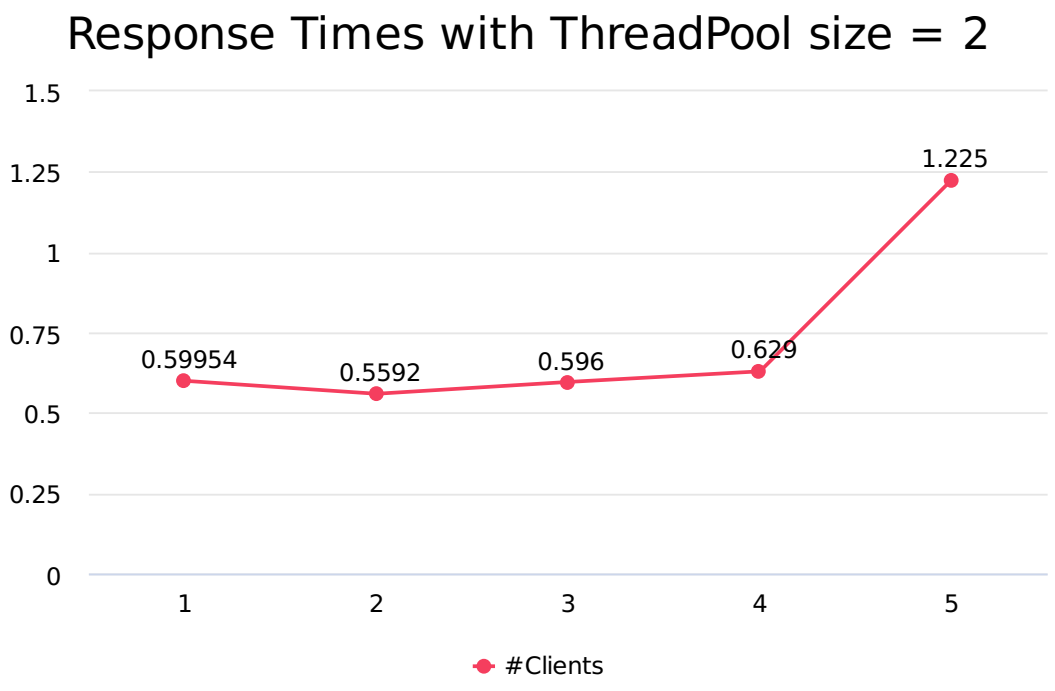
Evaluation Doc

PART 1 : Socket Connection and Handwritten Thread Pool

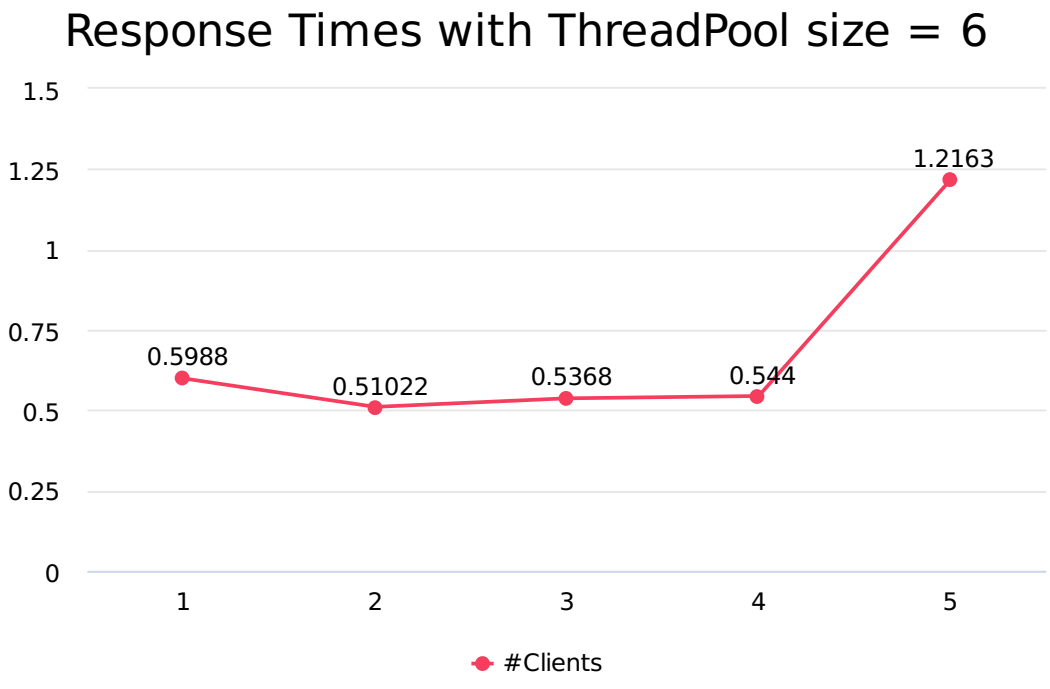
- **Lookup Method : For 3000 iterations, ThreadPool size = 2, the response time is in milliseconds**

# Clients	1	2	3	4	5
Average Times	0.59954	0.5592	0.596	0.629	1.225



- **Lookup Method : For 3000 iterations, ThreadPool size = 6, the response time is in milliseconds**

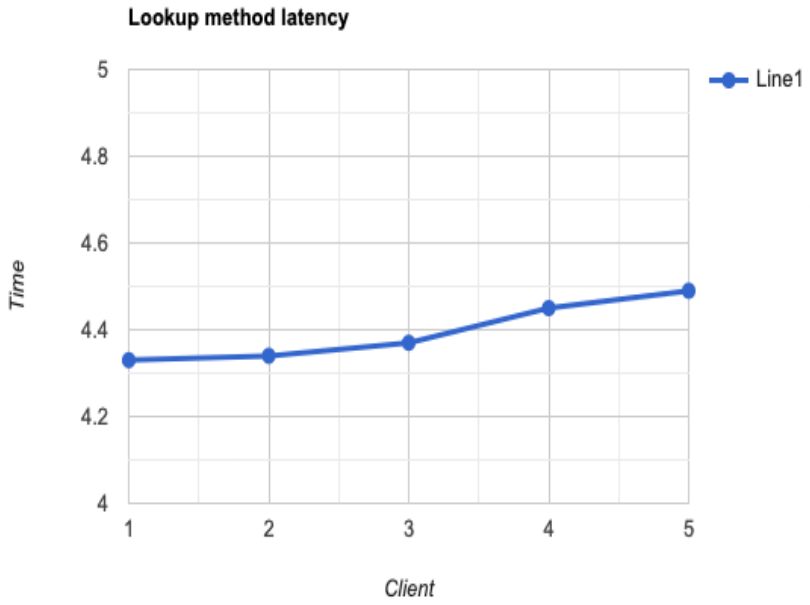
# Clients	1	2	3	4	5
Average Times	0.5988	0.51022	0.5368	0.544	1.2163



PART 2 : GRPC

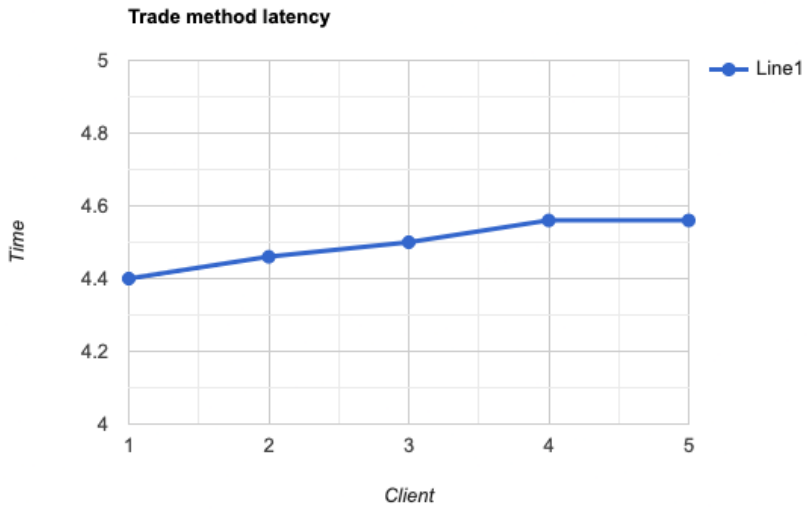
- **Lookup Method : For 3000 iterations, the response time is in milliseconds**

# Clients	1	2	3	4	5
Average Times	4.33	4.34	4.37	4.45	4.49



- **Trade Method : For 3000 iterations,** the response time is in milliseconds

#Clients	1	2	3	4	5
Average Time	4.40	4.46	4.55	4.56	4.56



Q1. How does the latency of Lookup compare across part 1 and part 2? Is one more efficient than the other?

Ans.

The Part1 latency covers :

1. s = socket.socket()
2. s.connect(hostname, port)
3. s.send()/s.recv()
4. s.close()

The Part2 latency covers :

1. channel = grpc.insecure_channel(f"{hostname}:{port}")
2. stub = rpc_pb2_grpc.StockBazaarStub(channel)
3. response = stub.Lookup(rpc_pb2.stockLookupMessage(stockName=stockName))

While calculating the client latency, the network overhead and server processing overhead is included i.e., the socket and channel establishment costs. The latency/lookup response time is less in the case of Part1 than Part2. gRPC sets up the channel once and subsequent requests share that channel but in socket, the socket is getting created for each request. So the communication establishment cost is higher in Part 1. But gRPC operates over HTTP which runs on top of TCP, while Part 1 is simple sockets

which operates over TCP. So communication cost is higher for gRPC requests in Part 2 than sockets in Part 1. Hence, the overall latency is less in case of Part 1 than Part 2.

Q2. How does latency change as the number of clients(load) is varied? Does a load increase impact response time ?

Ans. In Part 2, some change in latency is observed as the number of clients is varied. Although observed change is not significant, increasing concurrent connections(clients) to the server adds some latency, as the increased threads handling client calls incur some overheads. In Part1, as the number of client increases, the response times increase. This is because more number of client processes results in more number of concurrent processes trying to run on a thread. This results in more waiting time for the requests. Thus, the latency increases as the number of clients(load) is increased.

Q3.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?

Ans. The latency of lookup is lower as compared to trade. This is due due to the presence of synchronization such as locks used in trade method for update operation on shared variable of traded volume of stocks. Locks are essential to ensure that when one thread/process is updating a shared variable, another thread/process cannot alter that variable. This is done to maintain consistency.

Q4.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. In the first graph of Part 1, the number of clients is 5 while the size of the static thread pool is 2 while in the second graph of Part 1, the number of clients is 5 while the size of the static thread pool is 6. In most cases, the latency/response time of thread pool size 2 is more which can be because there are limited number of threads as compared to the number of client processes. Thus, more number of client processes end up waiting for a thread in case of a smaller ThreadPool. This results in more overall waiting time in case the static thread pool size is less than number of clients. With more number of threads, multiple client processes can run in parallel on different threads, which decreases the waiting time and hence the latency decreases.