Overview

To develop a file transfer server capable of transferring files to multiple clients simultaneously and to analyse both the number of clients and the size of the file affects the transfer efficiency of the server.

Design and Implementation

Server

I decided to go with a multi-threaded server design, as this offered a fairly simple implementation whilst still being able to concurrently serve multiple clients.

To run the server, compile the code by running

```
Javac Server.BasicServer.java
```

You can then run the server using the command

```
Java Server.BasicServer <number of clients to connect>
```

(Commands should be run in the *src* directory)

The argument passed to the server determines how many connections the server should wait for until it begins file distribution. With each connection made to the server, a new thread object and added to an ArrayList. Once the size of the list equals the amount of desired connections, the server records the current time and starts all the threads. When all the threads are finished and every client has a copy of the desired file, the difference in start and finishing times is calculated. This is how long it took to transfer each copy of the file to every client.

When the thread is run, the client tells the server what file they want. The server will then send the client the size of the incoming file before beginning to send the file in chunks. This reduces strain on the system. Before implementing this, I was getting errors due to filling the JVM heap when trying to send the larger files to multiple nodes.

Client

The client is essentially a generic socket. To compile the code, run the command

```
Javac Client.BasicClient.java
```

Once compiled, you can run the client by using the command

```
Java Client.BasicClient <numerical value for desired file>
```

(Commands must also be run within the *src* directory)

When the client is running, it will attempt to establish a connection with the server. The IP of the server is hard-coded as constant value. Simply alter this value to your own servers IP to allow for the clients to successfully connect.

Once connected, the client will tell the server which file it wants a copy of, as defined by the given argument. The options are:

0 – text file, 1 – mp4 file, 2 – xml file

Once the server has acknowledged the client's choice, the appropriate file is located, and the file size is sent to the client, so the buffer can be set to the appropriate size before reading in the file chunks.

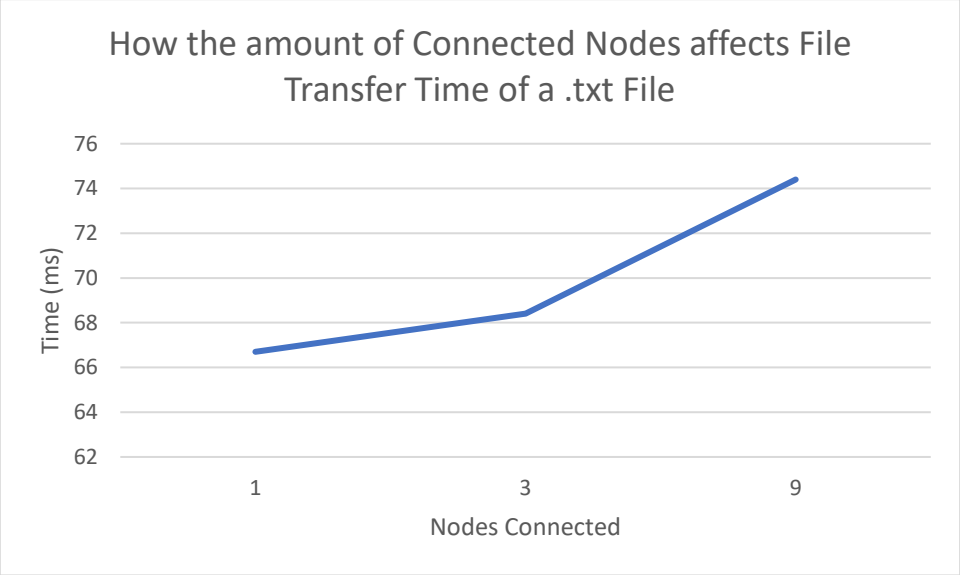
N.B. It should be noted that file paths for both server and client side should be altered for your own use. For the server, file paths should direct to where the files can be found. For the client side, file paths should be to where the received file should be saved.

Analysis

The server timed how long it took for all connected clients to receive a copy of the desired file. The timing would not begin until the predetermined number of clients had a successful connection. Once this condition is met, the current system time will be recorded and all the threads will be started. Only once every thread has terminated will a second time be registered and the difference calculated to give the total time to distribute all copies to all clients.

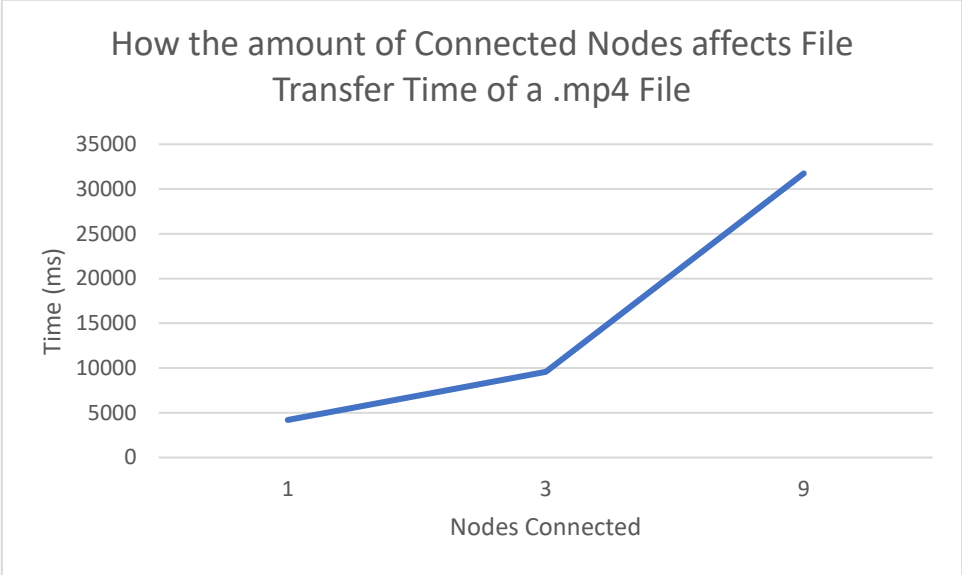
I recorded all my data manually, running the server on my machine and then sshing into 1, 3 and 9 different machines and running my client software on each of them. After all copies are transferred and the server terminates, the time taken is printed to the server console. I did 10 runs for each file to 1, 3 and 9 nodes, plotting the average time for each combination of file and number of connected nodes. Multiple runs were required in order to nullify the effect of outliers on the overall average. Some nodes may have been used by other students, which would mean that that particular node would take longer to receive the file, which means the time for total dispersion would also be affected.

Txt File



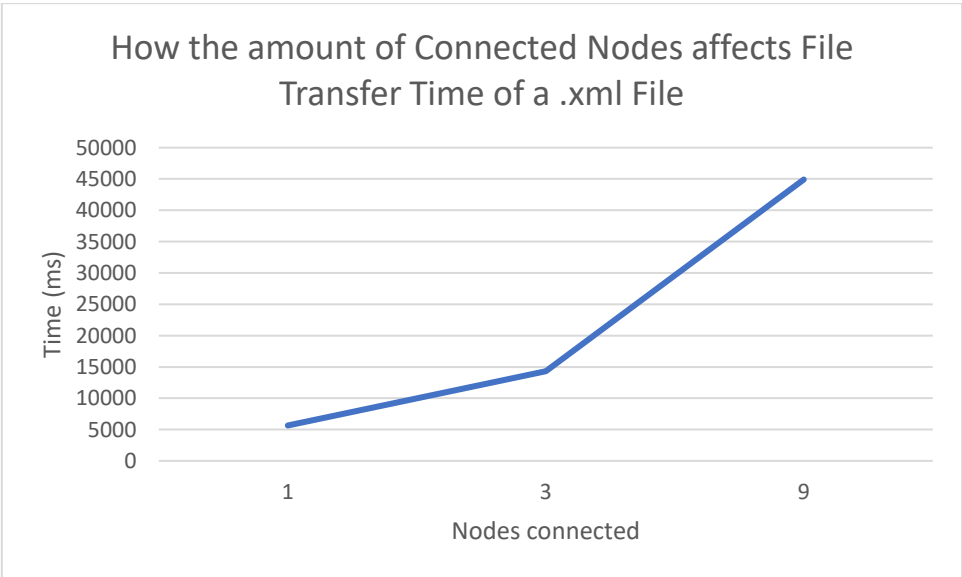
A parallel transfer of a small file like a pure text file is incredibly efficient, with the time necessary to distribute to multiple nodes being substantially less than a linear increase. This can be seen above, as it only took an average of roughly 67 milliseconds to transfer the file to one node. If it were to be a linear increase with respect to the amount of nodes requesting the file, you would expect 3 nodes to take roughly 190 milliseconds, however it only took an average of 68 milliseconds. The same can be said for 9 nodes, with an expected linear increase to roughly 600 milliseconds, but in reality only taking an average of 74. There is still an increase in the amount of total transfer time, but nowhere near what was expected.

Mp4 File



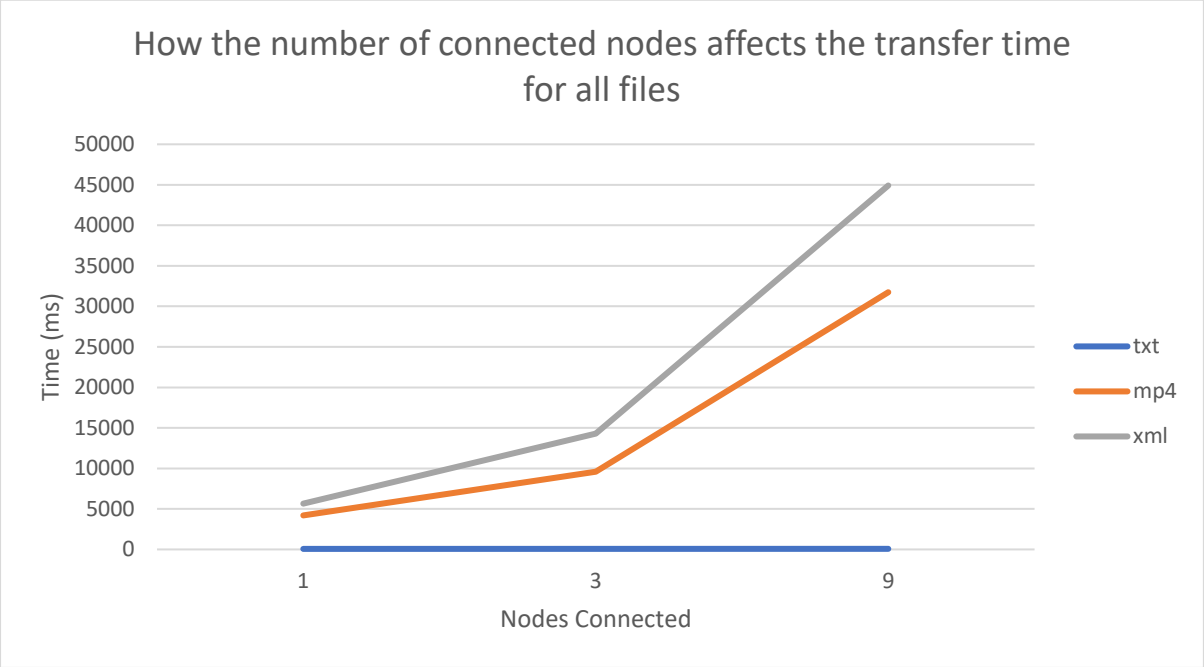
Here we can start to see the effect the size of the file has on the transfer time. The .mp4 file was the second largest of the 3 files, and you can see a dramatic increase in the time needed to send just one copy. Whereas before it took 66 milliseconds, it now takes an average of roughly 5 *seconds*! This is also starting to show a much more linear growth to transfer time. The difference between 1 and 3 nodes is essentially double, but the difference between 3 and 9 nodes is triple, which is definitely a linear jump.

Xml File



The largest of the three files behaved the closest to what was expected. From 1 to 3 nodes, the time taken went from 5 to 15 seconds, and from 3 to 9 went from 15 to 45 seconds. When multiplying the number of nodes by 3, the average time taken was also multiplied by 3, showing a linear relationship between the amount of copies being distributed and the time required to complete the task for a file of this size.

All Three



Above displays just how much the size of the file affects the time required to distribute copies to any number of nodes. You can also see that although there was only a slight time difference between the mp4 and xml file with 1 node, there is a large difference for 9 nodes. This illustrates just how much even a small difference in file size effects transfer time in a parallel distribution system.

Nodes Used

All nodes used had conventional disks and were running Linux. Below is the list of nodes used for analysis.

Server run on pc2-015

Nodes:

pc2-034 (used for tests involving 1, 3 and 9 nodes)

pc2-010 (used for tests involving 3 and 9 nodes)

pc2-147 (used for tests involving 3 and 9 nodes)

pc2-142

pc2-061

pc2-134

pc2-150

pc2-137

pc2-140

Reliability

I checked the equivalence of the files my comparing MD5 hashes of the original the server used and the copy received by the client. The checksums were the same, suggesting my system provides a reliable way to transfer files.

Evaluation and Conclusion

I feel I have successfully implemented a file transfer system that can handle serving multiple clients in parallel, allowing for file transfer times to occur faster than a linear growth rate would suggest. This can be seen by the data I have recorded and the subsequent analysis that I performed.