Concurrent Socket Server

Joshua Malgeri, Andreas Ink, Colton Harris

CNT4504 Computer Networks & Distributed Processing

Professor John Kelly

Introduction

CNT4504 Computer Networks & Distributed Processing is a fascinating course that quite literally connects us all. During our class we learned the relationships of servers and clients and how they impact each other. Therefore, we were tasked to apply our learnings and understandings of server/client relationships. We built a server and client that was developed in Python 3.6 where both the server and client had simple tasks to perform. The server takes in several requests from the user and establishes a connection to the server where the requests are processed and distributed back to the user via a printed message. However, the server is now multithreaded (handles multiple user requests). Previously the server slowed down due to bottleneck without multithreading implementation. We aimed to examine how the server would be impacted by the number of requests from the client server and the types of requests. Further, to analyze the impact of the processing speed and the number of requests idling. This analysis documents the design process of each server from the client to the request server. We dive into the programming language and the syntax describing the steps of how it iterates and processes the requests, and encapsulates how data is collected and analyzed including at what speed the server is processing requests.

Client-Server Setup and Configuration

The client is composed of modular architecture to ensure maintainability and readability throughout the programing process. For example, we created methods such as getBasicEndpoint and getComplexEndpoint. These methods were used to abstract certain reusable components such as retrieving a basic endpoint that returns a double or float, and then a more complex method that returns a json. Moreover, it utilizes other helper methods such as fetch, which wraps the requests library in python and handles exceptions by printing an error message that is defined as a constant at the top of the Python script.

Similarly, the server utilizes modularity and abstraction to write cleaner code. We've abstracted each endpoint into a helper method that executes the Linux command and parses it as needed so it's digestible on the client side. We also made the server multithreaded in order to process the multitude of requests from the user. For example, 'network_connections' utilizes subprocesses to run 'netstat', which is then split into each line and extracted to be dumped in a json object. However there is one discrepancy in code uniformity with the use of camel case and snake case between the client and server, ideally we'd maintain the same coding conventions across the two scripts.

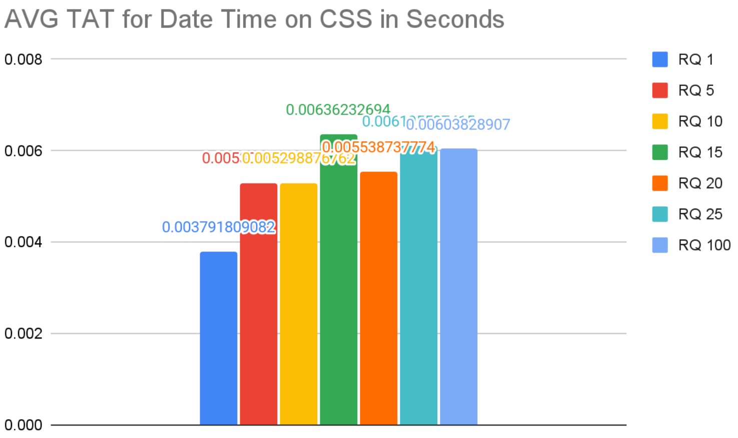
Overall, we decided to approach the code in this way to improve maintainability and readability which helped with the efficiency of development. Along with this, the multithreading of the server will also help with processing efficiency.

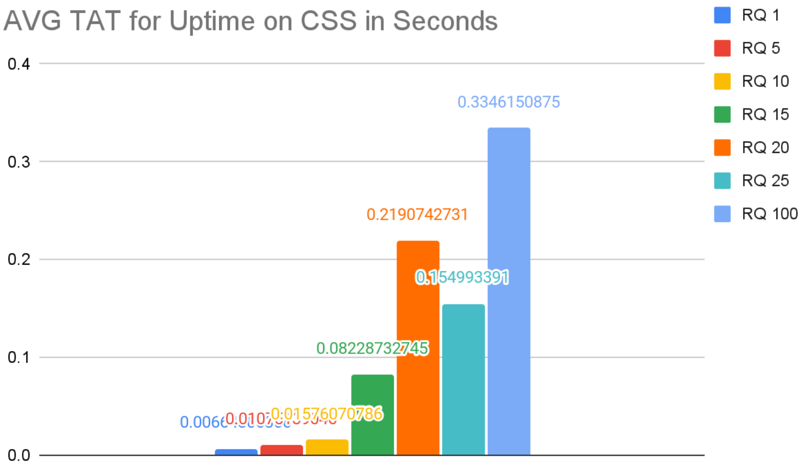
Testing and Data Collection

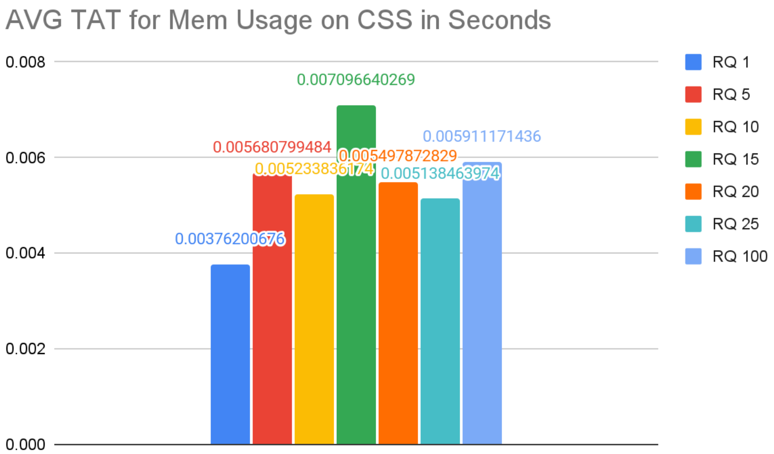
The final step of developing the concurrent socket server was by running and testing these servers through trial and error. The project was tested by running two python programs, a server program and a client program on the UNF servers. We also developed a test case app to help us with our development and testing which we will discuss later on. These programs were run by connecting to the server through the use of a VPN and a remote-in app such as Bitvise or Terminus. Once these servers were connected through the UNF server, the code to compile the python programs was executed on both the client and the server. We developed two client servers, one for testing as shown below on the graph and data visualization images. The other client was used in the UNF server as part of the final project of the iterative socket server.

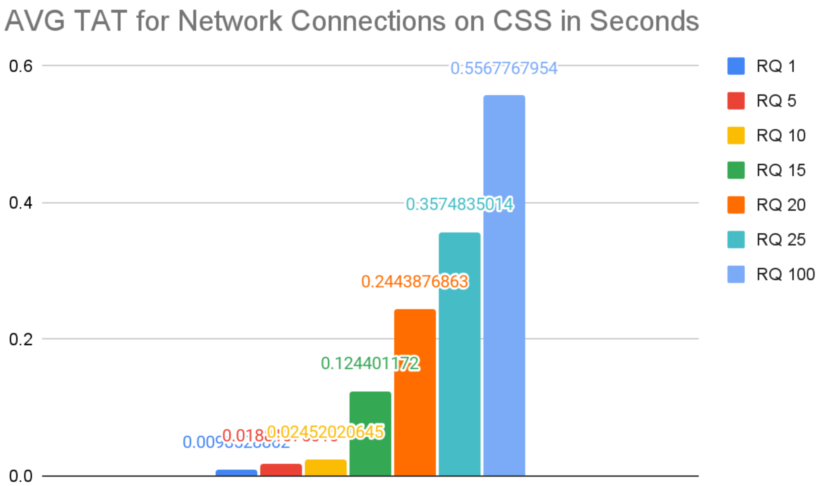
When the client makes a request to the server, the server sends back the requested data from the corresponding command and the turnaround time for the request is computed and printed. The data that was collected was the turnaround time for each individual request, which was grouped by the amount requested, the average turnaround time for each individual request in the group of requests, and the total turn around time of the entire group of requests. All of this data was then put into a spreadsheet to be analyzed.

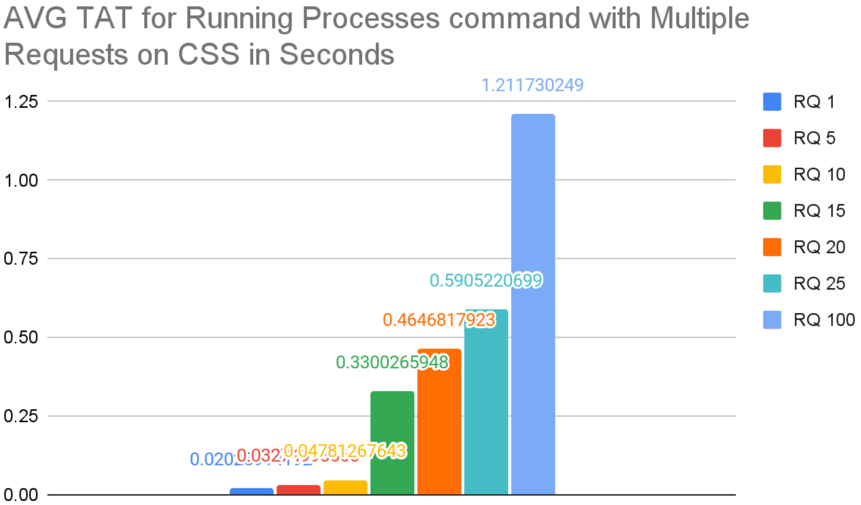
Charts of Average Turn Around Time of Commands on the Concurrent Socket Server

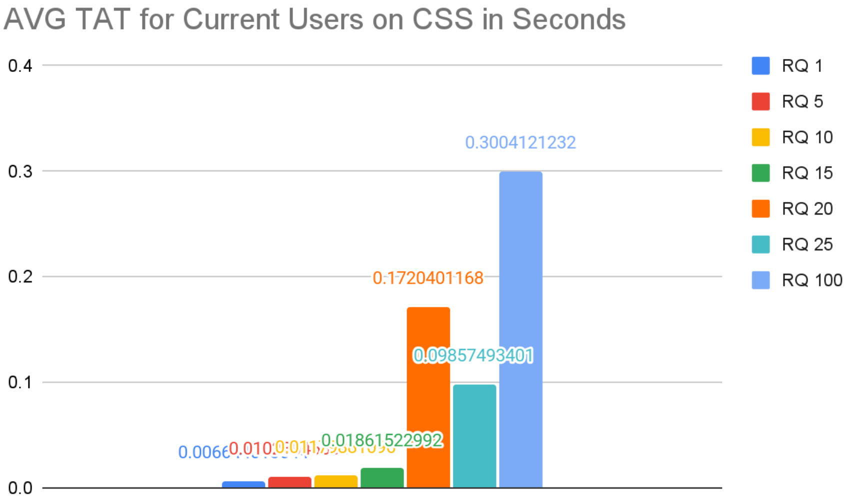



AVG TAT for Uptime on CSS in Seconds
0.4
0.3
0.2
0.1
0.0
0.3346150875
0.2190742731
0.154993391
0.08228732745
0.0066
0.0107
0.01576070786
RQ 1
RQ 5
RQ 10
RQ 15
RQ 20
RQ 25
RQ 100

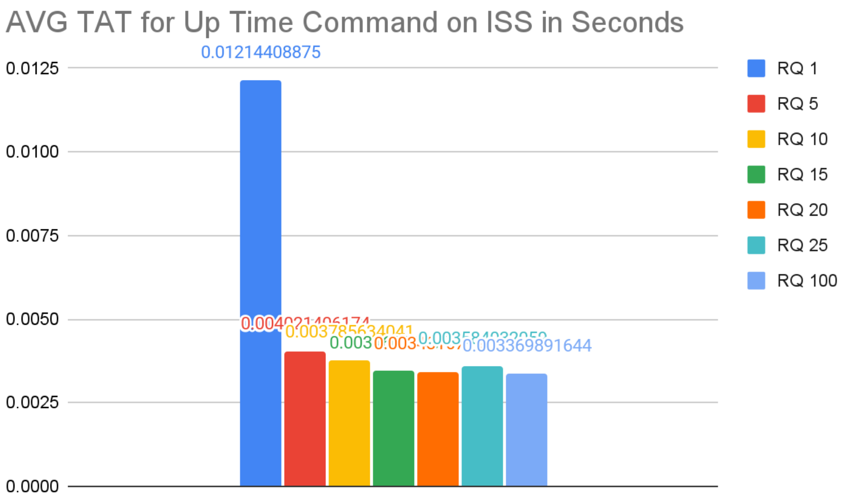

AVG TAT for Mem Usage on CSS in Seconds
0.008
0.006
0.004
0.002
0.000
0.007096640269
0.005911171436
0.005680799484
0.005497872829
0.005233836174
0.005138463974
0.00376200676
RQ 1
RQ 5
RQ 10
RQ 15
RQ 20
RQ 25
RQ 100

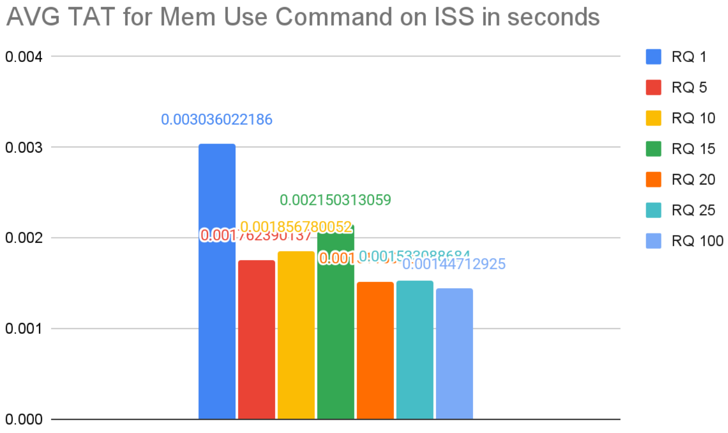

AVG TAT for Network Connections on CSS in Seconds
0.6
0.4
0.2
0.0
0.5567767954
0.3574835014
0.2443876863
0.124401172
0.009
0.018
0.02452020645
RQ 1
RQ 5
RQ 10
RQ 15
RQ 20
RQ 25
RQ 100

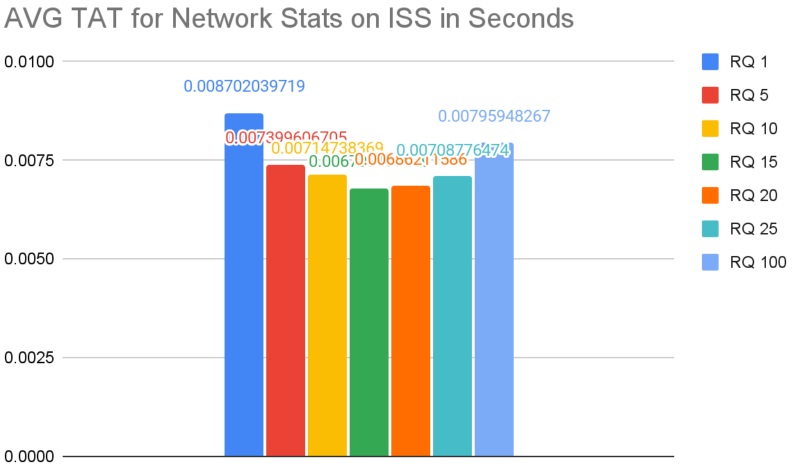

AVG TAT for Running Processes command with Multiple Requests on CSS in Seconds

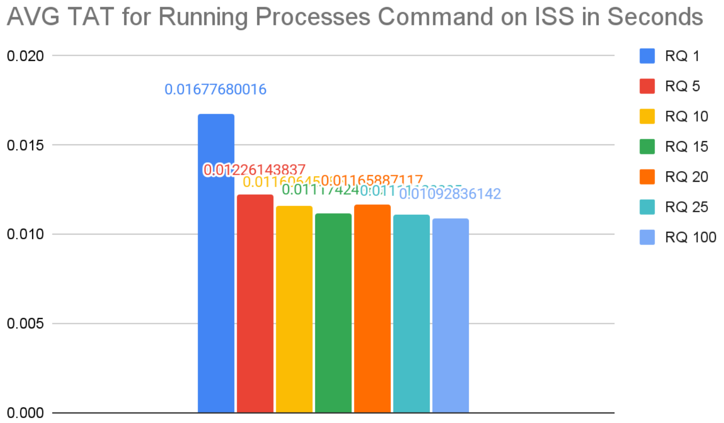

AVG TAT for Current Users on CSS in Seconds

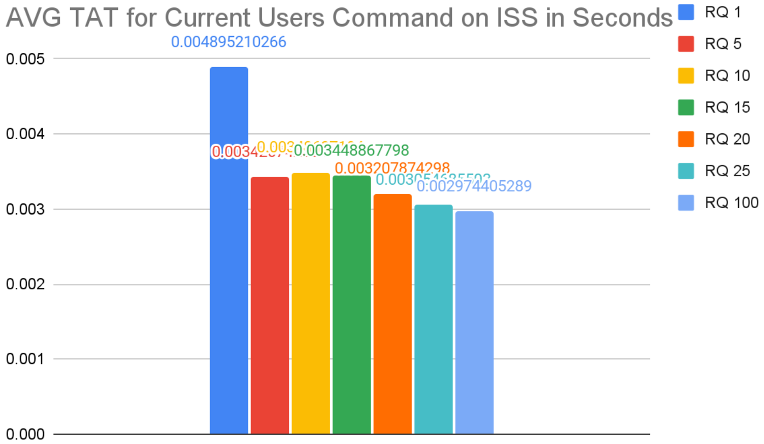
Charts of Average Turn Around Time for Commands on Iterative Socket Server








AVG TAT for Running Processes Command on ISS in Seconds


AVG TAT for Current Users Command on ISS in Seconds

Data Analysis

Increasing the number of clients does in fact lead to longer turnaround time for individual clients since the servers capacity is limited to support a fixed size of customers. However, effectively managing the requests with prioritization, and the proper allocation of resources can mitigate this problem. Additionally, clear communication with clients about the expected turnaround times and potential delays is essential for managing expectations and maintaining client relationships. Increasing the number of clients can greatly affect the average turnaround time due to the increased workload of a large client base. In order to fix this, the same steps can be taken as above. The primary cause of the effect on individual client turn-around time and the average turnaround time is that each is affected by the workload that is brought onto them by the client. Further, the capacity of how many requests the server can hold is a huge factor in the increasing and decreasing of efficiency. Without the changing of capacity, the amount of clients will overwhelm the server with requests and cause it to slow its processing speed. Thus, these factors contribute to the increased average turnaround time and the individual turnaround time.

Both the iterative socket server and the concurrent socket server have uses in different scenarios. The Iterative server is best used when the server is not going to be bombarded with complex requests by multiple clients, and the Concurrent server is best used when there are going to be requests from many clients and the server's hardware is powerful enough to effectively execute all of the requested processes concurrently.

Conclusion

Based on the data analysis and from our testing of the client/server, we can strongly conclude that server response time does in fact increase based on an influx of concurrent requests from the client server. We determined that the netstat command produced the slowest return time. Further, we found that the date time command was the fastest to return a response among the commands. We found these results based on multiple testing and data analysis performed by our group. Throughout the project we learned that modularity is important, especially in a group project with multiple teammates to isolate issues and resolve git conflicts and so that the process of the project can run more efficiently. Each team member was assigned their own portion of the project such as development of the server, development of the client, and the writing of the paper. Ultimately we all pitched in with each other's work load or process of the project. Also making TODO lists which gave us tasks to conquer for the project each week, a github repository for the code, and a task list were helpful when working as a team. We originally wrote the client and server in a version incompatible with the UNF server which led us to drastically revamp the server and client in a new way. To resolve this next time, we'd test the imports on the server to ensure we are not using incompatible technology.