrent requests is checking the load capacitance of the server and obtain the variety of result so that we can make an analysis on them.

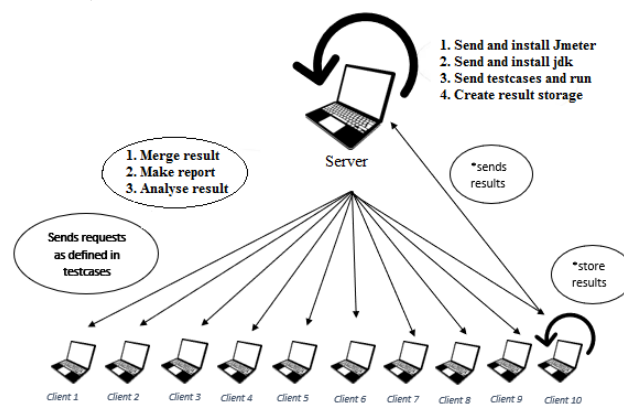


Figure 4.3: Concurrent Request Architecture

4.2.2 Cloud Server

From the experimental setup we got a container where we can upload single and multiple files according to the test cases we have prepared. With proper use of the command we can upload various types of files. These files can be later downloaded on demand too. After the upload process we can check from the stats about the state of the container. As in we can easily figure out the number of files, total size of it, and etc. For uploading large files the procedure was the same. We have uploaded large and small unstructured data. That means the likes of Text, Audio, Video, Zip etc. This type of data has different access patterns. In our container this data can easily be stored without any problems. The whole container can be extended according to demand. So, we don't have to worry about storage any more. As discussed before we have used replicator, this will work as a backup of the files on the container. Even if the container gets corrupted we have the backups of the files. So, data recovery will be a matter of time. The container has login credentials. So,

security will not be an issue. The user can feel secure while using this system. This container can come in handy for storing large files as well. We have uploaded large and small files both as we need to check how the server responds to different types of file. These files were uploaded according to the test cases we have designed.

4.2.3 Client PC

After the completion of making the main server, we need some client PC that will help us to send the request to the main server. In this process, we have chosen to set up the PC with different configurations. However, due to the ongoing pandemic situation we couldn't manage enough physical PCs and so we have used the VM to create different configurations. In each setup we have installed Apache JMeter as we need to collect data from each client PC along with the main server.

4.3 Test Case Design

To verify each and every task, we have designed our test cases based on some parameters such as the protocol, url, file size, and server load. Here, we are explaining each of our test cases to run our experiment in table 4.1:

ID	Protocol	File size		URL		Load	
	HTTP	Large	Small	Same	Different	Without	With
TC0	yes	yes		yes		yes	
TC1	yes	yes		yes			yes
TC2	yes		yes	yes		yes	
TC3	yes		yes	yes			yes
TC4	yes	yes			yes	yes	
TC5	yes	yes			yes		yes
TC6	yes		yes		yes	yes	
TC7	yes		yes		yes		yes

Table 4.1: Test Case Scenarios Under Different Parameters

TC0: In our first case, we have chosen HTTP as our protocol. Secondly, the file size should be large and the url will be the same. Furthermore, for the first case, we will run the test without any load on the server.

TC1: For our second test case, the url will remain the same and our file size will be large. But this time we will run the test with a load so that we can verify the connection.

TC2: For our third test case, the url remains the same but the file size will be small. This time we will again run the server without load.

TC3: For TC3 the file size and url will remain the same. But this time we will test the run with some load on the server.

TC4: For TC4, the file size will be large and this time we will use a different url.

Along with that we will run the test without any load.

TC5: For our sixth test we will change our url and keep the file size large having some load on the server.

TC6: In TC6, the file size will be small and the url will be different this time. Moreover, this time we will test the server without any load.

TC7: In our last case, the file size will be small and the url will be different. Also the server will have load on it while conducting this case.

4.4 Geographical Location and System Configuration

As we want to check the server load capacity, we need to send requests to the server from different geographical locations. And so we have distributed our client PCs in different geographical locations so that we can check the variety of results. Table 4.2 shows the geographical locations of the client and server.

Device	Location	CPU	RAM	System Type	Bandwidth
Cloud Server	Banasree	3.4GHz	16GB	64- bit Operating System	12 Mbps
Client1	Rampura	3.0GHz	8GB	64- bit Operating System	16 Mbps
Client2	Rampura	3.0GHz	8GB	64- bit Operating System	16 Mbps
Client3	Mirpur	2.5GHz	4GB	64- bit Operating System	10 Mbps
Client4	Mirpur	2.5GHz	4GB	64- bit Operating System	10 Mbps
Client5	Banasree	3.0GHz	16GB	64- bit Operating System	12 Mbps
Client6	Shymaoli	3.4GHz	8GB	64- bit Operating System	15 Mbps
Client7	Shyamoli	3.4GHz	8GB	64- bit Operating System	15 Mbps
Client8	Gazipur	2.1GHz	8GB	64- bit Operating System	10 Mbps
Client9	Gazipur	2.1GHz	8GB	64- bit Operating System	10 Mbps
Client10	Banasree	2.5GHz	4GB	64- bit Operating System	12 Mbps

Table 4.2: Geographical Location and System Configuration

4.5 Testing through JMeter

At first we completed the container creation and file uploading according to the test cases, then we tested the load of the server using a JMeter. JMeter is one of the most reliable parameters for this purpose. But for testing the load we needed to generate a URL. For that we gave the proper read write permissions and we were able to generate a URL for the load test we have done in JMeter. At first to test the server, we had to create a test plan. Test plan worked like a container which contained all the test plans we performed. From the test plan we created a thread group which contained threads. We have configured JMeter in our master pc and all the client pc. We have used a master pc for remotely controlling all the client pc. As we want to send concurrent requests to the server for every single test case. It

is impossible to handle all the 10 clients individually. So, we connect all the client pc under one master pc to run the concurrent request smoothly. Ubuntu has by default options for enabling the pc to control remotely. We have configured the `jmeter.properties` in master pc. Then the IP addresses of each client pc has been added in that folder that allows us to send concurrent requests from the client pc through the master PC's JMeter.

Next, in the thread group of JMeter we set everything according to our test case scenario. We set the number of threads, ramp up time 20 seconds and fix the loop count to only one. After setting up the thread group we have to add a sampler in the thread group to set what kind of request we will send to test. In our case we selected HTTP requests. This HTTP request had let us send HTTP requests to the server. In the HTTP request dialog box we gave the URL of our server as well as port number and path. After setting up JMeter we move to run our test cases for having the result. While checking for no load on the server we select any one IP of the client pc and conducte the remote run. While running the test with load, we select remote run all. Which let us send requests from all the client pc at the same time. When we conduct the remote run all JMeter sends requests at a time to the server's specific path which we set in JMeter based on our test case scenario. Moreover, to see the results of our test we have another element which is the listener. From the listener we closed what kind of output we wanted to see from there. We have selected the response time graph and table to view our results.

Chapter 5

Experimental Evaluation

5.1 Experimental Setup

5.1.1 Swift Setup

Firstly, we have installed Ubuntu 18.04. Then, 3 hard drives of 6 GB each have been added to the Linux environment. After selecting 3 hard drives of 6 GB, we have completed the VM setup. For the VM setup, we have started the terminal for writing the necessary commands for the whole process. Firstly, we have used `sudo apt-get upgrade`, this command will get the necessary upgrades done to the operating system. Depending on Ubuntu 18.04, we have to use commands to install new dependencies. Following the above command, we have used another command for the same purpose. For installing the Swift CLI (`python-swiftclient`) via git repository, we used the necessary command. After using some commands, we went from origin to a tree branch to complete the installation process smoothly. Then another command is used to install the necessary requirements. Then we installed swift from git repo similarly we have done in the `python-swiftclient`. A directory for swift needs to be created first and then the config files need to be stored there.

The externally added drives will get mounted here from this stage. After using the command, it showed us the drives and its status. Then an important command is used for showing known devices from the system. Then we used another command line to mount the mounted disks of the system. After that we have to label the drives for identification. Then we have to mount the drives in our virtual machine.

After mounting, we checked whether the mounting was done correctly or not. So, we have used a particular command for that purpose. After giving that command we get the confirmation that mounting is done successfully. Furthermore, we need to give read/write permission to the swift user so that they can have access. Without giving this permission, the swift can be used only for viewing the system. So, giving the permission is must.

We have used the `nano` command a lot. It is particularly used for editing the config

files which we have executed later. The executable programs are a necessary part of the whole setup process. So, for editing the config files this part is very essential.

Three ring builders are needed. They are account builder, container builder, object builder. In order to build that we have used a replication value of 3. The commands are given down below:

```
mohtasin@asif-18101096:~/swift/bin$ cd /etc/swift
mohtasin@asif-18101096:/etc/swift$ sudo swift-ring-builder account.builder create 3 3 1
[sudo] password for mohtasin:
mohtasin@asif-18101096:/etc/swift$ sudo swift-ring-builder container.builder create 3 3 1
mohtasin@asif-18101096:/etc/swift$ sudo swift-ring-builder object.builder create 3 3 1
mohtasin@asif-18101096:/etc/swift$
```

Figure 5.1: Ring Builder Creation

```
mohtasin@asif-18101096:/etc/swift$ sudo swift-ring-builder account.builder add r1z1-127.0.0.1:6002/d2 100
Device d1r1z1-127.0.0.1:6002R127.0.0.1:6002/d2_"" with 100.0 weight got id 1
mohtasin@asif-18101096:/etc/swift$ sudo swift-ring-builder container.builder add r1z1-127.0.0.1:6001/d2 100
Device d1r1z1-127.0.0.1:6001R127.0.0.1:6001/d2_"" with 100.0 weight got id 1
mohtasin@asif-18101096:/etc/swift$ sudo swift-ring-builder object.builder add r1z1-127.0.0.1:6000/d2 100
Device d1r1z1-127.0.0.1:6000R127.0.0.1:6000/d2_"" with 100.0 weight got id 1
mohtasin@asif-18101096:/etc/swift$ sudo swift-ring-builder account.builder add r1z1-127.0.0.1:6002/d1 100
Device d0r1z1-127.0.0.1:6002R127.0.0.1:6002/d1_"" with 100.0 weight got id 0
mohtasin@asif-18101096:/etc/swift$ sudo swift-ring-builder container.builder add r1z1-127.0.0.1:6001/d1 100
Device d0r1z1-127.0.0.1:6001R127.0.0.1:6001/d1_"" with 100.0 weight got id 0
mohtasin@asif-18101096:/etc/swift$ sudo swift-ring-builder object.builder add r1z1-127.0.0.1:6000/d1 100
Device d0r1z1-127.0.0.1:6000R127.0.0.1:6000/d1_"" with 100.0 weight got id 0
mohtasin@asif-18101096:/etc/swift$ sudo swift-ring-builder account.builder add r1z1-127.0.0.1:6002/d3 100
Device d2r1z1-127.0.0.1:6002R127.0.0.1:6002/d3_"" with 100.0 weight got id 2
mohtasin@asif-18101096:/etc/swift$ sudo swift-ring-builder container.builder add r1z1-127.0.0.1:6001/d3 100
Device d2r1z1-127.0.0.1:6001R127.0.0.1:6001/d3_"" with 100.0 weight got id 2
mohtasin@asif-18101096:/etc/swift$ sudo swift-ring-builder object.builder add r1z1-127.0.0.1:6000/d3 100
Device d2r1z1-127.0.0.1:6000R127.0.0.1:6000/d3_"" with 100.0 weight got id 2
```

Figure 5.2: Ring builder configuration for the mounted drives

Then we have used the rebalance command for ring build. These commands take all the partitions and assign them to devices. This also makes sure each drive is subscribed according to its destined weight. Then for each account, container and object builder we need to assign specific ports so that the builders file gets the access. We have used necessary commands and for the bash command line of the script we set up the port. That means for account builder we have assigned port number 6002. Similarly, the port 6001 and 6000 are assigned for container builder and object builder. Then we set up the proxy server. Again, using the nano command and the script is bind port: 8080. To set the permissions, we configured the swift file.

As we know Swift has suffix and prefix. So, to finish configuration we have used nano command for the configuration of swift. And in the script, we have input the values of suffix and prefix. Then we restarted our proxy server with the command sudo swift-init proxy restart. After that we need to do the authentication and authorization of the swift account. We have used memcached. The config file of the proxy-server got updated with the user and its necessary credentials. After doing all this we boot up the servers.

For Authentication, we have used the command attached. It will give an Auth storage token. This token will be then used for the verification process. In our case every time we started the system, we got a unique storage token.

Account verification has been done with the storage-token given from TempAuth. In our case the code is: curl -v -H 'X-Storage-Token: AUTH_tk36b797284aae44cb86cca942ad042a86'.

```

mohtasin@asif-18101096:~$ cd /etc/swift
bash: cd: /etc/swift: No such file or directory
mohtasin@asif-18101096:~$ cd /etc/swift
mohtasin@asif-18101096:~$ sudo swift-init account start
[sudo] password for mohtasin:
Starting account-server...(/etc/swift/account-server.conf)
mohtasin@asif-18101096:~$ sudo swift-init container start
Starting container-server...(/etc/swift/container-server.conf)
mohtasin@asif-18101096:~$ sudo swift-init object start
Starting object-server...(/etc/swift/object-server.conf)
mohtasin@asif-18101096:~$ sudo swift-init proxy restart
No proxy-server running
Starting proxy-server...(/etc/swift/proxy-server.conf)
mohtasin@asif-18101096:~$ sudo swift-init main status
proxy-server running (1727 - /etc/swift/proxy-server.conf)
container-server running (1693 - /etc/swift/container-server.conf)
account-server running (1673 - /etc/swift/account-server.conf)
object-server running (1709 - /etc/swift/object-server.conf)

```

Figure 5.3: Swift Starting

```

mohtasin@asif-18101096:~$ curl -v -H 'X-Auth-User: myaccount:me' -H 'X-Auth-Key: secretpassword' http://localhost:8080/auth/v1.0/
* Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET /auth/v1.0/ HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.58.0
> Accept: */*
> X-Auth-User: myaccount:me
> X-Auth-Key: secretpassword
>
< HTTP/1.1 200 OK
< X-Storage-Url: http://localhost:8080/v1/AUTH_myaccount
< X-Auth-Token-Expires: 85917
< X-Auth-Token: AUTH_tk36b797284aae44cb86cca942ad042a86
< Content-Type: text/html; charset=UTF-8
< X-Storage-Token: AUTH_tk36b797284aae44cb86cca942ad042a86
< Content-Length: 0
< X-Trans-Id: tx5ca140b9260d4164be60a-00611d75ae
< X-Openstack-Request-Id: tx5ca140b9260d4164be60a-00611d75ae
< Date: Wed, 18 Aug 2021 21:03:42 GMT
<
* Connection #0 to host localhost left intact

```

Figure 5.4: Account Authentication

Container creation is the third step in the Swift account activation process. By doing so we can easily see if the built container is there or not. Figure 5.5 is giving the confirmation of the creation of the container.

```

mohtasin@asif-18101096:~$ curl -v -H 'X-Storage-Token: AUTH_tk36b797284aae44cb86cca942ad042a86' -X PUT http://127.0.0.1:8080/v1/AUTH_myaccount/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_myaccount/mycontainer HTTP/1.1
> Host: 127.0.0.1:8080
> User-Agent: curl/7.58.0
> Accept: */*
> X-Storage-Token: AUTH_tk36b797284aae44cb86cca942ad042a86
>
< HTTP/1.1 201 Created
< Content-Length: 0
< Content-Type: text/html; charset=UTF-8
< X-Trans-Id: tx91689f2f016b41ada1636-00611d7642
< X-Openstack-Request-Id: tx91689f2f016b41ada1636-00611d7642
< Date: Wed, 18 Aug 2021 21:06:10 GMT
<
* Connection #0 to host 127.0.0.1 left intact

```

Figure 5.5: Container Creation Confirmation

5.1.2 Testing Tool Setup

Load testing is mainly to see whether the system under test is able to handle the required number of concurrent user accesses on a web server without any failure. Among all the load testing tools, JMeter is the best solution since it can test both static and dynamic resources. JMeter aids in the detection of concurrent users on a website and gives a variety of graphical performance statistics. JMeter is an open source tool so everyone can download it for free. Another reason to use JMeter is that it is easy to use. There are no prerequisites to use this tool which means if one

person wants to use JMeter he does not need any skill or domain knowledge. The downloading process is also so simple. JMeter is run on a pc where java is installed. To check if it is installed we have to open the command prompt and check for the java version. If we have java installed then all set to download jmeter. To install JMeter we have to go to the official website which is jmeter.apache.org. From this website we have to download any one of the binaries. Then from the downloaded zip file we have to unzip it. Then we can easily access the JMeter without any installation process like other traditional installation systems. From the bin folder of the unzipped file we have to find JMeter.bat to start JMeter in GUI mode. After that when we open that file we can see the GUI of Apache JMeter.

5.2 Experimental Result

We have designed 8 test cases. Each test case is of unique characteristics. According to the characteristics, they give different types of results in JMeter. The result briefly depends on the unique scenarios of the test cases. After doing the load tests with these test cases on JMeter we have got graphs and result tables. We have described all the test cases and their corresponding graphs and result tables down below:

5.2.1 TCO

For TCO we have selected a large file and kept the URL the same, we ran the test without having any load on our server.

Test#	Start Time	Thread Name	Label	Sample Time	Status	Latency
1	00:07:18.198	Load Test 1-1	HTTP Request	19	Ok	14
2	00:07:18.307	Load Test 1-2	HTTP Request	8	Ok	3
3	00:07:18.409	Load Test 1-3	HTTP Request	9	Ok	5
4	00:07:18.515	Load Test 1-4	HTTP Request	4	Ok	2
5	00:07:18.607	Load Test 1-5	HTTP Request	4	Ok	2
6	00:07:18.705	Load Test 1-6	HTTP Request	5	Ok	3
7	00:07:18.806	Load Test 1-7	HTTP Request	6	Ok	4
8	00:07:18.906	Load Test 1-8	HTTP Request	7	Ok	4
9	00:07:19.006	Load Test 1-9	HTTP Request	4	Ok	2
10	00:07:19.105	Load Test 1-10	HTTP Request	4	Ok	2

Table 5.1: Result Table of TCO

From the above table, we can see connectivity status is okay so the connection has been established properly. The sample time started from 19 milliseconds as we have tested this time with no load and small numbers. The average we can see from the table is around 4-6 milliseconds. The latency was 14 at the beginning which returned to normal by time.

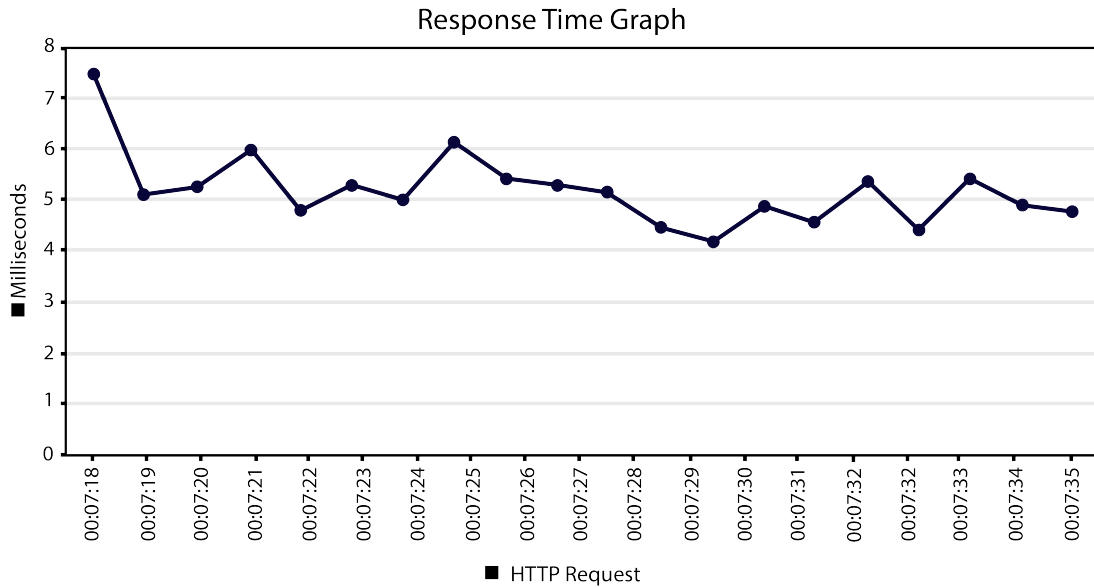


Figure 5.6: Response Time Graph of TC0

From the response time graph, we found that the graph remains between 4 milliseconds to 6 milliseconds. As there was no load in action this time. For TC0 the average response time is 5.2 milliseconds.

5.2.2 TC1

For TC1 we have selected a large file and kept the URL the same, we ran the test with having any load on our server.

Test#	Start Time	Thread Name	Label	Sample Time	Status	Latency
1	23:44:01.096	Load Test 1-1	HTTP Request	46	Ok	40
2	23:44:01.096	Load Test 1-2	HTTP Request	46	Ok	40
3	23:44:01.169	Load Test 1-3	HTTP Request	6	Ok	3
4	23:44:01.271	Load Test 1-4	HTTP Request	6	Ok	3
5	23:44:01.370	Load Test 1-5	HTTP Request	7	Ok	3
6	23:44:01.473	Load Test 1-6	HTTP Request	6	Ok	3
7	23:44:01.573	Load Test 1-7	HTTP Request	8	Ok	4
8	23:44:01.674	Load Test 1-8	HTTP Request	5	Ok	3
9	23:44:01.770	Load Test 1-9	HTTP Request	5	Ok	3
10	23:44:01.873	Load Test 1-10	HTTP Request	6	Ok	3

Table 5.2: Result Table of TC1

In this table, we can see connectivity status is okay so the connection has been established properly. The sample time started from 46 milliseconds as we have tested this time with load and then it was reduced to lower numbers. The average we can see from the table is around 5-9 milliseconds. The latency was very high because of the load. Later the latency came to a normal level.

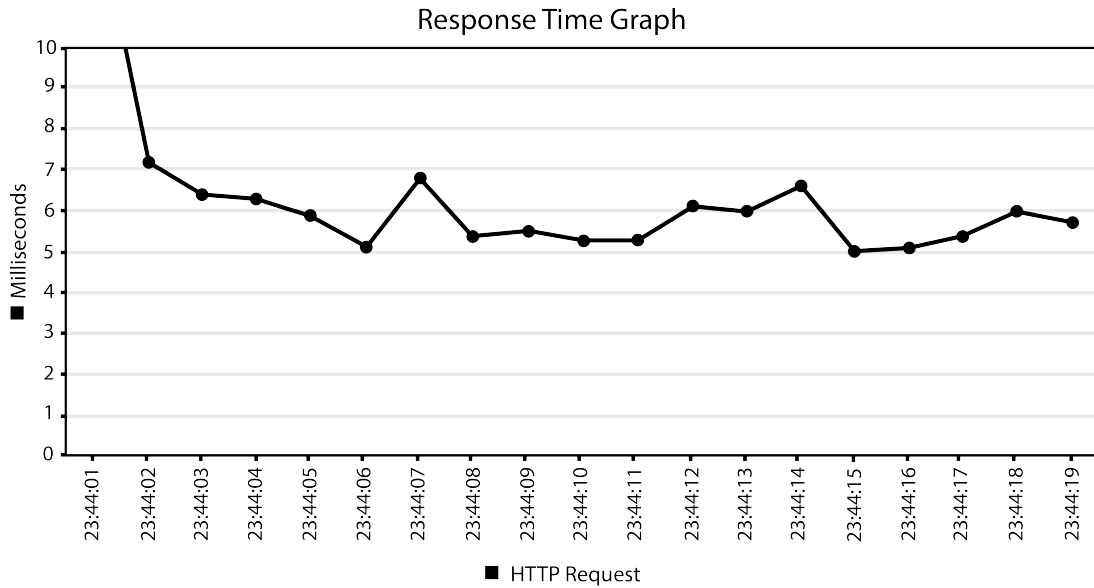


Figure 5.7: Response Time Graph of TC1

From the response time graph, we found that the graph remains between 5 milliseconds to 7 milliseconds. As there was load on the server initially the response time was higher. The average response time for TC1 is 6.47 milliseconds.

5.2.3 TC2

For TC2 we have selected a small file and kept the URL the same, we ran the test without having any load on our server.

Test#	Start Time	Thread Name	Label	Sample Time	Status	Latency
1	23:55:20.121	Load Test 1-1	HTTP Request	17	Ok	10
2	23:55:20.237	Load Test 1-2	HTTP Request	9	Ok	5
3	23:55:20.333	Load Test 1-3	HTTP Request	5	Ok	2
4	23:55:20.432	Load Test 1-4	HTTP Request	6	Ok	3
5	23:55:20.520	Load Test 1-5	HTTP Request	7	Ok	3
6	23:55:20.631	Load Test 1-6	HTTP Request	6	Ok	3
7	23:55:20.731	Load Test 1-7	HTTP Request	5	Ok	2
8	23:55:20.831	Load Test 1-8	HTTP Request	6	Ok	4
9	23:55:20.931	Load Test 1-9	HTTP Request	5	Ok	2
10	23:55:21.031	Load Test 1-10	HTTP Request	4	Ok	2

Table 5.3: Result Table of TC2

In this table, we can see connectivity status is okay so the load for the system is connected properly. The sample time started from 17 milliseconds and then it was reduced to 4-5 milliseconds. The latency was on the higher side as well at first and then it was decreased.

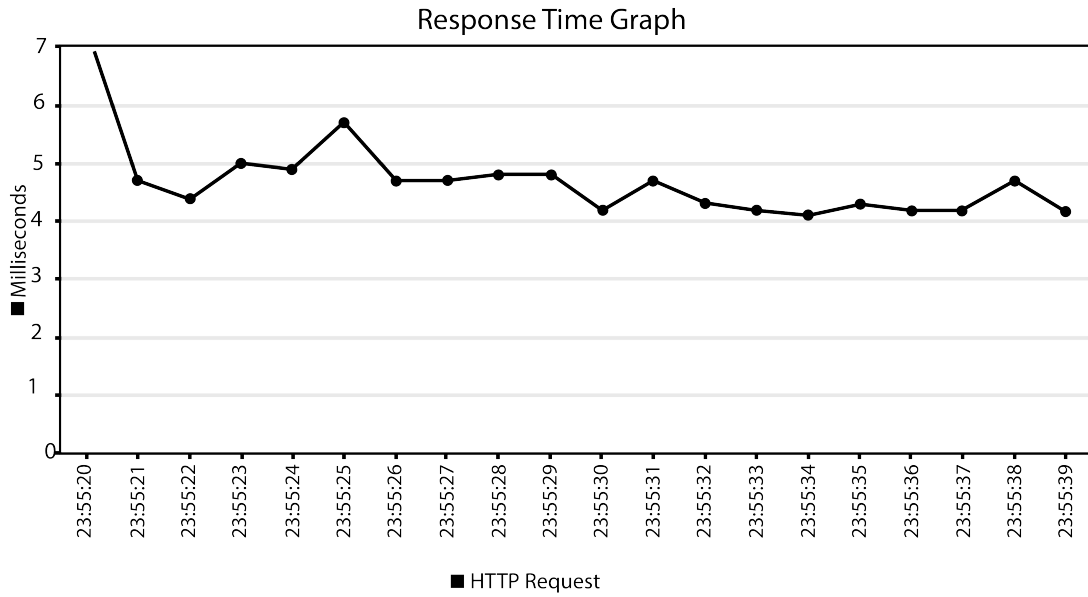


Figure 5.8: Response Time Graph of TC2

From the response time graph, we found that the graph remains between 4 milliseconds to 6 milliseconds. As there was no load on the server. The average response time of TC2 is 4.9 milliseconds.

5.2.4 TC3

For TC3 we have selected a small file and kept the URL the same, we ran the test with load on our server.

Test#	Start Time	Thread Name	Label	Sample Time	Status	Latency
1	00:03:05.751	Load Test 1-1	HTTP Request	37	Ok	32
2	00:03:05.750	Load Test 1-2	HTTP Request	38	Ok	32
3	00:03:05.823	Load Test 1-3	HTTP Request	16	Ok	7
4	00:03:05.923	Load Test 1-4	HTTP Request	17	Ok	7
5	00:03:06.020	Load Test 1-5	HTTP Request	8	Ok	4
6	00:03:06.121	Load Test 1-6	HTTP Request	9	Ok	4
7	00:03:06.223	Load Test 1-7	HTTP Request	5	Ok	2
8	00:03:06.324	Load Test 1-8	HTTP Request	5	Ok	3
9	00:03:06.422	Load Test 1-9	HTTP Request	6	Ok	3
10	00:03:06.522	Load Test 1-10	HTTP Request	9	Ok	4

Table 5.4: Result Table of TC3

The above table shows that connectivity status is okay. This test case was generated for small files with load so starting response time was 37 milliseconds. The average we can see from the table is around 5-7 milliseconds. The latency was higher compared to the previous test cases at first at 32.

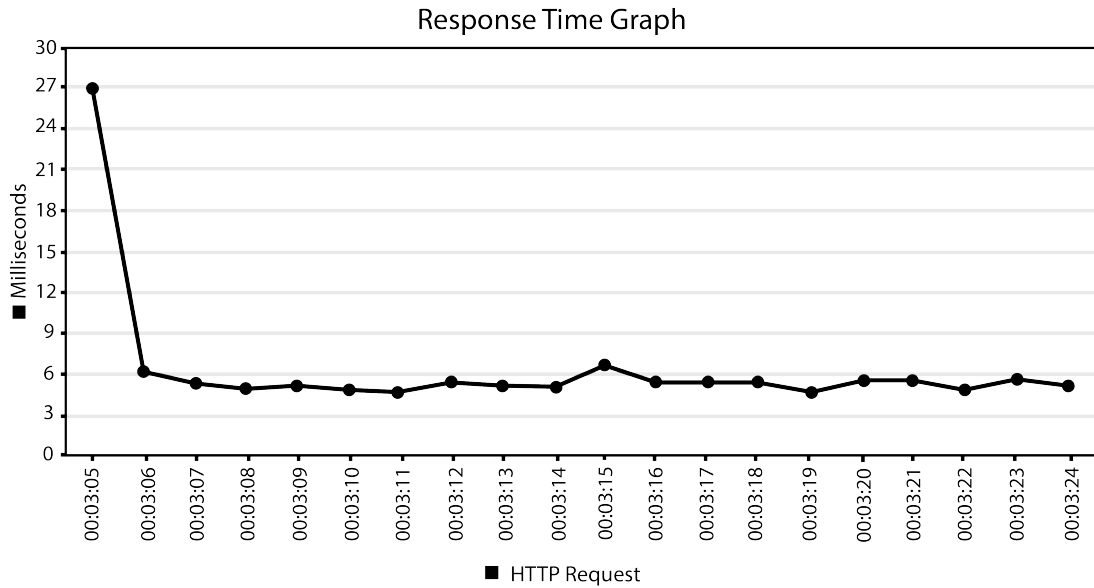


Figure 5.9: Response Time Graph of TC3

In this response time graph, the graph remains between 4 milliseconds to 7 milliseconds. As there was with load in so the response time was very high at first This affected the average response time as well which is 6.45 milliseconds.

5.2.5 TC4

For TC4 we have selected a large file and chose a different URL, we ran the test without having any load on our server.

Test#	Start Time	Thread Name	Label	Sample Time	Status	Latency
1	00:30:30.965	Load Test 1-1	HTTP Request	12	Ok	4
2	00:30:31.066	Load Test 1-2	HTTP Request	7	Ok	3
3	00:30:31.165	Load Test 1-3	HTTP Request	4	Ok	2
4	00:30:31.264	Load Test 1-4	HTTP Request	6	Ok	3
5	00:30:31.365	Load Test 1-5	HTTP Request	4	Ok	2
6	00:30:31.466	Load Test 1-6	HTTP Request	5	Ok	2
7	00:30:31.565	Load Test 1-7	HTTP Request	4	Ok	2
8	00:30:31.664	Load Test 1-8	HTTP Request	5	Ok	2
9	00:30:31.766	Load Test 1-9	HTTP Request	4	Ok	2
10	00:30:31.866	Load Test 1-10	HTTP Request	5	Ok	2

Table 5.5: Result Table of TC4

The connectivity status is okay as shown in the above table. This test case was generated for large files with no load but with a different URL. The starting response time was 12 milliseconds which was very low compared to the above cases. The average is around 4-6 milliseconds. The latency was normal all the time during the load test.

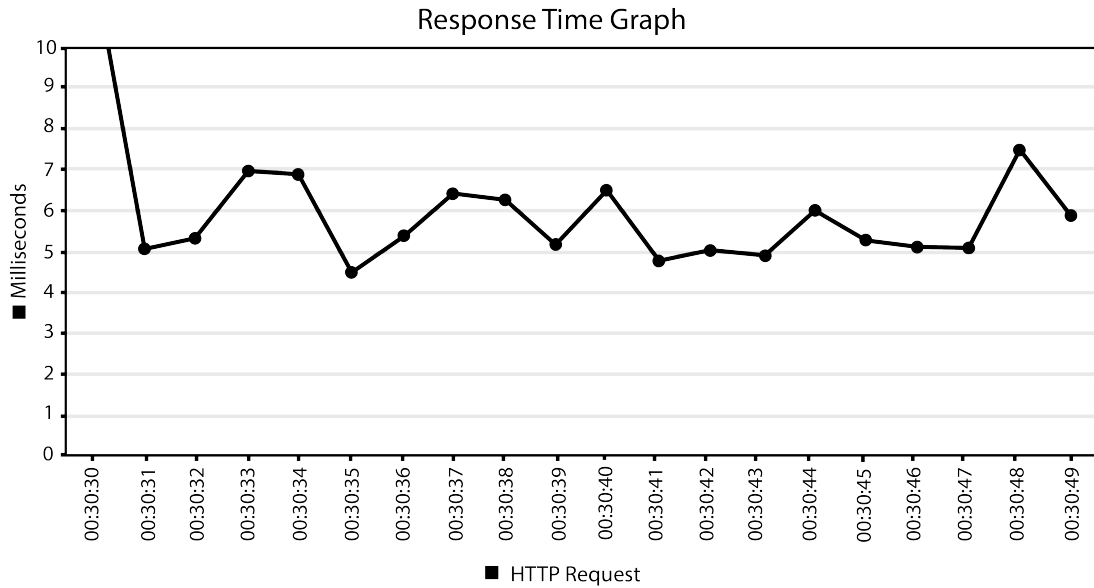


Figure 5.10: Response Time Graph of TC4

We can see that the graph remains between 4 milliseconds to 8 milliseconds most of the time. As there was no load with different URL so on a lower range compared to the previous cases. at first the average response time is 5.96 milliseconds.

5.2.6 TC5

For TC5 we have selected a large file with a different URL. This time we ran the test with load on our server

Test#	Start Time	Thread Name	Label	Sample Time	Status	Latency
1	00:26:18.379	Load Test 1-1	HTTP Request	47	Ok	39
2	00:26:18.379	Load Test 1-2	HTTP Request	47	Ok	39
3	00:26:18.437	Load Test 1-3	HTTP Request	7	Ok	3
4	00:26:18.535	Load Test 1-4	HTTP Request	7	Ok	4
5	00:26:18.635	Load Test 1-5	HTTP Request	7	Ok	3
6	00:26:18.734	Load Test 1-6	HTTP Request	8	Ok	3
7	00:26:18.835	Load Test 1-7	HTTP Request	7	Ok	3
8	00:26:18.940	Load Test 1-8	HTTP Request	5	Ok	3
9	00:26:19.039	Load Test 1-9	HTTP Request	4	Ok	3
10	00:26:19.138	Load Test 1-10	HTTP Request	6	Ok	2

Table 5.6: Result Table of TC5

The connectivity status is okay as shown in the table attached above. This test case was generated for large files with load and with a different URL. The starting response time was 47 milliseconds this time for that reason. The average is around 6-10 milliseconds. The latency was high at first because of the load and it returned to normal with time.

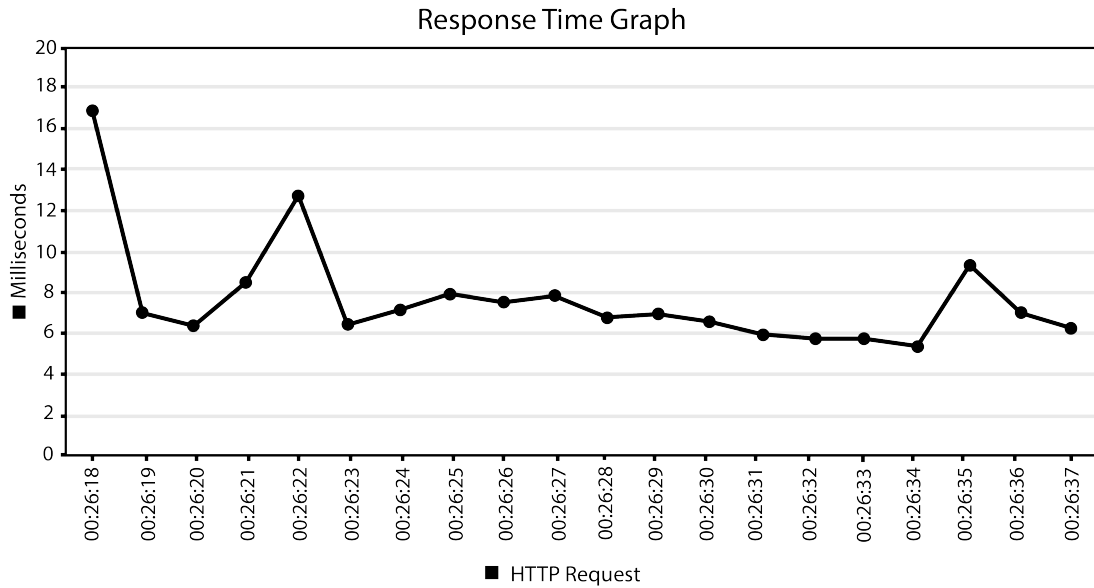


Figure 5.11: Response Time Graph of TC5

From the response time graph, we found that the graph remains between 5 milliseconds to 10 milliseconds. As there was load in action this time. For TC5 the average response time is 7.5 milliseconds.

5.2.7 TC6

For TC6 we have a small file with a different URL. The test was done without any load on the server.

Test#	Start Time	Thread Name	Label	Sample Time	Status	Latency
1	00:53:36.051	Load Test 1-1	HTTP Request	8	Ok	4
2	00:53:36.157	Load Test 1-2	HTTP Request	12	Ok	5
3	00:53:36.263	Load Test 1-3	HTTP Request	5	Ok	2
4	00:53:36.357	Load Test 1-4	HTTP Request	7	Ok	3
5	00:53:36.451	Load Test 1-5	HTTP Request	7	Ok	3
6	00:53:36.551	Load Test 1-6	HTTP Request	4	Ok	2
7	00:53:36.651	Load Test 1-7	HTTP Request	4	Ok	2
8	00:53:36.751	Load Test 1-8	HTTP Request	4	Ok	2
9	00:53:36.851	Load Test 1-9	HTTP Request	4	Ok	2
10	00:53:36.950	Load Test 1-10	HTTP Request	5	Ok	2

Table 5.7: Result Table of TC6

In this table, we can see connectivity status is okay so the connection has been established properly. The sample time started from 8 milliseconds which is the lowest compared to the prior cases. In this case, we have tested this time with no load for small files with different URL. The average we can see from the table is around 4-6 milliseconds.

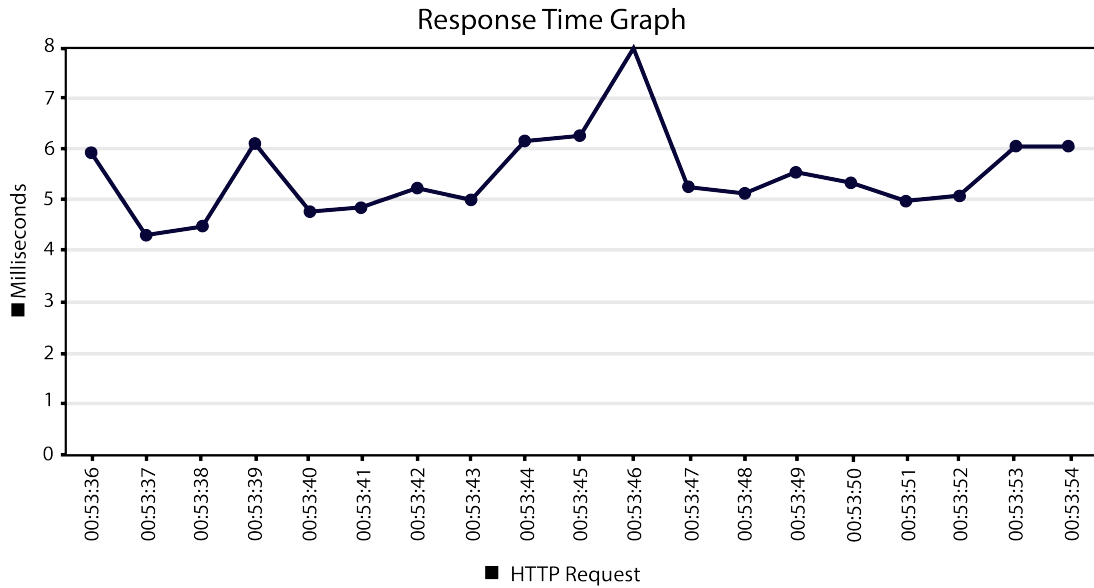


Figure 5.12: Response Time Graph of TC6

In this response time graph, the graph remains between 4 milliseconds to 6 milliseconds. As there was no load, the response time was low at first. The average response time as well which is 5.24 milliseconds.

5.2.8 TC7

For TC7 we have a small file with a different URL. This time we ran the test with load on our server.

Test#	Start Time	Thread Name	Label	Sample Time	Status	Latency
1	00:48:25.010	Load Test 1-1	HTTP Request	48	Ok	65
2	00:48:25.010	Load Test 1-2	HTTP Request	42	Ok	65
3	00:48:25.010	Load Test 1-3	HTTP Request	36	Ok	65
4	00:48:25.100	Load Test 1-4	HTTP Request	9	Ok	4
5	00:48:25.206	Load Test 1-5	HTTP Request	7	Ok	3
6	00:48:25.305	Load Test 1-6	HTTP Request	7	Ok	4
7	00:48:25.406	Load Test 1-7	HTTP Request	7	Ok	3
8	00:48:25.503	Load Test 1-8	HTTP Request	10	Ok	6
9	00:48:25.607	Load Test 1-9	HTTP Request	7	Ok	3
10	00:48:25.706	Load Test 1-10	HTTP Request	9	Ok	4

Table 5.8: Result Table of TC7

From the above table, we can see connectivity status is okay so the connection has been established correctly. The sample time started from 77 milliseconds as we have tested this time with small files with different URL with load. The average we can see from the table is around 5-7 milliseconds. The latency was 65 at the beginning which returned to normal after the first few seconds.

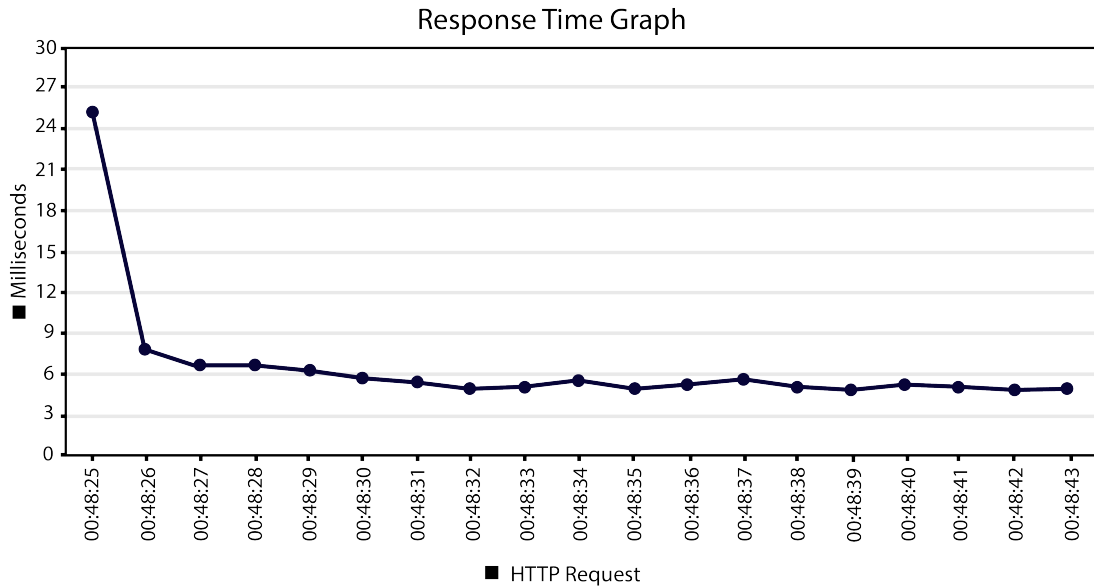


Figure 5.13: Response Time Graph of TC7

From the response time graph, we found that the graph remains between 4 milliseconds to 7 milliseconds. As there was load on the server initially the response time was higher. The average response time from TC7 is 6.495 milliseconds.

5.3 Experimental Findings

We have tested the all the test cases 4 times each. After doing so, we have taken the highest average response time. As the highest average time is considered to be the worst case. We have followed this procedure for all the 8 cases. The test cases are influenced by CPU usage and broadband speed. As a result, we can have different results in different environments for the same cases.

When the results are analyzed, it is evident that the average response time is always faster when the server is not under load. In our intended scenario TC0, a large file with the same URL but no load has an average response time of 5.20 milliseconds, however in TC1, everything remains the same except for the server load. Due to the increased strain on the system, the response time increased to 6.47 milliseconds. Again, we intended to conduct the same type of test, but with a smaller file rather than a large one. Thus, for TC2, we have a small file with the same URL but no load on the server. This resulted in an average response time of 4.90 milliseconds. On the other side, TC3, which uses nearly identical criteria and includes server load, returned an average response time of 6.45 milliseconds. Additionally, we attempted the same four test instances listed above but with a different URL for the remaining four cases. In TC4, we used a huge file with a different URL and obtained an average response time of 5.96 milliseconds without load. Likewise, TC5 operates under the identical conditions as TC4 except that it has a server load, which results in an average response time of 7.50 milliseconds. Similarly, we used a short file, a different URL and no load for TC6, which resulted in an average response time of

5.24 milliseconds. For our final test case, TC7, we have everything identical to TC6, except that the server is loaded. This test yielded an average response time of 6.49 milliseconds.

Test Case ID	Average Response Time (milliseconds)
TC0	5.2
TC1	6.47
TC2	4.9
TC3	6.45
TC4	5.96
TC5	7.5
TC6	5.24
TC7	6.495

Table 5.9: Average Response Time for Test Cases

From the above test it is clearly seen that while the server has load it take much time to response. Moreover, the response time also vary because of the file size. Furthermore, we discovered that when the URL is changed, the response time increases. The highest average response time was 6.47 milliseconds with the same URL and a loaded server in TC1. Whereas, when the URL was different and the server was loaded in TC5, the highest average response time was 7.5 milliseconds. On the other side, having small file size, same URL and no load in TC2, the highest average response time was 4.9 milliseconds. Meanwhile, the highest average response time was 5.24 milliseconds while using a different URL, small file, and no load in test case TC6. To summarize, based on the findings of the aforementioned test cases, we determined that the test case TC2 with the shortest average response time of 4.9milliseconds. This test used a small file with the same URL but no load. On the other hand, the highest average time is for test case TC5, which involved a large file, a different URL and a load is 7.5 milliseconds.

Chapter 6

Future Work

In our research we have tried to set up an OpenStack Swift server with the limited resources for a distributed system. But there is plenty of room for future work. Firstly, we can create a backend server to save all the caches. A cache server helps us to work really fast. As it remembers the most frequent website or servers we use and later use that knowledge for fast connectivity. Backend server will come in handy for storing all the required cache. For hosting, backend servers plays a vital role. The most important thing is security when it is used as a storage. For instance, if a piece of information is shared over the internet, it can use the help of having a backend server in between that will keep the data secure for the user from each other. And, sometimes passwords can be breached by using various technologies. So, the security system has to be hack proof. Backend server can ensure security. When there is no risk factor in a system it's easy to win the trust of the consumers. If we get enough financial backup for this we will make this available for https as well. This will make the system more secure and reliable. Also, it serves to help prevent attacks and prohibit data espionage. HTTPS sites give reasonable assurance that the site is not fraudulent and that content and data are transmitted securely. We know traditional cloud storage systems are very costly. We tried to make this as much cost efficient as possible than usual. With more efficient planning we can try to cut the cost to a more affordable range to attract more and more consumers.

Chapter 7

Conclusion

This paper presents a great addition to a shared-mode resource allocation for cloud-based load testing, which assures the cost-friendly aspect to use virtual machine resources in the cloud. Initially, OpenStack Swift setup was done as our cloud server. Furthermore, we have designed some test cases in order to run the testing process. The test cases are arranged based on metrics such as load, protocol, file size and URL. Then we used Apache JMeter as our load testing tool in order to run the testing for our test cases. The best average response time was 4.9ms for TC2. It was received when the load test was done on a scenario of a small file, same URL and no load. The highest average response time was 7.5ms. This was for TC5 which consisted of a large file, different URL and load. Here we have enlarged our user number to find out the sealing capacity of load capacitance for the servers. The main purpose of our research is to observe how the system will act under different parameters using the testing tool and compare the outcome obtained from there.

Bibliography

- [1] E. Luke, "Defining and measuring scalability," in *Proceedings of Scalable Parallel Libraries Conference*, 1993, pp. 183–186. DOI: 10.1109/SPLC.1993.365568.
- [2] T. Shih, C.-M. Chung, Y.-H. Wang, Y.-F. Kuo, and W.-C. Lin, "Software testing and metrics for concurrent computation," in *Proceedings 1996 Asia-Pacific Software Engineering Conference*, 1996, pp. 336–344. DOI: 10.1109/APSEC.1996.566768.
- [3] T. Yamaura, "How to design practical test cases," *IEEE Software*, vol. 15, no. 6, pp. 30–36, 1998. DOI: 10.1109/52.730835.
- [4] D. Menasce, "Load testing of web sites," *IEEE Internet Computing*, vol. 6, no. 4, pp. 70–74, 2002. DOI: 10.1109/MIC.2002.1020328.
- [5] H. Sofian, R. M. Saidi, R. Yunos, and S. A. Ahmad, "Analyzing server response time using testing power web stress tool," in *2010 International Conference on Science and Social Research (CSSR 2010)*, 2010, pp. 1120–1125. DOI: 10.1109/CSSR.2010.5773700.
- [6] I. Ali and N. Meghanathan, "Virtual machines and networks - installation, performance, study, advantages and virtualization options," *CoRR*, vol. abs/1105.0061, Jan. 2011. DOI: 10.5121/ijnsa.2011.3101.
- [7] P. Nirpal and K. Kale, "A brief overview of software testing metrics," *International Journal on Computer Science and Engineering*, vol. 3, Jan. 2011.
- [8] V. Arutyunov, "Cloud computing: Its history of development, modern state, and future considerations," *Scientific and Technical Information Processing*, vol. 39, Jul. 2012. DOI: 10.3103/S0147688212030082.
- [9] M. Kamra and R. Manna, "Performance of cloud-based scalability and load with an automation testing tool in virtual world," in *2012 IEEE Eighth World Congress on Services*, 2012, pp. 57–64. DOI: 10.1109/SERVICES.2012.54.
- [10] M. S. Bali and S. Khurana, "Effect of latency on network and end user domains in cloud computing," in *2013 International Conference on Green Computing, Communication and Conservation of Energy (ICGCE)*, 2013, pp. 777–782. DOI: 10.1109/ICGCE.2013.6823539.
- [11] F. Douglass and O. Krieger, "Virtualization," *IEEE Internet Computing*, vol. 17, no. 2, pp. 6–9, 2013. DOI: 10.1109/MIC.2013.42.
- [12] A. Ezugwu, S. Buhari, and S. Junaidu, "Virtual machine allocation in cloud computing environment," *International Journal of Cloud Applications and Computing (IJCAC)*, vol. 3, pp. 47–60, Apr. 2013. DOI: 10.4018/ijcac.2013040105.

- [13] X. Lu, N. S. Islam, M. Wasi-Ur-Rahman, *et al.*, “High-performance design of hadoop rpc with rdma over infiniband,” in *2013 42nd International Conference on Parallel Processing*, 2013, pp. 641–650. DOI: 10.1109/ICPP.2013.78.
- [14] Z. Duan and C. Yizhen, “The implementation of cloud storage system based on openstack swift,” vol. 644-650, Sep. 2014. DOI: 10.4028/www.scientific.net/AMM.644-650.2981.
- [15] N. M. Ms and V. Suma, “A study on cloud computing testing tools,” Feb. 2014.
- [16] E. N. Narciso, M. E. Delamaro, and F. D. L. D. S. Nunes, “Test case selection: A systematic literature review,” in *International Journal of Software Engineering and Knowledge Engineering*, vol. 24, 2014, pp. 653–676. DOI: 10.1142/S0218194014500259.
- [17] M. Arslan, U. Qamar, S. Hassan, and S. Ayub, “Automatic performance analysis of cloud based load testing of web-application amp; its comparison with traditional load testing,” in *2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, 2015, pp. 140–144. DOI: 10.1109/ICSESS.2015.7339023.
- [18] S. Bairagi and A. Bang, “Cloud computing: History, architecture, security issues,” Mar. 2015.
- [19] S. Kiran, A. Mohapatra, and R. Swamy, “Experiences in performance testing of web applications with unified authentication platform using jmeter,” in *2015 International Symposium on Technology Management and Emerging Technologies (ISTMET)*, 2015, pp. 74–78. DOI: 10.1109/ISTMET.2015.7359004.
- [20] X. Lu, D. Shankar, S. Gugnani, H. Subramoni, and D. K. Panda, “Impact of hpc cloud networking technologies on accelerating hadoop rpc and hbase,” in *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2016, pp. 310–317. DOI: 10.1109/CloudCom.2016.0057.
- [21] S. Gugnani, X. Lu, and D. K. Panda, “Swift-x: Accelerating openstack swift with rdma for building an efficient hpc cloud,” in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2017, pp. 238–247. DOI: 10.1109/CCGRID.2017.103.
- [22] M. A. Putri, H. N. Hadi, and F. Ramdani, “Performance testing analysis on web application: Study case student admission web system,” in *2017 International Conference on Sustainable Information Engineering and Technology (SIET)*, 2017, pp. 1–5. DOI: 10.1109/SIET.2017.8304099.
- [23] A. S. Rumale and D. N. Chaudhari, “Cloud computing: Software as a service,” in *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, 2017, pp. 1–6. DOI: 10.1109/ICECCT.2017.8117817.
- [24] M. Ruan, T. C. Thierry, E. Zhai, *et al.*, “On the synchronization bottleneck of openstack swift-like cloud storage systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, pp. 1–1, Feb. 2018. DOI: 10.1109/TPDS.2018.2810179.
- [25] T. Kamalakannan, K. Senthil, C. Shanthi, and D. Radhakrishnan, “Study on cloud storage and its issues in cloud computing,” Jun. 2019.

- [26] H. Li, X. Li, H. Wang, J. Zhang, and Z. Jiang, “Research on cloud performance testing model,” in *2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE)*, 2019, pp. 179–183. DOI: 10.1109/HASE.2019.00035.
- [27] A. Rashid and A. Chaturvedi, “Virtualization and its role in cloud computing environment,” *INTERNATIONAL JOURNAL OF COMPUTER SCIENCES AND ENGINEERING*, vol. Vol.-7, pp. 1131–1136, Apr. 2019. DOI: 10.26438/ijcse/v7i4.11311136.
- [28] E. Setiawan, A. Setiyadi, and R. Wahdiniwati, “Quality analysis of mobile web server,” *IOP Conference Series: Materials Science and Engineering*, vol. 662, p. 022043, Nov. 2019. DOI: 10.1088/1757-899X/662/2/022043.
- [29] M. Gull, S. Bai, and T. Bak, “A review on design of upper limb exoskeletons,” *Robotics*, vol. 9, p. 16, Mar. 2020. DOI: 10.3390/robotics9010016.
- [30] R. Kaur and S. Chopra, “Virtualization in cloud computing : A review,” *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, pp. 01–05, Jul. 2020. DOI: 10.32628/CSEIT20641.
- [31] A. Feldmann, O. Gasser, F. Lichtblau, *et al.*, “Implications of the covid-19 pandemic on the internet traffic,” in *Broadband Coverage in Germany; 15th ITG-Symposium*, 2021, pp. 1–5.