Part3: Evaluation and Performance Measurement

Our team conducted a straightforward load test to evaluate the performance of Part1 and Part2. We utilized our machine to deploy clients ranging from 1 to 5, and these clients simultaneously sent requests to the server, which we hosted on Edlab. We carefully measured the performance metrics for both Part1 and Part2 during the load test.

We have presented the results of our load test in the form of plots, where the number of clients is shown on the X-axis, and the time taken to process the requests is displayed on the Y-axis, in seconds. The plots for Part1 and Part2 show how the time taken to process requests varies with the number of clients.

The latency numbers below are for 1000 requests per client

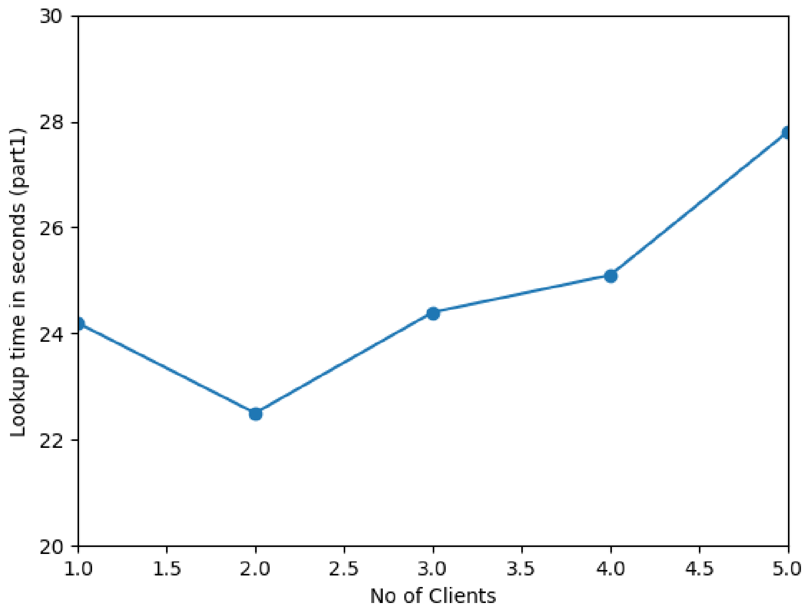

Fig 1: Lookup (Part1)

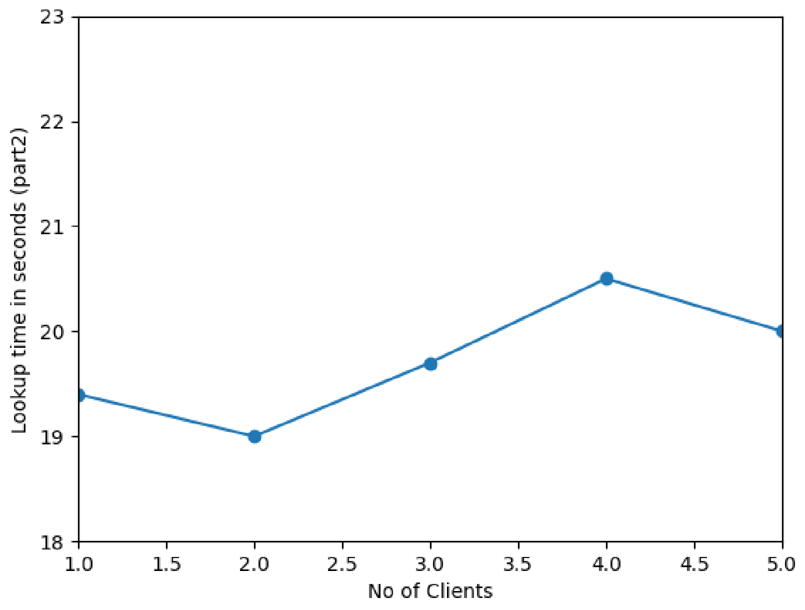

Fig 2: Lookup (Part2)

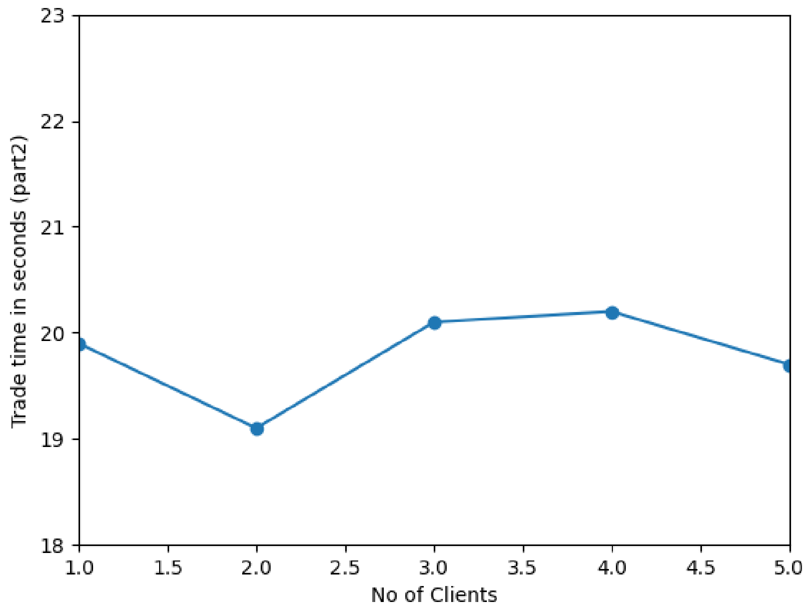

Fig 3: Trade (2)

Here are our observations:

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

   Our findings indicate that while there is a similar pattern in the variation of time taken to process increasing numbers of clients in both Part1 and Part2, there is a noticeable difference in the latency of lookup between the two. Specifically, we observed that the time taken to perform lookup in Part1 is significantly longer compared to Part2. This provides strong evidence that the server utilizing gRPC technology is more efficient compared to the traditional server. Therefore, the results of our load test support the use of gRPC technology for improved server performance.

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

   Our analysis of Part1 shows that the latency, or the time taken to complete a request, tends to increase when the number of clients exceeds 3. This is because we had only initialized 3 threads in the thread pool, which means that any additional requests beyond this limit are being queued. As the number of clients sending requests to the server concurrently increases, the number of requests queued also increases, leading to an overall increase in latency. In other words, the bottleneck in the system is the limited number of threads in the thread pool, which is causing a delay in processing requests.

   However, in Part2, we found that the average latency appears to be independent of the number of clients connected to the server. For the given load, the server was able to handle up to 5 clients making multiple concurrent rpc calls without any significant increase in latency. However, it is worth noting that this may not be the case if the number of clients increases beyond 5, since we only initialized 3 threads with a maximum of 30 concurrent rpc calls processing at a time. Thus, it is important to consider the thread pool size and maximum concurrent rpc calls when scaling up the number of clients.

3. **How does the latency of lookup compare to trade?**

   Based on our observations, we found that the latency of lookup calls and trade calls is quite similar on average. This suggests that the performance of the server is consistent and stable for both types of calls. However, it is important to note that there may be variations in latency for specific requests due to factors such as network congestion or server load, but on average, the latency is similar for both types of calls.

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

   As we have discussed in the previous question, we found that when the number of clients connected to the server is greater than the number of threads initialized, the response time tends to increase. This is because a larger number of requests are queued, which leads to an increase in the overall waiting time for requests to be processed. In other words, as the number of clients sending requests to the server increases, it is important to ensure that the number of threads initialized is sufficient to handle the load and prevent queuing of requests, which can lead to longer response times.