Concurrent Socket Server

Avery Anjard and Evan Baker

CNT4504 - Computer Networks & Distributed Processing

Professor John Kelly

Introduction

In the previous project, we created a multi-threaded client, which sends from 1 up to 25 commands, and a single-threaded server, which runs the commands. The more we sent to the server, the harder it struggled to keep up, especially with the more difficult commands such as netstat. For this project, however, we made our server multi-threaded, as well. The goal of doing this is to make the server more capable of taking on more commands, no matter how difficult they are to run.

In the next section of this paper, we shall discuss the setup and configuration of our client and server. Then, we shall explain how the server was tested and present the data that was collected. Then, we shall analyze the data and discuss how increasing the number of clients affects turnaround time, as well as compare how the iterative socket server performs to the concurrent socket server. Finally, we shall draw our conclusion and discuss the lessons we learned while doing this project.

Client-Server Setup and Configuration

The design of the client for the concurrent server socket project is pretty much the same as the client used for the iterative socket server. The only difference now is that the client can send 100 commands at a time, when it previously could only send up to 25. The client accepts 2 arguments when it is run: the IP address and the port number of the server. The client will then ask the user to select a command they would like to run; this is done in a while loop which does not end until the user selects “Q” to quit the program. The user is then asked to select how many queries they would like to send: 1, 5, 10, 15, 20, 25, or 100. Then, for the number of queries that the user selected, the client creates a new thread, starts a timer, runs the thread with the command, stops the timer, and calculates the turnaround time of the thread. At the end of queries, the total turnaround time, as well as the average turnaround time is calculated and displayed.

The thread class contains a constructor with the command, a PrintWriter, and a BufferedReader. When the thread is created in the main class, the command, the PrintWriter, and the BufferedReader are all passed to the constructor. When the thread is run, it simply sends the command to the server, and then outputs the results of that query to the console.

As for the server, the design had to be completely redone. In the previous project, the server was a simple bash program which took a single letter as an argument and used that to determine which command to run. For this project, the server was created using Java. The class for the thread is contained in a separate file. The constructor for the thread takes a socket as an argument. The run function creates an InputStream, a BufferedReader, an OutputStream, and a PrintWriter. Using a while loop, the server takes in input, which is formatted as a single letter, and uses that to determine what string to set the command variable as. Then, this command is used to write the script that will be used to run the command. The script is then executed. After the first while loop is done executing, a second while loop is used to output the results from the query to the client. The class for the actual server takes the port number from the args, creates a ServerSocket, and then uses a while loop to create new sockets and threads to handle all of the queries.

Testing and Data Collection

To test the concurrent socket server, each type of command was run using 1, 5, 10, 15, 20, 25 and 100 queries each time. The average turnaround time, as well as the turnaround time for the first and last request were collected. For the netstat command, however, the turnaround time for the first request was unable to be gotten due to how long the output from the netstat command is, meaning that we were not able to scroll all the way up to the first request.

Below are the charts for the iterative socket server.

A graph with different colored bars

Description automatically generated

A graph with different colored bars

Description automatically generated

A graph of a number of colored bars

Description automatically generated with medium confidence

A graph of a number of people

Description automatically generated with medium confidence

A graph of different colored bars

Description automatically generated

A graph of different colored bars

Description automatically generated

Data Analysis

Increasing the number of clients does not have seem to have a very large effect on individual clients. The turnaround time among individual clients tends to stay within a certain range, with that range depending on the command, no matter how many queries are being made. For average turnaround time, however, there appears to be a general trend of it decreasing as clients increase until it gets to 25, where it slightly increases. There are some exceptions, though. For the date command, for instance, the average turnaround time increases. The netstat command’s average turnaround time just increases. Another thing of note is how the turnaround time for the first request of a set of queries always tends to be higher than the average turnaround time, as well as all the turnaround times of the other requests.

Comparing the iterative socket server and the concurrent socket server, there are some commands where the multi-threaded server makes a significantly large difference in the average turnaround time, and some that do not. For example, the average turnaround time for the date command is not that much different between servers except for 5 and 10, where the concurrent socket server is better. There is an even larger difference for the uptime command – especially with 15 queries – with the concurrent socket server once again on top. For the current users command, however, the concurrent socket server appears to be quite a bit worse than the iterative socket server, except for 20 and 25 queries, which are relatively the same between the two servers. The average turnaround times for the memory use command between the two servers are quite similar. Running processes is the same, except for the average turnaround time for 25 queries, where the concurrent socket server is somewhat better. Finally, the concurrent socket server has better average turnaround times for netstat, especially for 5 and 15 queries, although the average turnaround times in the iterative socket server are most likely outliers.

From the analysis of this data, we can conclude that it would be better to use the concurrent socket server if you frequently use the uptime or netstat commands, and the iterative socket server for uptime commands. For the rest of the commands, it does not make a significant difference which type of server you use.

Conclusion

From the data analysis, we would conclude that while the concurrent socket server is somewhat faster for some regards, it is not significantly better than the iterative socket server. This is not quite what was expected for the server before the project was started. The explanation for this, however, is most likely about how the two servers were programmed. The iterative socket server was a simple bash program, which had only a few lines of code and was able to handle requests quite easily. The concurrent socket server, on the other hand, was a much longer and more complex Java program. Although it is easier to write a multithreaded program in Java, it comes at the expanse of performance.

Lessons Learned

Between these two projects, we were able to get a lot more experience with using sockets and threads that we did not have before. This included learning how to successfully implement a multithreaded program. We also had to overcome a problem with timing the threads. We initially tried to calculate the turnaround time for each thread within the thread run function. However, the stop time would not be recorded, resulting in the turnaround time being 0 microseconds no matter what. We were able to figure out that putting the time functions in the main function around the thread start call made it work.