Evaluation Document

By: Aishwarya Malgonde, Sahil Jindal
Course: CS 677 | Professor Prashant Shenoy

Lab 1: Asterix and the Stock Bazaar

February 26, 2023

Note: We have varied the number of threads (#threads) and number of clients (#clients/load) from 1 to 5, and calculated latency for all the 25 combinations.

For part2, MAX_CONCURRENT_RPC is set to 20 for all the experiments.

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

   Comparing the latency of the Lookup function across both parts, we can observe from the figures below that - the overall average latency of part2 (Fig 2) is higher than the part1 (Fig 1). The average is around 0.04 sec for part1 and 0.11 sec for part2. We think part2 has a higher average latency because it has a http/2 overhead (grpc) as compared to part1 which has a normal socket connection.

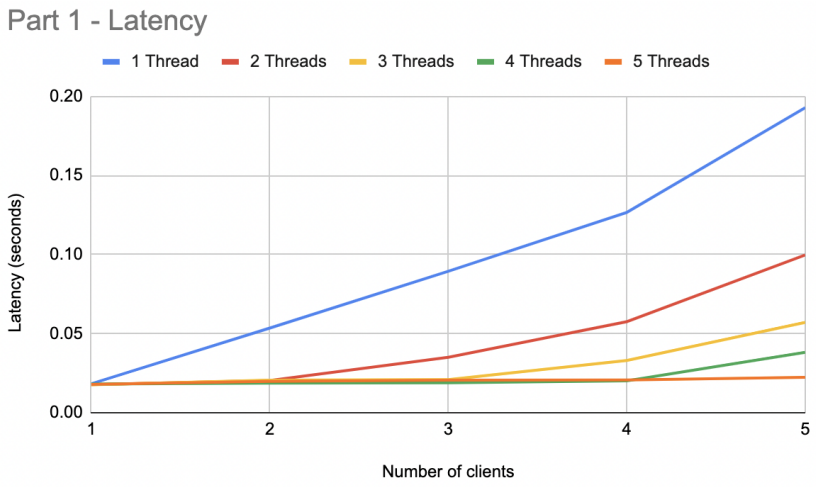

Fig 1. Part 1 Latency

But if we carefully analyze the trend for increasing #clients, we can conclude that part2 has better ability to handle multiple clients and provide a better throughput. In part1 the spikes are very evident when #clients grow bigger than #threads, but the lines in part2 are flatter and have mild slope increase as we increase the load across the 5 threadpool scenarios.

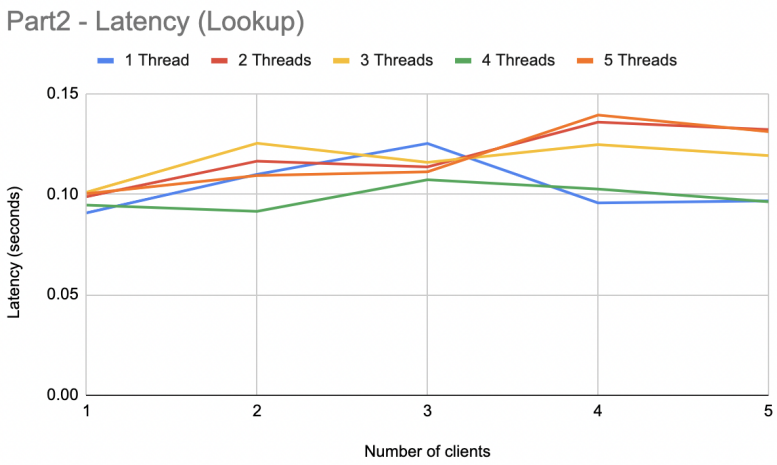

Fig 2: Part 2 latency for the Lookup method

Speaking of efficiency, we think that part2 (grpc) would be more efficient in situations when the load is higher compared to #threads, because once the grpc sets up a channel all concurrent requests coming from the clients share the same channel. Part1 would perform better in scenarios where the #clients is not going to exceed #threads, because as clients increase, there would be more waiting in the queue.

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

Part1 (Lookup) - In the graph below (Fig 3) we can see a general trend of increase in latency as the load increases across the different threadpool scenarios. The latency overall seems to be decreasing as we increase the #threads, which makes sense as we have more threads to process incoming requests and less waiting.

	1 Thread	2 Threads	3 Threads	4 Threads	5 Threads
1 Client	0.018308	0.017822	0.017673	0.018123	0.017914
2 Client	0.053502	0.020331	0.020549	0.018614	0.019833
3 Client	0.089436	0.035063	0.02093	0.018894	0.020526
4 Client	0.126554	0.057529	0.033003	0.020097	0.020625
5 Client	0.192891	0.099753	0.057062	0.038147	0.022393

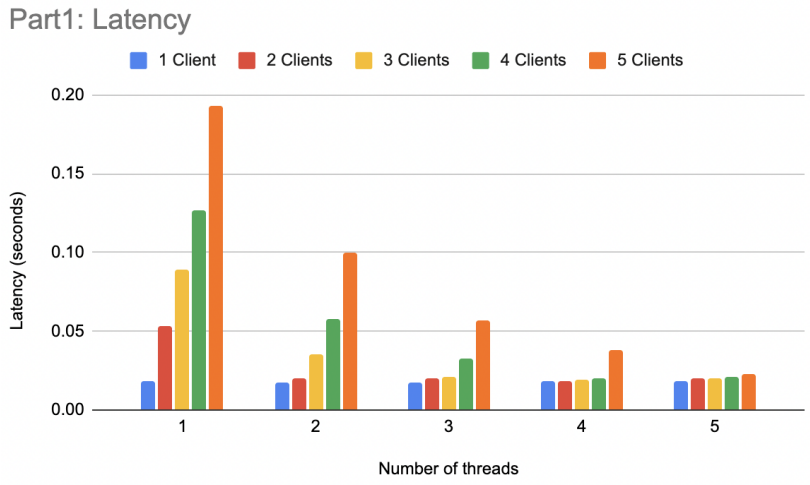

Fig 3. Part 1 latency across different threadpool scenarios for increasing #clients

Part2 (Lookup) - In the graph below (Fig 4) we can see a slight increase in latency as the load increases across the different threadpool scenarios. This increase is very mild and not as dramatic as seen in Part1 above

	1 Thread	2 Threads	3 Threads	4 Threads	5 Threads
1 Client	0.090682	0.098767	0.100947	0.09455	0.100274
2 Client	0.109877	0.116451	0.125289	0.091487	0.109262
3 Client	0.125199	0.113566	0.115806	0.107169	0.11113
4 Client	0.095667	0.135807	0.124683	0.10259	0.139343
5 Client	0.096632	0.132151	0.119156	0.096282	0.131125

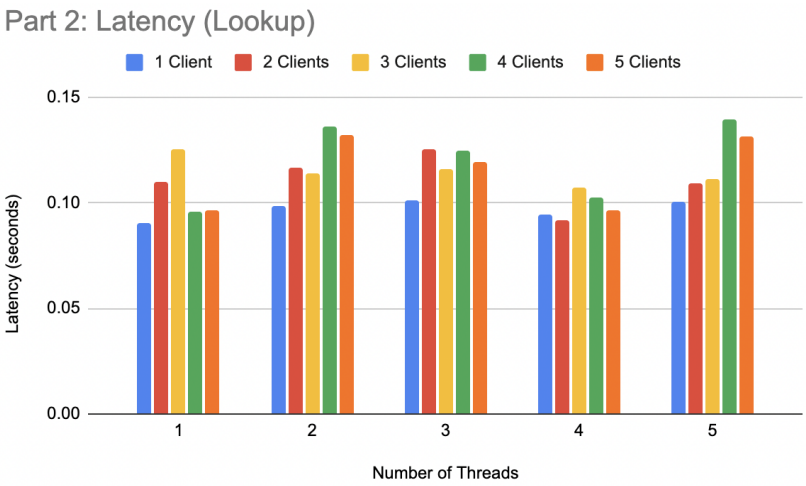

Fig 4. Part 2 latency for Lookup method across different threadpool scenarios for increasing #clients

Part2 (Trade) - Similar to the Lookup requests, as the number of clients increases the latency also increases. The average latency is higher than Part2 - Lookup graph shown above, because the trade method is updating the database, whereas Lookup is just reading.

	1 Thread	2 Threads	3 Threads	4 Threads	5 Threads
1 Client	0.18583	0.158354	0.165043	0.191128	0.150116
2 Client	0.158503	0.208717	0.179132	0.286448	0.165103
3 Client	0.301468	0.26329	0.161381	0.189771	0.187124
4 Client	0.456153	0.203672	0.238701	0.232131	0.270361
5 Client	0.450697	0.208399	0.26559	0.339483	0.336228

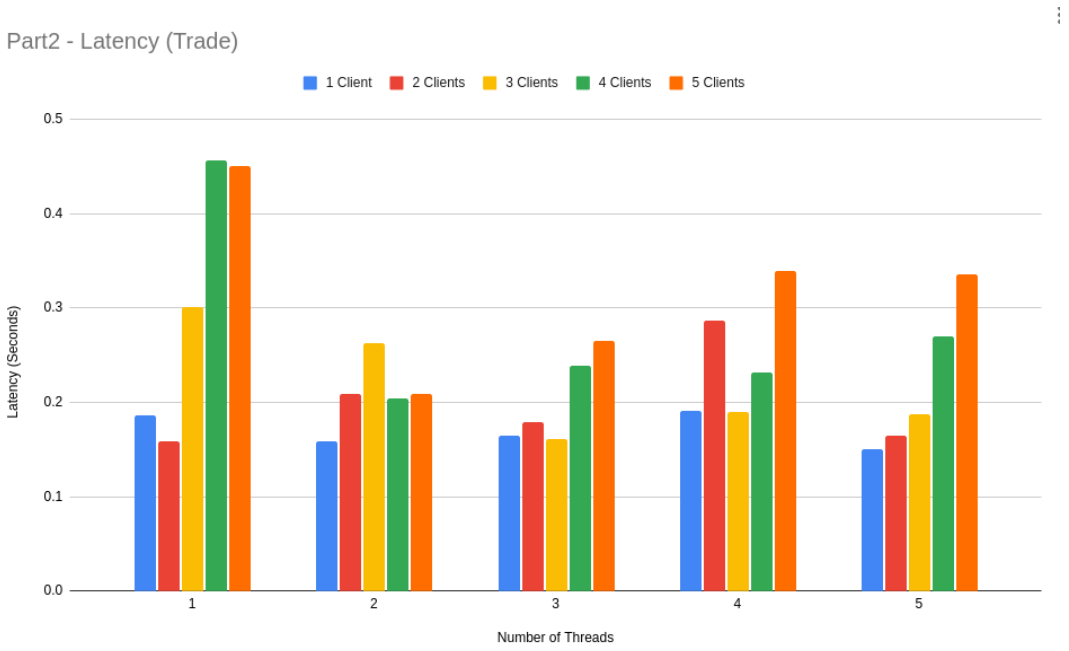

Fig 5. Part 2 latency for Trade method across different threadpool scenarios for increasing #clients

Part2 (Lookup + Trade + Update) - Here we are running an update client (every 1 second) along with a normal client (which randomly sends requests for Lookup/Trade). In this scenario, the number of maximum workers in the dynamic threadpool is 4. Compared to the graph in Fig 5 above, when x-axis (#threads) is 4, we can see a similar trend with increasing latency as seen below. Though the average latency in this case is much higher than the previous graph, because here we have multiple requesting updating the database.

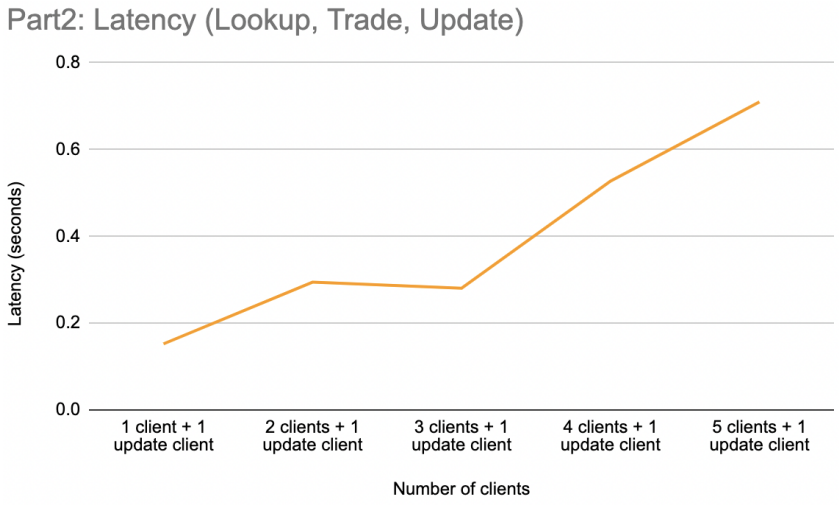

Fig 6. Part 2 latency for Lookup+Trade+Update method across different threadpool scenarios for increasing #clients

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 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.

   The average latency of lookup is comparatively faster than trade. The figures below show the latency for both the methods, with variations in threadpool size and number of clients.

   Yes, synchronization plays a role in our design. The database has a lock which needs to be acquired by the threads before it can complete the request from the queue. As seen by the figures, the trade requests have a higher slope and higher latency on average.

   For part 2, we have used in-built gRPC libraries which dynamically allocates threads. When the threads receive a lookup request it locks the stocks database and tries to read and get the price of the stock. Whereas, when a Trade request is received, the thread performs both read and write operations after acquiring the lock. Write operations are

expensive which makes the latency of trade requests higher.

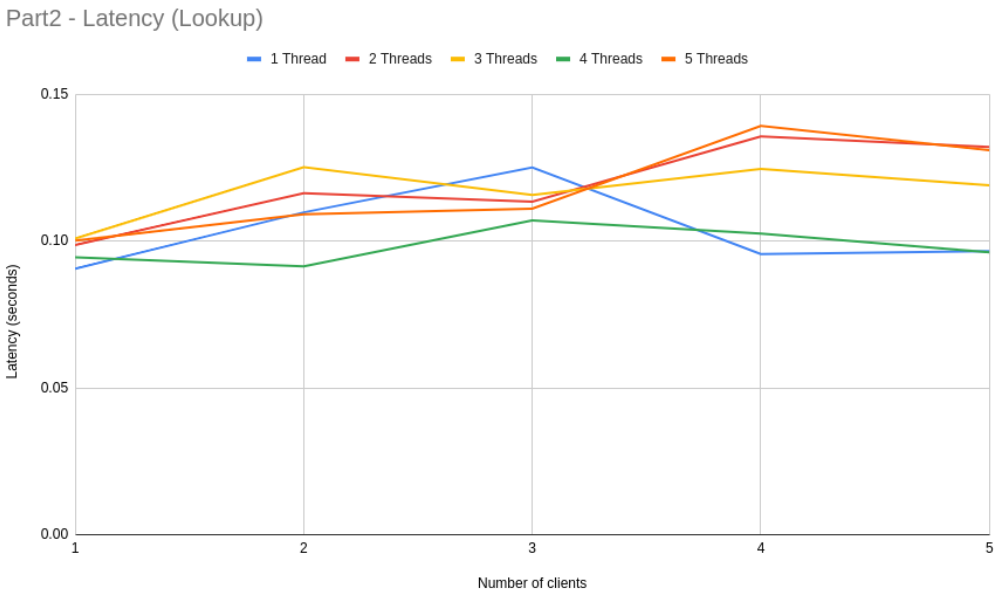

Fig 7. Part 2 latency for Lookup method across different threadpool scenarios with #clients in x-axis

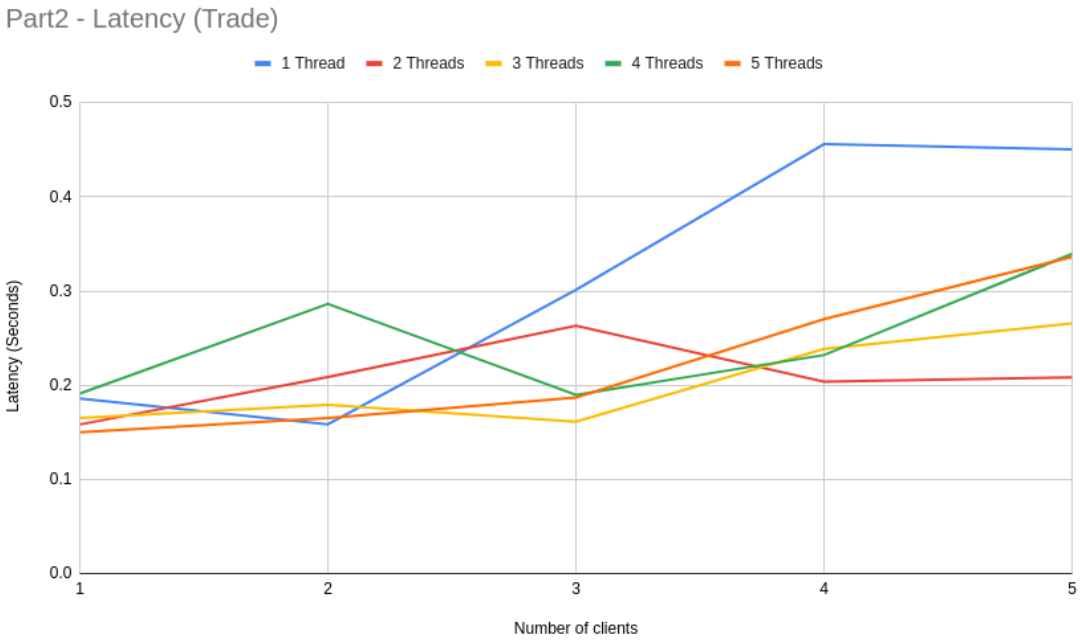

Fig 8. Part 2 latency for Trade method across different threadpool scenarios with #clients in x-axis

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?

   From the figure below Fig. 9 we can see that the latency has a general increasing trend as we increase the number of clients. An interesting observation is that when the #clients is less than or equal to the #threads, the latency doesn't vary significantly. For example, if we look at the yellow line which represents latency for 3 threads running on server side, for #clients less than or equal to 3, the line is somewhat flat, but after this point it deflects with a higher slope. This trend is consistent with the five threadpool scenarios represented in the graph below. This indicates that the client requests are waiting in the queue on the server side when the #clients become larger than the #threads, because at any given time all the threads are busy and at least these many {*#clients - #threads*} requests are queued by the server.

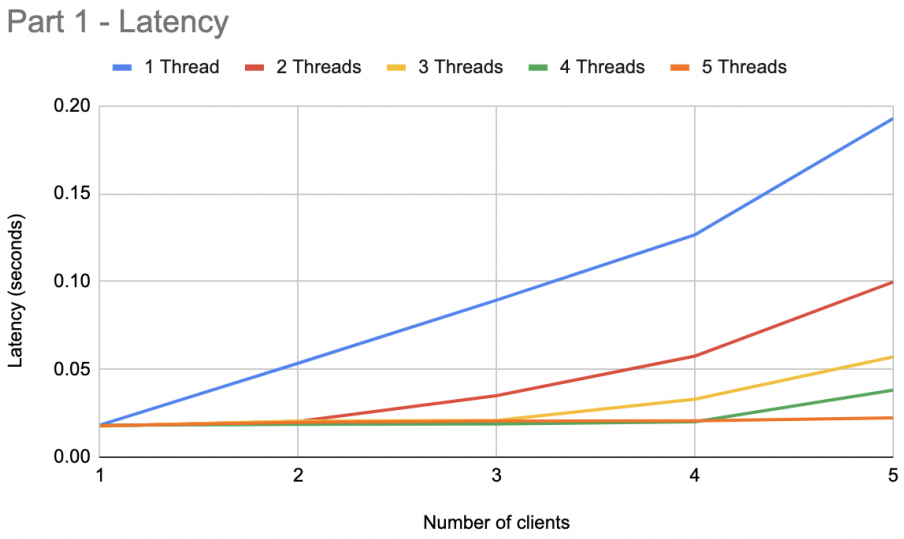

Fig 9. Part 1 latency vs #clients