Assignment 3

Author
Wanjing Hu / fng685@alumni.ku.dk

December 12, 2024

1 Question 1

Data Generation Procedure: We initialized the bookstore with 120 books, with random 7-digit ISBN, title of "Book"+ISBN, author with "Author"+ISBN, price randomly between 10 and 100, and number of copy randomly between 350 and 400. Also, 30% of the books were marked as Editor Picks, randomly distributed. We construct the ISBN based on the valid ISBN definition, and use HashSet to make sure each book has a different ISBN. We setup the Editor Picks to run through the "runFrequentBookStoreInteraction" work load interaction. We introduce the randomness to the copy of books so that we could simulate a various k smallest books in stock.

In case there are books that are NOT_AVAILABLE, we remain the workload configuration of Copied of Books to Buy be 1 and number of books to buy be 5, so that the stock could remain positive with 10 worker threads consuming concurrently buying 5 books of 1 copies for each call on runFrequentBookStoreInteraction. Since there are 100 warm up calls and 500 actual calls, and the percentage of calling Custom Interaction is 60%, the initial copy should be at least 360 to guarantee there is no sales miss. Here our number of copy is random between 350 and 400.

Hardware Employed: All our experiments were conducted on a single machine equipped with an Apple M1 processor. The M1 is an ARM-based SoC with 8 cores, comprising 4 high-performance cores and 4 energy-efficient cores, clocked dynamically to optimize power usage and performance. The chip includes a shared 16MB L2 cache and an integrated 8-core GPU. The system is configured with 8 GB of unified memory (RAM), which is shared across the CPU and GPU, providing high bandwidth and low latency for efficient processing. The machine has 496 GB of SSD storage, ensuring fast read and write operations during the experiments. Experiments were executed on macOS, a 64-bit operating system optimized for the M1 architecture. The system environment ensures tight integration between hardware and software, leveraging macOS's capabilities for optimized performance in computational workloads. The JVM max heap size is 1GB.

Measurement procedures: We have 10 concurrent Workload, indicating number of concurrent Threads is 10. We also repeat to run CertainWorkload for 7 times. We use VM-option to control whether we are conducting a local test or a RPC test. For performance measurement we have **throughput** averaged in each worker and then averaged by the number of worker, because the workers are in different threads and the successful interactions

should be averaged separately. The number of successful interactions per second (goodput) was recorded for each configuration. Both read (e.g., fetching book details) and write (e.g., buying/rating books) operations were included. Results were aggregated across all client threads. We also have **latency** for performance, indicating the response time. Latency was measured as the time taken for a client request to complete, averaged over all interactions. We show both average values and deviation values of the **successful interaction counts**. They are useful for detecting anomalies or trends. Averages can vary due to random factors in workloads or system state, but they stabilize over multiple runs. Deviation may change slightly with workload variations but remains consistent if the workload and environment are stable. Their reliability depends on how closely the workloads mirror actual usage patterns, including peak load, bursty traffic, and a mix of read and write operations. To see whether the total throughput and the goodput are close enough, we have **Total Interactions** and **Total Successful Interactions** metrics too. To check that the customer interactions percentage is 60%, we also have Customer Interaction Percentage reported, which is calculated by $totalFrequentCustomerInteractions/totalInteractions$.

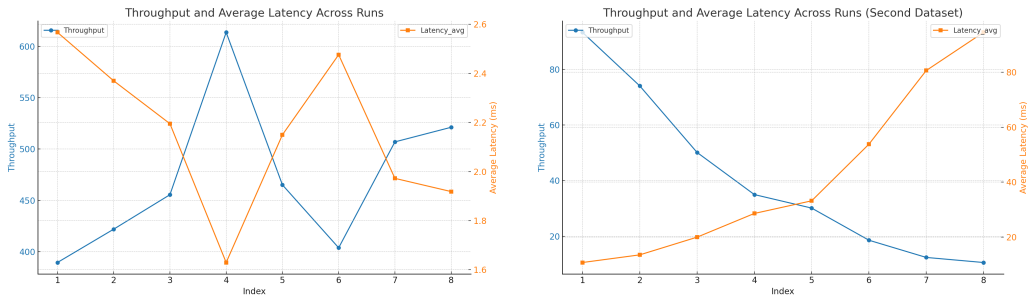
The goal was to evaluate throughput and latency under different configurations of client threads and operational setups Repetitions and Averaging: Each experiment was repeated 10 times for statistical reliability. The reported metrics are the mean values, and standard deviation was calculated to assess variability. Throughput Measurement:

Client Thread Configurations: Experiments were conducted with 1, 10, 20, 50, and 100 client threads to analyze scalability. Address Space Configurations: Experiments were repeated in two setups: Same address space: The client and server ran within the same JVM. Different address spaces: RPC-based communication between separate client and server JVMs. Data Validation: The fraction of unsuccessful interactions was monitored to ensure they remained below 1%. Customer interactions were validated to comprise approximately 60% of the workload.

2 Question 2

For executions in the same address in 1(a), the Throughput shows a generally increasing trend until a peak is reached, followed by slight stabilization. And the Latency shows a decreasing trend, inversely related to throughput. The increase may due the the local cache and the decrease may due to some fluctuation. Compared with the RPC version, the local test's Throughput is generally much higher and the latency is much lower. The local test benefits from reduced network latency and fewer external dependencies, allowing the system to handle higher interactions per second efficiently, and they have minimal communication overhead.

For executions across address in 1(b), the Throughput Shows a steady decline as latency increases and the Latency exhibits an increasing trend, worsening as the throughput decreases. The RPC test involves network calls and potential contention for resources, leading to higher latency and reduced throughput as the workload increases or external factors come into play. Also there may be some un-releasing resource, leading to the performance getting worse and worse. The increased communication overhead and potential network delays cause higher latency and reduced throughput.



(a) Executions in the same address space(LocalTest) (b) Executions across address space(RPC)

Figure 1: Schedules in Question1

For the RPC version, we modify the serializer to binary. Because with the origin configuration in BookStoreConstants, we found that we use XML as response serializer and it is too large that it would use up the memory.

```
public final class BookStoreConstants {
   public static final boolean BINARY_SERIALIZATION = false;
}
```

For the third time execution, we got this error, indicating we have allocated too much BufferingResponseListeners and they are not released properly after the request ends, and also they are not reusable.

```
Caused by: java.lang.IllegalArgumentException: Buffering capacity exceeded
```

3 Question 3

We use request latency, throughput, and error rates as the metrics. **Throughput** represents the system's capacity to handle requests or interactions per second successfully. It reflects overall system performance and efficiency. Latency may fluctuate due to network conditions, contention for resources, or background processes, especially in distributed or cloud systems. We calculate **Latency** for each worker thread individually and average them afterwards over the number of workers, so that we could keep the differences of different worker. **Additional metrics** such as system resource utilization (CPU, memory, and disk I/O), queue lengths for processing requests, and scalability under increased load would provide deeper insights. Furthermore, including user-centric metrics like perceived response time could demonstrate the bookstore's performance from an end-user perspective. These supplementary metrics would help ensure a comprehensive evaluation, making the results more actionable and representative of real-world operations.