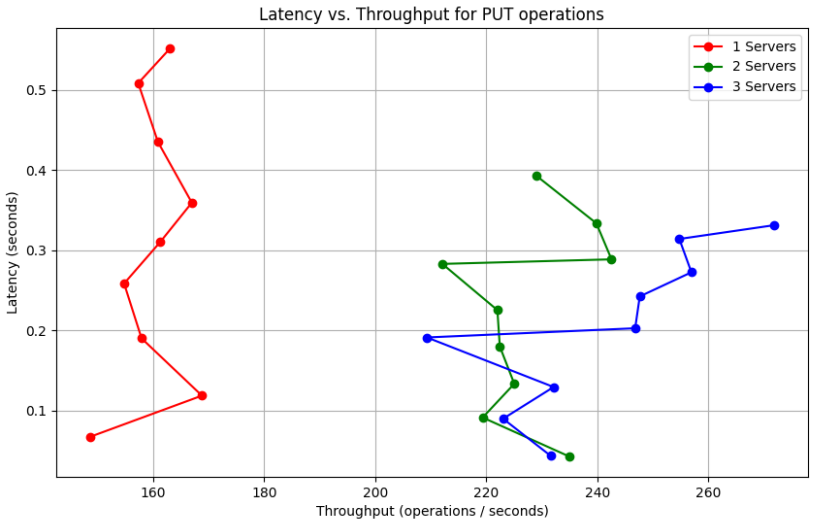
Coding Assignment #2 Report

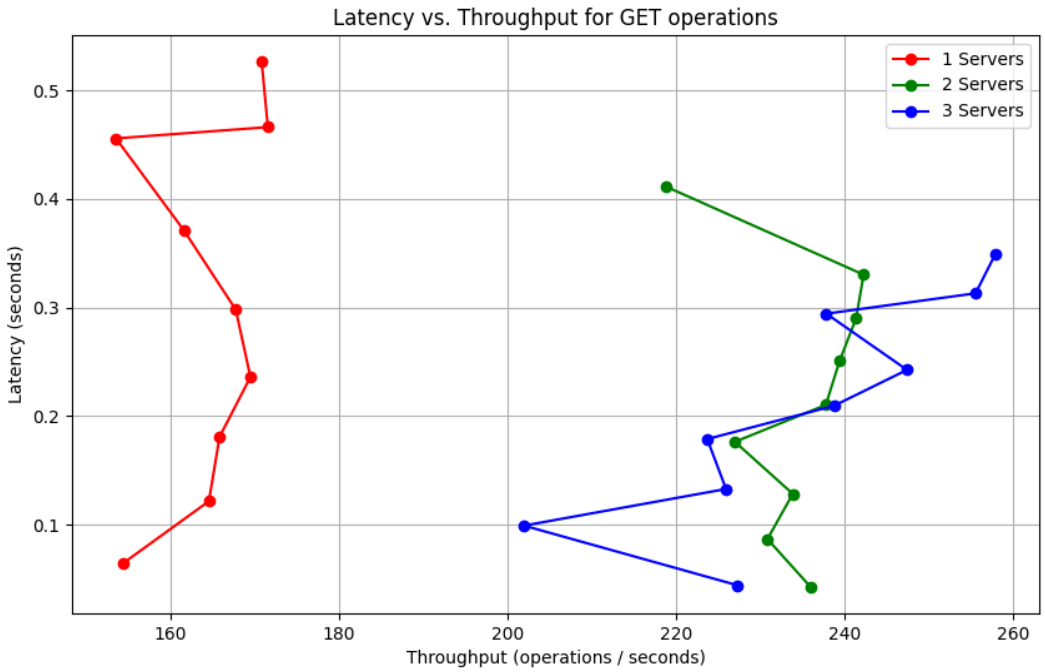
The server.py implements the old version that was present in the GitHub repository. However, the client.py, multithreading_client.py, and testing_and_measurement.py are implemented to use consistent hashing to evenly distribute the load across multiple key value stores, ensuring a more fault-tolerant system. All of the client files were implemented with the assumption that the number of servers is between one and three, and the server.py servers are executed on ports between 65432 and 65434. Note that the consistent hashing was done using uhashring library and hashlib library, and the client implementations distribute the key value stores and handling the loading as per the requirements. Moreover, 10 virtual nodes were used to provide a more balanced distribution of the data. Finally, for the testing_and_measurement.py file the latency vs. throughput was analyzed for different number of servers, different number of threads, and different number of requests.

First, we execute run `py -3 testing_and_measurement.py PUT --value some_value`
We assume that we will start with 1 server and scale up to 3 servers. Furthermore, the default parameter values of 10 threads and 100 PUT requests are used with some random value for each 100 keys generated. Therefore, the code will iterate starting from 10 to the number of requests (which is 100 in this case) and obtain a latency and throughput values which are stored for each server number scale (see the bottom of testing_and_measurement.py for greater detail). As a result, the latency vs. throughput plot is plotted below.



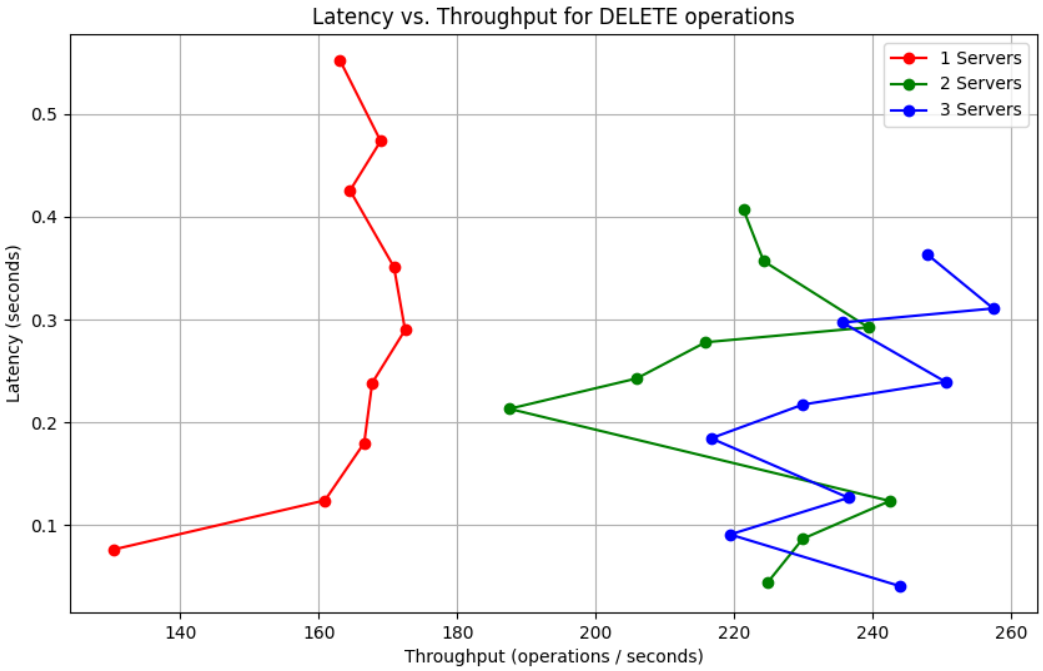
We can see that there is improvement in the latency and throughput when the key value store amount increases. When we use 1 server, 2 servers, and 3 servers we see a positive relationship between the latency vs. throughput.

Next, we execute run `py -3 testing_and_measurement.py GET`
We assume that we will start with 1 server and scale up to 3 servers. Furthermore, the default parameter values of 10 threads and 100 GET requests are used. Therefore, the code will iterate starting from 10 to the number of requests (which is 100 in this case) and obtain a latency and throughput values which are stored for each server number scale (see the bottom of testing_and_measurement.py for greater detail). As a result, the latency vs. throughput plot is plotted below.



We can see that there is improvement in the latency and throughput when the key value store amount increases (except this is not as obvious for the 2 and 3 servers). When we use 1 server, 2 servers, and 3 servers we see a positive relationship between the latency vs. throughput.

Next, we execute run `py -3 testing_and_measurement.py DELETE`
We assume that we will start with 1 server and scale up to 3 servers. Furthermore, the default parameter values of 10 threads and 100 DELETE requests are used. Therefore, the code will iterate starting from 10 to the number of requests (which is 100 in this case) and obtain a latency and throughput values which are stored for each server number scale (see the bottom of testing_and_measurement.py for greater detail). As a result, the latency vs. throughput plot is plotted below.



We can see that there is improvement in the latency and throughput when the key value store amount increases (except this is not as obvious for the 2 and 3 servers). When we use 1 server, 2 servers, and 3 servers we see a positive relationship between the latency vs. throughput.

I have faced various challenges. First, I was not entirely sure how to properly implement the project so I did the consistent hashing with the server.py file without thinking about the bottleneck this would create. Afterall, if my client files perform consistent hashing to determine which server to perform an operation on, it was redundant to do consistent hashing again in the server.py file. Therefore, I reverted my server.py code back to what it was in the beginning of October and made all of my consistent hashing implementation in the client files (client.py, mulithreading_client.py, and testing_and_measurement.py). Also, I was having a hard time getting thetesting_and_measurement.py file to work because the latency vs. throughput would

not properly be outputted. Moreover, when I tried to run testing_and_measurement.py immediately after it executed or a few minutes after it executed, I would receive a "[WinError 10048] Only one usage of each socket address (protocol/network address/port) is normally permitted" error. I was able to fix many of the small issues in testing_and_measurement.py and get the latency vs. throughput results to be properly plotted, but I still have not fixed the WinError that I keep encountering. This WinError and the various issues initially present in testing_and_measurement.py made me try to use nginx. Therefore, I played around with using nginx_client.py and nginx_testing_and_measurement.py and was able to get a somewhat close implementation, but its still not complete.