Evaluation & Performance Measurement

In terms of **functional test result**, please check out the **"output" file** for details.

We also provide completed **load test screenshots** in **"output" file**. Hence, in this document, we mainly focus on analyzing the results we have.

In this part, we perform a simple load test and performance measurements. The goal here is understand how to perform load tests on distributed applications and understand performance.

1. Load Test Results of Part 1

Here we vary the number of clients from 1 to 5 and measure the average latency as the load goes up.

Table1: Load test result of Part 1

Number of Clients	Average Latency / s
1	0.0108
2	0.0135
3	0.0158
4	0.0162
5	0.0174

Then we make a plot showing number of clients on the X-axis and response time/latency on the Y-axis.
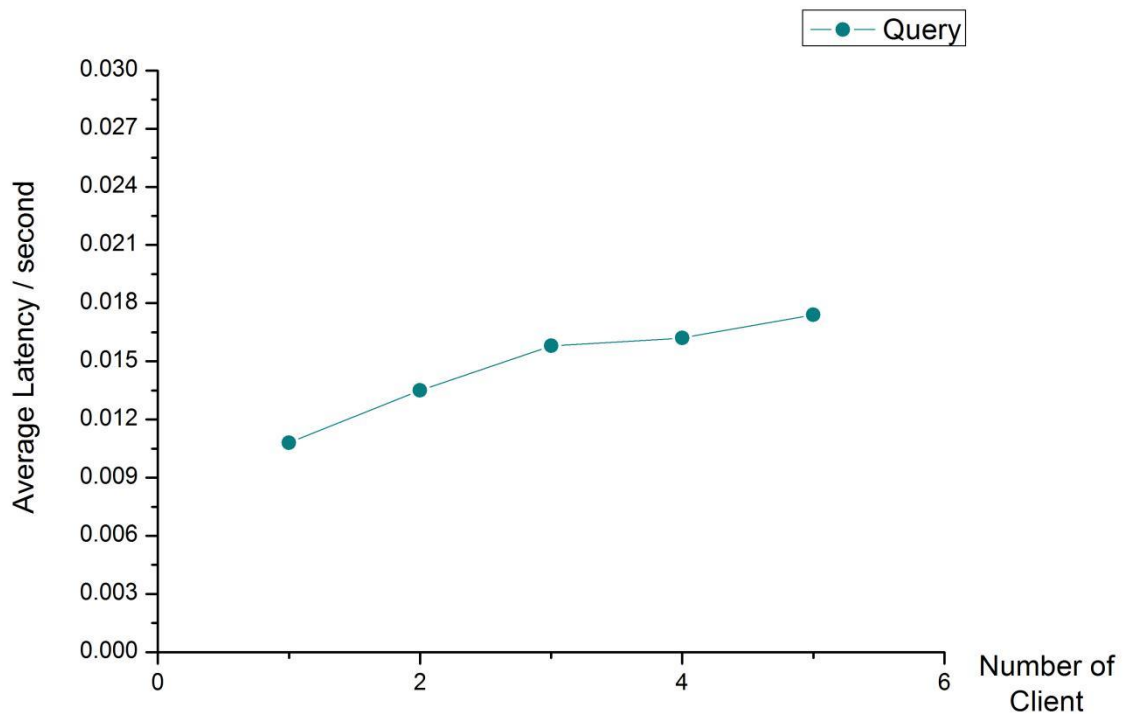

Figure1: Load test result of Part 1

2. Load Test Results of Part 2

Here we vary the number of clients from 1 to 5 and measure the average latency as the load goes up for methods query and buy separately.

Table2: Load test result of Part 2

Number of Clients	Query Latency / s	Buy Latency / s
1	0.0371	0.0373
2	0.0373	0.0374
3	0.0395	0.0395
4	0.0405	0.0407
5	0.0424	0.0435

Then we make a plot showing number of clients on the X-axis and response time/latency on the Y-axis.
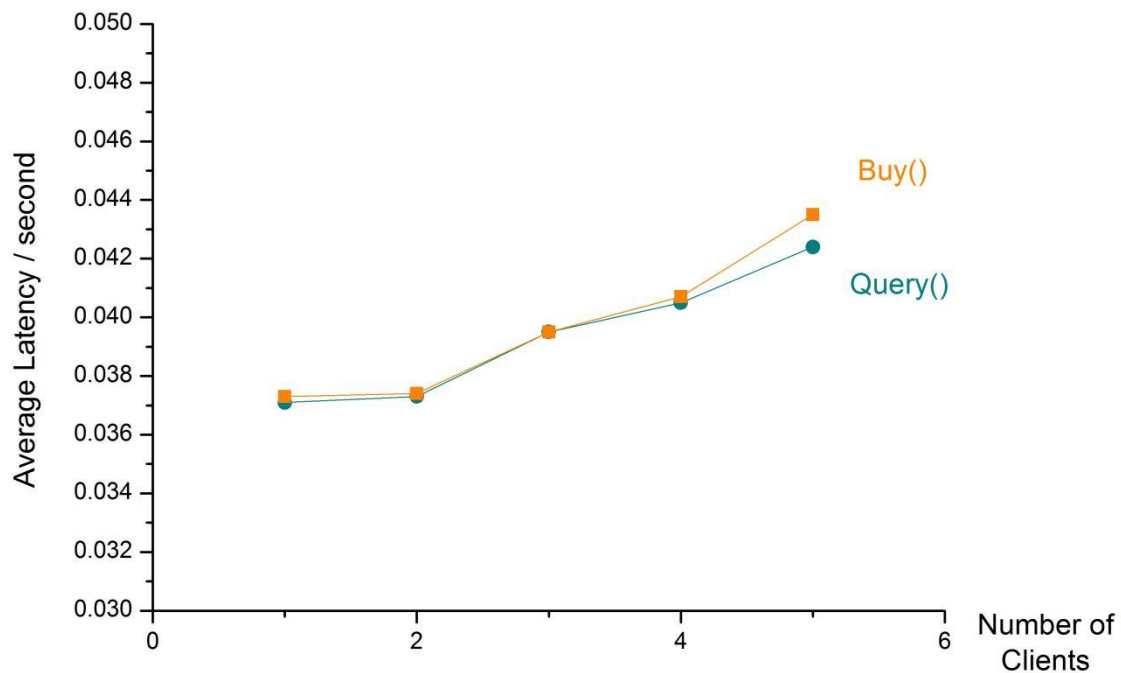

Figure2: Load test result of Part 2

3. Questions in Part 3

In this section, let's analyze load test results by answering following questions.

Problem 1

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

Solution:

Looking at **table 1** and **table 2** as shown above, it is obvious that the latency of Query method in Part2 is about 2 or3 times larger than that in Part1.

Only considering the average latency of each Query request we measure, we could say the way we implement Query in Part1 is more efficient. But as we know gRPC has lots of advantages, such as working across multiple languages and platforms.

Problem 2

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

Solution:

As the **figure 1** shows, the average latency recorded a growth from 0.0108 seconds to 0.0174 seconds, when the number of clients increases from 1 to 5.

As the **figure 2** shows, the average Query latency recorded a growth from 0.0371 seconds to 0.0424 seconds, when the number of clients increases from 1 to 5. The average Buy latency recorded a growth from 0.0373 seconds to 0.0435 seconds, when the number of clients increases from 1 to 5.

Load increase is literally able to make the response time longer. However, when the number of clients is smaller than the size of thread pool, the amount of latency increase can be really small, that is because the server is actually not overloaded. In this case, we find out that sometimes you may not even see an increase.

But if the number of clients is larger than the size of thread pool, which means the server is kind of overloaded. The increase of latency could be more pronounced. For more details, you can check out my answers in Question4.

Problem 3

How does the latency of query compare to buy? You can measure latency of each by having clients only issue query requests and only issue buy 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 query requests (with read locks) to be be faster than buy requests (which need write locks). Your design may differ, and you should observe if your measurements show any differences for query and buy based on your design.

Solution:

As the **table 2** and **figure 2** shown above, we can clearly see that the latency of Query method is actually sort of smaller than the latency of Buy method, but not very much in our load test.

In order to solve the concurrency&synchronization problems, we implement the toy store using both locks and Condition variables. Note that a condition variable is always associated with some kind of lock.

Specifically, querying and buying will read from or write to the product catalog, which makes it a shared data structure. So the underlying lock of condition variables is applied to the product database. However, the use of read-write locks should ideally cause query requests (with read locks) to be be faster than buy requests (which need write locks). Hence, from our measurements, we can observe the latency of querying is smaller than the latency of buying.

Problem 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?

Solution:

Open 6 terminals and type in the toy name as follows. In this case, the number of clients(that is 6) is larger than the size of static thread pool(that is 5). Now simulate different number of clients, and we can check out the latency of each query as shown below.

When the number of clients is larger than the size of static thread pool, even though "consumer" are all busy, the "producer" will push requests into the queue constantly. That is, those requests in the queue have to wait for the next available "consumer" thread. Hence, the latency could increase.

For instance, looking at the screenshot, sometimes you can see a super large latency, such as **0.0469 seconds** in the first terminal, which is about 4 times of other latency.

Hence, in this case, the server is kind of overloaded. the average latency is 0.0176 seconds, which is larger than all of the latency in **figure 1**. The response time increases due to the request waiting in the request queue.