EFFICIENT LIVE FILESYSTEM MIGRATION

JIAYI FU

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

May 2017

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

EFFICIENT LIVE FILESYSTEM MIGRATION

JIAYI FU

Research Consultant: Prof. George Samaras

Committee Member: Dr Andreas Pamporis

Μάιος 2017

ABSTRACT

In this thesis, we are going to provide an implementation of a Migration mechanism in a Hyper-Responsive system, naming this mechanism Efficient Live Filesystem Migration (ELFM). Furthermore, the ELFM have to adapt all the main goals of any Distributed Filesystem which are replication, performance and consistency.

The problem with the migration nowadays is usually the servers are sending a huge amount of data in between them and thus creating network traffic extending the waiting time of the users. With that in mind, with the implementations of OverlayFS and the “Diff and Patch” methodology in ELFM we managed to reduce the size of package which server exchange by sending only the necessary data while reducing the waiting time of the user.

The experimental evaluation was tested on the Common Open Research Emulator (CORE) on different scenarios from 100MB initial filesystem up to 2500 MB, using random workload generator to change the contain of the filesystem by 10-40% and migrate the filesystem between different machines with ELFM.

With the current version, the results showed that the implementation works perfectly with small number of version differences and under forty percentage changes of filesystem before the migration.

TABLE OF CONTENTS

Chapter 1

Introduction

1.1 Motivation

1.2 Thesis Structure

A Distributed filesystem (DFS) or a network filesystem in computing is a set of client and server service. Additionally, this service allows users to have access to their own data from multiple hosts via a computer network. What's more, DFS has a variable of benefits regarding the resources management and accessibility. In other words, the resources management and accessibility exist in order to provide users access to all resources through a single point. Moreover, fault tolerance means that shares can be replicated. That is to say, if a server from one location goes down then the resources will still be available to users. In other words, DFS allows administrators to distribute shared folders and workload across several servers for more efficient network and server resources use [13]. All things considered, the workload management seems to be better than any other type of filesystem.

1.1 Motivation

With the progressive development of software and technology, users nowadays demand applications that are more Hyper-Responsive. More specifically, a user application requires a low-latency access to real-time backend services. For instance, in online gaming and augmented reality the client side applications need to be more responsive to the users. Significantly, this is the case in order to minimize the latency access to real-time backend services [14].

Augmented reality is one of the most viral topic nowadays, most of the users in this topic are mobile users. To put it another way, a mobile user is the kind of user that changes his location constantly and exchanges messages and data with the server continuously. Therefore, the result is the creating of a big traffic to the network and massive amount of processes to the server.

In order for the system to avoid all that traffic and processes it is reasonable to share them to a number of distributed filesystem servers. Additionally, users will change their connection with the most efficient server compared with all the other servers by speed and ping [14]. Importantly, those servers must have the same data when a user changes his connection from one to another so it is necessary to migrate the filesystem. Above all, the purpose of this thesis is to make the migration phase more efficient. Hence, this particular thesis aims to reduce the time of migration and to scale down the size of data which the two servers are exchanging.

1.2 Thesis Structure

This particular thesis will follow the structure below:

Chapter 2: Basic information about different system and tools that the ELFM system used.

Chapter 3: Related Studies about Filesystem Migration.

Chapter 4: Basic information about how the ELFM was developed base on this thesis.

Chapter 5: Description of the scenario which was used, in order to take the results.

Chapter 6: Conclusion with the result sets and possible future work.

CHAPTER 2

Background

The second chapter presents the basic concepts and structures of Linux Filesystem. More specifically, the chapter analyses how the OverlayFS functions. Lastly, the current chapter deals with Tar computer software utility as well as the sockets.

2.1 Linux Filesystem

Filesystem is one of the common features of Operating system and it can control how the data is stored and retrieved. The main function of filesystem is to separate a big amount of data, placed in a storage like hardware into smaller pieces. Additionally, the filesystem gives a name to the smaller pieces and it describes where the smaller pieces begin and stop in the storage. Notably, it also provides valid information's that are identified by the user.

There is a simple description of the UNIX system which is also applicable to Linux. Most specifically "on a UNIX system, everything is a file; if something is not a file it is a process" [1]. The above statement describes how the systems made things easier and simpler since "there are special files that are more than just files (named pipes and sockets, for instance)" [1].

The type of files included in the Linux system are: Directories, Special files, Links, Sockets, Named pipes. Furthermore, Directories are files that contain lists with the names of other files. Special files are used for input and output of the system. In addition, Links files include the location of the original file which is located in another place in the system's file tree. Sockets is "a special file type, similar to TCP/IP sockets, providing inter-process networking protected by the file system's access control" [1]. Last but not least, the last type of files in the Linux system is Pipes which are similar with Sockets. Most specifically, Pipes and Sockets are similar except from the fact that Pipes deals with communication of local process instead of network communication.

Symbol	Meaning
-	Regular file
d	Directory
l	Link
c	Special file
s	Socket
p	Named pipe
b	Block device

Figure 1. Using ls -l in terminal of Linux to display type of file with the first character of output line

2.2 OverlayFS

OverlayFS is a filesystem service for Linux, it was merged into the Linux kernel mainline in kernel version 3.18 and after [2].

An overlay filesystem combines two types of filesystem, an ‘upper’ and ‘lower’

-The ‘lower’ filesystem usually contains read-only files or directories and it can be any filesystem supported by Linux which does not need to be writable.

-The ‘upper’ filesystem frequently contains writable files or directories. In addition, any changes in the overlay (merged) filesystem automatically changes or creates the content of the ‘upper’ filesystem.

-The overlay (merged) filesystem usually is the place which users can have access to read and write the contents included. Furthermore, it contains all the files and directories from ‘lower’ and ‘upper’ filesystem. Hence, when an object appears in both ‘upper’ and ‘lower’ filesystem with the same path and name, then the object in the ‘lower’ filesystem is hidden. However, the visible object is the one located in the ‘upper’ filesystem.

At the mount time the two directories are given as mount options which are called “lowerdir” and “upperdir”. Moreover, these particular two directories are combined into one merged directory. Additionally, a ‘workdir’ which needs to be an empty directory on the same filesystem as upperdir.

mount -t overlay -o lowerdir=”/lower”,upperdir=”/upper”,workdir=”/work” overlay “/merged”

In order to support the deletion of an object without changing the lower filesystem, an overlay filesystem keeps a record in the upper filesystem. Furthermore, that object has been removed by creating a character device (special file) with 0/0 device number with the same name of that object, which is called whiteout. Also, when a whiteout is detected in the upper level of a merged directory, any matching name in the lower is ignored and that file is hidden in the overlay filesystem [3].

2.3 Tar

Tar or tarball is a computer software utility for grouping and packaging multiple files into one archive file. Notably, its purpose is for distribution or backup. The tar archive format collects any number of files, directories, and other file system objects into a single stream of bytes [4].

Due to some historical reasons, there are several formats of tar archives. More specifically, all of them are based on the same principles, but have some subtle differences that often make them incompatible with each other [5].Therefore, every format creates and handles their archives in its own way.

Format	UID	File Size	File Name	Devn
gnu	1.8e19	Unlimited	Unlimited	63
oldgnu	1.8e19	Unlimited	Unlimited	63
v7	2097151	8GB	99	n/a
ustar	2097151	8GB	256	21
posix	Unlimited	Unlimited	Unlimited	Unlimited

Figure 2. Summary of limitations of each format [5]

2.4 Sockets

Sockets in a computer network is a mechanism for end-point communication between different machines across a network or the internet. To a programmer, a socket looks and behaves much like a low-level file descriptor. This is precisely because commands such as read() and write() work with sockets in the same way they do with files and pipes[6].

A Unix like socket is applicate in Linux and is usually used in a client-server application framework which uses protocols like FTP, SMTP and POP. Moreover, with those specific protocols sockets create a connection between the client and the server. Thereafter, they exchange messages and data between the two which are depended from the program.

The types of sockets which are available to the users are called Stream Sockets, Datagram Sockets, Raw Sockets and Sequenced Packet Sockets. More specifically, the first two are the ones most commonly used and the last two are rarely used [6].

Stream Sockets use TCP(Transmission Control Protocol) for data transmission and make the delivery guaranteed a success without the sender having received an error indicator. Thus, the messages and data that the receivers will gather will be in the same order the sender forwarded them.

Datagram Sockets use UDP(User Datagram Protocol) for data transmission so the delivery is not guaranteed.

Raw Sockets are not intended for the general user, they have been provided mainly for developing new communication protocols or for gaining access to some of the more cryptic facilities of an existing protocol [6].

Sequenced Packet Sockets are similar to a stream socket with the exception that record boundaries are preserved. Moreover, it allows the user to manipulate the Sequence Packet Protocol (SPP) or the Internet Datagram Protocol (IDP) headers on a packet or a group of packets [6].

2.5 Ubuntu Operation System

Ubuntu is a Debian-based Linux operating system for a personal computer. It also provides specific repositories of free software and it recommends non free ones. Furthermore, non free software are under the Ubuntu name in some of their distribution channels. What's more, Ubuntu offers the option to not only install free packages but also non free packages. In addition, the version of Linux and the kernel are included in Ubuntu which contains firmware blobs [7].

With Ubuntu default installation comes a wide range of software which include LibreOffice, Firefox, Thunderbird and Transmission etc. Another key to remember, many other additional software can be downloaded and installed from the build-in Ubuntu Software Center as well as the APT-based package management tools.

In the security sector, Ubuntu by default limits the user by running programs with low privileges. Having said that, this case depicts precisely that the users cannot corrupt the operating system or have access to other user's files. To have an administrator permission users can use the sudo tool which can give temporary privileges to them. Additionally, in Ubuntu operating system by default most of network ports are closed to prevent hacking. Importantly, there is a build-in firewall to allow users who install network servers to gain control access.

Efficient Live Filesystem Migration (ELFM) system uses Ubuntu 16.04 LTS Xenial Xerus because it was the last stable release. Another key thing to remember, the Software Center offers a wide range of softwares as well as a simple user friendly environment with a very responsive developer community. All things considered, Ubuntu is the best option for an amateur Linux developer to start with.

2.6 Python 2.7

Python is a widely used high-level programming language for general-purpose programming and it was first released in 1991. Significantly, its design philosophy can emphasize code's readability by using white space to determine code blocks. That is the case so it can avoid using curly braces or keywords like other programming languages. Additionally, Python's syntax allows programmers to express more concepts in fewer lines of code.

Python features a dynamic type system and an automatic memory management which supports multiple programming paradigms. That is to say, Python features include object-oriented method, imperative, a functional programming and procedural styles. Hence, Python has a large and comprehensive standard library.

Python interpreters are available for the most operating systems and it's code can run on a wide variety of systems.

The core design philosophy of the language includes aphorisms such as: "Beautiful is better than ugly, explicit is better than implicit, simple is better than complex, complex is better than complicated, flat is better than nested, sparse is better than dense and readability counts" [12].

The language support multi-paradigm programming includes object-oriented and structured programming as well as many features like functional programming and aspect-oriented programming.

The amount of Python libraries are very large and the are one of the many characteristics of Python's greatest strengths. Furthermore, Python can support many tasks for many applications such as Internet-facing applications which have many standard formats and protocols called MIME and HTTP. In addition, there are many modules for graphical user interfaces development as well as a relational database connector, a pseudo-random numbers generator, an arithmetic with arbitrary precision decimal, a manipulation of regular expressions and a unit testing.

Python 2.7 is the last major release in the 2.x series and with a large stable libraries. However, the maintainers have shifted their focus on the new feature development of Python 3.x series. This means that Python will continues receive different community support, but there will be no new full feature releases or standard library for the language. What's more, Python 2.7 will remain in the place of popularity for many years to come by providing a stable and supporting base platform that the Python 3 did not port yet [8].

2.7 Bash Scripts

Bash is a Unix shell and a command language which was first released in 1989. Moreover, Bash shell has been distributed widely as the default shell for Linux distributions and Apple's macOS and a version for Windows is currently available since 2010 [9].

Bash is a command processor that is typically run in a text window and users can type different commands as well as cause actions. In addition, Bash can read commands from a file which is called a script or bash script. Having said that, like all Unix shells Bash can

support filename globbing, piping, here documents, command substitutions, variables and control structures.

Bash can execute the vast majority of Bourne shell scripts without modification. With this in mind, when a user presses the tab key, Bash automatically uses command line completion to match partly typed program names, filenames and variable names. Moreover, the Bash command line completion system is very flexible and customizable and it can also package functions that complete arguments and filename for specific programs and tasks.

A Bash script is a plain text file which contains a series of shell commands. Therefore, these scripts are a mixture of commands usually typed in the shell text window. Furthermore, the name of this text file usually has an extension of .sh(foo.sh for example). Hence, Linux is an expressionless system so it doesn't necessarily have to have this specific characteristic to run the script [10].

2.8 Terminal

In a Linux system, terminal is a communication tool between users and machines. Additionally, Users can insert shell commands or script files into the terminal and it tells the server’s operating system what to do. Notably, there are several shells that are widely used such as: Bourne shell (sh) and C shell (csh). Each shell, for instance, has its own feature set and intricacies regarding how commands are interpreted. Nonetheless, from the fact that each shell contain their own feature set, all of them feature input and output redirection, variables, and condition-testing, among other things [11].

2.9 Linux Containers

Linux Containers (LXC) is a userspace interface for the Linux kernel containment features. Furthermore, one Linux host can run multiple isolated Linux containers. Having said that, Linux containers can easily give users the ability to create and manage a system or an application containers [15].

Every single Linux Container can share the same operating system kernel from the host and can also isolate the application processes from the rest of the system. To put it in another way, running apps and services of every container can stay lightweight and run swiftly in parallel.

2.10 Common Open Research Emulator (CORE)

The Common Open Research Emulator (CORE) is a tool for building virtual networks. Furthermore, as an emulator, CORE can build a representation of a real computer network that runs in real time using script and programs from the local host machine. Another key thing to mention, CORE runs as opposed to simulation and uses abstract models. Additionally, the live-running emulation can connect to physical networks and router besides from virtual network. Lastly, CORE can provide an environment for running real applications and protocols by taking advantage the virtualization provided by the Linux or FreeBSD operating systems [16]. What's more, in Linux operating system Core uses Linux containers to emulate the computer network.

Chapter 3

Related Studies

3.1 The Design and Evolution of Live Storage Migration in VMware ESX

The team in “The Design and Evolution of Love Storage Migration in VMware ESX” presents their experience with the design and implementation of three different approaches to live storage migration which are Snapshotting, Dirty block tracking and IO Mirroring. Furthermore, they designed different trade-offs by the way of the impact on the guest performance, overall migration time and atomicity.

The first two approaches exhibit several shortcomings. For example, Snapshotting imposes substantial overheads and lacks of atomicity and makes long distance migration fragile. Moreover, Dirty block tracking adds atomicity and by working at the block level allows a number of new optimizations, but also cannot guarantee convergence and zero downtime for every migration.

Their latest approach based on IO mirroring offers guaranteed convergence, atomicity and zero downtime with only a slightly higher IOPS penalty form the second approach. With the latest approach they have achieved consistent reduction in total migration time, having the total love migration duration close to that of a plain disk copy [17].

3.2 Workload-Aware Live Storage Migration for Clouds

In this paper the team has demonstrate that the existing migration solution can have poor I/O performance during migration that could be mitigated by taking a workload-aware approach to storage migration. Furthermore, they developed their own insight on workload characteristic by collecting I/O traces of five representative applications to validate the extent of temporal locality, spatial locality and access popularity that widely exists. Moreover, they demonstrated the improvements introduced by work-load aware scheduling on a fully implemented system for KVM and through a trace-driven simulation framework.

After demonstration they summarized the benefits of workload-aware scheduling under different conditions and workloads using the pre-copy migration model as an example. With the Network bandwidth the benefits of scheduling increases as the amount of available network bandwidth for migration decreases. With the Image size the benefits of scheduling increases as the image size gets larger even if the active I/O working set remains the same. With the I/O rate they became more intense, the benefits of scheduling increases, that mean that with high I/O intensity, the probability that any previously migrated content becomes dirt is higher. And last with I/O characteristics, as the extent of locality or popularity becomes less pronounced, the benefits of scheduling decreases [18].

Chapter 4

Design and Development

4.1 Baseline approach of migration

The Baseline approach of Migration was a simple designed program with python programming language. Importantly, baseline approach scoped to simulate the basic cases of a network filesystem migration. On the one hand, in the system a client side exists and it is responsible for the file packaging as well as sending the packages to a specific server. On the other hand, the server side is responsible for receiving the package from the client and also adapting it to the local filesystem.

4.1.1 Client

Firstly, the Client program is responsible for creating a socket with a server address and a port which can send the messages to the server. In addition, the program will establish the directory that the user wants to migrate and package. Another key thing to remember, all files and directories will be packaged into one Tar file. Furthermore, the client program will wait for a “ready” message from the server in order to acknowledge that the server is ready to receive that package. With this in mind, when the “ready” message is received then the sending process of the package will start using the socket. Lastly, in order for the program to be terminated the client must delete all the unnecessary packages for the filesystem.

4.1.2 Server

One key thing to mention, is that the server program will run in the background of all linux machines and they will be ready to get connection from the client. Importantly, whenever the server program receives a connection from a client it will firstly establish the directory that user wants to migrate. Furthermore, the next step is for the server to sent a “ready” message to the client. Therefore, after the server sends the message, the program will be in

a position to receive the package which includes all of the changes of the filesystem. Another important thing to remember, is that the package is in Tar file form and it's significantly necessary to decompress the Tar file. What's more, when the Tar file is decompressed is in order to adapt all files into the filesystem by replacing all directories and files. All things considered, through the end of the program the server will delete any packages that are not necessary for the filesystem.

4.2 ELFM Optimization 1

With the first optimization of ELFM system it is reasonable that by sending all the files is not really an efficient action. In addition, it is for this reason that the OverlayFS was introduced in ELFM's optimization, in order to avoid sending all the files to migrate in the filesystem. Similarly, with the baseline approach there is still a client side program and a server side program. However, there some differences between the two programs.

4.2.1 OverlayFS in ELFM

It is important to refer again to some of OverlayFS characteristics since they are necessary in the process of ELFM's first optimization. In other words, one of the characteristics of the OverlayFS is, for instance, the merged filesystem which is the place that users can access on the contents. Additionally, any changes in the merged filesystem automatically changes or creates the content of the 'upper' filesystem. However, if the user doesn't make any changes to an existing file then that file will only appear in the 'lower' filesystem. Furthermore, if a file or directory is deleted in the merged filesystem then the 'upper' filesystem will keep a character device (special file) with the same name of that object. In conclusion, if the 'upper' filesystem at the mount phase of OverlayFS is empty then it is possible to detect which files are modified, created or deleted.

4.2.2 Client

Similarly with the baseline approach migration system, the Client program creates a socket with a server address and a port which can send the messages to the server. Additionally, the program will establish the directory that the user wants to migrate. However, in the packaging phase there are some changes from the baseline approach. More specifically, the changes which would package all the files and directories in the 'lower' filesystem, the program will go through the 'upper' filesystem and will examine all the files. Moreover, if a file is a character device then the program will add the file's path into one text file and then delete it from the 'lower' filesystem and the character device from the 'upper' filesystem. Therefore, when this process is over as a result the 'upper' filesystem will contain only files that are created or modified. Furthermore, the program will package the text file which contains all paths of deleted files and directories as well as the 'upper' filesystem into a Tar file. Additionally, the client program will wait for a "ready" message from the server in order to acknowledge that the server is ready to receive that package. With this in mind, when the "ready" message is received then the sending process of the package will start using the socket. Lastly, in order for the program to be terminated the client must delete all the unnecessary packages for the filesystem, replace all the files from the 'upper' filesystem to 'lower', empty the 'upper' filesystem and remount the OverlayFS.

4.2.3 Server

The server program of ELFM first optimization is similar with the baseline approach it also needs to run in the background of all Linux machines and needs to be ready to get connection from the client program. Furthermore, whenever the server program receives a connection from the client it will firstly establish the directory that the user wants to migrate. Additionally, the server will sent a "ready" message to the client. Therefore, after the server sends the message, the program will be in a position to receive the package which includes all of the changes of the filesystem. Another important thing to remember,

is that the package is in Tar file form and it's significantly necessary to decompress the Tar file. What's more, when the Tar file is decompressed, the program will read from the text file that contains all deleted files paths and delete them from the "lower" filesystem. Subsequently, the program will replace all the files from the package into the "lower" filesystem as a result to complete the adapt phase. All things considered, the server program will delete any files that are unnecessary for the filesystem and remount the OverlayFS.

4.2.4 Versioning

With the above two programs, the ELFM system will apply for only one case which is a network consisted of two linux machines. In addition, the result is that the machines will have maximum one version difference. Furthermore, in order to support more machines and maximize the level difference it is necessary to have one version control mechanism. However, to import this mechanism simply to create a folder named "storage" at the same level as "upper" and "lower" filesystem. That is to say, the "storage" folder will contain the Tar file which is the same package that the client generates. Having said that, every machine's version will be determined from the number of files in "storage" folder. Furthermore, both programs will exchange versions between them before the "ready" message phase. Hence, both machines have to save the package into their "storage" folder to keep track of their versions.

4.3 ELFM Optimization 2

Along with the importation of the OverlayFS and the versioning technique in ELFM system there was a success to package only modified files in order to migrate another filesystem. However, with the scenario which there is a big difference of versions between two

machines there is a high probability that the package of client program prepares in the first optimization will be more large than the first ones. Furthermore, to increase the efficiently of ELFM system it is important to send less data without losing the concept of migration. Thus, for this reason the ‘diff’ and ‘patch’ was introduced in ELFM system.

4.3.1 Diff

‘Diff’ is a shell command for Linux and Unix OS. Additionally, ‘diff’ can analyze two files and prints the lines that are different. Essentially, the outputs of this command is a set of instruction for how to change one file to make it identical to the second file. Furthermore, ‘diff’ command has a big variety of options. However, in ELFM system ‘diff’ is used by default options. Furthermore, if the outputs are redirected into a file then that file is called patch file.

Figure 3. This is example of diff command with default option

4.3.2 Patch

Patch is also a shell command for Linux and Unix OS. Moreover, patch takes the patch file that diff generates and applies it to the original file in order to produce the patched version.

4.3.3 Client

In the second optimization of ELFM system, the client program firstly creates a socket with a server address and a port which can send the messages to the server. Additionally, the program will establish the directory that the user wants to migrate. Moreover, the program will start to pack the Tar file with a new method. To give an illustration, the program will go through the ‘upper’ filesystem in order to examine all the files. Therefore, if a file is a character device then the program will add the path of the file into one text file. Furthermore, the program deletes that file from the ‘lower’ filesystem and the character device from the ‘upper’ filesystem. However, if a file is not a character device then the program will run the diff command with its parameters. Most specifically, diff command’s parameters are consistent with that file and a file with the same path in the ‘lower’ filesystem. What’s more, the program will save the diff result into one variable. Moreover, if that variable is not empty it means that the file was modified. Having said that, the program will save that variable into one patch file, the path of that file and the name of the patch file will be added into one modified manager text file. With this in mind, the program will replace the modified file from the ‘upper’ filesystem to the ‘lower’ filesystem as well as deleting it from the ‘upper’ filesystem. Last but not least, all modified files will be removed from the ‘upper’ filesystem which means that in the filesystem only created files will remain. Additionally, the program will move all the files from the ‘upper’ file system to the package.

After the packaging phase of the program the client will exchange the version with the server in order to recognize how many versions does the server needs to migrate. In addition, the next step is for the program to wait for a “ready” message. With this in mind, when the “ready” message is received then the program will send all the version packages that the server needs, including the current one to the server with the socket. Thus, in order to terminate the client program it is necessary to move the current version package into the

‘storage’ folder, delete all the unnecessary packages, empty the ‘upper’ filesystem and remount the OverlayFS.

4.3.4 Server

The second optimization of ELFM system server program is similar to all other versions and it must be run on the background of all Linux machines as well as being ready to get connected from the client program. In addition, when the server receives a connection from the client, it will first establish the directory that the user wants to migrate. Moreover, the server will exchange versions will the client in order to acknowledge the difference of the versions in between them. Additionally, the server will send a "ready" message to the client. Therefore, after the server sends the message, the program will be in the position to receive the package. Whenever the package is received the program will decompress the package from the tar file form. What’s more, when the package is decompressed the program will start reading from the deleted text file with the aim to delete those files in the ‘lower’ filesystem. With this in mind, what follows is for the program to start reading from the modified manager text file in order to patch all necessary files in the ‘lower’ filesystem with the compatible patch file using patch command. Lastly the program will move all the created files from the package into the ‘lower’ filesystem with the result of completing the adapting process.

After the adapting process, the server program will move all the packages into the ‘storage’ folder, delete any files that are unnecessary for the filesystem and remount the OverlayFS.

Chapter 5

Evaluation

5.1 Scenario

5.2 Results

5.1 Scenario

For this particular thesis, CORE was used to compare the efficiency between two versions by time and data traffic. Furthermore, the topology of the network is consisted by three Machines named Machine A, Machine B, Machine C and one switch. Moreover, all links have bandwidth 100 MB per second.

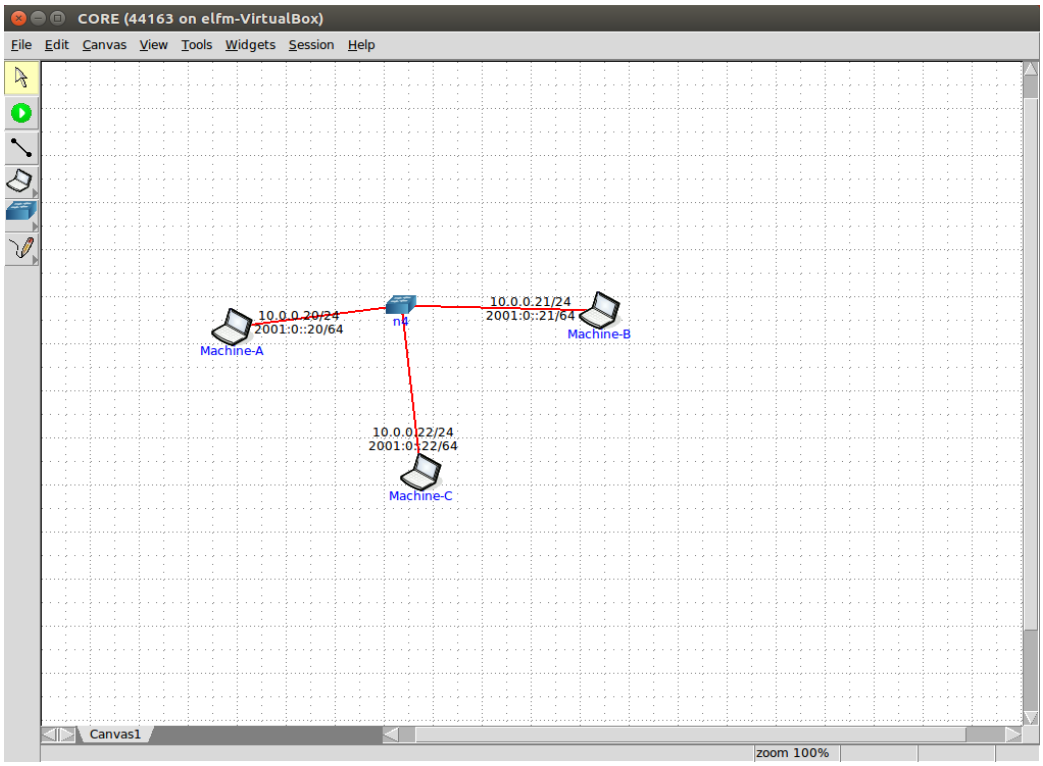

Figure 4. Network topology

This network topology was tested with two categories of filesystem sizes which specifically are small and big filesystem. Furthermore, the small filesystem contains filesystems with the initial sizes 100MB, 200MB, 300MB, 400MB and 500MB and the big filesystem contains filesystems with initial sizes 500MB, 1000MB, 1500MB, 2000MB and 2500MB.

The route of this particular scenario is A -> B -> C -> B -> A. That is to say, with this route migrations are involved with one version difference, two versions differences and three versions differences. Moreover, before every migration the filesystem is tested with 10%, 20%, 30% and 40% changes. Additionally, the changes of a filesystem are determined by the create file, delete file, modify file. As well as, in the modify file there are separated by insert lines, delete lines and modify lines.

Machine A	Machine B	Machine C	Status
0	0	0	Initialize
1	**0**	0	A make changes
1	**1**	0	Migrate A to B
1	2	**0**	B make changes
1	2	**2**	Migrate B to C
1	**2**	3	C make changes
1	**3**	3	Migrate C to B
1	4	3	B make changes
4	4	4	Migrate B to A

Figure 5. Table of every machine version

The results are tested between the first and the third version of ELFM system. In order to show the efficiency of OverlayFS in ELFM system.

5.2 Results

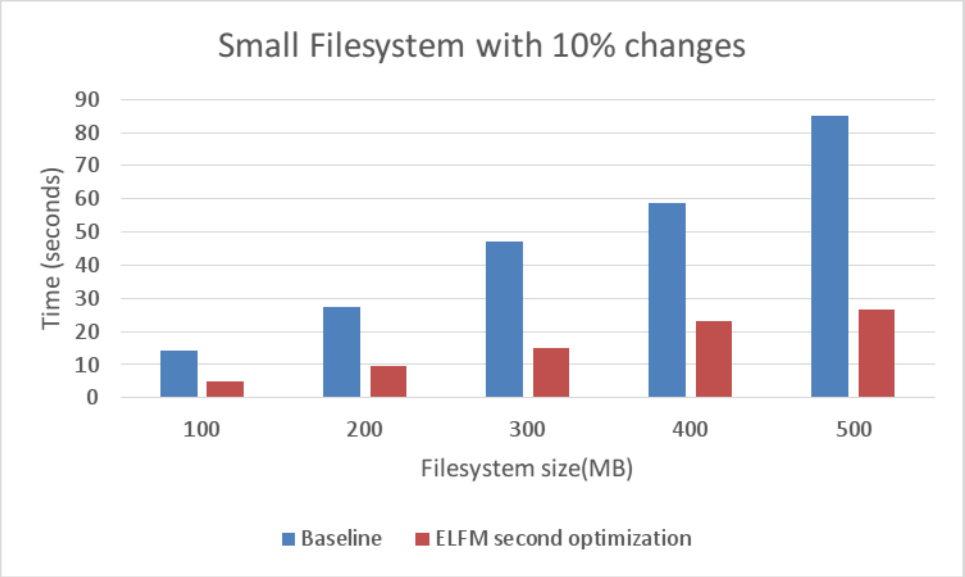

Figure 6. First set of result 1/8

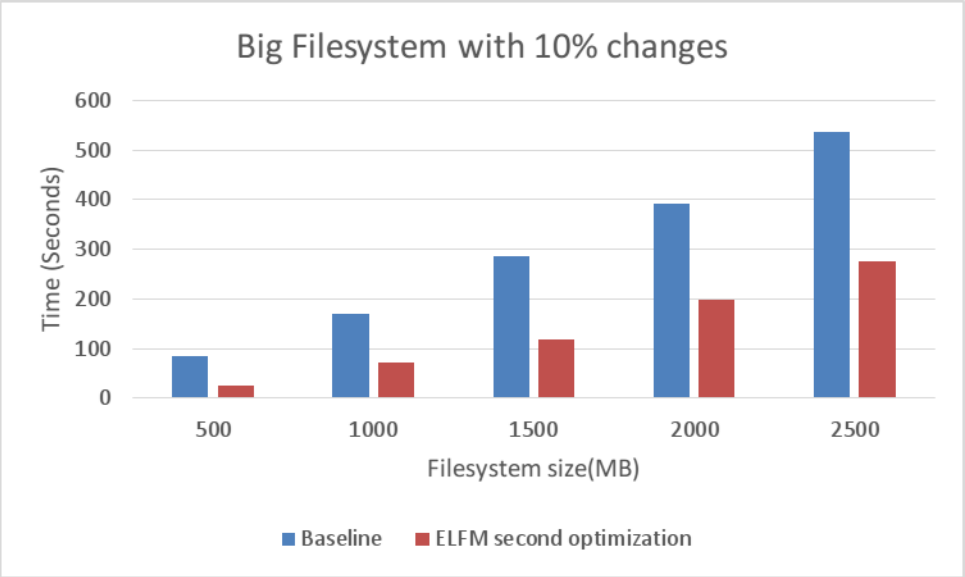

Figure 7. First set of result 2/8

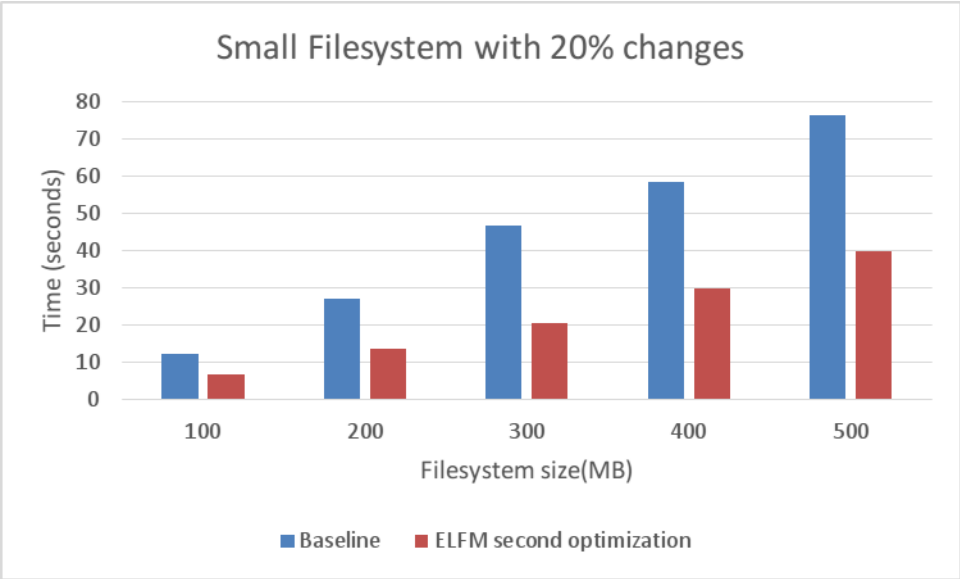

Figure 8. First set of result 3/8

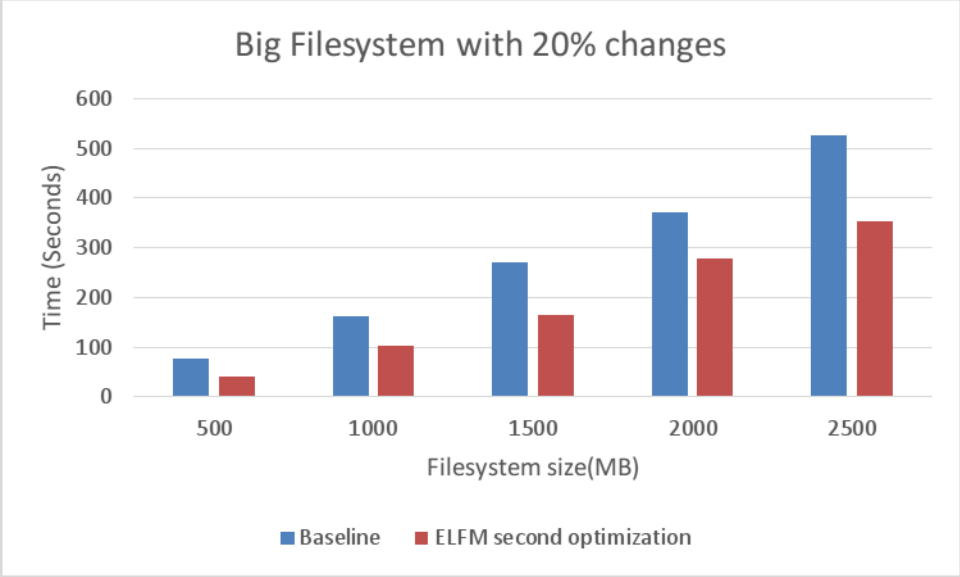

Figure 9. First set of result 4/8

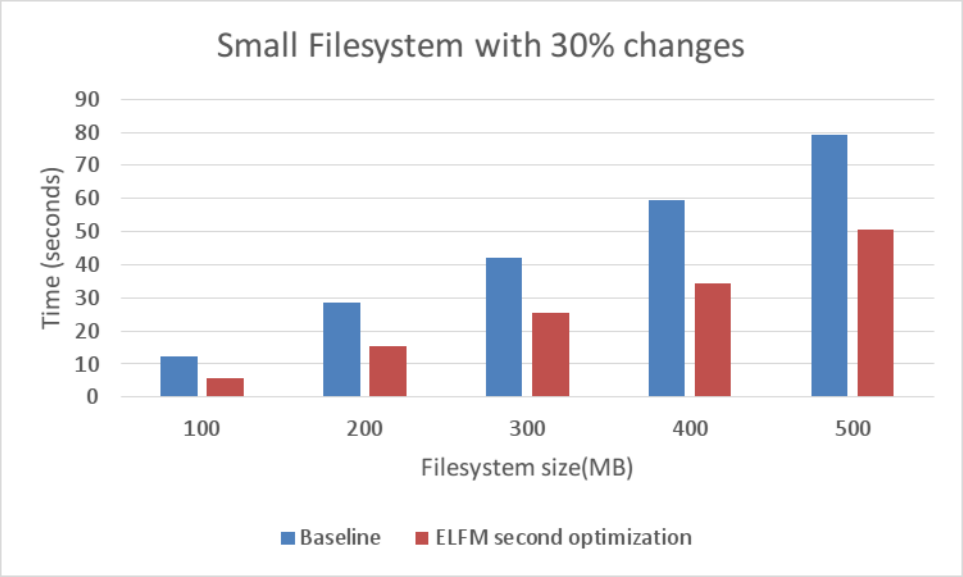

Figure 10. First set of result 5/8

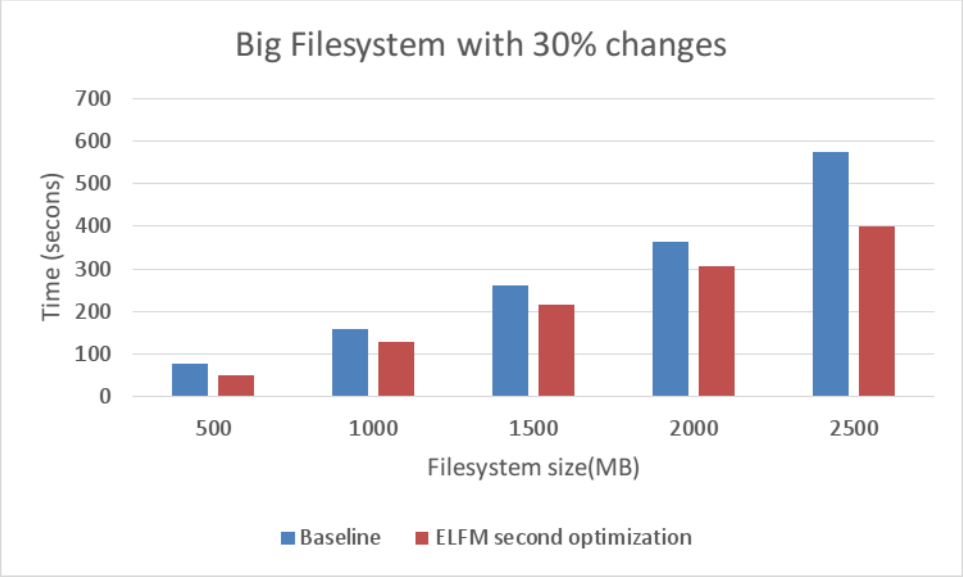

Figure 11. First set of result 6/8

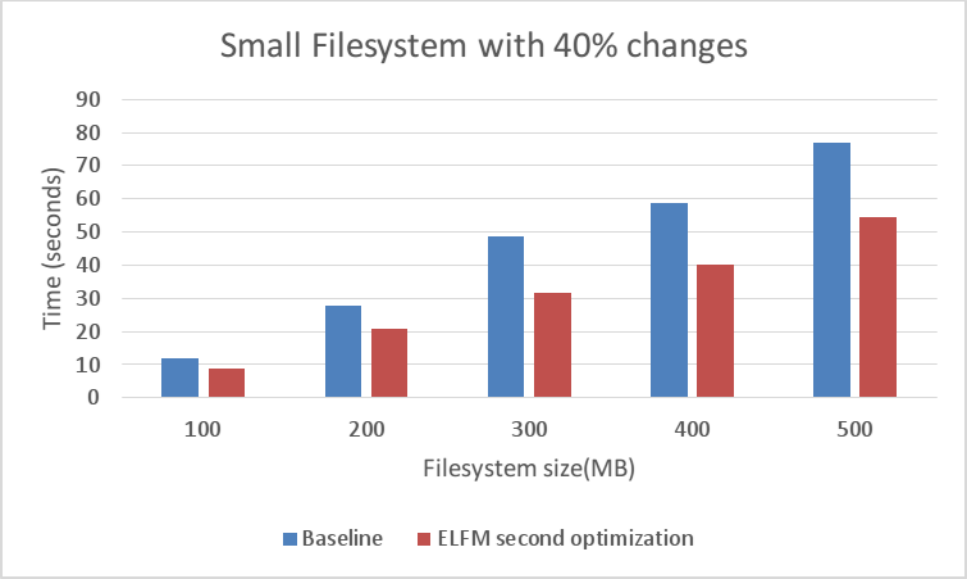

Figure 12. First set of result 7/8

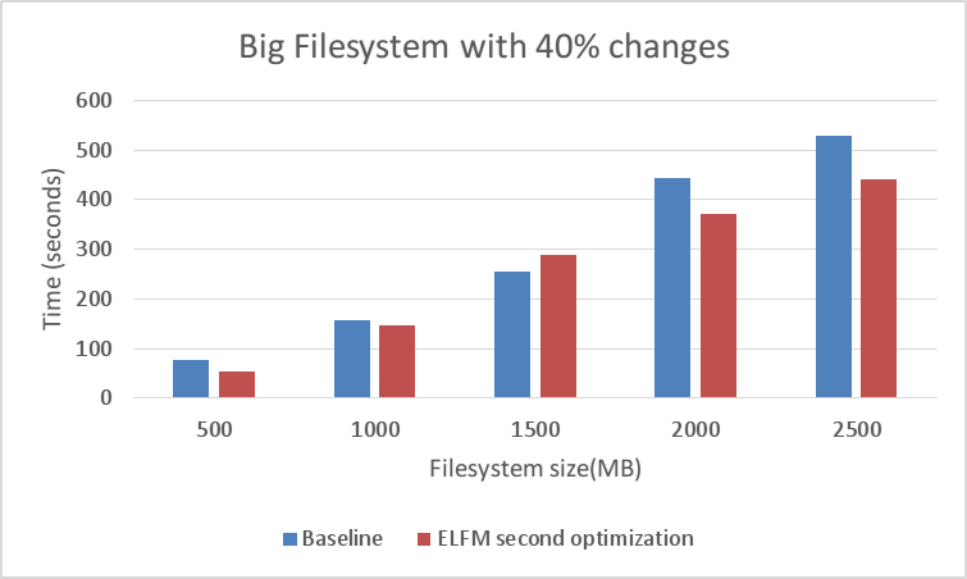

Figure 13. First set of result 8/8

The first set of the result showed that the average time of each scenario the ELFM second optimization have better time than the average time of the baseline approach. Furthermore, the improvement of the efficiency in average time comparing by the second optimization of ELFM with the baseline approach are over 50% in ten percentage of changes, over 40% in twenty percentage of changes, over 35% in thirty percentage of changes and over 20% in forty percentage of changes.

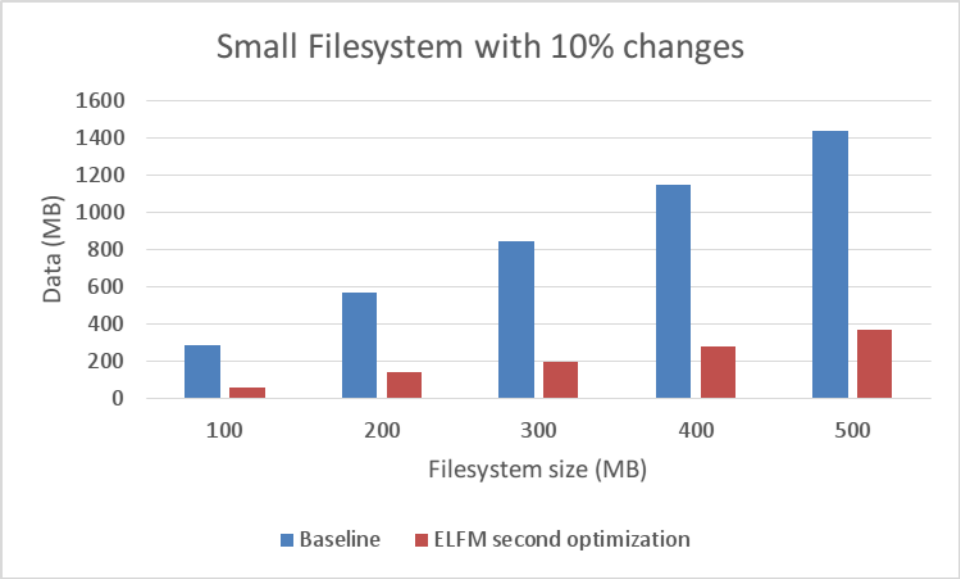

Figure 14. Second set of result 1/8

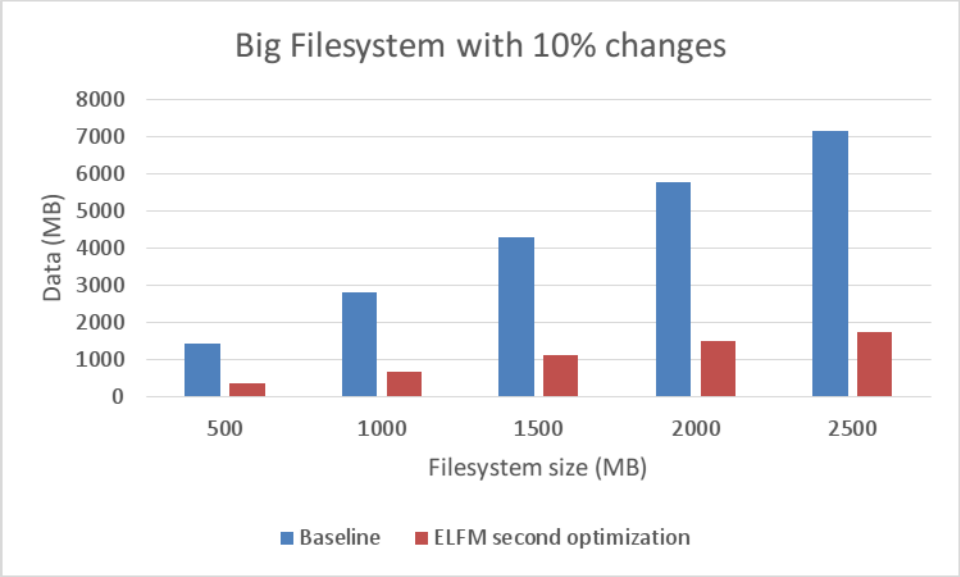

Figure 15. Second set of result 2/8

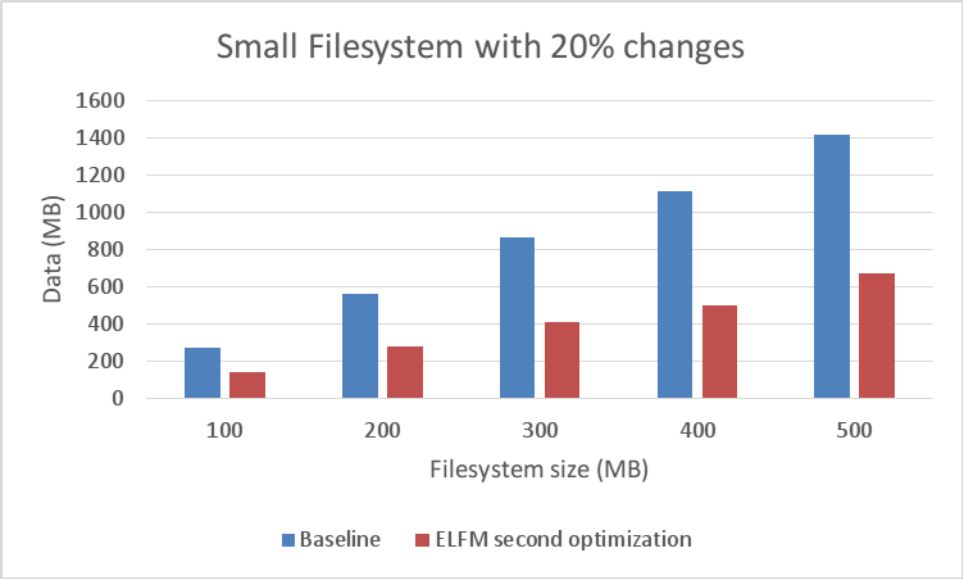

Figure 16. Second set of result 3/8

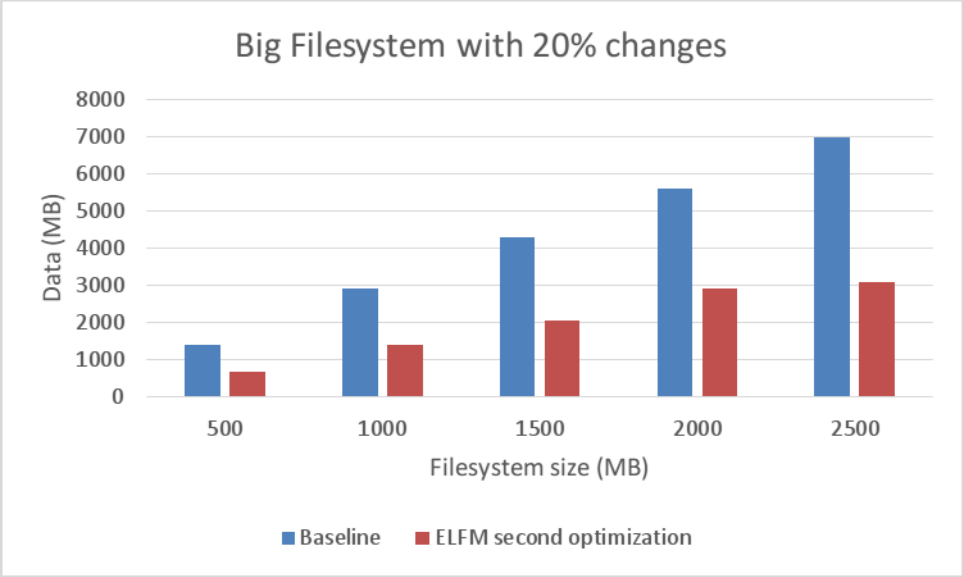

Figure 17. Second set of result 4/8

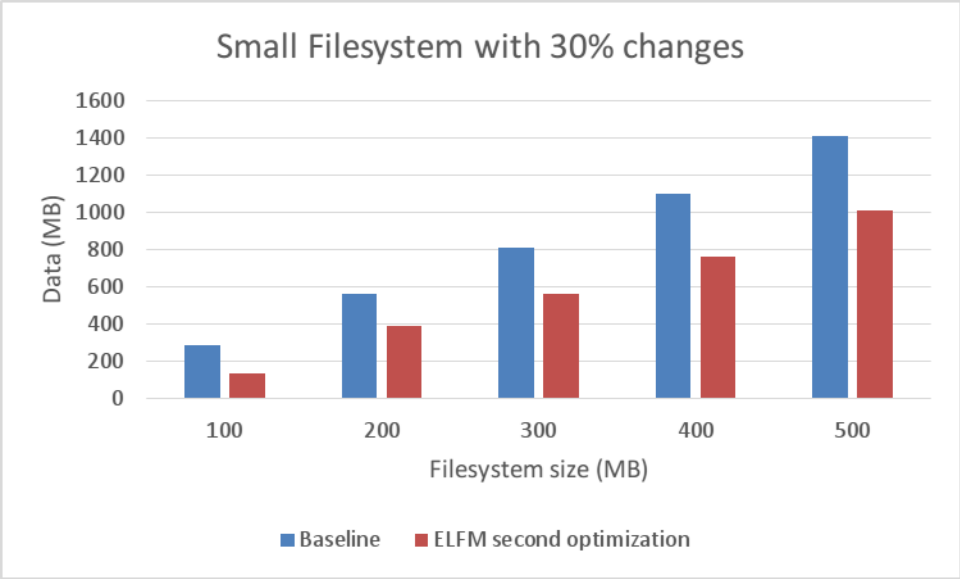

Figure 18. Second set of result 5/8

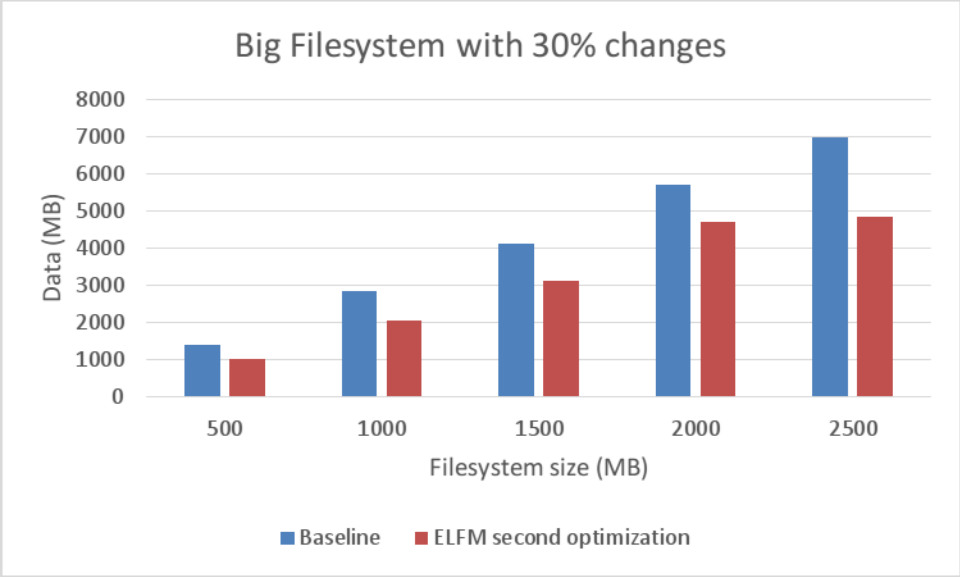

Figure 19. Second set of result 6/8

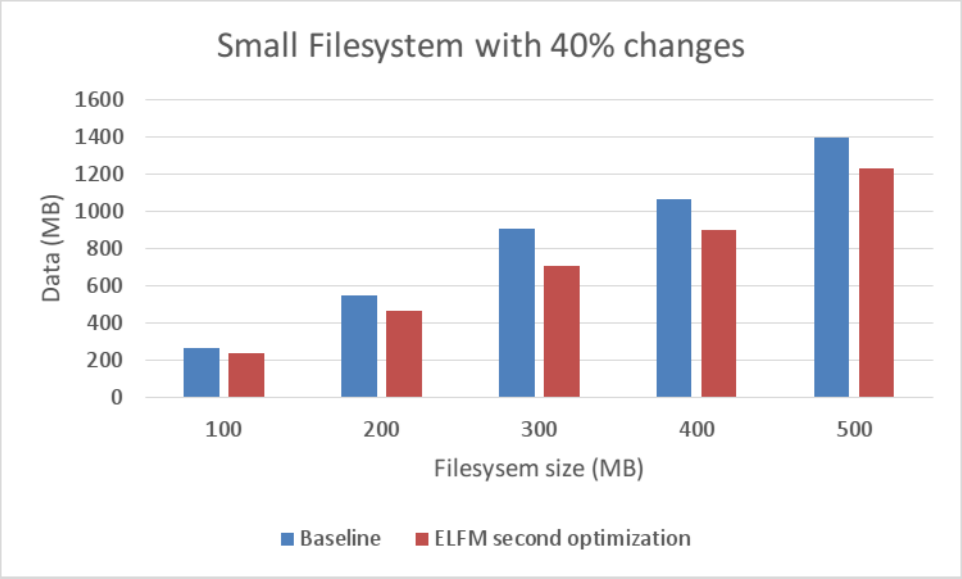

Figure 20. Second set of result 7/8

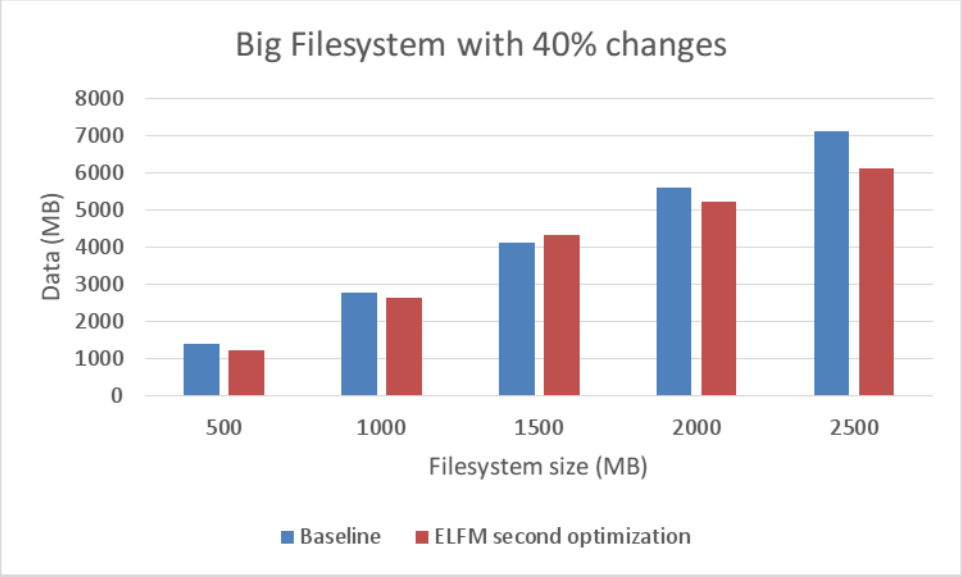

Figure 21. Second set of result 8/8

With the second set of result it is obvious that the second optimization of ELFM sends less data from the baseline approach ones sends. Moreover, the improvement of the efficiency in data sends comparing by the second optimization of ELFM with the baseline approach are around 70% in ten percentage of changes, around 60% in twenty percentage of changes, around 30% in thirty percentage of changes and around 15% in forty percentage of changes.

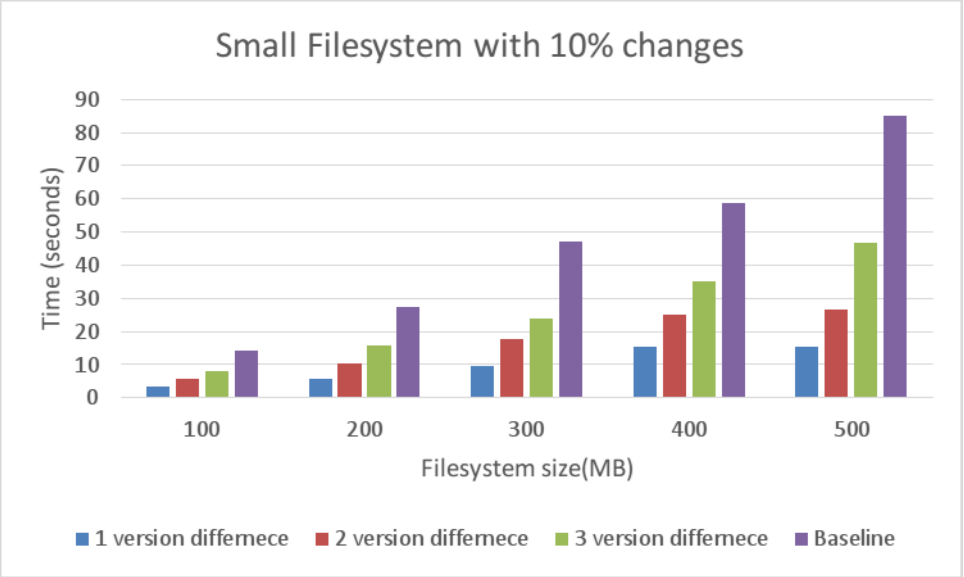

Figure 22. Third set of result 1/8

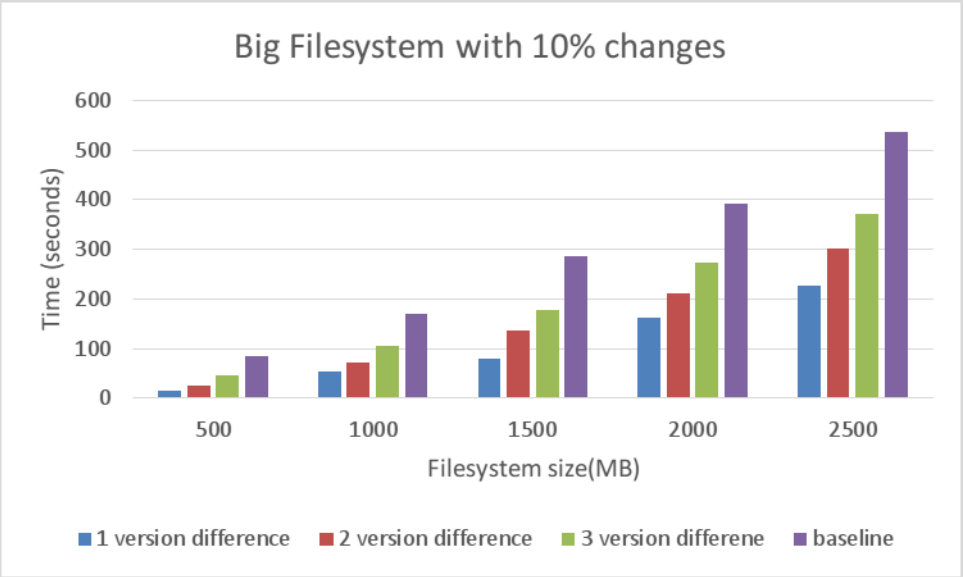

Figure 23. Third set of result 2/8

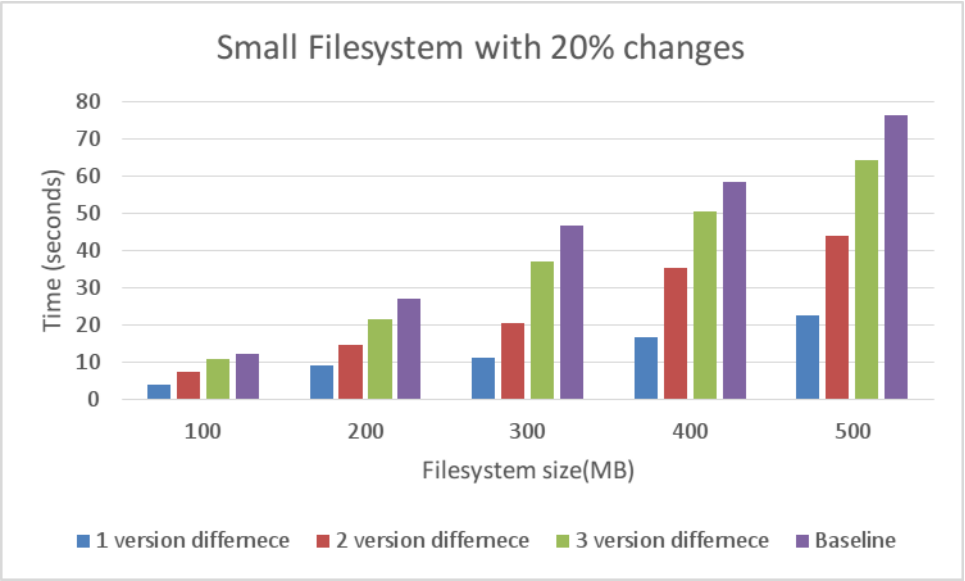

Figure 24. Third set of result 3/8

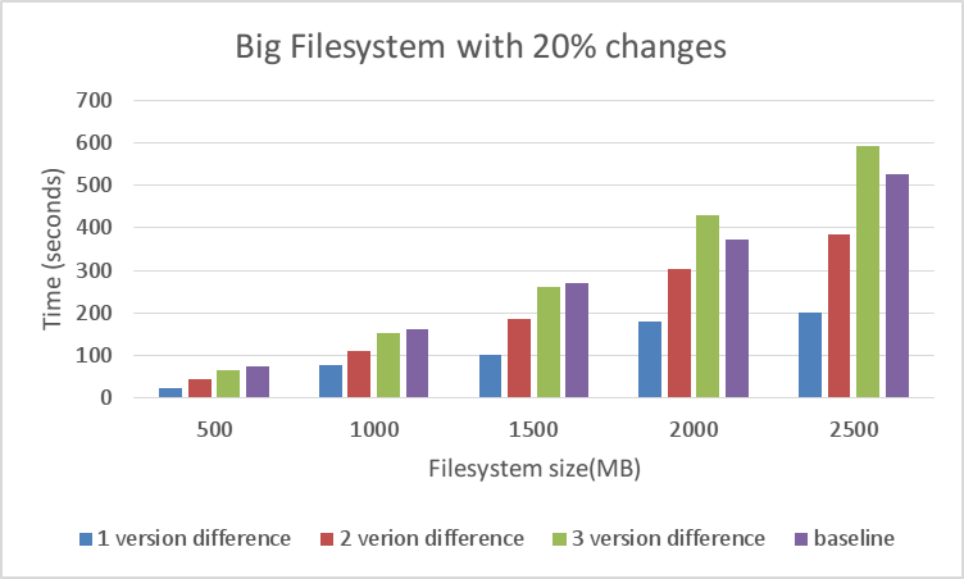

Figure 25. Third set of result 4/8

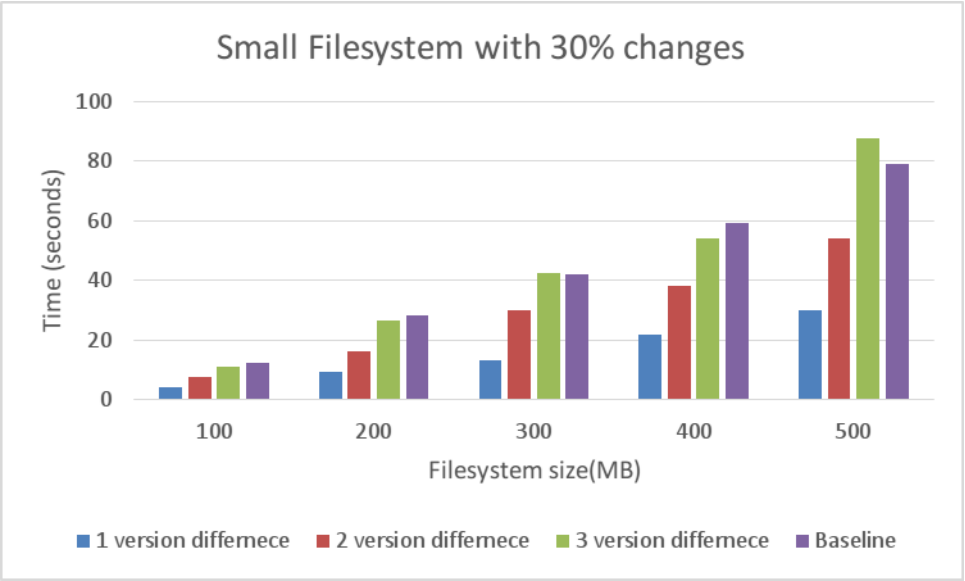

Figure 26. Third set of result 5/8

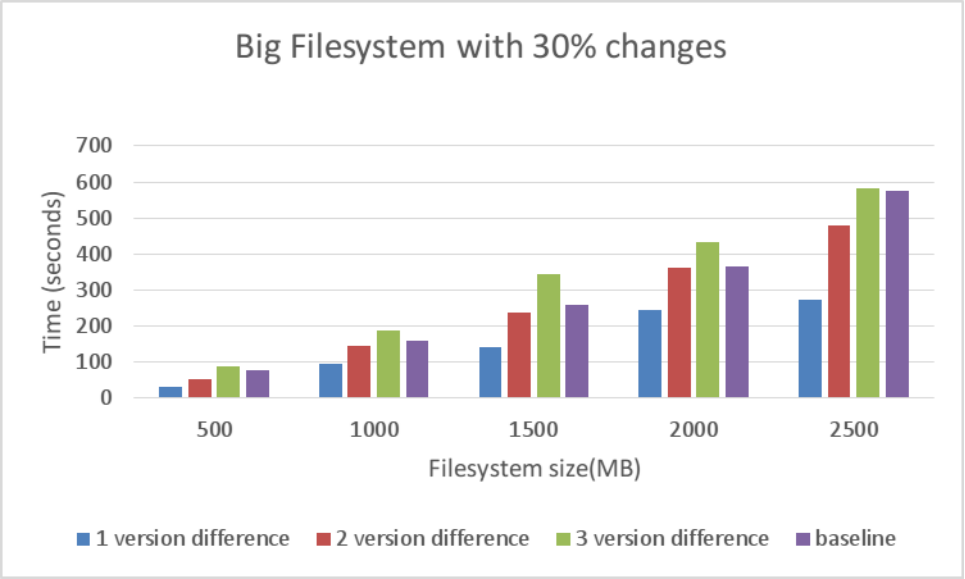

Figure 27. Third set of result 6/8

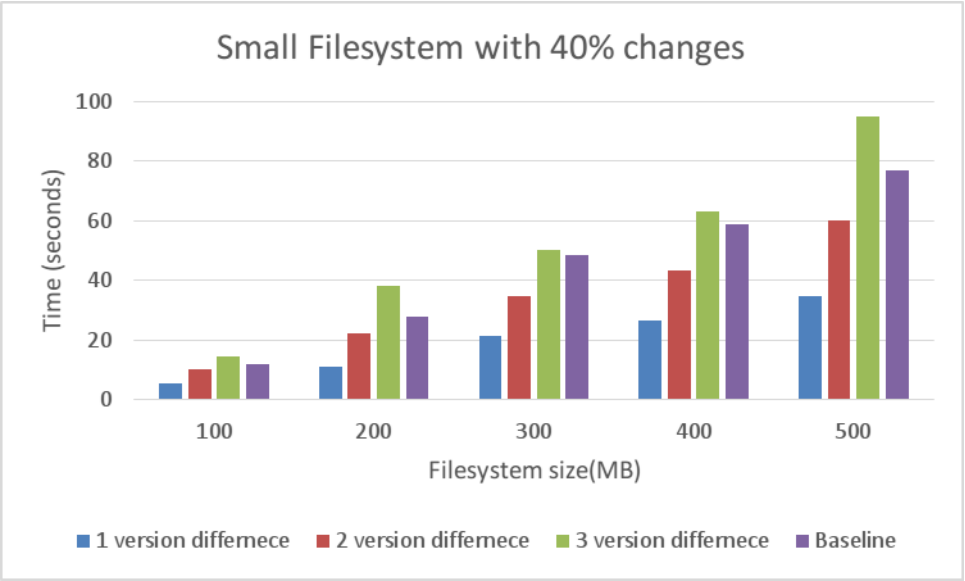

Figure 28. Third set of result 7/8

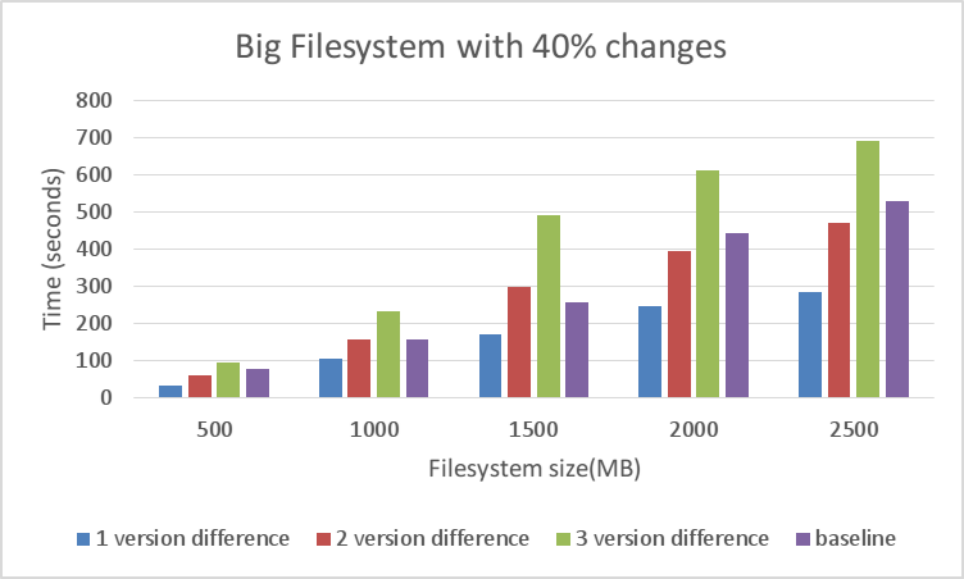

Figure 29. Third set of result 8/8

The third set of result showed that with more version differences the system needs more time to migrate. By contrast, in the baseline approach of migration, all migrations of each route have approximately the same time. Moreover, when the second optimization of ELFM deal with 3 version differences only 21 of 40 scenarios are more efficient than the baseline approach. However, migrating 1 version difference and 2 version differences it is clearly that the second optimization of ELFM is more efficient.

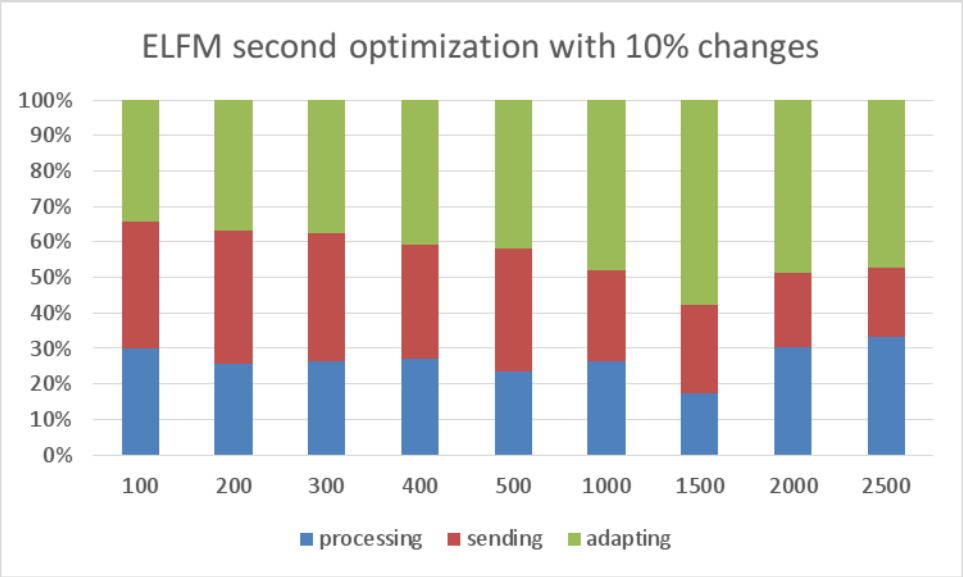

Figure 30. Fourth set of result 1/8

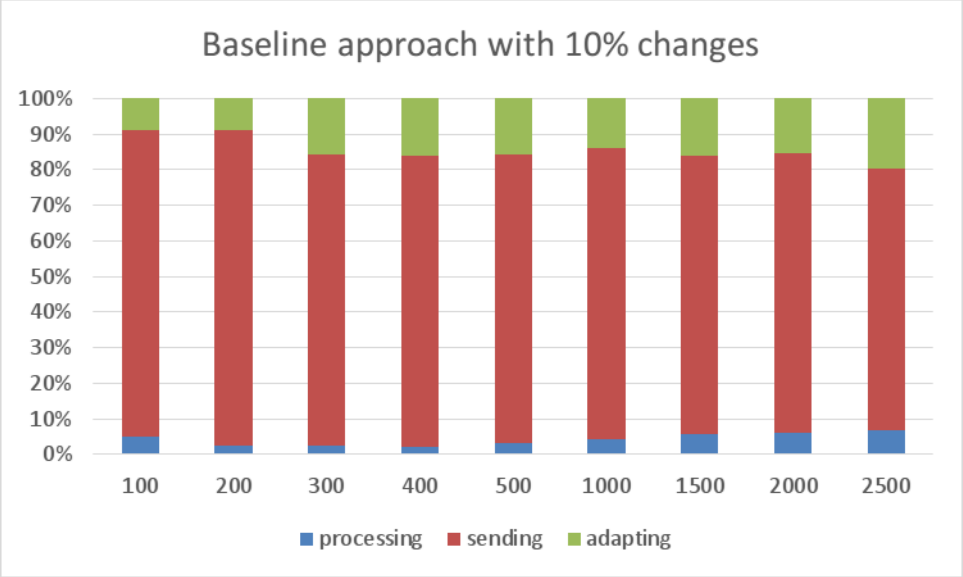

Figure 31. Fourth set of result 2/8

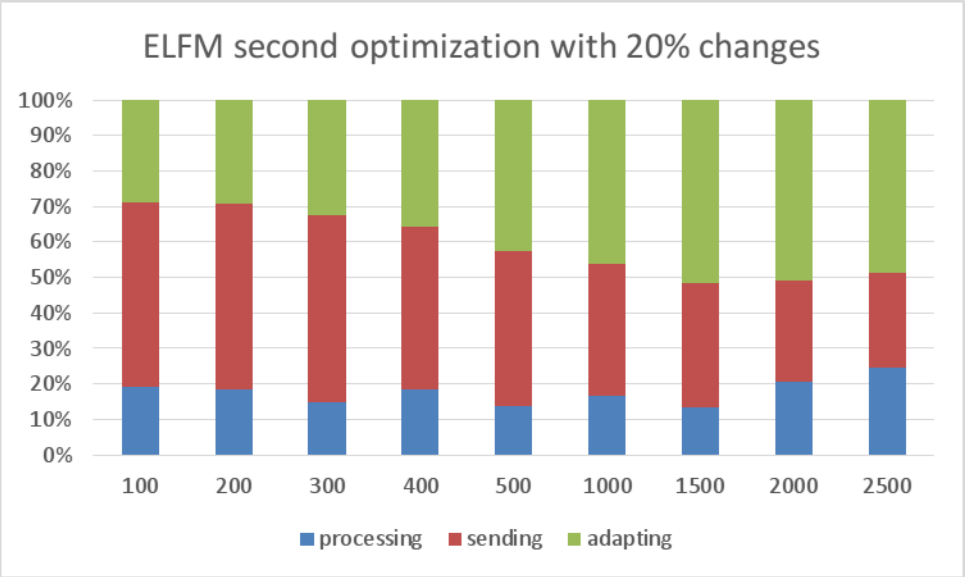

Figure 32. Fourth set of result 3/8

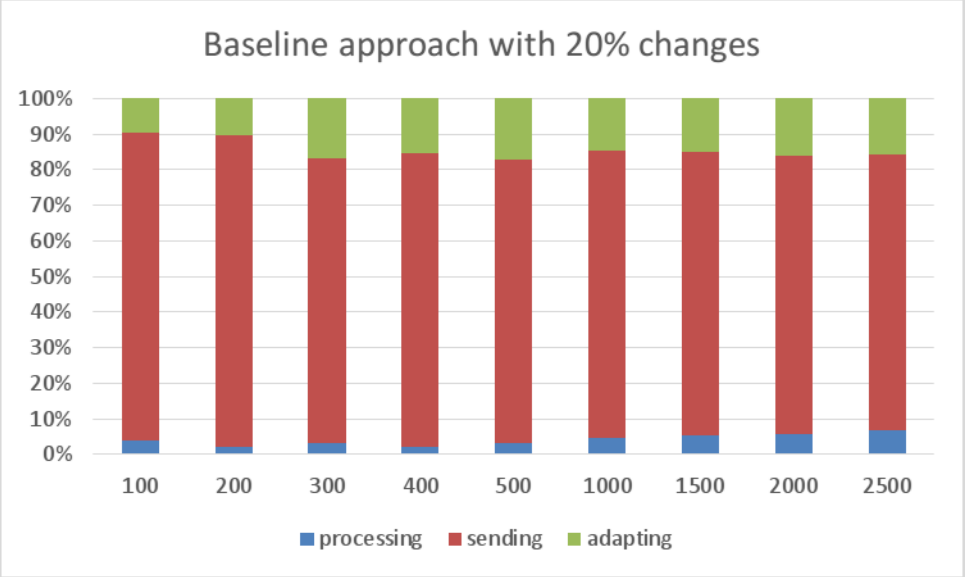

Figure 33. Fourth set of result 4/8

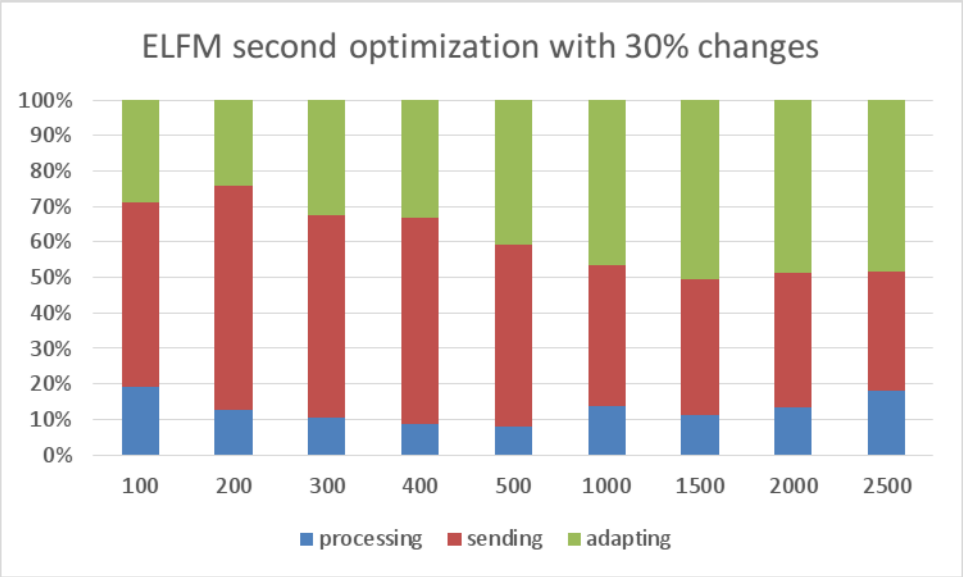

Figure 34. Fourth set of result 5/8

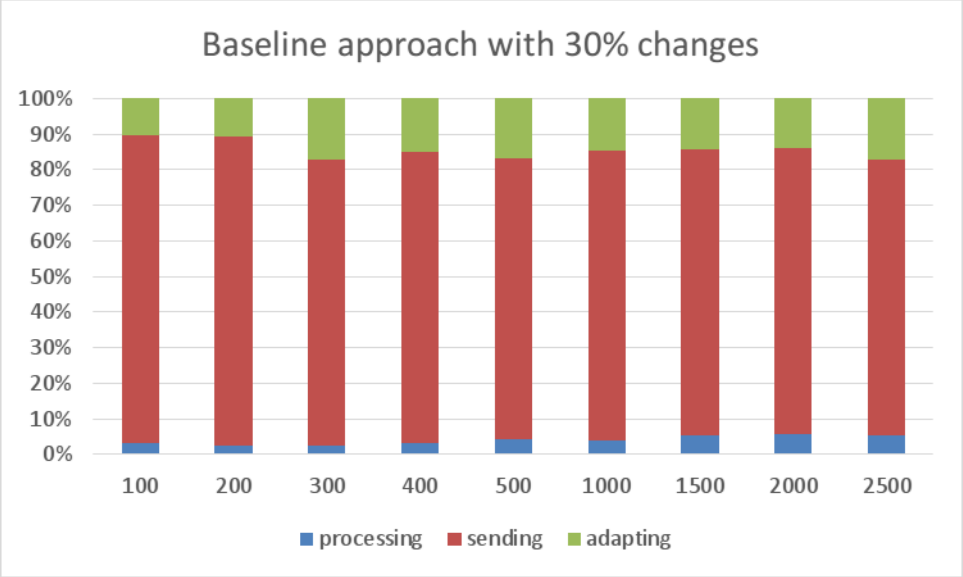

Figure 35. Fourth set of result 6/8

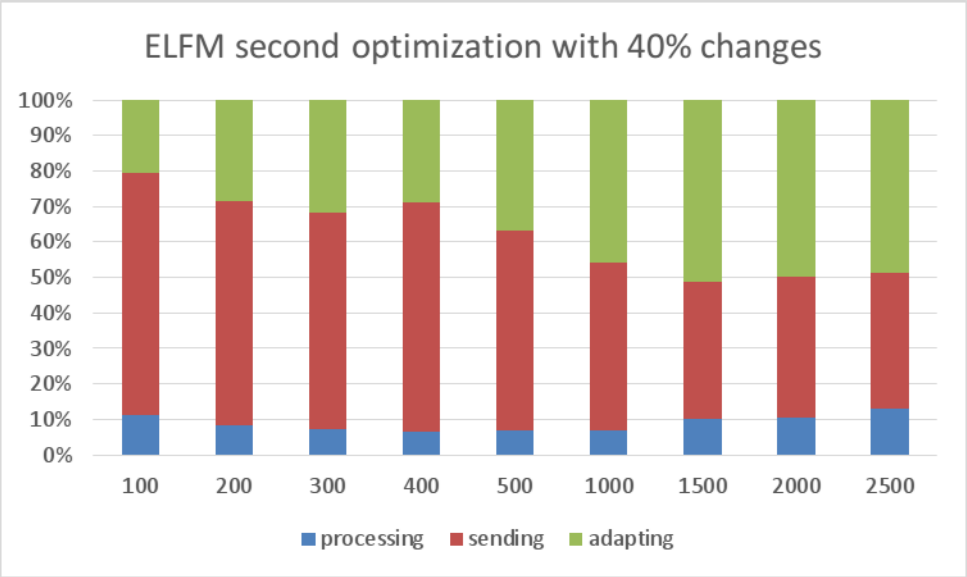

Figure 36. Fourth set of result 7/8

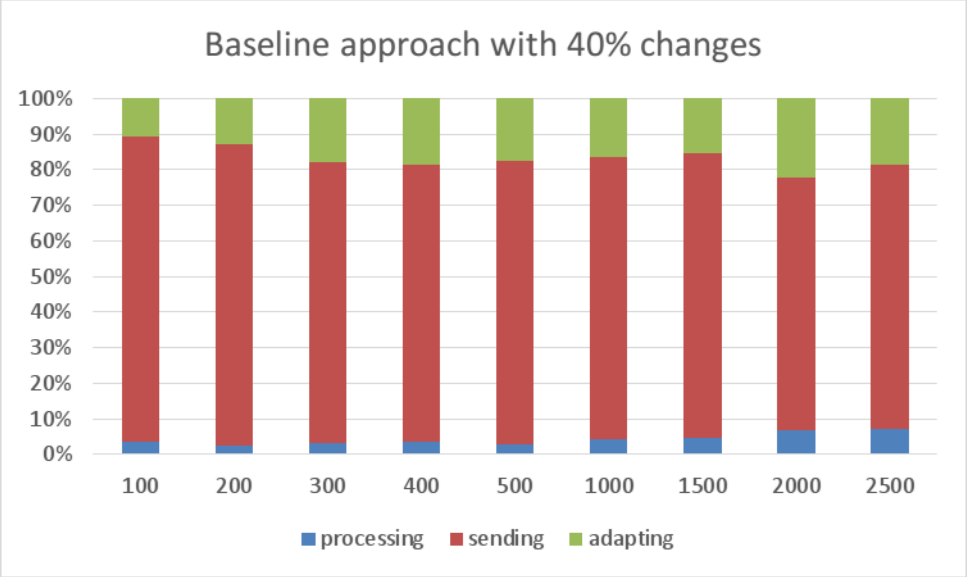

Figure 37. Fourth set of result 8/8

The fourth set of results depicted precisely that the processing and the adapting time for the second optimization of ELFM takes the more percentage of the whole migration time than the baseline approach. However with more optimization the ELFM system can reduce those times with different techniques which are mentioned in Chapter 6 – Future work.

Chapter 6

Conclusion and Discussion

6.1 Summary

In order to improve the efficiency of the migration, the implementation of “OverlayFS” in the ELFM system and the “Diff and Patch” methodology in the processing phase and adapting phase was used. Therefore, it is obvious that the migration average time and the data sending size both are decreased, since the system can acknowledge which files are modified and created.

Furthermore, the above implementations leaded the system to adapt the simple versioning management. Hence, with the versioning management any Linux machine in the system can theoretically recover from any system crash or losing data from the merged filesystem.

However, in the case when a migration between two machines have big version differences and large percentage of filesystem changes, the ELFM system will need more improvements in order to become more efficient.

6.2 Future Work

In order to make the ELFM system more efficient, the system should make the processing and adapting phase in lesser time. With this in mind, the system can create a daemon (long-running background process) which can calculate the differences every ten minutes in an active Linux machine so at the migration time the system will have the package ready to send. With this technique the system will save time in the processing phase which is consisted of over twenty five percentage in the whole end to end time.

Another solution in order to reduce the time of the processing phase is to send all the version difference package to all the other machines in the background while the machine is active. Although, this mechanism will determined if it is necessary by the characteristics of the whole network system.

Furthermore, the current ELFM system adapting phase which is divided by adapting the Created files process, adapting the Modified file process and deleting file process. Thus, the system can run those three processes parallel, in order make the adapting phase faster.

References

[1] Machtelt Garrels “Introduction to Linux” http://www.tldp.org/LDP/intro-linux/html/sect_03_01.html

[2] Michael Larabel “OverlayFS Proposed For The Linux 3.18 Kernel” https://www.phoronix.com/scan.php?page=news_item&px=MTc5OTc

[3] Neil Brown “Overlay Filesystem” https://www.kernel.org/doc/Documentation/filesystems/overlayfs.txt

[4] FreeBSD File Formats Manual https://www.freebsd.org/cgi/man.cgi?query=tar&apropos=0&sektion=5&manpath=FreeBSD+7.0-RELEASE&arch=default&format=html

[5] “Controlling the Archive Format” https://www.gnu.org/software/tar/manual/html_section/tar_68.html

[6] “What is a Socket?” https://www.tutorialspoint.com/unix_sockets/what_is_socket.htm

[7] “Ubuntu GNU/Linux” https://www.gnu.org/distros/common-distros.html#Ubuntu

[8] A.M. Kuchling “What’s New in Python 2.7” https://docs.python.org/2/whatsnew/2.7.html

[9] Jack Hammons “Bash on ubuntu on Windows” https://msdn.microsoft.com/en-us/commandline/wsl/about

[10] “What is Bash” https://tiswww.case.edu/php/chet/bash/bashref.html#What-is-Bash_003f

[11] Mitchell Anicas “An Introduction to the Linux Terminal” https://www.digitalocean.com/community/tutorials/an-introduction-to-the-linux-terminal

[12] “PEP20 – The Zen of Python” https://www.python.org/dev/peps/pep-0020/

[13] “What is DFS (Distributed File System)” http://www.maxi-pedia.com/what+is+DFS

[14] Miguel Baguena, Andreas Pamboris, Peter Pietzuch, George Samaras, George Samaras, Mihail L. Sichitiu, Pietro Manzoni “Towards Enabling Hyper-Responsive Mobile Apps Through Network Edge Assistance” https://spiral.imperial.ac.uk/bitstream/10044/1/26539/2/main_ccnc16-camera.pdf

[15] “LXC/Introduction” https://linuxcontainers.org/lxc/introduction/

[16] “Common Open Research Emulator (CORE)” https://www.nrl.navy.mil/itd/ncs/products/core

[17] Ali Mashtizadeh, Emre Celebi, Tal Garfinkel, Min Cai “The Design and Evolution of Live Storage Migration in VMware ESX”
http://static.usenix.org/legacy/events/atc11/tech/final_files/Mashtizadeh.pdf

[18] Jie Zheng, Kunwadee Sripanidkulchai “Workload-Aware Live Storage Migration for Clouds”
https://pdfs.semanticscholar.org/7319/8ff0481448f4286d816af2d57ae316cc8de8.pdf