PCSD Assignment 3

This assignment is due via Absalon on December 15, 23:59. This assignment is worth 10 points total, as announced in the course description. While individual hand-ins will be accepted, we strongly recommend that this assignment be solved in groups of two students. Groups may at a maximum include three students.

A well-formed solution to this assignment should include a PDF file with answers to all exercises as well as 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:

- Discuss basic concepts in recovery for durability of a two-level memory under a restricted set of fail-stop failures.
- Explain in detail the functioning of the ARIES algorithm in a recovery scenario.
- Implement workload generation procedures in an experimental framework.
- Design and conduct an experiment to measure performance of a system.

Exercises

Question 1: Recovery Concepts

Regarding recovery techniques, answer the following questions:

1. In a system implementing force and no-steal, is it necessary to implement a scheme for redo? What about a scheme for undo? Explain why.

2. What is the difference between nonvolatile and stable storage? What types of failures are survived by each type of storage?
3. In a system that implements Write-Ahead Logging, which are the two situations in which the log tail must be forced to stable storage? Explain why log forces are necessary in these situations and argue why they are sufficient for durability.

Question 2: ARIES

Consider the recovery scenario described in the following, in which we use the ARIES recovery algorithm. At the beginning of time, there are no transactions active in the system and no dirty pages. A checkpoint is taken. After that, three transactions, T1, T2, and T3, enter the system and perform various operations. The detailed log follows:

```
LOG

LSN     LAST_LSN     TRAN_ID     TYPE         PAGE_ID
---     --------     -------     -----        --------
1       -            -           begin CKPT   -
2       -            -           end CKPT     -
3       NULL         T1          update       P2
4       3            T1          update       P1
5       NULL         T2          update       P5
6       NULL         T3          update       P3
7       6            T3          commit       -
8       5            T2          update       P5
9       8            T2          update       P3
10      6            T3          end          -
```

At this point in the execution, i.e., after LSN 10, the system crashes. Apply the ARIES recovery algorithm to the scenario above. Show:

1. the state of the transaction and dirty page tables after the analysis phase;
2. the sets of winner and loser transactions;
3. the values for the LSNs where the redo phase starts and where the undo phase ends;
4. the set of log records that may cause pages to be rewritten during the redo phase;
5. the set of log records undone during the undo phase;
6. the contents of the log after the recovery procedure completes.

For each of the items above, justify your answer.

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 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 Assignment 1 in your performance measures, i.e., ***`rateBooks`****,* ***`getTopRatedBooks`****, and* ***`getBooksInDemand`****, or additionally include any of your implementation for 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).

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

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 in demand, and adds copies for these books.

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 is 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]

[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

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. This function is used in the customer interaction to select books to be bought.
- `nextSetOfStockBooks`: generates a random set of ImmutableStockBooks of size n. This function is used in the new stock acquisition interaction to generate candidate books for insertion.

In addition to the *BookSetGenerator* class, you should implement the `initializeBookStoreData` method of the `com.acertainbookstore.client.workloads.CertainWorkload` class. This method 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 `getBooksInDemand` and then invokes `addCopies` for the list of books returned.
- `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.

measurements in your particular hardware setup.

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 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 *WorkloadConfiguration* constructor). 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.

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.

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

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

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 other effects. Explain why you observe these trends and how much that matches your expectations.

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?