UMEÅ UNIVERSITET
Department of Instutition

April 6, 2023

Distribuerade system

Assignment 1

version 1.0

Name	Mathias Hallberg, Gustaf Söderlund
Username	c19mhg, et14gsd

Contents

1 Introduction

In this assignment we're supposed to implement two clients that receives data from a video application. One client using sequential transfer and another client using parallelism. The performance of transferring data from the video application to the client will be measured. In this assignment we decided to measure the packet latency, packet drop rate, frame throughput and bandwidth utilization. The video application is part of a distributed system, distributed over a set of hosts.

The data sent from the hosts are called a stream and each stream contains several frames where each frame contains a set of blocks and each block contains a set of pixels, RGB (red-green-blue).

2 System Explanation

The system is built using the given interface *FrameAccessor*. The *FrameAccessor* interface contains several other interfaces like *Frame*, *PerformanceStatistics* and *Factory*. These interfaces are implemented inside the three different classes *Factory*, *ParallelClient* and *SequentialClient*.

2.1 Factory

The *Factory* Class implements *FrameAccessor.Factory* and the purpose of this class is to distinguish if the system is going to use the sequential client or the parallel client. The Class contains two methods of *getFrameAccessor* and the purpose of the method is to return a FrameAccessor object.

Depending on if the user wants to use the sequential client or the parallel client. The arguments given to the method decides which frameaccessor is going to be used. The sequential client is chosen and returned if the StreamServiceClient argument is not a list of clients. If the argument is a list the parallel client will be returned instead.

2.2 ParallelClient

The *ParallelClient* class implements the *FrameAccessor* interface and has the purpose to pull data from the video application using different clients. The thread library that's being used in this class is *ExecutorService*. The idea of pulling data using threads is done by making a separate thread for each client. Where the number of blocks within a frame is fairly distributed to each thread. The threads can then retrieve data spontaneously and by making sure the system is synchronized, race condition wont happen during the execution.

The performance is analysed during the execution of threads and the measured values are safely stored within different *hashmaps*.

2.3 SequentialClient

The *SequentiallClient* class implements the *FrameAccessor* interface and has the purpose to pull data from the video application using one single client. The retrieval and the measurement of the performance is done in the same way as the *Parallelclient* but instead of creating threads for each clients the retrieval of data is done on the EDT.

3 Performance Analysis

In this section the analysis of the performance will be presented.

To measure the performance we ran a program that runs the sequential client eight times with the four different clients and the parallel is run 8 times with two, three and four clients. The performance is sent to a file to plot data. We only used one stream which had the least amount of frames and smallest size of the frame to make the testing doable.

We've decided to measure the performance of Sequential Client from different hosts to compare the difference between the hosts. When measuring the performance of parallel client we decided to compare the different number of hosts.

The performance is represented in box plot diagrams. In the plots we can tell that 50% of the the KPI is within the blue box, 25% of the KPI is within the quantiles(lines) and the yellow/orange line represents the median. The outliers is represented as a small circle.

3.1 Packet Latency

In Figure 1 we're presented by a box plot diagram showing the packet latency in the four different hosts using the Sequential Client.

By analyzing the statistics we can tell that the latency differs a bit depending on the host. The most consistent host seems to be *salt* and the most inconsisten host is *itchy*.

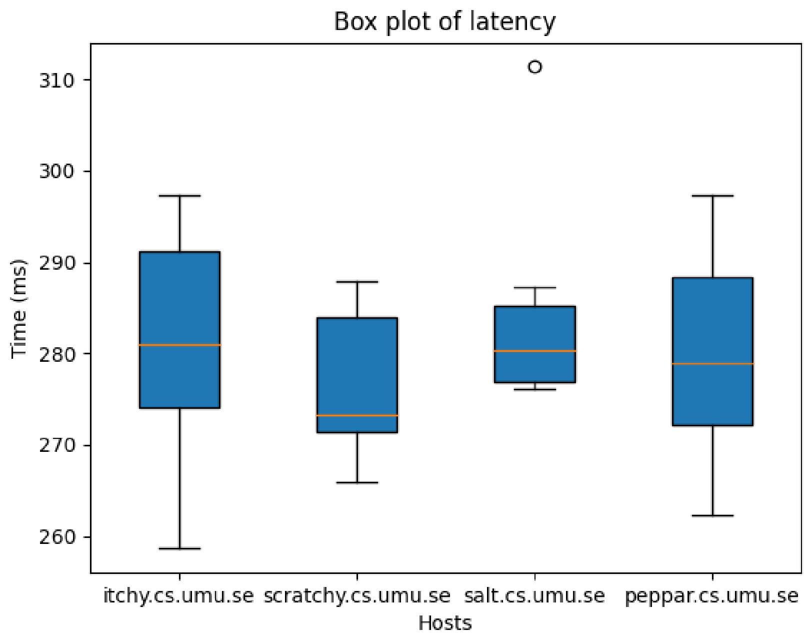

Figure 1: Representation of the packet latency with the four different hosts.

In Figure 2 we're presented by a box plot diagram showing the packet latency using the Parallel Client with different number of hosts.

By analyzing the statistics in Figure 2 we can tell that using three hosts we get a more consistent latency. The median is also at it's lowest around 193 ms which makes it more reasonable to use three hosts then two or four.

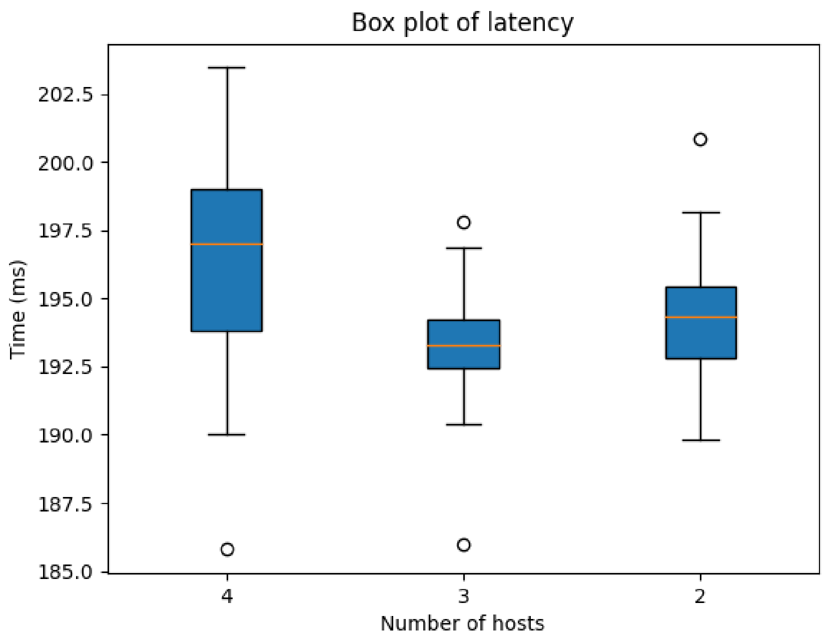

Figure 2: Representation of the packet latency with different of hosts.

3.2 Packet Drop Rate

In Figure 3 we're presented by a box plot diagram showing the packet loss of the four different hosts using the Sequential Client.

From these graphs we can tell that *salt* has a more consistent Drop rate than the rest of the hosts, *itchy* is more inconsistent and has a more spread.

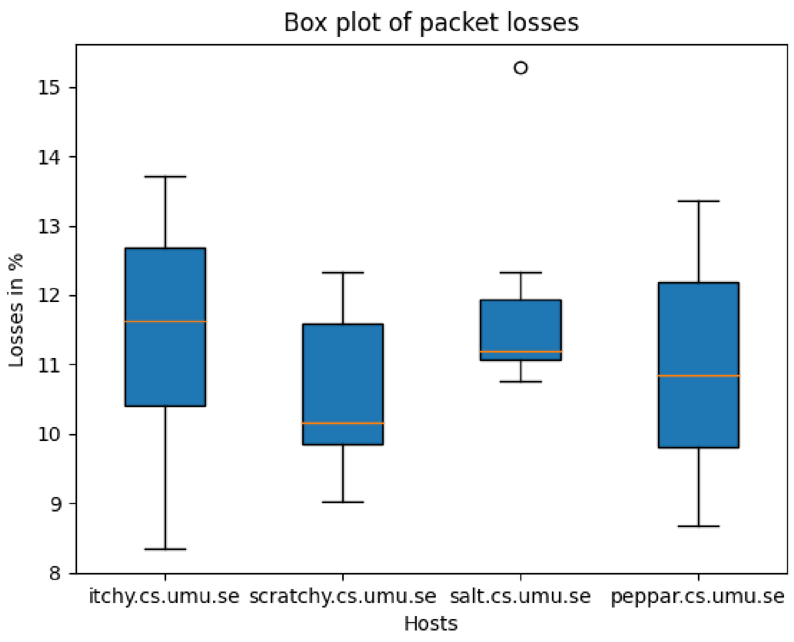

Figure 3: Representation of the packet losses from different hosts.

In Figure 4 we're presented by a box plot diagram showing the Packet drop rate using parallel client using different numbers of hosts.

From the statistics presented in Figure 4 we can see that the packet loss differs a little bit. We can see that three numbers of hosts is the most consistent one where 50% is between 9%- and 11% packet loss.

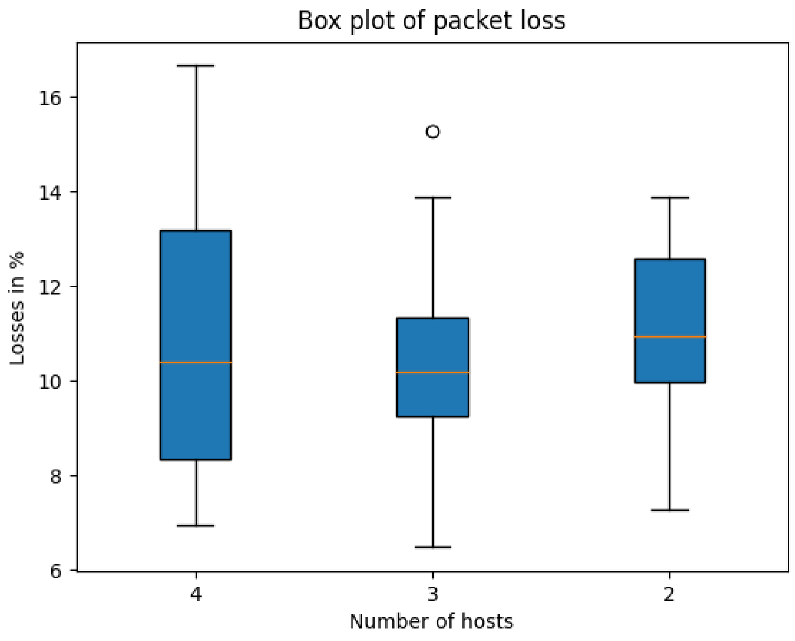

Figure 4: Representation of packet loss with different hosts.

3.3 Frame Throughput

In Figure 5 we're represented by a box plot diagram where the frame throughput has been measured using Sequential Client.

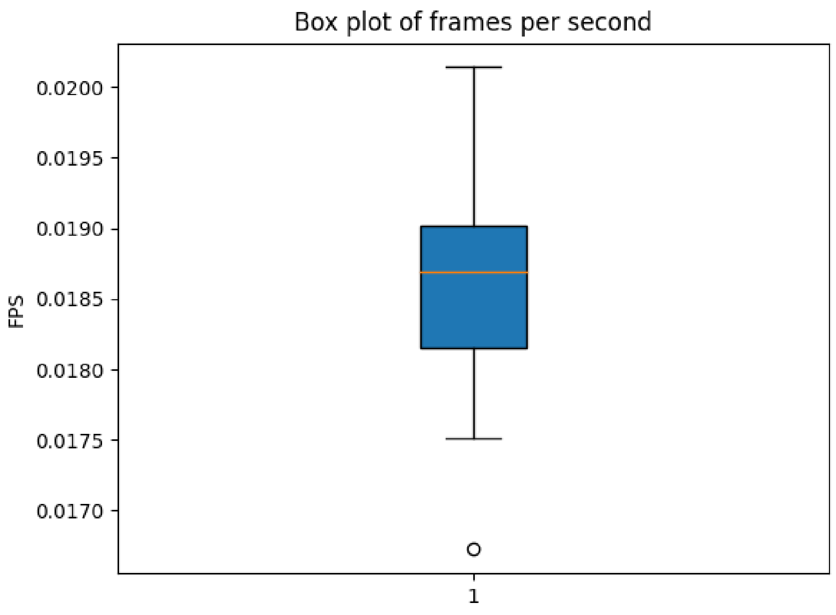

Figure 5: Representation of fps with sequential client implementation.

In Figure 6 we're presented a box plot diagram where the frame throughput has been measured using parallel Client with a different number of hosts.

By inspecting the data given of the statistics in Figure 6 we can see that that using two hosts gives us highest frames per second and most consistent frames per second.

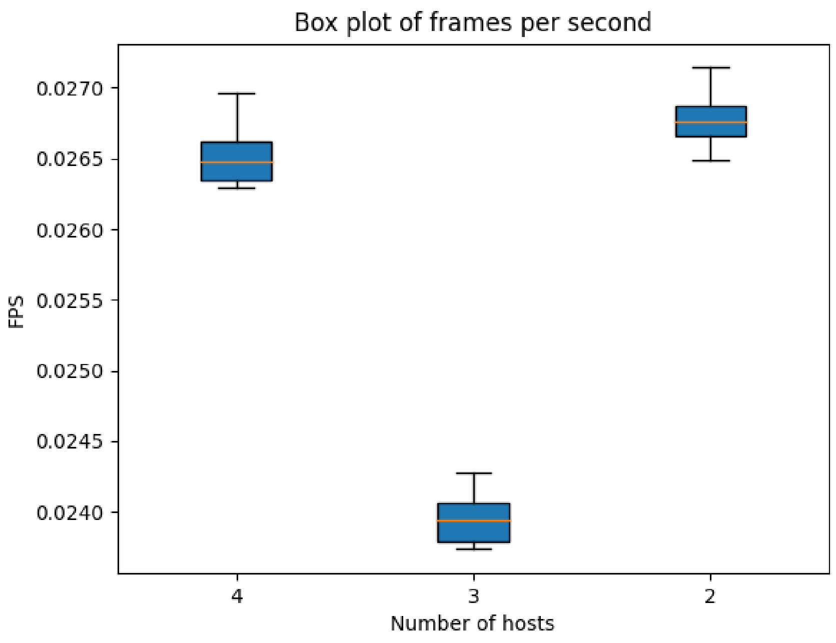

Figure 6: Representation of fps with parallel client and the amount of hosts used.

3.4 Bandwidth Utilization

In Figure 7 we're presented by a box plot diagram where the bandwidth utilization has been measured using sequential client.

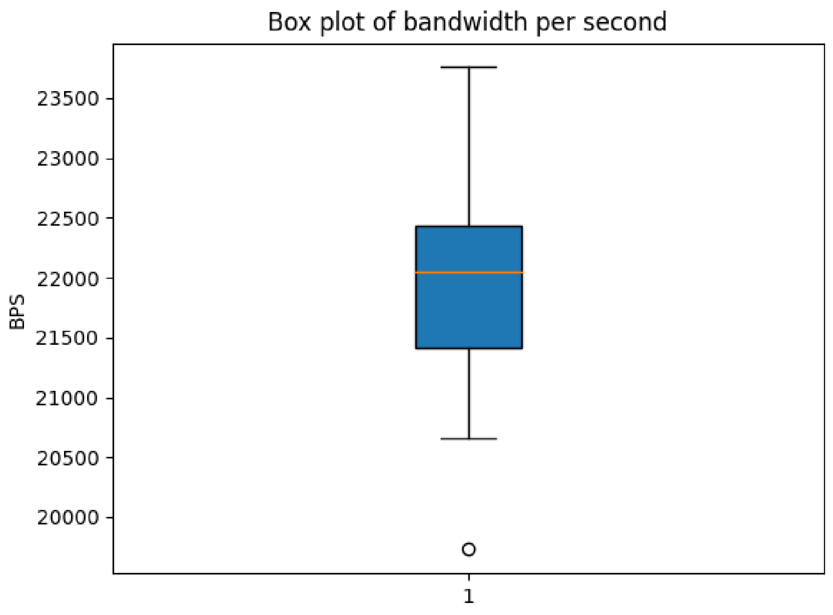

Figure 7: Representation of bps with sequential client implementation.

In Figure 8 we're presented by a box plot diagram where the bandwidth utilization has been measured using parallel client using different number of hosts.

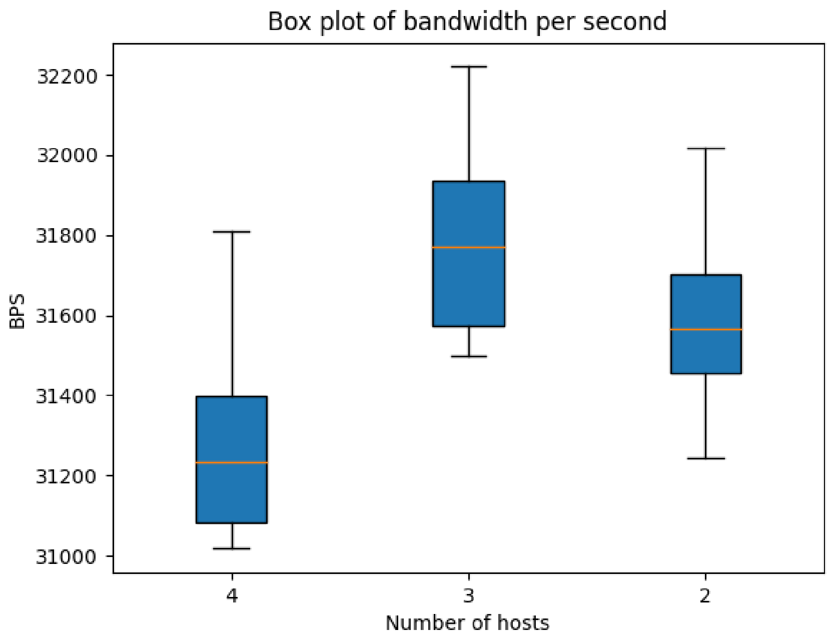

Figure 8: Representation of bps with parallel client and the amount of hosts used.

3.5 Interesting Values

When running our tests, we received one result that was quite interesting. As we can see in Figure 9 we got one run which had a packet loss which was around 50 %. From the graph we can read that this is abnormally high and makes the rest of the graph very hard to read since it is squished down, this was also the case for bps, fps and latency. We decided for all other graphs where we compare the different results that we nullify it. Why we received this value is something that is out of our hands and even if we wanted to be able to give a clear explanation of why this happened, it would be close to impossible. It could have been something on our computer that made it to sending these request fast enough our itchy not responding fast enough. This does however prove a point in that you do need more than one run to actually test something. If we only would have done one run and this was that run, it would have been a inaccurate analysis. Now we did eight runs of this and can with confidence say that this value is abnormal and should be nullified.

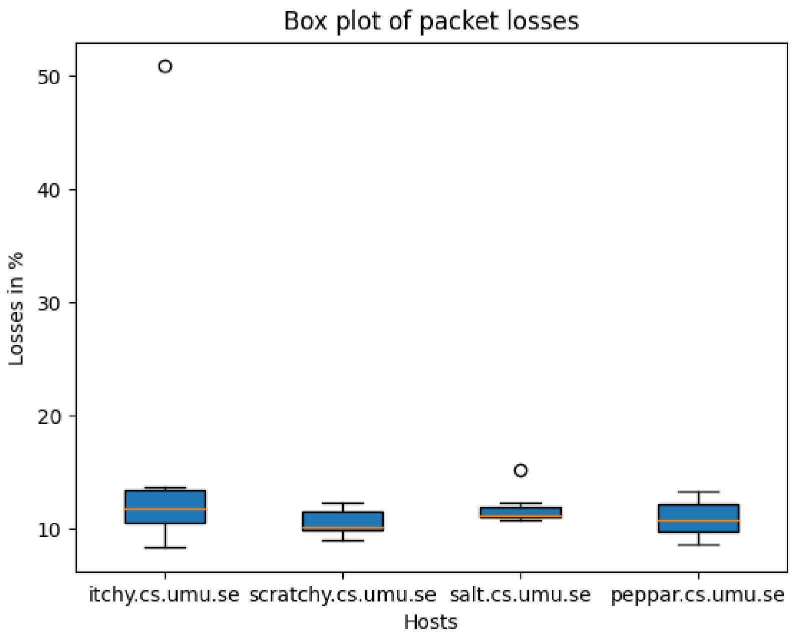

Figure 9: Representation of bps with parallel client and the amount of hosts used.

4 Discussion

As we can see from our graphs, if we have a higher percentage in drop rate, the latency also increases. This makes sense as if we have to retransmit a package the overall latency will decrease since have to spend more time handling one block as opposed to just going to the next one in the queue.

As the packet drop rate increases the Bandwidth utilization or bits per second decreases. This is because if we lose packets containing data and we need to retransmit the data and this will effect the bits sent per second each time we have to retransmit. This is the same principle as with latency as the metrics performance will be worse since packet drop rate is something we would like to be as low as possible.

To mediate this effect there are several techniques that can be used. We can increase the timeout to allow the packet to be received, this will most likely not affect the any other metrics since we will spend more time if the packet wasn't received.

One thing to take notice of is the result of the frames per second graph from the different hosts shown in Figure 6. The results are clumped together and doesn't have a large variance. The result of three hosts are also worse compared to four and two which really doesn't make sense if we compare it to the other metrics such as latency. We were thinking and a possible reason for this might be due to our algorithm when calculating the FPS using three hosts might not be optimized and that's why the statistics of the performance show some odd data.

If we would have used a video compression technique we would have seen a difference in frames per second and latency. We most likely wouldn't see a difference in packet drop rate since the packet was most likely being dropped anyways and if we would have compressed it, we can't seeing it making a difference. As for bandwidth utilization, it should change there either. The amount of bandwidth used should stay the same regardless of how small or large the packets are. The frames per second and latency should however see a change. Since the packets will be smaller we can receive more packets and that should result in more frames and a lower latency. It is hard to say how much a video compression technique will affect without testing but there should be a performance increase in frame throughput and latency.