ACS Programming Assignment 3

This assignment is due via Absalon on December 14, 23:59. While individual hand-ins will be accepted, we strongly recommend that this assignment be solved in groups of maximum two students. NOTE: The KU IDs of ALL group members MUST be stated on a separate `group.txt` file to ensure all group members get feedback and get the assignment accounted for in Absalon.

A well-formed solution to this assignment should include a PDF file with answers to all questions posed in the programming part of the assignment. In addition, you must submit your code along with your written solution. Evaluation of the assignment will take both into consideration. Note that all homework assignments have to be submitted via Absalon in electronic format. It is your responsibility to make sure in time that the upload of your files succeeds. While it is allowed to submit scanned PDF files of your homework assignments, we strongly suggest composing the assignment using a text editor or LaTeX and creating a PDF file for submission. Email or paper submissions will not be accepted.

Learning Goals

This assignment targets the following learning goals:

- Implement workload generation procedures in an experimental framework.
- Design and conduct an experiment to measure performance of a system.

Programming Task

Performance of a Certain Bookstore

The team of `acertainbookstore.com` has tasted success and their concurrent bookstore service has earned them a name in the industry. They want to publish performance measurements of their application to let everyone know how good it is. They have assigned you the task to design, implement, and conduct an experiment to achieve this goal.

For this assignment, you are provided with a code handout that includes and extends the one in Programming Assignment 1. The only difference is that there is an additional `com.acertainbookstore.client.workloads` package, which contains the client workloads which you need to implement. For this assignment, you have to measure the performance of the *original, non-concurrent* version of the bookstore (*CertainBookStore*

class). It is entirely optional if you want to additionally include the unimplemented methods of Programming Assignment 1 in your performance measures, i.e., `rateBooks`, `getTopRatedBooks`, and `getBooksInDemand`, or additionally include any of your implementation for Programming Assignment 2. This assignment only asks for measurements against the original implementation of the *CertainBookStore* class, which was handed out. In this way, this assignment remains independent from the previous assignments.

Your task in this assignment consists of two major parts:

- Extend the code appropriately to implement workload generation and reporting of results. The workload you will implement consists of three types of *interactions* that are executed against the service, each modeling a typical application use scenario. The workload generator executes a given mix of these interactions; however, your results will report exclusively the throughput and latency of the customer-facing interaction against the *BookStore* interface.[1] In addition, only *successful interactions* should contribute to throughput and latency measurements, where a successful interaction is defined as an interaction which did not raise an error condition through an exception.

- Design and execute an experiment that analyzes one of the factors that influence performance: the number of clients concurrently accessing the bookstore. The results of your experiments should consist of two plots with number of clients in the x-axis, and throughput and latency in the y-axis, respectively. You will need to control and document a number of other parameters that affect performance as part of your experimental design (e.g., hardware setup, details of the workload, fraction of successful interactions, number of books managed by the bookstore).

Workload

We model three main interactions against our bookstore: one customer interaction against the *BookStore* interface, and two stock management interactions against the *StockManager* interface. We describe these interactions as follows:

Customer Interaction, Frequent *BookStore* interaction, 60% of interactions in workload: In this interaction, we model customers who buy books from the bookstore. The interaction starts with the customer querying books which are marked as editor picks. After that, the customer selects a subset of the editor picks queried, and buys a number of copies of each of the selected books.

Stock Replenishment Interaction, Frequent *StockManager* interaction, 30% of interactions in workload: This interaction models a stock manager who queries for books with low stock, and adds copies for these books.

[1] This is similar in spirit to what is done in application-level benchmarks such as TPC-C (http://www.tpc.org/tpcc/).

New Stock Acquisition Interaction, Rare *StockManager* interaction, 10% of interactions in workload: In this interaction, the stock manager queries for all books in the bookstore, then adds books to the catalogue which are not already there.

NOTE: None of the interactions are atomic, even though they must call functions from the *BookStore* and *StockManager* interfaces which individually respect before-or-after atomicity.

Implementation

The `com.acertainbookstore.client.workloads` package contains a number of classes that you should extend to implement workload generation and use to obtain experimental results.

The workload experiment is executed by running the *CertainWorkload* class. This class spawns multiple worker threads (*Worker* class) which try to select one of the three different workload interactions while trying to preserve the distribution of the interactions configured as parameters. The workers also take a configuration object (*WorkloadConfiguration* class) which contains all the details of the parameters and *all* the necessary information to run the workload interactions.

The worker threads first run a number of warm-up runs, and then run actual measurement runs. After running measurements, each worker returns a result consisting of the total number of interactions run, the number of successful interactions run, the respective numbers of customer interactions, and the time taken by this worker.[2]

Implement BookSetGenerator and data initialization

The `com.acertainbookstore.client.workloads.BookSetGenerator` class is a helper class used to generate input parameters to function calls in the interactions of the workload. It is behaviorally similar to the "Random" class. The *BookSetGenerator* class consists of 2 methods:

- `sampleFromSetOfISBNs`: selects a given number n of unique ISBNs out of a given input set at random using a uniform distribution. This function is used in the customer interaction to select books to be bought.
- `nextSetOfStockBooks`: generates a set of ImmutableStockBooks of size n with random values. This function is used in the new stock acquisition interaction to generate candidate books for insertion.

In addition to the *BookSetGenerator* class, you should consider the `com.acertainbookstore.client.workloads.CertainWorkload` class. In this class,

[2]Since the workers do not run indefinitely and their readings are not taken over various time intervals, the readings are not steady state readings. This was intentionally chosen for this assignment for simplicity. You may need to change these parameters to get reliable measurements in your particular hardware setup.

implement the `initializeBookStoreData` method, which should initialize the bookstore with a given initial number of books.

NOTE: You will need to decide details of the procedure for data generation, e.g., how many books you will initially load the bookstore with, how long should book titles be, or how many copies of the books are given initially to the books added to the bookstore. Remember to document these decisions in your experimental setup description asked for below.

Implement the workload interactions

In order to measure the performance, you need to implement the following three methods `com.acertainbookstore.client.workloads.Worker` class, which represent the three different client interactions:

- `runRareStockManagerInteraction`: This method invokes `getBooks` and then gets a random set of books from an instance of *BookSetGenerator* by calling `nextSetOfStockBooks`. It then checks if the set of ISBNs is in the list of books fetched. Finally, it invokes `addBooks` with the set of books not found in the list returned by `getBooks`.
- `runFrequentStockManagerInteraction`: This method invokes `getBooks`, selects the k books with smallest quantities in stock, and then invokes `addCopies` on these books.
- `runFrequentBookStoreInteraction`: This method invokes `getEditorPicks`. It then selects a subset of the books returned by calling `sampleFromSetOfISBNs`, and buys the books selected by calling `buyBooks`.

For each of the above methods, if you need parameters, look at the defined parameters in the *WorkloadConfiguration* class. These parameters should be self-explanatory. You can add more parameters to this class during the implementation if you feel the need.

Implement reporting of metrics

In order to measure the performance, we use the metrics of *throughput* and *latency*. The throughput and latency are both measured on the client side. The throughput measured is the aggregate throughput of *successful customer interactions* for all workers.[3] The latency measured is the average latency of these interactions across all workers. To provide the aggregation and reporting of metrics, you should implement the method `reportMetric` of class `com.acertainbookstore.client.workloads.CertainWorkload`.

You should choose a configuration of the parameters (see the *WorkloadConfiguration* class) and then vary the number of client threads to measure the throughput and latency values for different number of client threads. We provide you with a set of default parameter values in the *WorkloadConfiguration* class, but note that these values are not

[3] $agg\ throughput = \sum_{i \in Workers} \frac{number\ of\ successful\ customer\ interactions_i}{total\ time\ taken\ in\ actual\ runs_i}$

authoritative. You may need to tune these values according to your setup (and document your reasons in the setup description asked for below).

You should also repeat the experiments in the *same address space* and *across different address spaces* (i.e., both using and not using RPC by setting the localTest in the *CertainWorkload* class). It is entirely optional if you also want to measure across address spaces on different machines.

NOTE: Above, we are concentrating on successful interactions only, which is sometimes referred to as "goodput". You should ensure in your measurements that the total throughput and the goodput are close enough, i.e., only a small fraction (say $< 1\%$) of the interactions are unsuccessful. Also, you should check that the customer interactions constitute roughly 60% of the total interactions processed. You may need to tune your data generation procedures to achieve this. Remember to document your decisions in your experimental setup description asked for below.

NOTE: *The code handout has a number of tuning parameters which are set to default values e.g., serialization type (binary/text),*[4] *thread-pool size in client proxy classes,*[5] *thread-pool size in bookstore server.*[6] Remember that the thread-pool in client proxy classes is a Jetty-specific artifact and is separate from the worker threads you implement for this assignment. You may wish to tune these parameters for your experiments if you encounter any bottlenecks and want to investigate them.

Questions for Discussion on the Performance Measurements

In addition to the implementation above, reflect about and provide answers to the following questions in your solution text:

1. Discuss in detail the setup you have created for your experiments. In particular, document your data generation procedures, hardware employed, measurement procedures (e.g., number of repetitions, statistics used such as average or deviation), and any other considerations you made. In the evaluation of this question, we will consider not only your thoroughness, but also whether you provide a brief justification/rationale for your decisions. [Hints: *For examples of describing setup for computer systems experiments*, see the first paragraph in Section 5.1 (Experimental Setup) in the paper describing Silo[7] and Section 4.4 (Experimental Setup) in Schad et. al.'s paper on measurements of runtime in public clouds.[8]]

2. Show and explain the plots for throughput and latency that you obtained. As described above, we expect two plots: one for throughput and one for latency. Each plot should include two curves: one for executions in the same address space, and one for executions across address spaces. Describe the trends observed and any

[4] `com.acertainbookstore.utils.BookStoreConstants.BINARY_SERIALIZATION`

[5] `com.acertainbookstore.client.BookStoreClientConstants.CLIENT_MAX_THREADSPOOL_THREADS`

[6] `com.acertainbookstore.server.BookStoreHTTPServer.{MIN_THREADPOOL_SIZE, MAX_THREADPOOL_SIZE}`.

[7] `http://people.csail.mit.edu/stephentu/papers/silo.pdf`

[8] `https://infosys.uni-saarland.de/publications/SDQ10.pdf`

other effects. Explain why you observe these trends and how much that matches your expectations. You must ensure that your graphs are readable (proper use of labels, scales, axes, legends to name a few). *Unreadable graphs will be rejected.* See these references[9,10] for graph-plotting guidelines.

3. How reliable are the metrics and the workloads for predicting the performance of the bookstore? Are the metrics well chosen? What additional metrics would you choose to demonstrate the performance of the bookstore?

[9] `https://www.matrix.edu.au/how-to-draw-a-scientific-graph-correctly/`
[10] `http://www3.nd.edu/~pkamat/pdf/graphs.pdf`