COMPSCI-677: Lab 3 Eval Doc

Will Lillis and Pranav Shekar

May 1, 2023

In order to evaluate the performance of our system, we ran the dockerized version of our application on an AWS ec2 `t2.micro` instance (the instructor response in this piazza post stated that using something besides an `m5a.xlarge` instance was OK). These tests were conducted by running the `test()` function within `src/client/client.py`. The tests issues 100 requests for each concurrent client, and returns the average latency of all of these requests. The results from all clients in a given run were then averaged together, and then displayed in the graph below. We ran 5 clients concurrently for each test, all of them started via the `src/client/test.sh` script. The results from all clients in a given run were then averaged together, and then displayed in the graph below.

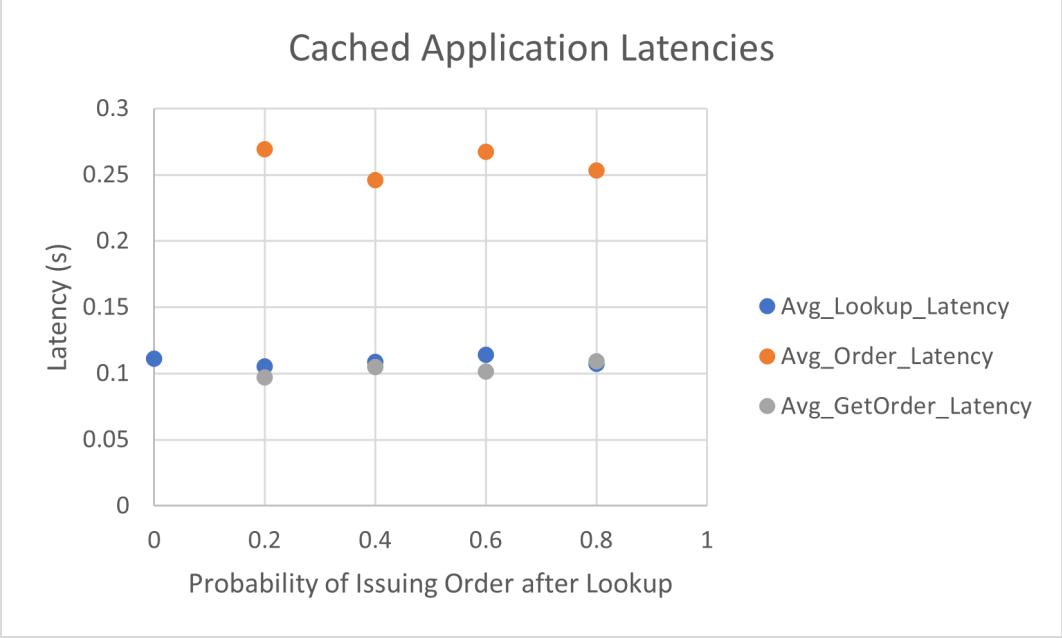

Fig. 1: A scatterplot depicting the system's performance while utilizing caching for various values of the probability p of issuing an order after making a Lookup request.

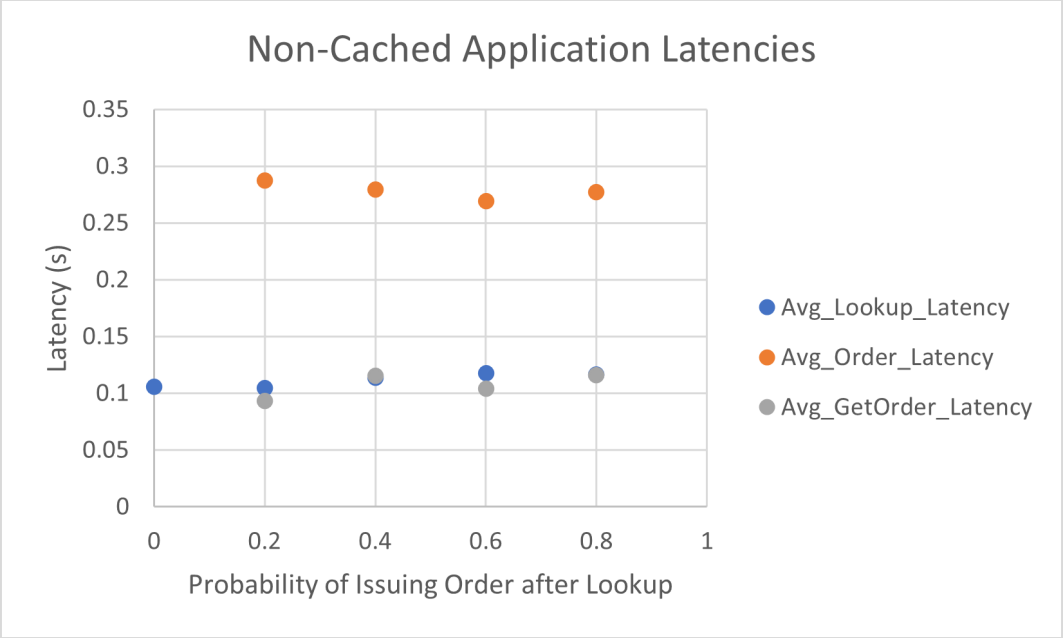

Fig. 2: A scatterplot depicting the system's performance without caching for various values of the probability p of issuing an order after making a Lookup request.

We'll first note that there isn't any data for "`Avg_Order_Latency`" and "`Avg_GetOrder_Latency`" for $p = 0$, as there aren't any orders to lookup when none are made. Next, we will note the overall (surprising) lack of a trend in the plots above. The system's performance is very stable for all values of p. We will note that we re-tested several times to ensure this wasn't a fluke. Given this, we have reached the conclusion that network latency dominated the performance of the application. Due to this, we saw no virtually no difference in performance in `Lookup` request latency between the cached and non-cached application tests.

We simulated crash failures by killing random order service replicas while the client was running (both locally and while running on AWS). This included killing both the leader replica as well as follower replicas. Such failures were transparent to the clients. Furthermore, we tested crash recovery extensively and found that all order service replicas end up with the same database file following recovery on their restart.