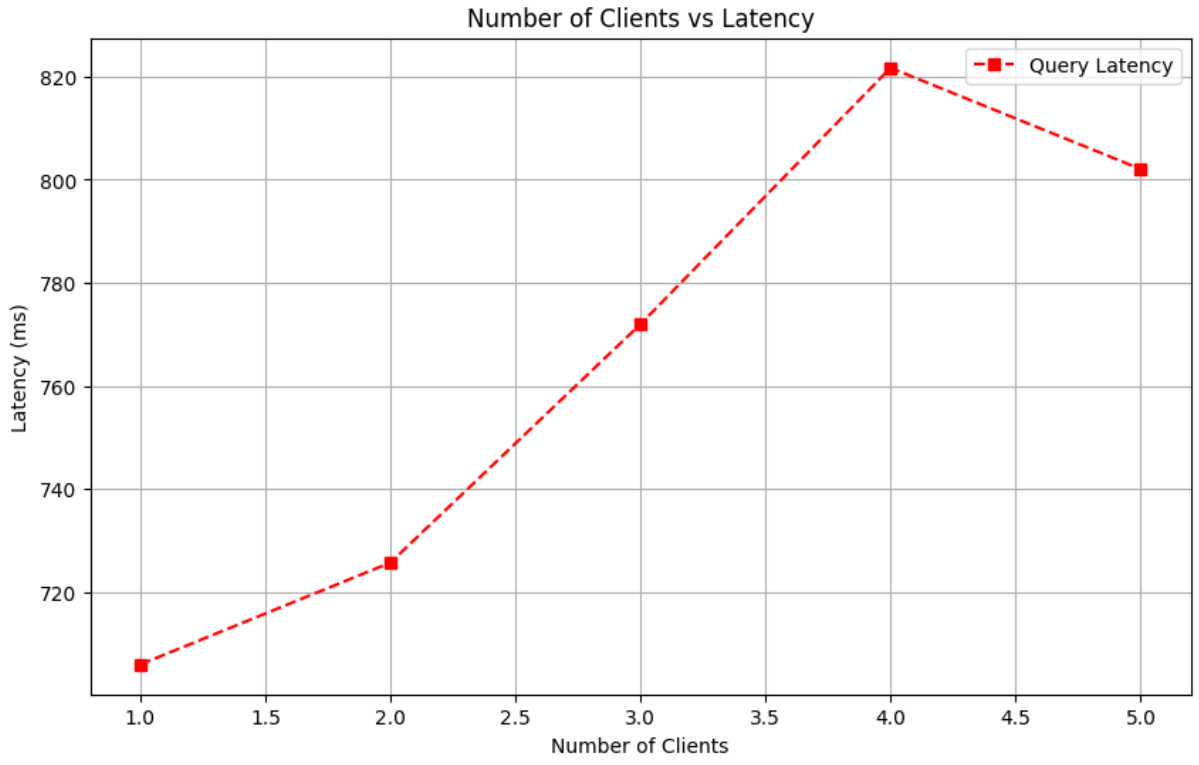
Evaluation Document

Plots for Part 1:

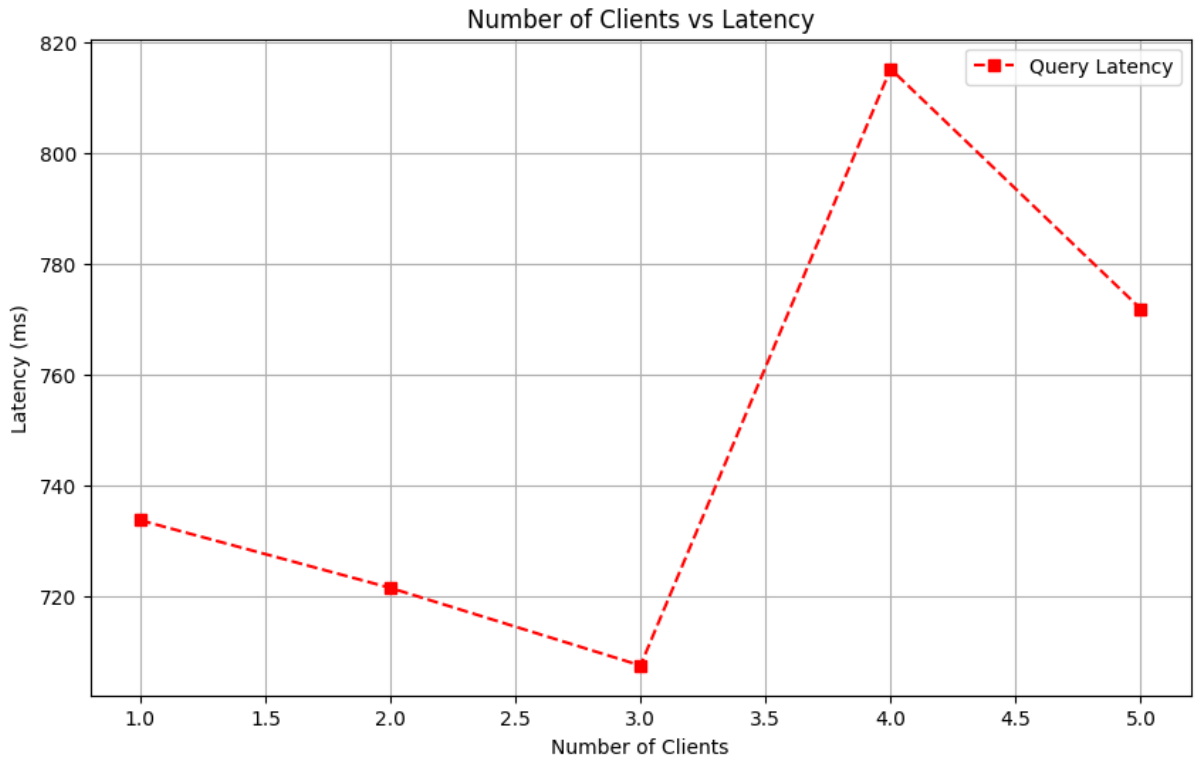
The latencies vary as the no. of clients increase and the no. of server threads increase. Tested these using EDLab server as Server and my local machine as Client.

Server with 2 threads in the Threadpool:



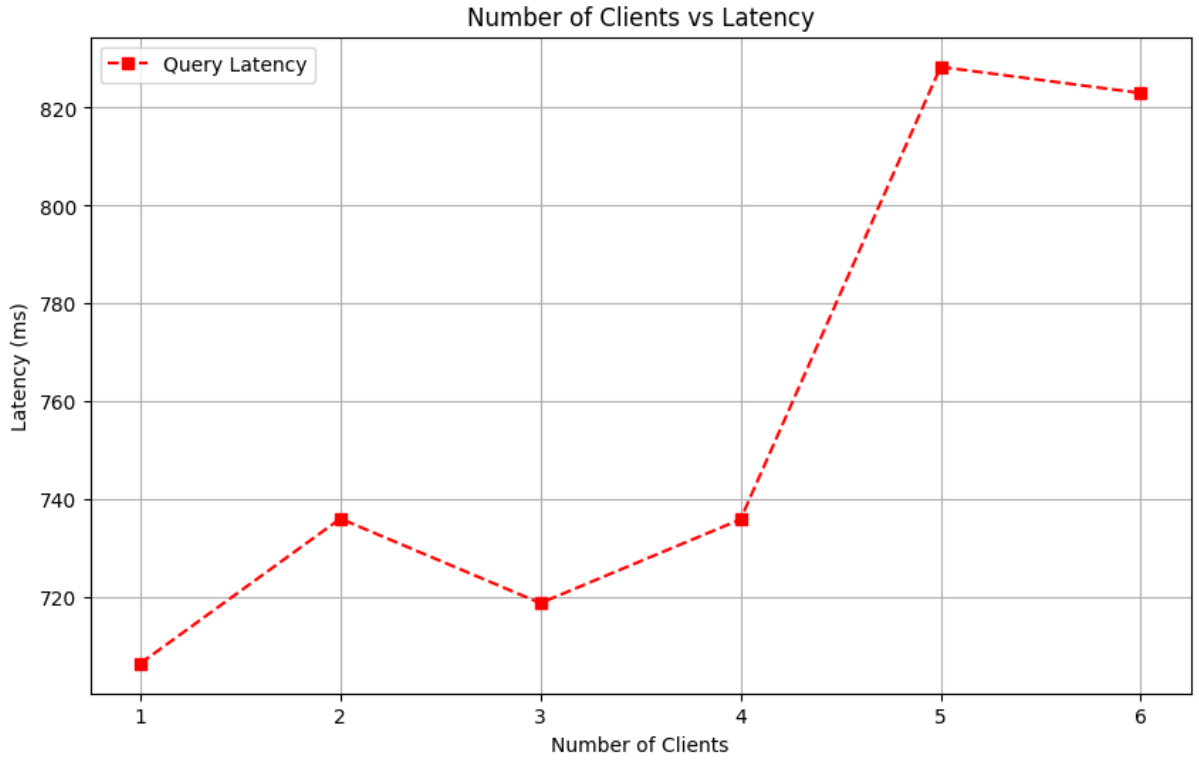
As you can see, with 2 threads the latency increases as the no of clients increase. the jump from 1 to 2 clients is not significant compared to jump from 2 to 3 clients. This is the evidence that as clients > threads in server, the latency increases due to waiting.

Server with 4 Threads in the Threadpool:

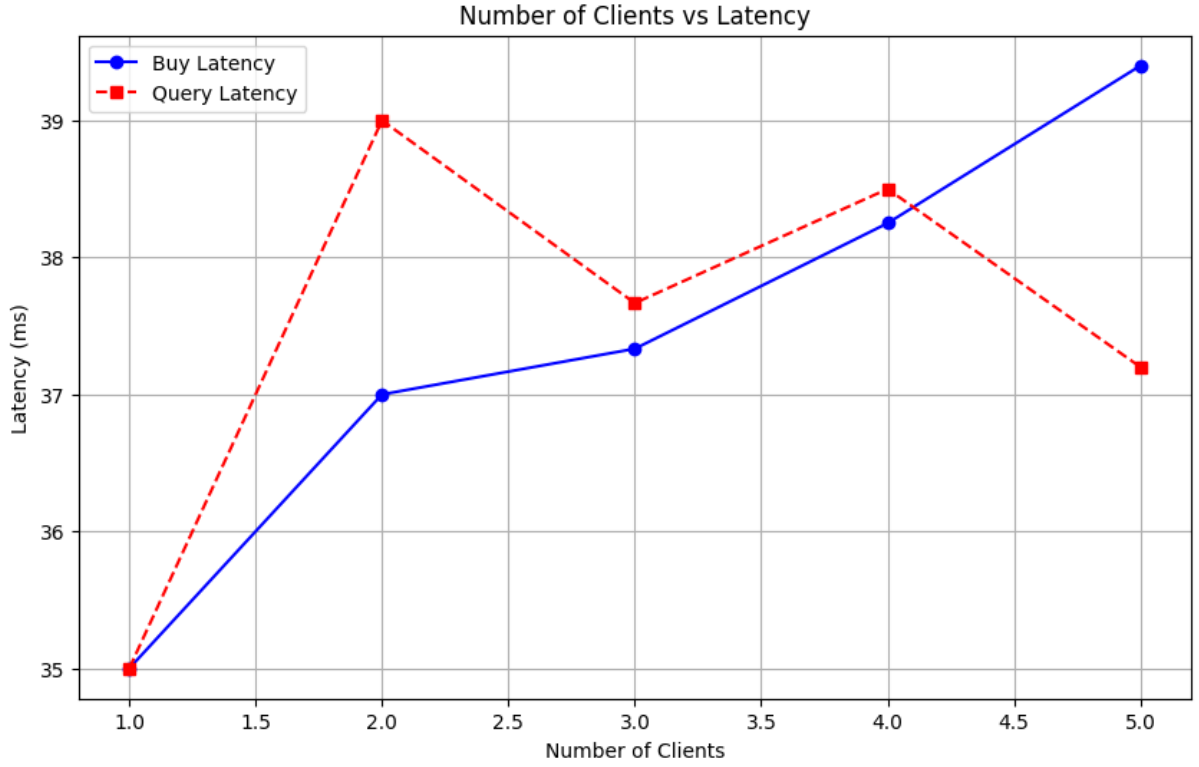


As the no of threads increase the latency decreases. But when no.of threads = no. of clients the latency shoots showing increased load on the server (almost 100 ms).

Server with 8 Threads in the Threadpool:



Plots for Part2:



As per my implementation, as the no of clients increased the latency for buy call increased and the latency for query call almost stayed the same. The buy and query latency are almost comparable for 3, 4 clients. And the buy latency increased at 5 clients crossing over query latency which is in line with “the use of read-write locks should ideally cause query requests (with read locks) to be faster than buy requests (which need write locks).”.

Answers:

Q1. The Query latency for part 2 is much lesser than query latency of part 1. Both were run EDLab server and my local machine. The part 2 implementation of the threadpool is more efficient than part 1. Because using a dynamic thread pool, which adjusts the number of threads based on workload, offers several advantages over a static custom-made thread pool. Dynamic thread pools can improve resource utilization by scaling down during low activity, reducing resource consumption, and scaling up as demand increases, enhancing throughput and response times.

Q2. As the number of clients increase, we can see an uptrend in the latencies in both part1 and part2. As the load increases, response times increases as increased demand can saturate server resources (CPU, memory, network bandwidth), leading to longer processing times for each request.

Q3. As per my implementation, as the no of clients increased the latency for buy call increased and the latency for query call almost stayed the same. The buy and query latency are almost comparable for 3, 4 clients. And the buy latency increased at 5 clients crossing over query latency which is in line with “the use of read-write locks should ideally cause query requests (with read locks) to be faster than buy requests (which need write locks).”.

Q4. From plots in part1 :

2 Threads: When the number of clients exceeds the number of server threads designated to handle requests, additional requests must wait for an available thread to become free. This waiting time contributes to increased latency. The significant jump in latency moving from 2 to 3 clients, compared to the jump

from 1 to 2 clients, highlights the threshold at which the server's capacity to handle simultaneous requests is exceeded, leading to queuing and increased wait times for processing.

4 Threads: When the number of server threads equals the number of clients, each client theoretically has a dedicated thread for handling its requests, which should ideally minimize waiting times and thus reduce latency. However, my observation of increased latency under these conditions suggests that the server is under increased load, potentially due to context switching, thread management overhead, or resource contention within the server. When every thread is utilized simultaneously, any inefficiency in handling the requests or additional computational demand placed on the server can lead to increased response times. For 5 clients, the overall trend is pointing upwards.