TESTING AND EVALUATION DOCUMENT

for

Mini Trading Application

COMPSCI 677 Lab 1

Prepared by: Mudit Chaudhary (32607978)
Girish Baviskar (33976648)

Contents

1 Evaluation methodology

1.1 What do we evaluate?

We perform evaluation by measuring latency in milliseconds for both part 1 and part 2. To make the comparison fair between part 1 and part 2, we measure the time required to open connection, process and receive response, close connection. Example of latency measuring code snippets are shown below:

- Part 1: Here `send` creates a connection, sends request, receives response, and closes the connection.
```
methodStartTime = System.currentTimeMillis();
Object serverResponse = send(this.hostname, this.port, methodCall);
elapsedTime = System.currentTimeMillis() - methodStartTime;
```
- Part 2: Here `performLookup` calls the lookupService using a gRPC blockingstub.
```
methodStartTime = System.currentTimeMillis();
this.performLookup(chosenStockName);
elapsedTime = System.currentTimeMillis() - methodStartTime;
```

1.2 How do we evaluate?

We run the server on an Edlab machine and run the clients on a local machine. The local machine is connected to Edlab port using SSH Port Forwarding, which we have described in `README.md`. All the clients were run in parallel.

We run the following load testing experiments:

- **Part 1** Varying the number of clients and threadpool from 1 to 5.
  1. Send 1000 random lookup requests and average the response times.
- **Part 2** Varying the number of clients from 1 to 5 and setting max threadpool size to 10.
  1. Send 1000 random lookup requests and average the response times.
  2. Send 1000 random trade requests and average the response times.
  3. Send 1000 random lookup requests with the update client running in the background and average the response times.
  4. Send 1000 random trade requests with the update client running in the background and average the response times.

We run the each experiment 2 times and report its average.

2 Evaluation Results

Fig. 2.1 shows the variation in latency for varying number of clients and threads in the thread pool for part 1. We can observe that increasing the number of clients shows a higher average latency. Moreover, the latency decreases with increasing the number of threads in the thread pool. Table 4.4 shows the raw results. Fig. 2.2 shows the variation in latency for varying number of clients between lookup calls of part 1 and part 2. For this evaluation, we do not run the update client in the background for part 2. We can observe that part 2 is consistently performing better than part 1. Fig. 2.3 shows the

Figure 2.1: Part 1 Latency (milliseconds) for varying number of clients and threads

difference in latency between trade and lookup calls for part 2. For this evaluation, we have an update client running in the background, which sends price update calls to the server at random intervals. We notice that lookup operation has a lower latency compared to trade. This is because our implementation uses separate locks for read and write operations (ConcurrencyHashMap). As update and trade calls both perform a write operation, there is a contention for the locks, leading to slow down.

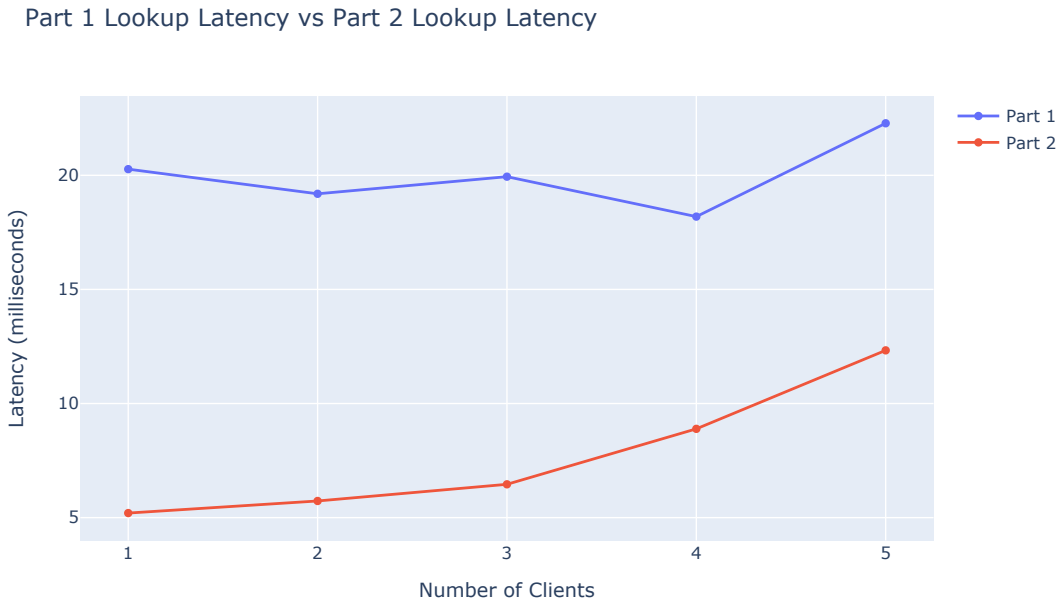

Figure 2.2: Latency variation between part 1 and part 2 lookup calls

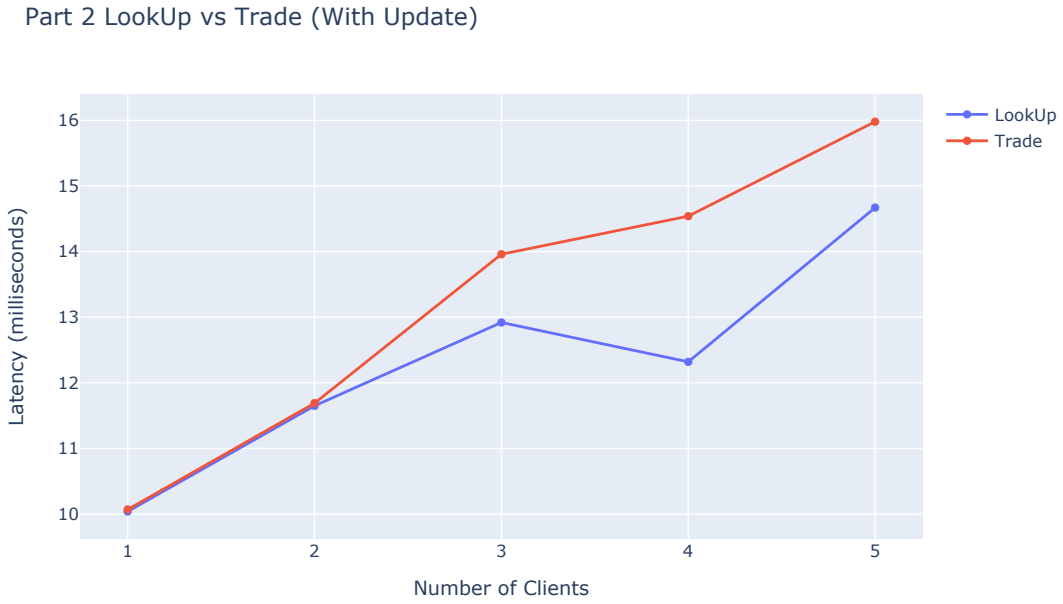

Figure 2.3: Latency variation between part 2 lookup and trade calls

3 Lab Questions

1. How does the latency of Lookup compare across part 1 and part 2? Is one more efficient than the other?
   Ans: Part 2 is more efficient than part 1 as we can observe in Fig.2.2

2. How does the latency change as the number of clients (load) is varied? Does a load increase impact response time?
   Ans: We observe that as the number of clients increases, the response time/latency also increases.

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 play 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.
   Ans: We observe that lookup operation has a lower latency compared to trade. Yes, synchronization plays a role in our design and impact the performance of each. This is because our implementation uses separate locks for read and write operations using ConcurrencyHashMap. As update and trade calls both perform a write operation, there is a contention for the locks, leading to higher latency.

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?
   Ans: For part 1, if the number of clients is larger than the threadpool, the latency rises. The latency increase would be sharper if the lookup request was more time consuming, however, due to its lightweight nature the increase is small.

4 Appendix

Clients	Thread	Latency (millseconds)
1	10	5.205
2	10	5.73
3	10	6.46
4	10	8.89
5	10	12.33

Table 4.1: Part 2 latency for lookup call without update client

Clients	Thread	Average response time (millis)
1	10	10.075
2	10	11.695
3	10	13.965
4	10	14.545
5	10	15.98

Table 4.2: Part 2 latency for trade call with update client

Clients	Thread	Latency (milliseconds)
1	10	10.048
2	10	11.65
3	10	12.9265
4	10	12.32
5	10	14.67

Table 4.3: Part 2 latency for lookup call with update client

Number of Clients	Number of Threads	Latency (milliseconds)
1	1	27.38
2	1	28.83
3	1	29.685
4	1	36.675
5	1	36.895
1	3	20.64
2	3	26.475
3	3	26.96
4	3	33.195
5	3	35.28
1	5	20.27
2	5	19.19
3	5	19.945
4	5	18.185
5	5	22.28

Table 4.4: Part 1 Latency (milliseconds) for varying number of clients and threads