Advanced Computer Systems

Programming Assignment 3

University of Copenhagen

Julian Sonne Westh Wulff <bkx655@alumni.ku.dk>

Toke Emil Heldbo Reines <bkx591@alumni.ku.dk>

December 22, 2021

1 Experiment

The goal of the experiment is to measure and analyze how the number of clients concurrently accessing the bookstore affects the performance of the system, in particular latency and throughput. Thus the experiment is structured around three types of client interactions with different frequencies, that are close to the kind of behavior and frequencies we would expect from the application running in a production environment. The workload used for the experiment thus consists of these 3 parameterized interactions modeled as described in the assignment text, namely: a *Customer Interaction*, a *Stock Replenishment Interaction*, and a *New Acquisition Stock Interaction*. The frequency of these interactions used in the workload was set at 60%, 30%, and 10% respectively. Running the experiment, the workload is then used by each client to test how latency and throughput is affected when we increase the number of clients running the workload concurrently.

As we did not have any specific domain-specific knowledge about the application's usage other than what was given in the assignment text we choose to run the experiments against 3 different initial sizes of the bookstores (number of books initial in the bookstore) that we deemed were realistic. Thus we choose the initial bookstore sizes to be 0, 100, and 1000 books. Notably, 0 was chosen as this would be the starting point when the application is deployed, though in a realistic scenario the frequencies of the different interactions would most likely be different, especially the *New Acquisition Stock Interaction* would most likely be the most dominant interaction. But to keep the experiments comparing the frequencies was kept static for all of them.

To populate our experiments with data we used a number of different generators. For generating books, both for the initial bookstore and for *New Acquisition Stock Interaction* interaction, we used a random books generator. The purpose of the generator was to generate book objects that resemble what we would expect in a 'real world' deployment of the application in terms of the size of the objects. Thus the books generator generates books with a title and an author consisting of a random alphanumerical string of 32 characters, as this seemed reasonable. We choose to keep the number of characters fixed rather than dynamic (e.g. between 2 and 32 characters) to keep the total size of the initial bookstore consistent between individual runs with a different number of clients. Furthermore, as the *Customer Interaction* looks up editor picks, we choose that 10% of the books generated would be marked as an editor pick to make this interaction meaningful. As the rest of the fields of a book are integers we did not change anything particular for these as it does not affect the size.

To make sure only successful interactions contributes to the measurements we made sure to reject experiments where the fraction of unsuccessful interaction was less than 1%. Furthermore, the experiment was both carried out using local testing and RPC.

The experiments were run on a single machine with an AMD Ryzen 5600X CPU with 6 cores, 12 threads, 3MB L2 Cache, 32MB L3 Cache, and a Clock frequency of 3.7-4.6 GHz, and 32 GB 3600 MHz memory.
Because of this setup, we wanted to run the tests with a thread count in which we would exploit the number of threads. Knowing that memory was not an issue, we wanted to run tests with a worker thread-count +/-

of the amount of physical and virtual (threads) cores in the CPU, as a multiplicative of those and tests with a worker thread-count way above the core count. We initial ran all the tests with a worker thread-count from 1 to 128, but the final and reported results, we reduced this to $1 - 60$, as these results did not show any new interesting changes in the trend, and due to the exponential increase in the time to run the tests.

For a detailed description of the parameter, configuration see Table 1 and Table 2 in the appendix in subsection 4.1. These parameters were chosen because we deemed that they seemed realistic within the use case of the application and gave reliable results.

2 Results

In all 6 charts in Figure 1 and Figure 2, the left y-axis belongs to the Local runs and the right y-axis belongs to the RPC runs - the unit is the same. The plotted numbers are the averaged results of 4 runs per configuration. The tests give a clear indication of the overhead/latency at play in the RPC tests. The tendency/trend of the latency graphs are similar from local to RPC runs, but the difference is more than ten-fold in the actual numbers (in milliseconds).

If we look at the throughput graphs in Figure 1 we observed that the maximum throughput was reached when the number of clients was close/equal to the number of cores in the CPU, which is to be expected. Furthermore, we observe that for all the experiments the latency increased exponentially with the number of clients, though the latency for RPC calls was with much higher factor than for the local test. One reason for this behavior might be that when the number of clients increases the chance of having cache misses in the CPU also increases. In other words, using exactly 100% CPU leaves no threads waiting. Having 11 working threads on a CPU with 10 cores (110% usage) leads to 1/11 threads waiting. Using 200% (20 threads on a 10 core CPU), leaves 10/20 threads waiting. Scaling this theory gives us a trend similar to what we observe in the graphs.

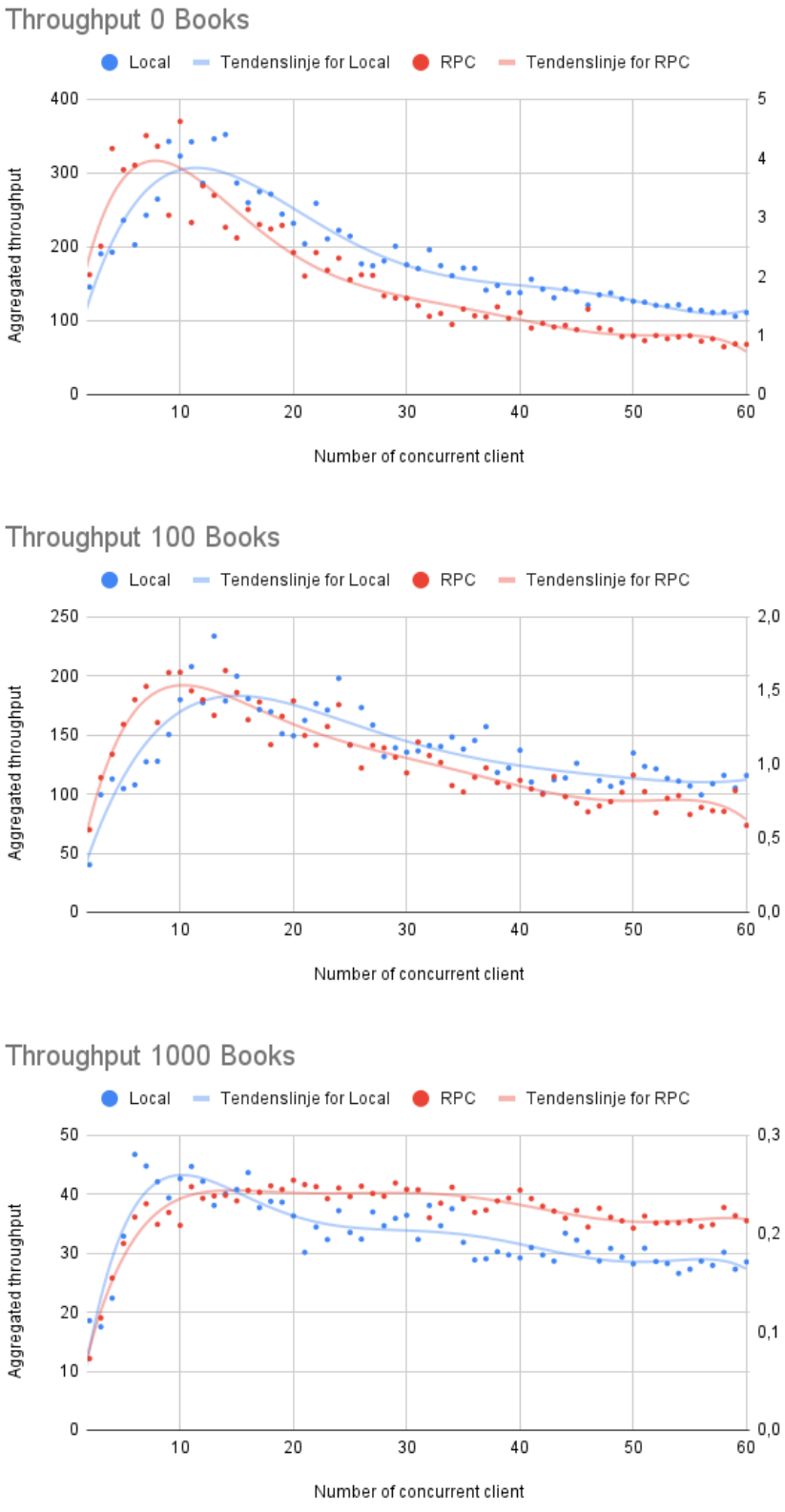

Figure 1: Graphs of the average measured aggregated throughput vs. number of clients (threads) for experiments using a local and RPC bookstore and the initial bookstore described in section 1 with either 0, 100, or 1000 books.

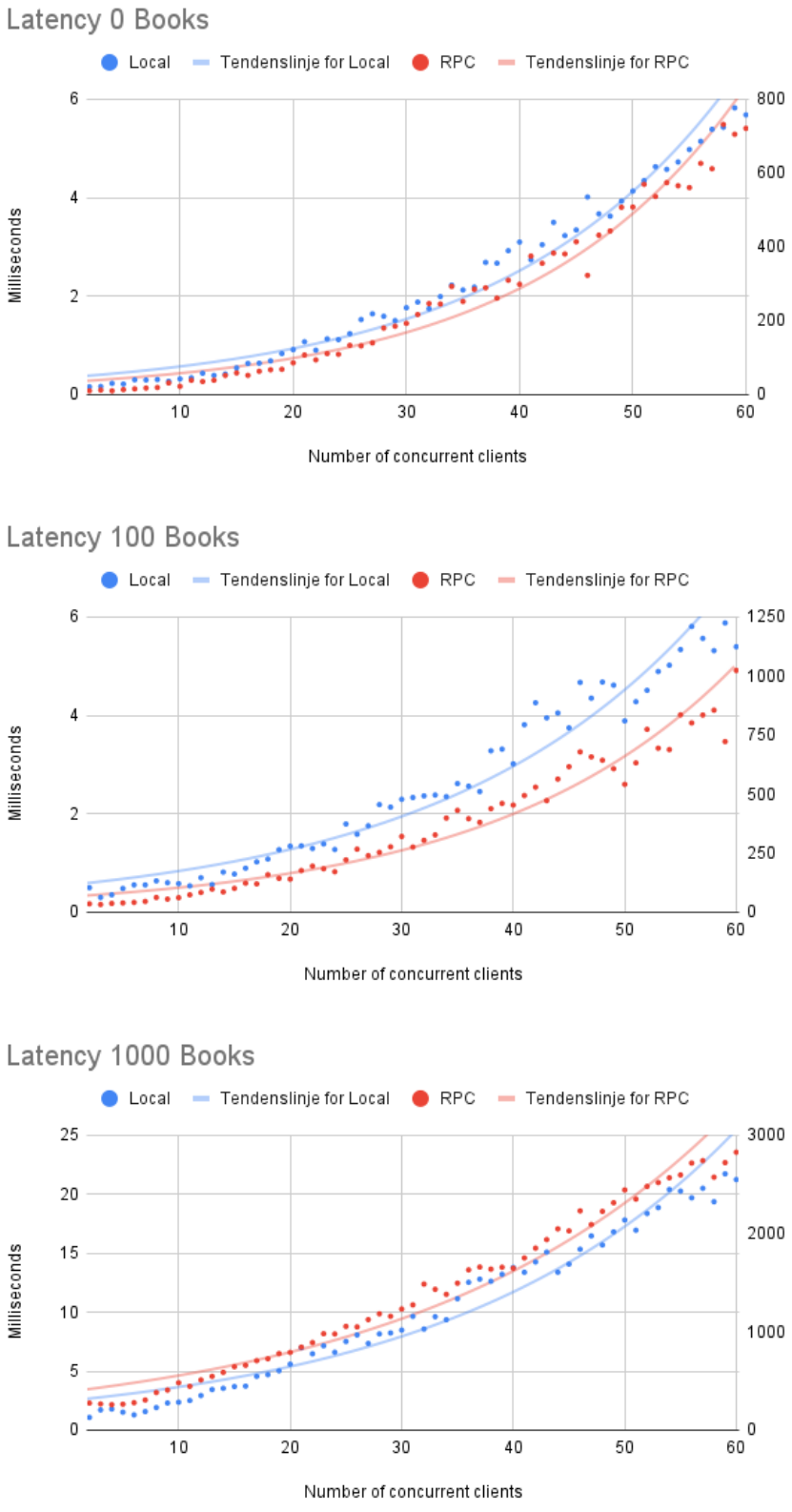

Figure 2: Graphs of the average latency vs. number of clients (threads) for experiments using a local and RPC bookstore, and the initial bookstore described in section 1 with either 0, 100, or 1000 books.

3 Discussion

3.1 Workloads

We are testing the system as it is designed to be used. We might also want to test how the system would behave when misused. We could flip the interaction ratios, such that rare interactions became frequent. If we want to test the robustness of our system, we should test not only the intended usage of our system but also corner cases and varying interaction frequencies. Some interactions are not implemented / not used, so the tests do not provide a completely realistic usage of the system, but an approximate, and appropriate usage.

3.2 Metrics

We are testing overhead, throughput, and latency. We are not properly testing scalability, as we are limited to our hardware. Testing scalability would be nice to have, as we would be able to better pinpoint bottlenecks. The database could be a bottleneck, as the server proxies could scale with the number of client proxies, and a scalability test could potentially show this.

4 Appendix

4.1 Parameter configuration used in the experiments

name	value	description
`numBooksToBuy`	5	The number of books to buy by the *Customer Interaction*
`numBookCopiesToBuy`	1	The number of copies of each book to buy by the *Customer Interaction*
`numEditorPicksToGet`	10	The number of editor picks to get by *Customer Interaction*
`numAddCopies`	10	The number of copies to replenish by the *Stock Replenishment Interaction*
`numBooksToAdd`	5	The number of new books to acquire by the *New Acquisition Stock Interaction*
`numBooksWithLeastCopies`	5	The number of books to replenish by the *Stock Replenishment Interaction*
`warmUpRuns`	100	The number of warm up runs for each workload
`numActualRuns`	500	The number of actual meassured runs for each work load
`percentRareStockManagerInteraction`	10%	The frequency of the *New Acquisition Stock Interaction*
`percentFrequentStockManagerInteraction`	30%	The frequency of *Stock Replenishment Interaction*

Table 1: Workload configuration

name	value	description
`numCopies`	10	The number of copies in a random generated book
`bookTitleLength`	32	The fixed length of the title of a random generated book
`authorNameLength`	32	The fixed length of the author name of a random generated book
`editorPickProbability`	10%	The probability of a random generated book being marked as an editor pick

Table 2: Book-set generator configuration