PART 3: EVALUATION DOCUMENT

Please note that for all the below experiments, we executed the server in edlab machines and we executed the client programs from our local machines concurrently like in the following scenarios.

For 5 clients: 3 clients from person1 machine and 2 clients from person2 machine running concurrently.

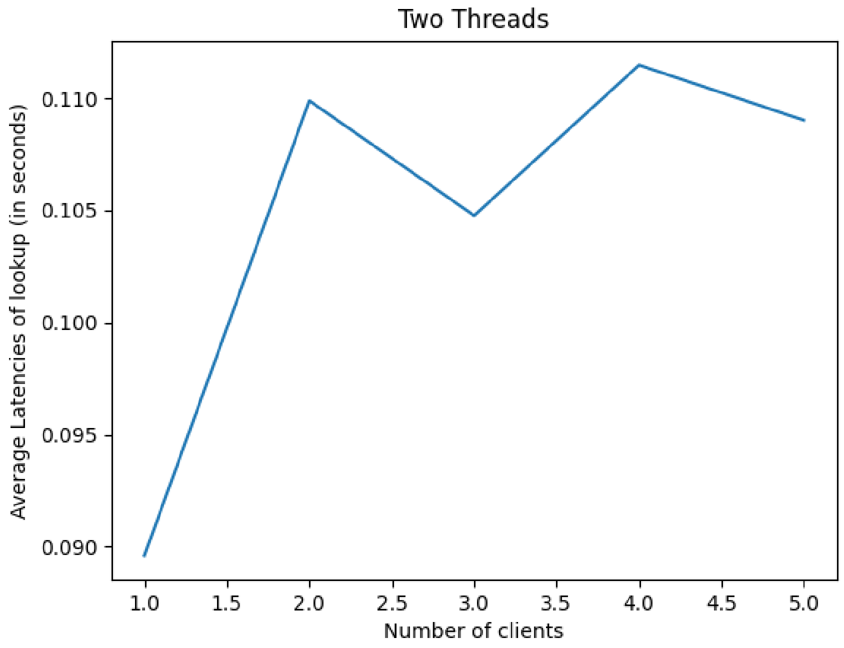
This is because we intended to replicate the real-world scenarios as the clients must be hitting the servers from different locations.

Part1:

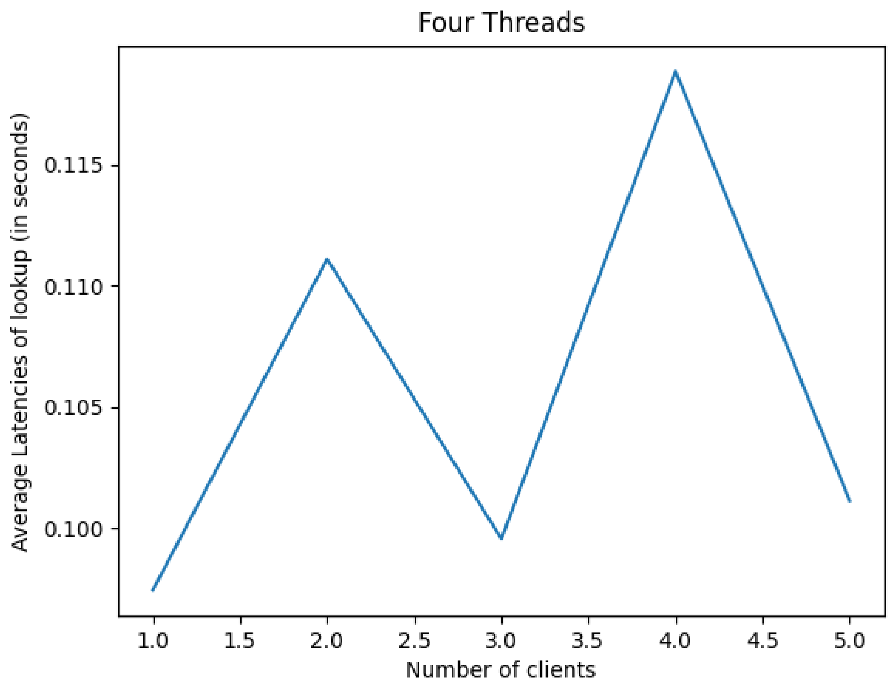
To load test the socket server, we created different scripts.
client_runner.py script: It runs client.py in multiple processes, which sends sequential requests to the server.
run.sh: It is a shell script, which runs the client_runner.py concurrently varying the number of clients from 1 to 5.

By running the above run.sh script by varying the number of clients from 1 to 5, we generated the below graphs.

For the number of threads = 2 in the server,



For the number of threads = 4



Observation for part1 lookup:

Since we are using 2 threads and 4 threads to generate graphs. For 1 and 2 clients, the number of threads doesn't matter as the number of clients here is <= 2 (minimum number of threads). So, if we have to compare clients for 3, 4, and 5, then on average by increasing the number of threads the latency is decreasing which is expected behavior.

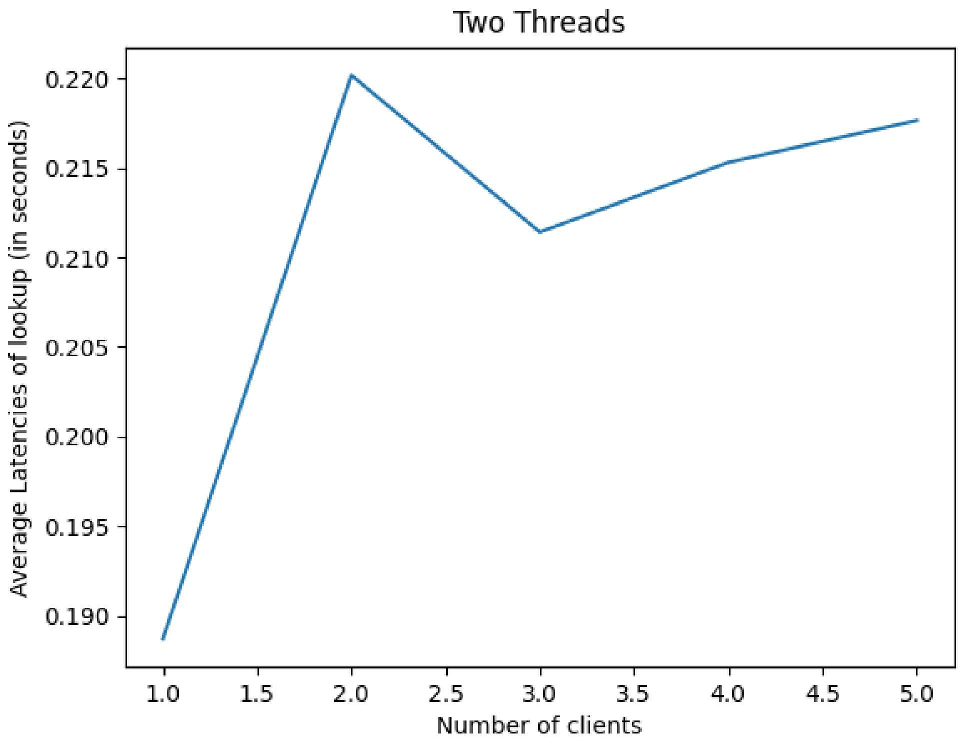
Part2:

To load test the gRPC server, we have created different scripts as follows:
lookup_runner.py: It runs lookup.py in multiple processes, which sends sequential requests to the server.
trade_runner.py: It runs trade.py in multiple processes, which sends sequential requests to the server.
run.sh: It is a shell script, which runs either lookup_runner.py or trade_runner.py concurrently varying the number of clients from 1 to 5. We can configure which file to run during the runtime execution (please see src/README.md file on how to run the scripts)
NOTE: Please note that, during performing the load test for gRPC, we are executing the price_updater.py, which updates the price of the stocks, parallelly.

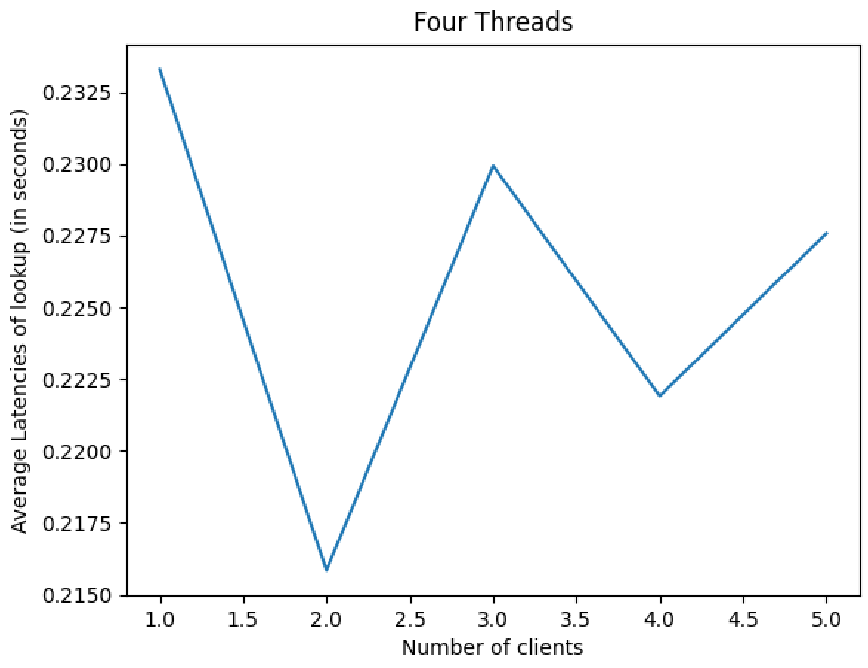
By running the above run.sh script by varying the number of clients from 1 to 5, we generated the below graphs.

For Lookup:

Number of threads = 2 in the server



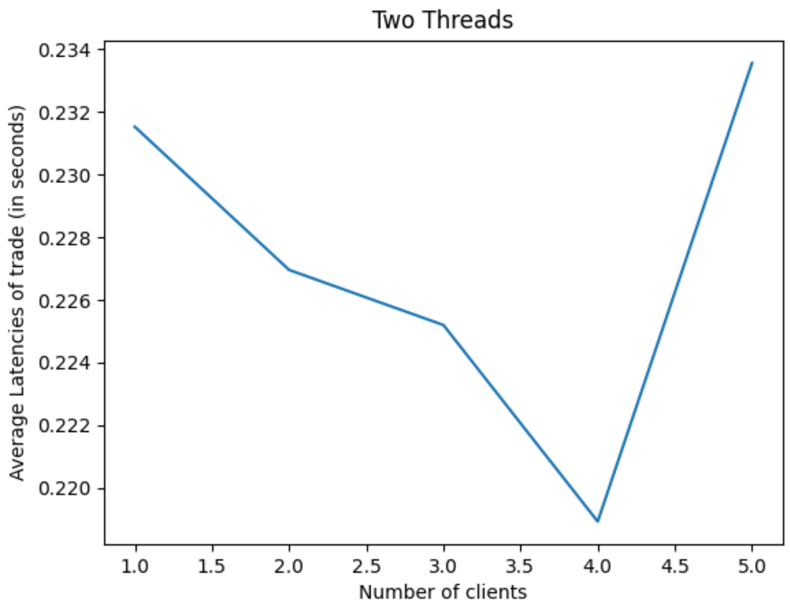
Number of threads = 4 in the server



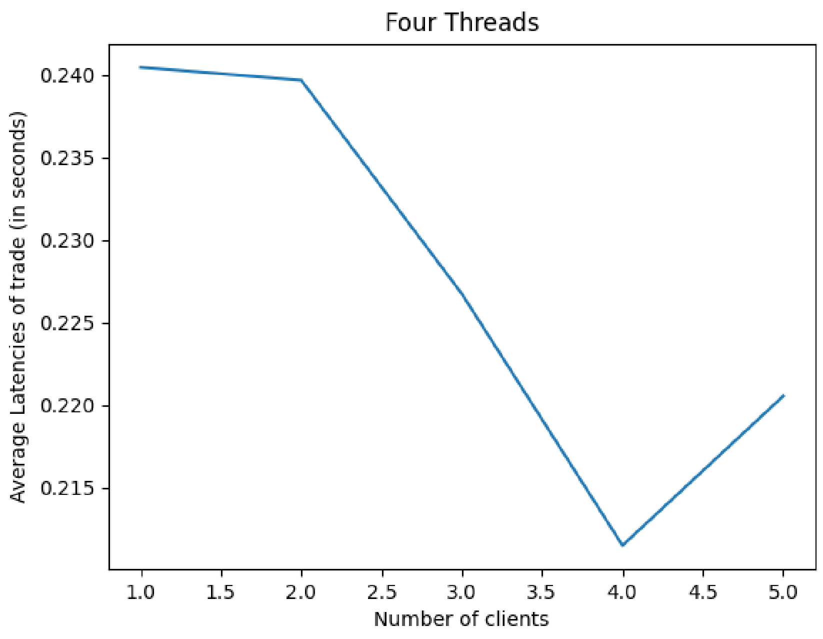
Observation for lookup:

Generally speaking, the average latency should decrease by increasing the number of threads as the number of requests is equally split among the threads.

For Trade:
Number of threads = 2



Number of threads = 4



Observation for trade:
Since we are using 2 threads and 4 threads to generate graphs. For 1 and 2 clients, the number of threads doesn't matter as the number of clients here is <= 2 (minimum number of threads). So, if we have to compare clients for 3, 4, and 5, then on average by increasing the number of threads the latency is decreasing which is expected behavior.

1. How does the latency of Lookup compare across part 1 and part 2? Is one more efficient than the other?

   The latency of lookup with sockets is lesser than the latency with gRPC.

When the latency of lookup is compared in part 1 and part 2, in our experiments, socket implementation is working more efficiently than the gRPC implementation. In general, gRPC servers are more efficient when we are using high-computation client-server applications. For small client-server applications, then sockets may be a better option. In our case, since our application is a low computational application, socket is performing well.

2. How does the latency change as the number of clients (load) is varied? Does a load increase impact response time?

On average the latency is increasing as the number of clients is increasing, which is because the number of requests is increasing the load on the server. Increasing the load increases the response time as we are using a static number of threads.

3. How does the latency of lookup compare to trade? You can measure latency of each by having clients only issue lookup requests and only issue trade requests and measure the latency of each separately. Does synchronization pay a role in your design and impact performance of each? While you are not expected to do so, use of read-write locks should ideally cause lookup requests (with read locks) to be be faster than trade requests (which need write locks). Your design may differ, and you should observe if your measurements show any differences for lookup and trade based on your design.

In our graphs, the latency of lookup operations is lesser than the latency of trade operations as the trade operations involve both read and write operations. But in the case of lookup operations, only read operations are involved. So extra write operation is increasing the latency in the trade requests.

4. In part 1, what happens when the number of clients is larger the size of the static thread pool? Does the response time increase due to request waiting?

With 2 threads in part 1, on varying the clients from 3 to 5 on an average the requests load increases the threads become occupied, and the requests are waiting in the task queue which in turn increases the response time.