



UNIVERSITY OF AMSTERDAM

System and Network Engineering
Research Project 2
Master of Science Program

Academic year 2007–2008



*Streaming and storing CineGrid data:
A study on optimization methods*

by

Sevickson KWIDAMA
\$firstname.\$lastname@os3.nl

UvA Supervisor : Dr. Ir. Cees de Laat
Project Supervisors : Dr. Paola Grosso
MSc Ralph Koning

Research Project report for System and Network Engineering, University of Amsterdam, the Netherlands, for RP2 academic year 2007–2008.

This document is © 2008
Sevickson Kwidama `$firstname.$lastname@os3.nl`.

Some rights reserved: this document is licensed under the Creative Commons Attribution 3.0 Netherlands license. You are free to use and share this document under the condition that you properly attribute the original authors. Please see the following address for the full license conditions: <http://creativecommons.org/licenses/by/3.0/nl/deed.en>

Cover image: A combination of a movie reel and the world, is meant to symbolize the co-operation around the world through CineGrid. Source: <http://www.stockxpert.com/>.

Version 1.0.0 and compiled with L^AT_EX on July 7, 2008.

Abstract

CineGrid enables participants to stream high-quality digital media, for collaboration and demonstration purposes.

CineGrid has two possible streaming solutions, Scalable Adaptive Graphics Environment (SAGE) and NTT jpeg2000 codec.

SAGE is specialized middleware software for enabling data, high-definition video and high-resolution graphics to be streamed in real-time, this depends heavily on the hardware used.

NTT jpeg2000 codec is the other streaming solution that is discussed in this paper. This setup uses hardware implementation to compress and decompress streams. The advantage that this setup has on SAGE is that the hardware does not influence the stream. The compression rate of NTT jpeg2000 codec greater is than the rate of SAGE, this would suggest that less bandwidth is needed to stream high-quality digital media.

The local filesystem and NFS is used currently to store the CineGrid data, this form does not meet the requirements of the CineGrid network. The requirements are large and scalable storage space and read speeds higher than the local filesystem and NFS.

One of the possible solutions to meet the requirements is GlusterFS.

GlusterFS can be implemented in a test setup, to look at the stability over long term. The read speed of GlusterFS can become greater than the read speed of the local filesystem.

Contents

1	Introduction	5
2	CineGrid	6
3	Benchmarking	8
3.1	Streaming tests	8
3.2	Filesystem tests	9
4	SAGE vs NTT jpeg2000 codec	11
4.1	SAGE	11
4.1.1	Test Method	12
4.2	NTT jpeg2000 codec	14
4.2.1	Test Method	14
4.3	Results	15
4.3.1	SAGE	15
5	GlusterFS	20
5.1	Current situation	20
5.2	GlusterFS	20
5.3	Test Method	21
5.4	Results	22
5.5	Results w/ Performance Translators	24
6	Future Work	26
7	Conclusion	26
8	Acknowledgments	27
	Bibliography	28
	Appendices	29
A	GlusterFS Installation	29
B	GlusterFS Configuration	31
B.1	Server Configuration file	31
B.2	Client Configuration file	31
B.3	Server Configuration file w/ Performance Translators	32
B.4	Client Configuration file w/ Performance Translators	32

List of Tables

1	Digital Media Formats. Source: [7]	7
2	Video differences	13
3	Maximum read speed: Local, NFS, GlusterFS	23
4	Maximum read speed: GlusterFS with Performance Translators	25

List of Figures

1	GLIF World Map. Source: [6]	6
2	SAGE Architecture. Source: [2]	11
3	Infrastructure used in SAGE tests.	12
4	Infrastructure used in NTT tests.	15
5	Log SAGE Manager. T: TCP stream, B: UDP stream	16
6	CPU load UDP stream SAGE Manager/Renderer	17
7	CPU load Display node. L: UDP stream, R: TCP stream	18
8	Stream fluctuation	19
9	GlusterFS storage design. Source: [21]	21
10	Infrastructure used in GlusterFS tests.	22

1 Introduction

CineGrid[1] Open Content Exchange (COCE) is a platform and architecture to stream CineGrid 4K-content to different 4K-suitable locations. 4K stands for *fourthousand* \times *twothousand* pixels, this is approximately $4\times$ the HD quality nowadays.

Transport and streaming in CineGrid is done using either the Scalable Adaptive Graphics Environment (SAGE)[2] or the NTT hardware jpeg2000 codec[3]. These setups compress and decompress the stream to make it possible to travel over 1 Gbps connections and still be able to deliver 4K-content on screen.

These two setups have totally different approaches and a comparison has not yet been made between them.

The research question that summarizes this part of the research is:

How do SAGE and NTT jpeg2000 codec compare against each other, regarding network streams?

Another aspect of the research is the testing of GlusterFS[4] as a suitable storage system for CineGrid. The currently used system, NFS, does not meet the streaming requirements; while the research of a previous System and Network Engineering (SNE) student[5] pointed out that GlusterFS would be a good alternative. I will build a test setup to test the performance of GlusterFS to see if it truly can improve the current situation.

The research question that summarizes this part of the research is:

Can GlusterFS improve the performance of the CineGrid storage?

In this document I will be looking at the different aspects given above.

First I will give some background information about CineGrid in chapter 2 on the following page, and about benchmarking in chapter 3 on page 8 in conjunction with the different tools that I will use.

The two parts of my research project will be explained individually in chapter 4 on page 11 and chapter 5 on page 20.

Closing this document I will give possible future work in chapter 6 on page 26 and some conclusions in chapter 7.

2 CineGrid

CineGrid is the basis for my research. In this chapter I will explain what CineGrid is and what it's purpose is.

CineGrid, started in 2006 as a non-profit international organization. Nowadays CineGrid has members from different countries. Some of the countries that are members in this organization are: Japan, Netherlands, USA.

The mission of CineGrid as stated on their website[1]:

CineGrid's mission is to build an interdisciplinary community focused on the research, development, and demonstration of networked collaborative tools, enabling the production, use and exchange of very high-quality digital media over high-speed photonic networks.

The transport and streaming of high-quality digital media is made possible by a virtual international organization, the Global Lambda Integrated Facility (GLIF). This organization promotes the paradigm of lambda networking to support demanding scientific applications. Lambda networking is the use of different 'colors' or wavelengths of (laser) light in fibres for separate connections. Each wavelength is called a 'lambda'[6].

GLIF is one of the possible ways to transport and stream very high-quality digital media.

CineGrid uses the optical networks (lambda's) of GLIF for intercontinental transport and streaming of the CineGrid content.

In figure 1 I give the most recent map of the lambda connections around the world.

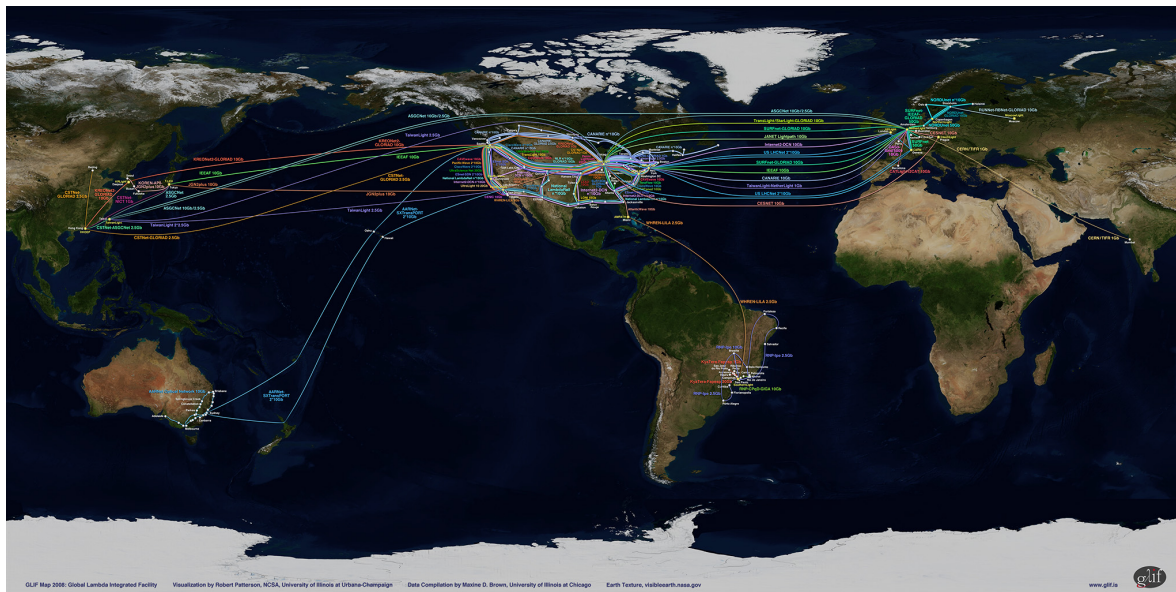


Figure 1: GLIF World Map. Source: [6]

One of the high-quality digital media formats that is available in CineGrid is called 4K. This is roughly $4 \times 1080p$ HD quality which is viewable on HD televisions. In table 2 on the following page I give a comparison between different formats. This comparison gives an idea of the difference between “smaller” media formats and 4K.

The characteristics of each format is displayed in a row in the format table.

X (pixels) is the amount of pixels horizontally;

Y (pixels) is the amount of pixels vertically;

Rate (fps)¹ is the amount of frames displayed in one second;

Color (bpp)² each pixel has a string of bits that is used to determine the color of that pixel;

Frame (pixel) sums up the amount of pixels in one frame, this is done by multiplying X with Y, with the following formula:

$$Frame(pixels) = X(pixels) \times Y(pixels) \quad (1)$$

Frame (MB) is calculated by multiplying the results of equation (1) with Color and dividing it by 8 to get the result in MB:

$$Frame(MB) = \frac{Frame(pixels) \times Color(bpp)}{8} \quad (2)$$

Flow (MB/s) this means the amount of frames in MB (equation (2)) sent in one second; this is calculated with the following formula:

$$Flow(MB/s) = Frame(MB) \times Rate(fps) \quad (3)$$

Stream (Gb/s) transforming the results of equation (3) to bits, we obtain the uncompressed stream:

$$Stream(Gb/s) = Flow(MB/s) \times 8 \quad (4)$$

Format	X (pixels)	Y (pixels)	Rate (fps)	Color (bpp)	Frame (1) (pixels)	Frame (2) (MB)	Flow (3) (MB/s)	Stream (4) (Gb/s)
<i>720p HD</i>	1280	720	60	24	921600	2.8	170	1.3
<i>1080p HD</i>	1920	1080	30	24	2073600	6.2	190	1.5
<i>2K</i>	2048	1080	24 48	36	2211840	10	240 480	1.2 2.4
<i>SHD</i>	3840	2160	30	24	8294400	25	750	6.0
<i>4K</i>	4096	2160	24	36	8847360	40	960	7.6

Table 1: Digital Media Formats. Source: [7]

Almost all videos used nowadays in 4K-demos are actually SHD³ videos, one of the reasons for this is that the NTT setup has a limitation that the signal converter outputs SHD and another reason is that the 4K-beamer is only capable of showing SHD videos. The 4K-beamer is still called a 4K-beamer because the reasoning behind it is that SHD is still a form of 4K, it is $4 \times 1080p$ HD.

There are two different setups at my research location, one streams 1080p HD (SAGE) and the other one streams SHD (NTT jpeg2000 codec).

In the analysis of the different streams I will start by using the table above and my test results to determine the compression rate of the streams and by multiplying those results I will get SHD-compatible results to compare the two setups with each other.

¹frames per second

²bits per pixel

³Super High-Definition also known as Quad HD

3 Benchmarking

Benchmarks are used to measure the time needed by a computer to execute a specified task. It is assumed that this time is related to the performance of the computer system and that the same procedure can be applied to other systems, so that comparisons can be made between different hardware/software configurations.

From the definition of a benchmark, one can deduce that there are two basic procedures for benchmarking:

- Measuring the time it takes for the system being examined to loop through a fixed number of iterations of a specific piece of code.
- Measuring the number of iterations of a specific piece of code executed by the system under examination in a fixed amount of time.[8]

In the benchmark tests that I will perform, the first procedure will be measured, meaning measure the time it takes a computer system to run a specific procedure.

3.1 Streaming tests

In the tests that I will perform between the two different architectures the following tests will be done:

- CPU load;
- Network performance and traffic behavior.

The CPU load will be monitored using the command “top”.

This program gives real-time information about processes running on the system. The program is used with the following syntax:

```
top -b -d 0.5 > load.output
# -b, batch-mode, useful for sending output to file
# -d, delay between samples taken
# > load.output, write the results to a file
```

I chose top because it is a standard tool available on Linux. By using a standard tool we increase the possibility in reproducibility and standardization.

The way that top calculates the load is by taken the CPU time of a process in a specific time interval, divide it against that time interval, multiply it by 100 to get the load percentage. An optional step is to divide that percentage against the amount of CPU cores in that system to get the load per CPU.

The CPU load percentage can be calculated with the following formula:

$$\% CPU\text{Load}(t) = \frac{\frac{\Delta T(t)}{t} \times 100}{\# CPU's}$$

To monitor and display the network performance, I chose Wireshark[9]. Wireshark is a proven tool for packet capturing and analysis, it also has a graphing possibility to display traffic behavior. I will first capture the packets with tcpdump, tcpdump is a command-line program for capturing packets, after that the captured packets are imported in Wireshark for analysis. To use tcpdump in command-line the following command will be used:

```
tcpdump -i eth -tt -v -w net.output host IP and tcp and not port ssh
# -i eth, is the interface to listen to
# -tt, print a timestamp
# -v, level 1 verbose
# -w net.output, file to write output to
# host IP and tcp and not port ssh is the capture filter
# used to capture only not-ssh-TCP packets from and to that IP
```

SAGE keeps a log during streaming that can be used for my analysis. The output is a framenummer, the bandwidth used for streaming the video to the display and bandwidth available for compression. Furthermore it outputs the frames per second (fps) compressed and sent to the diplay.

By using this setup, later analysis can be done on the output. A graph of the tcpdump can be made with the IO graph option in Wireshark, to display the traffic behavior. By using gnuplot graphs can be made of the load and the SAGE log.

3.2 Filesystem tests

In the GlusterFS tests the read throughput will be measured, with the following tools:

- dd
- iozone

Dd is a tool to convert and copy a file, according to the man page[10]. This tool is also a standard tool available on Unix and Linux systems.

The output of a copy operation can be used to measure the overall read throughput on block level. A block is a sequence of bytes or bits, with a nominal length (a block size). Most file systems are based on a block device, which is a level of abstraction for the hardware responsible for storing and retrieving specified blocks of data[11].

Different block sizes can have different throughput results.

Dd is used in the following manner:

```
dd if=inputfile of=outputfile/device
# if=inputfile, specify the file that will be used to read from
# of=outputfile/device, specify the file or device to sent the file to
# in this case /dev/null to measure only the read and not the write
# the output of this command will be a summary of the speed taken to copy file
```

Iozone[12] is a filesystem benchmark tool. This tool works on the file level.

File level storage is exchanged between systems in client/server interactions. The client uses the file's name and is not interested how the file is saved. The server performs the block level storage operations to read or write the data from a storage device[13].

This tool is used to benchmark how the read throughput is with the filesystem overhead.

The syntax and options used are:

```
iozone -e -i0 -i1 -r 16kb -s 7G -f iozone.tmp
# -e, flush timings are included in the results
# -i0, this is always needed, to make the file for the rest of the tests
# -i1, measure the performance of reading a file
# -r 16k, block size used
# -s 7G, size of the file used in the tests
# -f iozone.tmp, temporary file that was made with -i0 for the -i1 tests
```

In the example of *iozone* above I use a filesize of 7GB, this is also the filesize that is used in the tests with *dd*. I use a 7GB file to make sure that no caching influences the test results and that I get the throughput speed.

A rule of thumb is use $2\times$ the amount of RAM in the system to avoid caching[14].

I use the two tools above to measure different aspects of the read throughput:

- *dd* is a PIO transfer meaning the data is first sent through the processor, this measurement is done on block level;
- *iozone* is used to measure read throughput on file level.

The servers that I will use for my tests are part of the *Rembrandt cluster*[15].

The Rembrandt cluster is a cluster owned by UvA, dedicated to support the OptIPuter node at Amsterdam Lighthouse, the research lab at SARA.[15]

Nodes 1-7 in the Rembrandt cluster will be used in this project, the components that are relevant to my research project are:

- OS: Fedora Core 6
- Network: Gigabit Ethernet (GigE)
- Processor: Dual 64-bit CPUs (Opteron, 2.0 GHz)
- Memory: 4 GB (Rembrandt5 has less memory: 1 GB)
- Storage: Hardware-RAID (Serial ATA disk 250 GB \times 11, RAID 0)

4 SAGE vs NTT jpeg2000 codec

4.1 SAGE

Scalable Adaptive Graphics Environment (SAGE)[2] is specialized middleware for enabling data, high-definition video and high-resolution graphics to be streamed in real-time, from remotely distributed compression and storage clusters to tiled displays over ultra-high speed networks.[16] It can also be seen as a graphics streaming architecture. This architecture was primarily developed with the thought of supporting collaborative scientific visualization environments and allow user-definable layouts on the displays.[17]

I will use SAGE in this research project to stream videos to one high-definition display.

The architecture of SAGE, shown in figure 2, consists of the following components:

1. Free Space Manager (FSManager)
2. SAGE Application Interface Library (SAIL)
3. SAGE Receivers
4. UI clients

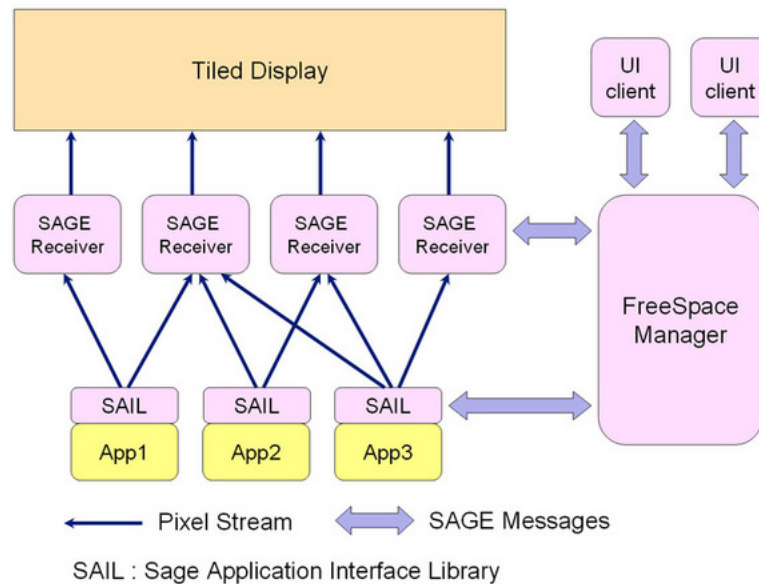


Figure 2: SAGE Architecture. Source: [2]

The *FSManager* is the window manager for SAGE. It can be compared with the Xserver on Linux systems. One of the most important differences with the Xserver is that this manager can scale to many more displays. This component manages the displays by receiving window positions and other commands from the UI clients and sends control messages to SAIL nodes or the SAGE receivers to update the displays.

SAIL is the layer between an application that is streaming and the displays. This component communicates with the *FSManager* and streams directly to the SAGE receivers. By using this component it becomes easy to program and use “standard” applications in the streaming architecture.

SAGE Receiver is a physical computer connected directly to one or more displays. This receiver receives commands directly from the FSManager or indirectly by receiving the streams from SAIL.

UI Clients can be a Graphical User Interface (GUI), text-based console or tracked devices. A user can use this to control the stream to the tiled displays by sending control messages (commands) to the FSManager and the UI client receives the status of the streams and displays.

SAGE can stream content with TCP or UDP.

TCP is used for short distances because the latency is negligible in retransmission, after a lost packet or NACK. UDP on the other hand is used in longer distances to stream content. The latency in longer distances is too great to retransmit without disturbing the displayed stream.

SAGE uses DXT to compress and decompress video, meaning it compresses and decompresses every frame independently. The output of the codec process is a sequence of still images.

DXT produces a compressed stream bandwidth of 250 Mb/s for a 1080p HD and 800 Mb/s for a SHD stream.[18]

SAGE functions in the following manner:

1. FSManager makes an ssh connection to the SAGE Receiver(s), to start the process that receives the stream, uncompresses it and streams to the display;
2. FSManager also sends the streaming commands to SAIL;
3. A streaming application compresses the stream and sends it with the help of SAIL to the Receiver.

4.1.1 Test Method

The infrastructure that I used in the tests is displayed in figure 3.

In this setup Rembrandt4 is the FSManager. The application that streams the video with SAIL is also available on Rembrandt4. The mac-mini is the SAGE receiver that sends the stream to the display.

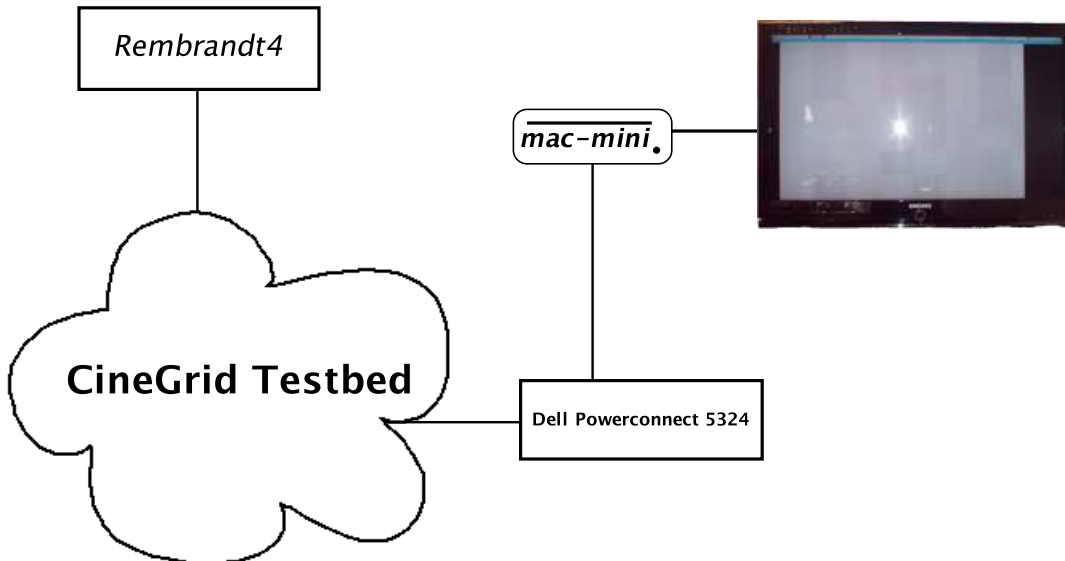


Figure 3: Infrastructure used in SAGE tests.

Doing the tests with SAGE, I made the following assumptions:

- tcpdump captures all packets in the limit of bandwidth;
- top is non-destructive, meaning it does not influence the SAGE setup.

The tools described in section 3.1 on page 8 will be used in the following manner in the SAGE tests:

top This tool will be executed on Rembrandt4 and the Mac-mini to register the percent of the CPU used by the SAGE utilities.

tcpdump This tool will also be executed on Rembrandt4 and Mac-mini to capture the SAGE packets, this tool is used to measure the bandwidth usage, lost packets (retransmissions) and it will be used to graph the traffic signature, the tools are executed on both endpoints to see if there is change on the way to the endpoints.

SAGE log The log is kept on the manager, this logs the compression bandwidth, bandwidth to the display, frames per second compressed and send to the display. This log is taken from the application layer, the results can be used to compare the bandwidth given by tcpdump.

The two tests that I have described above are the basic tests that I will use to get all the information to later analyze and document. I had the idea to run the tests five times to make sure that no unknown variables would influence the stream. But after doing the UDP stream test five times it was noted that doing the test five times was not necessary and two times would also suffice because the tests were done in a stable controlled environment.

I first made a script to automate the tests but when I started the tests, it was noticed that the use of the script takes too much processor time and therefore influences the stream by diminishing the amount of frames compressed and sent per second.

This would suggest that the compression process and streaming can be easily influenced.

That is why the script was eventually not used, the commands were executed in separate tests.

Two different videos were used in the tests:

7Bridges This is a video made from a boat while travelling under several bridges in Amsterdam.

PragueTrain This one is a video of a steam locomotive passing by in Prague.

In table 4.1.1 I give the similarities and differences between these two videos.

Video	Length (secs)	Size (GB)	Rate (fps)	Media Format
<i>7Bridges</i>	138	4.3	30	1080p HD
<i>PragueTrain</i>	97	2.5	24	1080p HD

Table 2: Video differences

I used these videos to see if there would be difference in the output, but no difference was noted. That is why the PragueTrain video will be used for further analysis in section 4.3 on page 15. I find the Rate difference, when looking at the output, negligible.

I chose this video because it is shorter, making it possible to gather more results in a shorter amount of time.

4.2 NTT jpeg2000 codec

NTT jpeg2000 codec has a different approach from SAGE, in the sense that it uses hardware encoding and decoding instead of software-driven. This approach is taken because a hardware implementation is faster than a software implementation in the setup of NTT jpeg2000 codec. Jpeg2000 is the method used to compress and decompress the video stream. Jpeg2000 is an intra-frame codec just as DXT in SAGE.

The NTT setup has primarily two components: “jpeg2000 real-time codec” and “jpeg2000 server”. The jpeg2000 real-time codec is the component that compresses and decompresses the data at a rate of 250 Megapixels per second, with a compressed data rate of up to 500 Mbps.[3] The jpeg2000 real-time codec is a hardware node with the following components:

- OS: Linux
- Network: Gigabit Ethernet (GigE)
- Video Encoding/Decoding: 4× PCI-X JPEG 2000 codec boards with JPEG 2000 processing chips and 1 hardware real-time clock card to sync the cards[3]

This hardware node is directly connected to the displays; the maximum amount of displays that one jpeg2000 real-time codec can handle is four.

As can be seen above this component only compresses and decompresses it doesn't store the data, for that purpose one needs the jpeg2000 server.

In iGrid 2005[19] they used a component called jpeg2000 server[3]. This hardware node does the storage and sending the compressed video to the jpeg2000 real-time codec. The components are:

- OS: Linux
- Network: GigE
- Processor: Dual CPUs (Xeon, 3.0 GHz)
- Storage: Software-RAID (Serial ATA disk 250 GB×7, RAID 0, XFS file format)[3]

Looking at the specifications of jpeg2000 server above, it can be noted that it is not a storage server specific for the jpeg2000 codec, this would mean that any storage server can be used.

The jpeg2000 server is implemented in software, this is installed on node41[20]. This program receives a data transfer command from the jpeg2000 real-time codec, reads the data from the storage and writes the data periodically to the GigE network interface.[3]

I will use this server for storage and the sending of the compressed video to the jpeg2000 real-time codec.

The transport method used in the NTT jpeg2000 codec is UDP.

4.2.1 Test Method

Due to unforeseen circumstances, I was not able to do the NTT jpeg2000 codec tests.

If it was possible to do the tests I would have used top and tcpdump in the same manner as explained in 4.1.1 on page 12.

The infrastructure that I would have used in the tests is displayed in figure 4 on the next page. The NTT jpeg2000 codec is placed at SURFnet. That is why the SURFnet network would have been used to stream the video from node41, located at UvA, to the NTT jpeg2000 codec. Node41 runs the jpeg2000 server process, called mserver, and the NTT jpeg2000 codec runs the mdecoder process, to decode and display the stream.

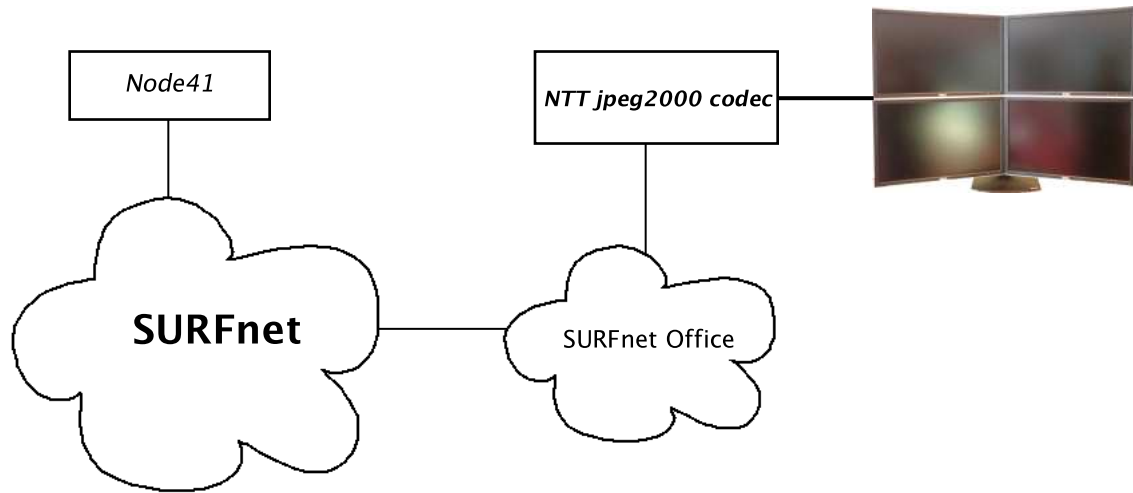


Figure 4: Infrastructure used in NTT tests.

4.3 Results

In this section I discuss the results from the SAGE setup. The tests were done several times but below I will only explain one test per subject, if there were noticeable differences between the tests I will point them out.

The results will be explained by means of graphs.

4.3.1 SAGE

The logs discussed below were taken from the SAGE Manager/Renderer. This is the node that compresses and sends the stream to the node connected to the display. Figure 5 on the following page gives two graphs: TCP stream of the SAGE Manager/Renderer and the UDP stream. The variables that are displayed are:

Display This is the network bandwidth used from the Manager/Renderer to the display.

Render The SAGE Manager/Renderer's internal bandwidth used to compress a video.

The Y-axis displays the amount of Mbps, the X-axis displays the seconds.

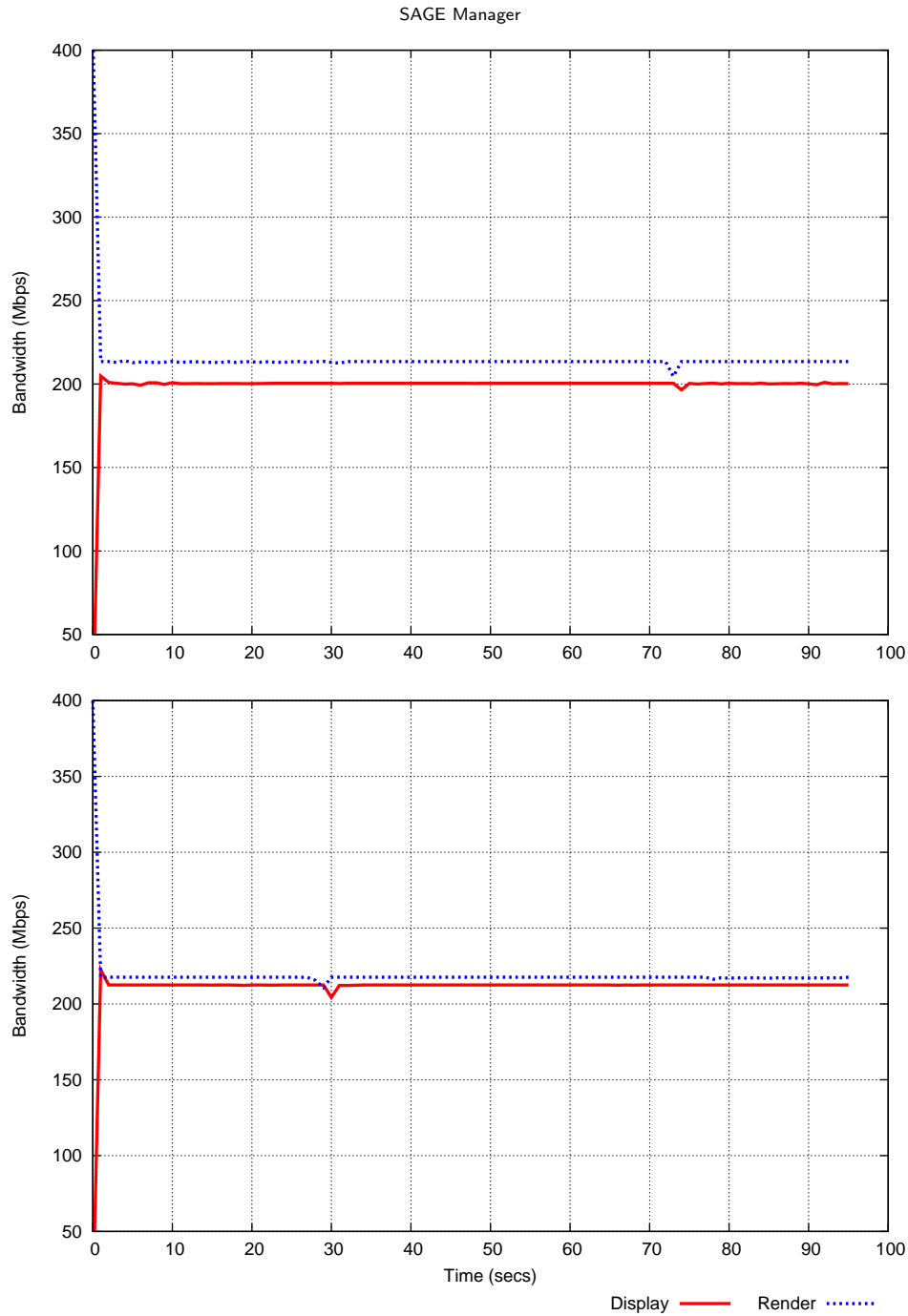


Figure 5: Log SAGE Manager. T: TCP stream, B: UDP stream

When the SAGE Manager/Renderer is started, the compression bandwidth starts at about 450 Mbps to fill the buffer. The display bandwidth starts at about 0 and reads from the buffer with an increasing speed, till a point is reached where the buffer is read from almost as fast as it is written to. In the graphs above it is clear that the compression bandwidth has direct influence on the bandwidth to the display, this would mean that if there is some influence on the compression bandwidth it would be noticeable in some manner on the display.

The compression bandwidth has also, but in lesser manner influence on the frames per second

compressed and sent.

I believe that the gap between Render bandwidth and the Display bandwidth can be contributed to TCP or UDP packet processing. In the UDP graph the gap is less than in the TCP graph because UDP takes less processing.

The read operation from the buffer has 1 sec delay from the write operation, if this buffer size is increased it might increase the reliability of the stream by softening the effect on the display bandwidth of, forexample drops in the compression bandwidth.

I would suggest to increase the buffer size and the delay so that the bandwidth to the display is not influenced. This could be applied in both TCP and UDP cases.

Now I will discuss the findings that have to do with the CPU load on the different components of SAGE in the test setup. Figure 6 displays the CPU load graph of a UDP stream on the SAGE Manager/Renderer.

I think that I can contribute the variations in the graphs to sampling in top.

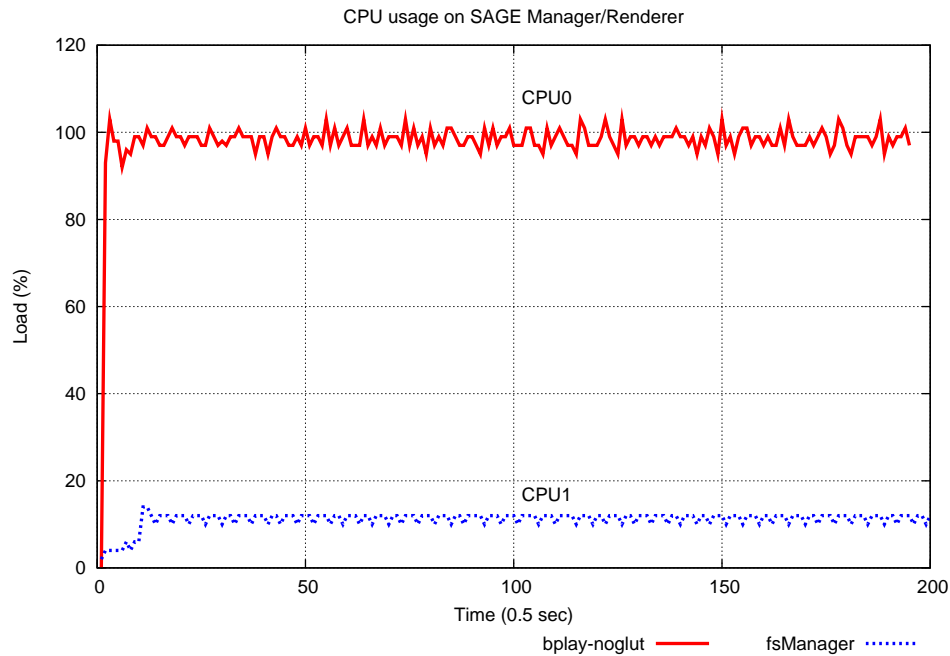


Figure 6: CPU load UDP stream SAGE Manager/Renderer

SAGE has two processes running on the Manager/Renderer these processes are:

fsManager is the Free Space Manager to control the displays by sending control messages.

bplay-noglut is the application used to compress and stream the videos.

These two processes run on different CPU's, by multi-threading the bplay-noglut process the load can be divided among available CPU's

On the display node side there is only one process that receives and sends the stream to the display.

This process is *sageDisplayMana*.

Below in figure 7 on the following page the graphs of a TCP stream and a UDP stream are displayed side by side for comparison.

There is an increase of about $\approx 14\%$ when a TCP stream is used. This is due to TCP processing, when a UDP stream is used the display node only receives a stream, but when a TCP stream is used the display node also needs to send ACK's and keep track of the sequence numbers.

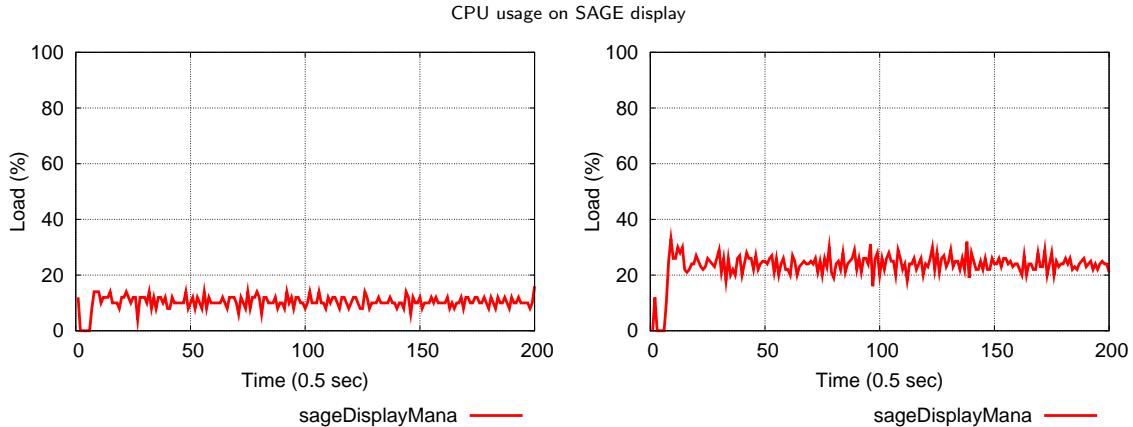


Figure 7: CPU load Display node. L: UDP stream, R: TCP stream

Now that I have discussed the SAGE log and the CPU load, I will be discussing the findings from the tcpdump analysis.

I will start with a summary of the amount of Mb/s send from the SAGE manager node to the display node:

- UDP: Average 218.4 Mb/s
- TCP: Average 208.1 Mb/s

Looking at these numbers I can conclude that they reflect the results of the SAGE Manager logs above.

The documented compression rate of SAGE is 6:1.[18]

Using the averages above and table 2 on page 7 I get a compression rate of 6.8:1 for UDP streams and 7.2:1 for TCP streams. It comes close to the documented compression rate, the difference might be caused by different measuring methods.

The ratio of SHD to 1080p HD is 4:1.

The averages above \times the ratio gives me 873.6 Mb/s for a compressed SHD UDP stream and 832 Mb/s for a compressed SHD TCP stream.

The NTT jpeg2000 codec setup streams SHD which is 6 Gb/s, this divided against the documented 500 Mb/s[3] compressed stream, gives a compression rate of 12:1.

Given these results are calculated correctly it would seem that NTT has a greater compression rate. The Δ is ≈ 300 Mb/s. This would mean that the hardware implementation of NTT jpeg2000 codec setup has a greater compression rate than the software implementation of SAGE.

After analyzing the traffic captured I concluded that the UDP streams are bursts of about 0.01 sec of 110Kb per 0,001 sec. After graphing the UDP stream at 0.01 sec I noted a peculiar fluctuation in the stream, it is displayed below in figure 8 on the following page. This fluctuation is also visible in the TCP streams.

I expected to see a constant amount of bits sent and/or received in the UDP and TCP stream. After some consideration I concluded that these fluctuations are not TCP or UDP bound, but are rather software or hardware boundaries. Boundaries that could have influence on the fluctuations displayed are:

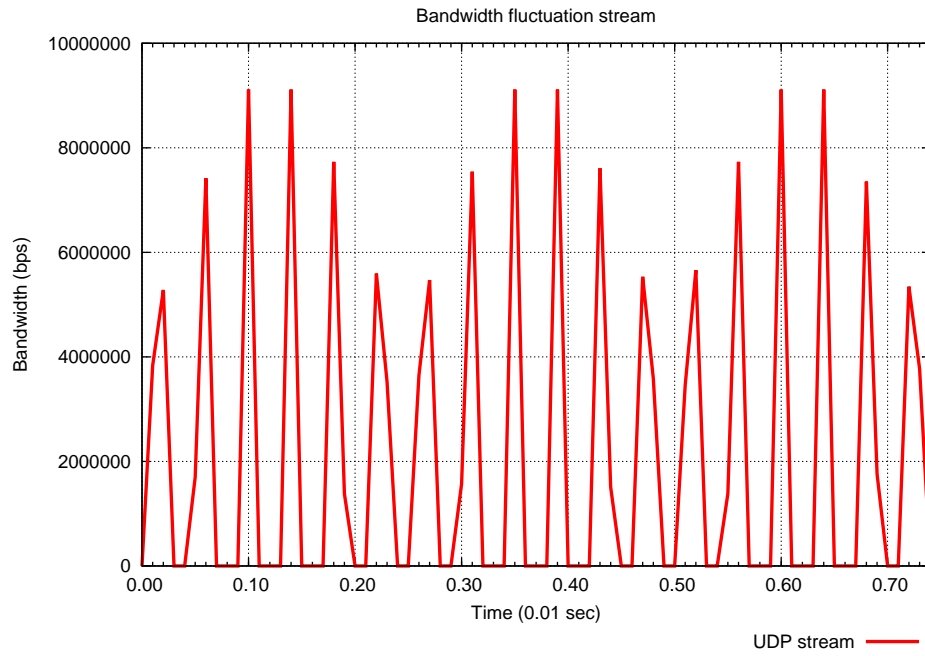


Figure 8: Stream fluctuation

- Queue and buffer of the network card;
- Time interval that tcpdump captures packets and time interval that packets are sent.

5 GlusterFS

This is the second part of my research project. Below I will give an overview of the current situation with NFS.

GlusterFS architecture and installation will be discussed afterwards and the test methods and results will be discussed.

5.1 Current situation

The setup that is currently used to store videos is to store them locally or by storing them on servers with NFS. NFS stores the videos independently on different servers, they are accessed with the NFS protocol.

These approaches have both their disadvantages when it comes to CineGrid. Local file disadvantages are, that the storage capacity does not increase fast enough to hold the terabytes of data sent and received from CineGrid. The storage bus speed is less than the network speed.

NFS removes the short coming of bus speed of local storage but has other disadvantages, one of the disadvantages is that the servers are accessed individually and the storage capacities are individually sorted meaning that they can not be summed up to increase the storage capacity.

5.2 GlusterFS

GlusterFS uses the idea of using clusters of independent storage units and combine them into one large storage server. This concept can have performance increase compared to other networked filesystems, because every node has its own CPUs, memory, I/O bus, RAID storage and interconnect interface. These units can have an aggregated performance increase. GlusterFS is designed for linear scalability for very large sized storage clusters.[21]

GlusterFS consists of different components:

GlusterFS Server and Client The server has the storage capability and exports the predefined storage amount so that the client can mount it.

GlusterFS Translators Translators are used to extend the filesystem capabilities through a well defined interface.

GlusterFS Transport Modules Transport Modules are implemented by the Translators and this module defines the method of transport. The methods defined at this moment in GlusterFS are TCP/IP, Infiniband userspace verbs (IB-verbs) and Infiniband socket direct protocol (IB-SDP).

GlusterFS Scheduler Modules The Scheduler Module does the load balancing between the different storage servers.

GlusterFS Volume Specification Volume Specifications are configuration files to connect the servers to the clients.

The design of GlusterFS is displayed in figure 9 on the next page. The storage bricks in the figure are actually the individual storage nodes. The storage clients are the clients that mount the storage system to make use of it.

On the bricks and the clients GlusterFS must be installed; there is one installation package for both server and client. By using different commands you can choose for both server and client executables or only one of them.

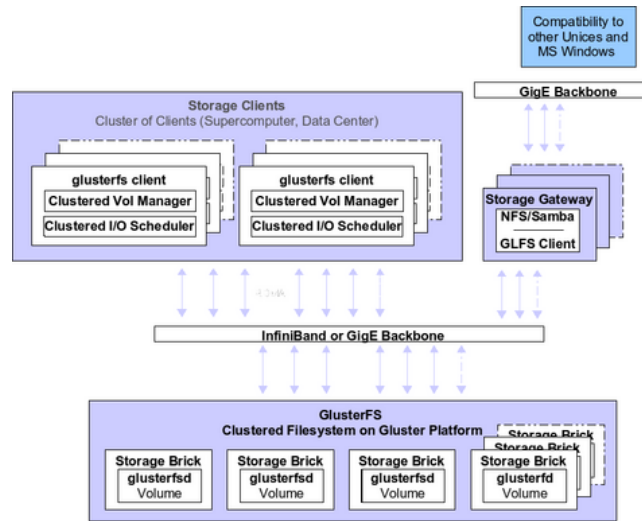


Figure 9: GlusterFS storage design. Source: [21]

5.3 Test Method

For testing the performance of a local file, NFS and GlusterFS, I first performed a `dd` and `iozone` of the local file system to have a base measurement. This base measurements was taken to calculate the performance improvements achieved with GlusterFS.

After taking the base measurements of the local filesystem I will also take the measurement of NFS to have an idea of the performance of the current setup. I will use Rembrandt1 as NFS client and Rembrandt2 as the NFS server.

In the tests I use different block sizes to see if there are differences in throughput.

GlusterFS has many “Translators” that can be used to configure the filesystem to fulfill specific storage needs. My research of GlusterFS will focus on performance improvements. The different translators that can improve performance are:

1. Performance Translators
 - (a) Read Ahead Translator, improve reading by prefetching a sequence of blocks
 - (b) Write Behind Translator, to improve the write performance
 - (c) Threaded I/O Translator, is best used to improve the performance of concurrent I/O
 - (d) IO-Cache Translator, improves the speed of rereading a file
 - (e) Booster Translator, removes FUSE (Filesystem in USERSpace) overhead
2. Clustering Translators
 - (a) Automatic File Replication Translator (AFR), mirroring capability
 - (b) Stripe Translator, striping capability
 - (c) Unify Translator, this sums up all the storage bricks and present them as one brick [4]

The stripe translator works a bit like RAID 0, in the sense that it divides the input file in blocks and saves that block on individual servers (disks in the case of RAID 0). This is done in the round-robin fashion. In theory this method is the best method to increase read performance, that is why I will use this method in my tests.

The other performance translators can later be added for more performance increase. I will only use the stripe translator, this will give an idea of the minimum performance increase that can be

achieved when using GlusterFS.

The installation of GlusterFS is discussed in Appendix A on page 29.

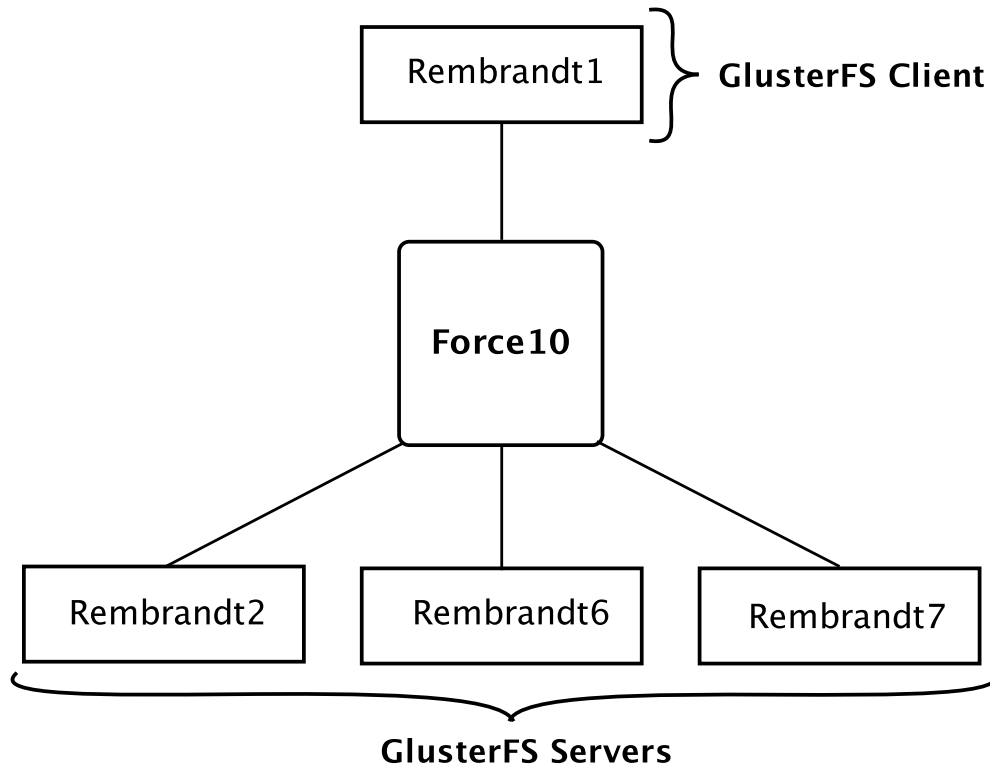


Figure 10: Infrastructure used in GlusterFS tests.

In figure 10 a schematic picture is given of the setup used in the storage tests.

In the setup displayed above I will use Rembrandt1 as the GlusterFS client and respectively Rembrandt2 for 1 server brick test, Rembrandt 2 and 6 for the two server test and Rembrandt 2, 6 and 7 for the three server test.

The setup used in the three server setup is displayed in Appendix B.1 and Appendix B.2.

5.4 Results

In this section I will be discussing my findings in the storage performance tests. Table 5.4 displays the results of the tests; the results are in MB/s.

The results displayed in the table are the results of doing each test twice and taking the bandwidth results with the maximum MB/s.

Local									
<i>Block Sizes</i>									
<i>Tool</i>	4KB	16KB	128KB	256KB	512KB	1MB	4MB	8MB	16MB
dd	85.9	87.4	90.9	90.4	90.4	89.4	87.4	86.2	88
iozone	83.1	84.3	89.7	86.6	86.6	84.1	89.4	86.1	83.1
NFS									
dd	73.8	74.7	74.8	74.2	74.3	74.4	73.8	74.6	74.6
iozone	73.8	74.6	74.6	74	74	73.3	74.2	73.6	73.9
GlusterFS 1 Server									
dd	56	54.5	54.8	53.7	54.6	54.5	54.6	54.2	54.6
iozone	53	55.3	56.5	56.5	54	54.8	53	53.9	54.6
GlusterFS 2 Servers									
dd	56.3	55.9	55.2	55	54.9	54.8	55.2	54.9	54.9
iozone	52.6	52.4	52.5	51.5	51	53.1	52.3	53.3	52.9
GlusterFS 3 Servers									
dd	104	105	105	104	103	103	103	104	103
iozone	87.6	89.1	86.8	92.9	92.4	90.2	87	86.6	86.7

Table 3: Maximum read speed: Local, NFS, GlusterFS

The results of dd and iozone must be handled separately because they are not results of the same test. Dd and iozone have different ways of testing the storage system making it impossible to put the results of dd and iozone in the same context.

The results of dd are theoretically higher than results of iozone because dd does not take the filesystem overhead into account.

In some of the results in the table above the dd is equal or lower than iozone. The reason for this can be that the dd and iozone tests are done separately or the filesystem is also accessed in dd including the filesystem overhead in the dd results. If the filesystem overhead is minimal, this can also be reflected in the results by means of near equal results.

After extensive testing of the setup described in section 5.3 on page 21, which the results of the maximum read speeds are displayed in 5.4. I could not find an optimal block size to achieve maximum read speed. There might be need to do tests on more servers in different environments to get an optimal block size.

To get a better view of the performance loss or gain I will use the results of 4KB to compare the results against the local filesystem, I use the 4KB results because it is the default block-size in most Linux OS's.

1. NFS:
 - dd 14.1% loss
 - iozone 11.2% loss
2. GlusterFS 1 server:
 - dd 34.8% loss
 - iozone 36.2% loss

3. GlusterFS 2 servers:
 - dd 34.5% loss
 - iohome 36.7% loss
4. GlusterFS 3 servers:
 - dd 21% gain
 - iohome 5.4% gain

Following the results above I can say with certainty that GlusterFS can surpass the read speed of the local filesystem, thus three or more server bricks are used.

It is interesting to note that there is minimal difference between the one server and the two server setup. I think the reason for this is that changing from one server to two servers with striping increases the network and filesystem overhead making it seem like there is no improvement in the results.

For real improvements a minimum of three server bricks must be used, if less servers are available it is better to keep using the local filesystem or NFS when talking about read performance.

After I concluded this, a chance was opened up to do GlusterFS tests on four and five servers. I thought it would increase when more servers were used. But I did not see increase in the read speed which I found strange. I thought maybe the maximum that one thread on the client can receive was reached, that is why I increased the amount of GlusterFS threads on the client, then on the server.

But no improvement was noted.

After discussing this problem with other members of SNE it was concluded that the 1 Gb connection between the GlusterFS client and the servers was full. Taking the bandwidth results of the three GlusterFS server setup, multiplying it with 8 to get the amount of Mb, and adding TCP overhead we immediately see that the total bandwidth is close to the 1 Gb maximum.

The servers that I used as bricks were all homogeneous, when I used a system with less memory a decline was noted in the read performance.

This would point out that GlusterFS is best used in a homogeneous architecture to get maximum performance.

On a side note I don't think GlusterFS is yet ready to be used in a distributed manner over a network connected to the Internet, the reason for this is that the version now available, version 1.3, does not include encryption. However they are planning in incorporating this feature in future releases [4].

Encryption is not a prerequisite in the architecture of CineGrid, as long as dedicated paths are used.

The observation that I have just given do not have consequences on the the CineGrid network and on my conclusions, but I thought it might be useful to keep this in mind when architecturing a GlusterFS network.

5.5 Results w/ Performance Translators

After it became clear that the 1 Gb connection between the GlusterFS client and GlusterFS servers was the bottleneck, the connection was upgraded to a 10Gb connection.

With this upgrade I decided to include some Performance Translators to give an idea of the performance increases that can be achieved with GlusterFS when using Performance Translators. The Performance Translators that I used were:

- Read Ahead Translator
- Threaded I/O Translator

I also included Rembrandt3 and Rembrandt4 to get results of four and five GlusterFS servers. In Appendix B.3 and Appendix B.4 I give the configurations that I used. The results in table 5.5 are given in the same manner as the results in table 5.4 on page 23.

GlusterFS 1 Server									
<i>Block Sizes</i>									
<i>Tool</i>	4KB	16KB	128KB	256KB	512KB	1MB	4MB	8MB	16MB
dd	89.9	95.6	83.4	91.4	93.3	88.4	83.4	79.7	91.7
GlusterFS 2 Servers									
dd	103	119	122	122	123	123	121	118	118
GlusterFS 3 Servers									
dd	294	323	323	304	265	262	259	269	261
GlusterFS 4 Servers									
dd	306	336	326	303	263	305	260	263	278
GlusterFS 5 Servers									
dd	315	331	333	308	270	260	260	265	260

Table 4: Maximum read speed: GlusterFS with Performance Translators

As you can see above, there is a big difference between GlusterFS results without Performance Translators and GlusterFS results with Performance Translators. This would suggest that it is a good idea to use different Translators for different storage needs.

Below I also give the performance gains in percentages compared to the Local Filesystem read speed.

1. GlusterFS 1 server:
 - dd 0.8% gain
2. GlusterFS 2 servers:
 - dd 19.9% gain
3. GlusterFS 3 servers:
 - dd 242.3% gain
4. GlusterFS 4 servers:
 - dd 256.2% gain
5. GlusterFS 5 servers:
 - dd 266.7% gain

6 Future Work

Now that you reached the end of my report on SAGE, NTT jpeg2000 codec and GlusterFS, I would like to make some suggestions for future work.

The NTT jpeg2000 codec test must be done, in the same manner I used in this document, to make a comparison between SAGE and NTT jpeg2000 codec.

I would also suggest to build a SAGE setup that streams SHD to make a realistic comparison between the SAGE setup and the NTT jpeg2000 codec setup.

The use of tools like strace can give a better understanding of the compression and streaming process of SAGE, and make it possible to make improvements to the SAGE setup.

A research project can investigate packet drops and increase the reliability of the streaming process. I did not discuss packet drops in this document, but it was brought to my attention during the research project that packet drops directly influence the stream displayed.

At the moment no clear explanation is available for the packet drops.

Regarding GlusterFS, I think it can be implemented in a test network to look at its stability over long term.

If the stability is satisfactory, it should be implemented in the production network of CineGrid. By using GlusterFS in CineGrid it can increase the read performance of the Rembrandt cluster when streaming over long distances.

There should also be some research on the different Translators to see which combination has the most performance increase.

7 Conclusion

I will mention the conclusions in the same order as this document.

SAGE is a implementation in the spirit of open source. It can be used on different Linux systems, but because it is implemented in software, it heavily depends on the hardware it runs on.

I assume the calculations made with the SAGE results to get SHD stream are correct.

The compression rate of NTT jpeg2000 codec is greater than the rate of SAGE. The difference between the setups is about ≈ 300 Mb/s.

The tests with NTT jpeg2000 codec were not possible, which means that I could not make an overall comparison.

NTT jpeg2000 codec is implemented in the hardware. This has the advantage over SAGE that the hardware does not influence the stream.

Although NTT jpeg2000 codec is used in many different CineGrid installations, there is near to no documentation in English on the Internet, most of the documentation is in Japanese. On the other hand SAGE has been documented in many academic papers available via their website[2].

I conclude that GlusterFS can be implemented in a test setup, to look at the stability over long term. The read speed of GlusterFS is greater than the local filesystem when three or more server bricks are used.

Good attention must be taken on the amount of bandwidth available when using GlusterFS, because it can easily become a bottleneck.

8 Acknowledgments

Dear reader,

First I would like to give my praise to God for giving me life and this opportunity.

I would like to extend my gratitude to my two supervisors Dr. P. Grosso and MSc R. Koning for their time and guidance.

I would also like to thank J. Roodhart for his help with GlusterFS.

SURFnet receives my thanks for their time spend in setting up the NTT jpeg2000 codec. Because of time restraints and unforeseen network problems I could not complete the tests with the NTT jpeg2000 codec setup.

I thank Dr. K. Koymans, J. van Ginkel and E. Schatborn for giving me the opportunity to do System and Network Engineering.

Dr. Ir. C. de Laat for the choice of this Research Project and his advice.

Last but not least I would like to thank my parents for their time spend reviewing this paper and my girlfriend for her patience.

With kind regards,

Sevickson Kwidama

References

- [1] Cinegrid. Cinegrid. Website. <http://cinegrid.org/index.php>.
- [2] Electronic Visualization Laboratory. Sage. Website. <http://www.evl.uic.edu/cavern/sage/index.php>.
- [3] Takashi Shimizu et. al. International real-time streaming of 4k digital cinema. *Future Generation Computer Systems*, 22, May 2006.
- [4] GlusterFS. Glusterfs. Website. <http://www.gluster.org/>.
- [5] Daniël Sánchez. Design of store-and-forward servers for digital media distribution. Master's thesis, University of Amsterdam, 2007.
- [6] GLIF. Glif: the global lambda integrated facility. Website. <http://www.glif.is/>.
- [7] Ralph Koning. Cinegrid on layer2 paths. Presentation at Terena Networking Conference, 2008. <http://tnc2008.terena.org/>.
- [8] André D. Balsa. Linux benchmarking - concepts. Website. <http://linuxgazette.net/issue22/bench.html>.
- [9] Gerald Combs et al. Wireshark: Go deep. Website. <http://www.wireshark.org/>.
- [10] die.net. dd(1) - linux man page. Website. <http://linux.die.net/man/1/dd>.
- [11] Wikipedia. Wikipedia, the free encyclopedia. Website. <http://en.wikipedia.org/>.
- [12] IOzone. Iozone filesystem benchmark. Website. <http://www.iozone.org/>.
- [13] Mark Farley. Block and file level storage. Website. http://searchstorage.techtarget.com/expert/KnowledgebaseAnswer/0,289625,sid5_gci548225,00.html.
- [14] Tim Verhoeven. Centos: How to test and monitor disk performance. Website. <http://misterd77.blogspot.com/2007/11/how-to-test-and-monitor-disk.html>.
- [15] Advanced Internet Research. Rembrandt cluster. Website. http://www.science.uva.nl/research/air/network/rembrandt/index_en.html.
- [16] Byungil Jeong et al. High-performance dynamic graphics streaming for scalable adaptive graphics environment. 2006.
- [17] Javid M. Alimohideen Meerasa. Design and implementation of sage display controller. 2003.
- [18] Jason Leigh Luc Renambot, Byungil Jeong. Realtime compression for high-resolution content. 2007.
- [19] *iGrid2005*, 2005.
- [20] Universiteit van Amsterdam. Cinegrid distribution center amsterdam. Website. <http://cinegrid.uva.netherlight.nl/>.
- [21] GlusterFS. Glusterfs user guide v1.3. Website. http://www.gluster.org/docs/index.php/GlusterFS_User_Guide_v1.3.

Appendices

A GlusterFS Installation

I first installed GlusterFS on my own test environment in OS3⁴, before installing it on the Rembrandt's at UvA. I did this to first understand how the GlusterFS installation works and how it can be configured. This was done in a test environment to minimize the impact that GlusterFS would have if there was some problem.

GlusterFS was first installed in the Ubuntu 8.04 DomU under Xen at OS3 but this installation was not successful, below I will discuss installation process and problems that I ran in while trying to get GlusterFS to work.

To install GlusterFS in Ubuntu I first took care of the dependencies by issuing the following command:

```
sudo apt-get install libtool gcc flex bison byacc
```

Also the kernel source (linux-source) are necessary if you want to compile the patched FUSE module for GlusterFS to use with the clients.

This FUSE module has improvements to increase the I/O throughput of GlusterFS.

The installation of GlusterFS can be done in two methods, compile from source or install from Debian package for Debian kernels or RPM and yum for ... kernels.

I used the compilation from source method to understand the installation process for GlusterFS. To install the server for GlusterFS and the client without the FUSE improvements the following commands were used:

```
wget http://europe.gluster.org/glusterfs/1.3/glusterfs-CURRENT.tar.gz
tar -xzf glusterfs-CURRENT.tar.gz
cd glusterfs-1.3.9
./configure --prefix= #without the --prefix I got problems with the libraries
```

```
##### This is the output of ./configure #####
#GlusterFS configure summary #
#===== #
## This line below informs that the client cannot be installed because FUSE #
## is not correctly configured #
#Fuse client : no #
## The line below is yes when Infiniband verbs is installed for transport, #
## default the transport method is TCP/IP #
#Infiniband verbs : no #
## This one is for the polling method for the server #
#epoll IO multiplex : yes #
#####
```

```
sudo make install
```

Like it is displayed above the installation does not install the client, this might be because FUSE is not installed or not correctly for configured to be used by GlusterFS.

Installation of FUSE was done with the following commands:

⁴SNE Master study. <https://www.os3.nl/>

```
wget http://europe.gluster.org/glusterfs/fuse/fuse-2.7.3glfs10.tar.gz
tar -xzf fuse-2.7.3glfs10.tar.gz
cd fuse-2.7.3glfs10
./configure --prefix=/usr --enable-kernel-module
make install
ldconfig
depmod -a
rmmod fuse # I receive an error when I try to remove the module, it says that it is in use
modprobe fuse
```

When I tried to install it on my Ubuntu 8.04 with kernel 2.6.24-19, I received the following error:

```
Making install in kernel
make[1]: Entering directory '/home/$user/fuse-2.7.3glfs10/kernel'
make -C /usr/src/linux-headers-2.6.24-19-generic SUBDIRS='pwd' modules
make[2]: Entering directory '/usr/src/linux-headers-2.6.24-19-generic'
  CC [M] /home/$user/fuse-2.7.3glfs10/kernel/dev.o
  CC [M] /home/$user/fuse-2.7.3glfs10/kernel/dir.o
/home/$user/fuse-2.7.3glfs10/kernel/dir.c: In function iattr_to_fattr:
/home/$user/fuse-2.7.3glfs10/kernel/dir.c:1027: error: struct iattr has no member named ia_file
make[3]: *** [/home/$user/fuse-2.7.3glfs10/kernel/dir.o] Error 1
make[2]: *** [_module_/home/$user/fuse-2.7.3glfs10/kernel] Error 2
make[2]: Leaving directory '/usr/src/linux-headers-2.6.24-19-generic'
make[1]: *** [all-spec] Error 2
make[1]: Leaving directory '/home/$user/fuse-2.7.3glfs10/kernel'
make: *** [install-recursive] Error 1
```

The conclusion that I made from this error is that Ubuntu made some changes in the function 'iattr_to_fattr' making it not possible to install the patched FUSE for the client in this version of Ubuntu.

That is why I downgraded my Ubuntu virtual machines to version 7.10 and I didn't receive this error anymore.

After successfully installing the GlusterFS server and client on Ubuntu 7.10 I tried these steps on the Rembrandt's. The Rembrandt's have Fedora Core 6 installed. When I tried to install it in my own home environment I got the following error:

```
test -z "/sbin" || mkdir -p -- "/sbin"
/usr/bin/install -c 'mount.glusterfs' '/sbin/mount.glusterfs'
/usr/bin/install: cannot create regular file '/sbin/mount.glusterfs': Permission denied
make[5]: *** [install-slashesbinSCRIPTS] Error 1
make[5]: Leaving directory '/home/$user/glusterfs-1.3.9/xlators/mount/fuse/utils'
make[4]: *** [install-am] Error 2
make[4]: Leaving directory '/home/$user/glusterfs-1.3.9/xlators/mount/fuse/utils'
make[3]: *** [install-recursive] Error 1
make[3]: Leaving directory '/home/$user/glusterfs-1.3.9/xlators/mount/fuse'
make[2]: *** [install-recursive] Error 1
make[2]: Leaving directory '/home/$user/glusterfs-1.3.9/xlators/mount'
make[1]: *** [install-recursive] Error 1
make[1]: Leaving directory '/home/$user/glusterfs-1.3.9/xlators'
make: *** [install-recursive] Error 1
```

First look at this error gives the idea that I'm trying to write in a directory that I don't have permission to write in, this is obvious because I only have write permissions in my own home directory.

But I receive this error also when I use my own home directory as environment for the /sbin. After discussing it with Jeroen Roodhart it was concluded that there was a bug in the Makefile. The bug is that the ./configure does not change all the path variables to the variables that I write as -prefix.

After this the GlusterFS worked on the Rembrandt's with no problem.

B GlusterFS Configuration

B.1 Server Configuration file

```
volume brick
  type storage/posix
  option directory /glusterfs/skwidama-exp
end-volume
```

```
volume server
  type protocol/server
  option transport-type tcp/server
  subvolumes brick
  option auth.ip.brick.allow 192.168.192.*
end-volume
```

B.2 Client Configuration file

```
volume remotel
  type protocol/client
  option transport-type tcp/client
  option remote-host 192.168.192.12
  option remote-subvolume brick
end-volume
```

```
volume remote2
  type protocol/client
  option transport-type tcp/client
  option remote-host 192.168.192.16
  option remote-subvolume brick
end-volume
```

```
volume remote3
  type protocol/client
  option transport-type tcp/client
  option remote-host 192.168.192.17
  option remote-subvolume brick
end-volume
```

```
volume stripe0
  type cluster/stripe
  option block-size *:1MB
  subvolumes remotel remote2 remote3
end-volume
```

B.3 Server Configuration file w/ Performance Translators

```
volume brick
  type storage/posix
  option directory /glusterfs/skwidama-exp
end-volume

volume iot
  type performance/io-threads
  option thread-count 2
  option cache-size 32MB
  subvolumes brick
end-volume

volume server
  type protocol/server
  option transport-type tcp/server
  subvolumes iot
  option auth.ip.iot.allow 192.168.192.*
end-volume
```

B.4 Client Configuration file w/ Performance Translators

```
volume remotel
  type protocol/client
  option transport-type tcp/client
  option remote-host 192.168.192.12
  option remote-subvolume iot
end-volume

volume remote2
  type protocol/client
  option transport-type tcp/client
  option remote-host 192.168.192.13
  option remote-subvolume iot
end-volume

volume remote3
  type protocol/client
  option transport-type tcp/client
  option remote-host 192.168.192.14
  option remote-subvolume iot
end-volume

volume remote4
  type protocol/client
  option transport-type tcp/client
  option remote-host 192.168.192.16
  option remote-subvolume iot
end-volume

volume remote5
  type protocol/client
```

B.4 Client Configuration file w/ Performance TranslatorsB GLUSTERFS CONFIGURATION

```
option transport-type tcp/client
option remote-host 192.168.192.17
option remote-subvolume iot
end-volume
```

```
volume stripe0
  type cluster/stripe
  option block-size *:1MB
  subvolumes remotel remote2 remote3 remote4 remote5
end-volume
```

```
volume readahead
  type performance/read-ahead
  option page-size 1MB
  option page-count 20
  option force-atime-update off
  subvolumes stripe0
end-volume
```