Concurrent Socket Server

Michael Beaver, Tyler Merrett, Michael Skipper

CNT 4504 - Computer Networks & Distributed Processing

Professor J. Scott Kelly

Introduction

The purpose of this assignment lies within strengthening students' ability to understand and manipulate rudimentary programs adhering to a client and server relationship; namely, to demand these programs fulfill specific commands, such as retrieving information about the data and the time or the memory usage on the server. This project has allowed practical implementations to fluff the blanket of mystique associated with computer networks and provide insight into a process which we will likely encounter along our paths. Specifically, adapting an iterative server into a concurrent server stood a grand feat to overcome, though upon further analyzing the data for our iterative server, we realized its concurrency in nature!
This project aims to provide students with practical insight about the theoretical topics which have been discussed all semester, preparing an outlet for students to develop proprietary servers and actually measure and analyze the efficiencies of different server implementations, namely converting an iterative server into a concurrent server and analyzing the effect of client size on the turnaround times for individual clients and in the average case. Our goals were to gain a better understanding of what iterative and concurrent servers are and to communicate effectively as a team.

In the remaining sections of the paper, we discuss the process of converting an iterative server into a concurrent server and our realization that our initial iterative server was nothing but a concurrent server in disguise. Moreover, we discuss the development considerations of the group involved with completing this project and the methodology for producing server and client programs, the functionality of basic operations utilized in this process, the manner testing as performed, an analysis of the data of the runtimes we collected and the graphs of this data, the effect of increasing the number of clients on turnaround time for individual clients and for the average turnaround time, the primary cause for the effect that increasing the number of clients has on these turnaround times, the conclusions drawn from analyzing the data and the lessons learned completing this project.

Client-Server Setup and Configuration

The design of our client program was identical to the design of the client program used for the iterative server, the difference lies within the design of our server program and the manner that clients are handled. The design decisions we began to contemplate included simulating a rapid influx of requests from separate client instances for our server to handle concurrently, as opposed to the iterative nature which clients were handled for the last project. Namely, requests such as displaying the current memory usage on the network and the network connections on the server. In order to simulate this, our connection sockets are encapsulated within their own threads on the client side to concurrently process the sending of requests and their recognitions returned by the server. We also tracked the turnaround time for these requests by taking the difference between the moment the clients' request was sent to the moment the request was received by the server, in order to understand the relationship between the number of

clients and the turnaround time for individual clients and in the average case. The server supports the requests by the client program by implementing a nested Runnable class 'ConnectionHandler' that handles all incoming Socket connections within their own instances, which allows the server to handle multiple terminal connections. Client requests are received as String values that are passed into each of the 'ConnectionHandlers' as constructor input. These String values are then passed to an additional function and switch block that then calls upon that unique Socket connection's Runtime object to make a query through the use of the Process object and the 'Runtime.exec()' method.

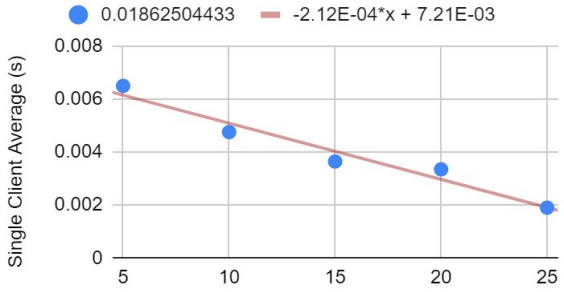
Since iterative servers handle both the connection request and the transaction involved in the call itself and do not last long, concurrent servers are constructed for lengthy operations which create child server processes to handle the connection already established with the server. Our concurrent server listens for client requests, and when a request is received, spawns a new server instance to handle the request, determines which operation is being requested, performs the operation, collects the resulting output, replies to the client request with the output from the operation performed, and destroys the server instance.

Testing and data collection

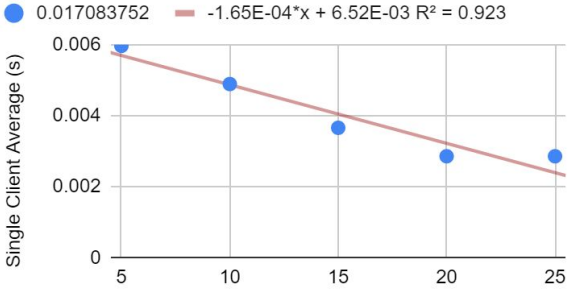
In order to test the concurrent server, we followed the same methodology as testing the iterative server to ensure parallelism in findings, so we used ‘nanoTime()’, which returns a value since some fixed, arbitrary origin time, to record the moment immediately before sending the client request to the server and another the moment immediately after the request is processed by the server and sent back to the client. Subtracting the latter from the prior results in the turnaround time for an individual client for a single operation. Repeating this process for each operation, these values are summed and divided by the total number of clients to reveal the average turnaround time of the concurrent server.

The data we collected reveals the duration of the turnaround time for a client request to be sent, processed, and returned by a concurrent server, and for varying amounts of client requests, we measured the time for clients to send data requests, which provide the date and time on the server; uptime requests, which provide how long the server has been running; memory use requests, which provide the current memory usage on the server; netstat requests, which list network connections on the server; running processes requests, which list programs currently running on the server, and current users requests, which lists the users currently connected to the server. The data also indicates that our server is concurrent because the single client turnaround time goes down with more trials.

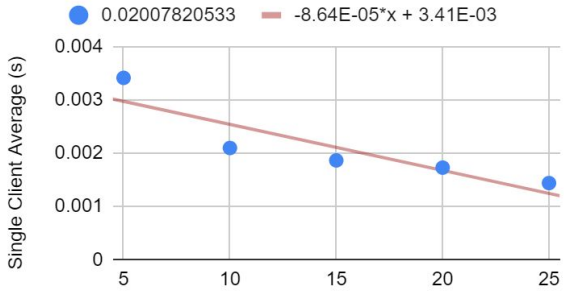
Date and Time



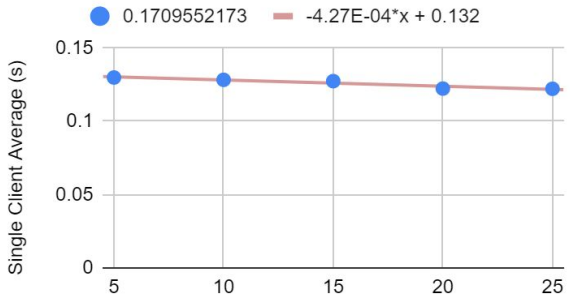
Uptime



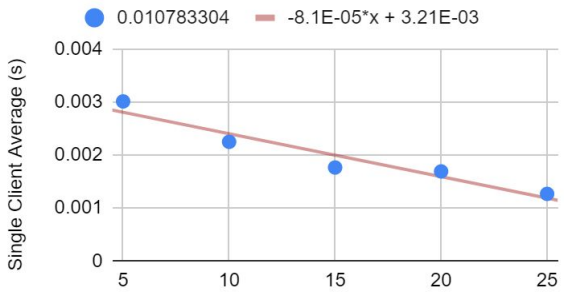
Memory



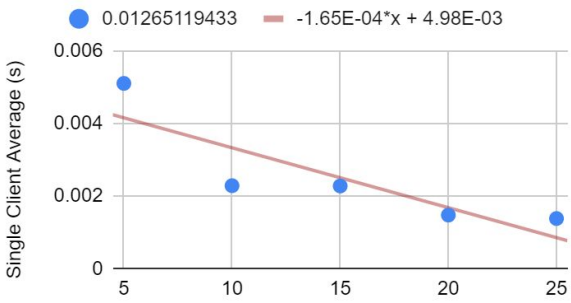
Netstat



Users



Processes



Data Analysis

Increasing the number of clients seems to steadily decrease the turnaround time for individual clients and the average turnaround time, showing a negative correlation between the number of clients and the turnaround time for individual clients and in the average case. While we didn't have any exact values for Iterative Servers, we can assume, since the individual clients are handled successively, it will take longer to handle larger quantities of clients. Iterative servers are usually utilized for transactions which do not take a very long time, because connection requests and transactions involved in the call are both handled by the server in this model. Because iterative servers are prepared for transactions which do not take a long time, if a transaction becomes long, queues begin to build up as clients who come later are not able to be serviced until the earlier clients are served. If transactions are expected to take a long time, the design considerations should turn toward concurrency, in which client connections are established before any request is made by a client, where a process in the child server actually handles the connection and allows requests to be performed without waiting for earlier clients to finish their processes.

Conclusion

We have found that our data aligns closely with that of a Concurrent Socket Server. This conclusion is both drawn from validation from a more knowledgeable source as well as the actual data itself. In an Iterative Socket Server, each individual client has to wait for the previous one in order to develop their own conversation with the server and thus would prolong the individual client time from the perspective of the initial request. In the case of concurrent, the more iterations we complete, or clients' requests, the closer we get to the time of the actual calculation and transmission of data from client to server and back. In larger quantities of requests, it seems to make more sense and be more natural to handle as many processes as possible in a parallel fashion making concurrent better in these situations.

Lessons Learned

This time around, having learned from our previous adventures completing a project, we managed to "strategically" procrastinate. The primary issue we had to overcome this time around, however, was the issue regarding our own knowledge and our code. Where we previously thought we created an Iterative Socket Server, we had actually created a Concurrent Socket Server. Issues arose when we were unable to distinguish the difference between these types of servers. After a conversation with our knowledgeable professor we were able to increase our understanding of the two types of servers.