ACS Assignment 4

This assignment is due via Absalon on December 18, 23:59. 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. 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 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:

- Perform simple system reliability calculations, while clearly stating underlying assumptions.
- Analyze the operation of the recovery method ARIES upon a DBMS crash failure.
- Implement a simple synchronous replication protocol while ensuring a level of consistency of operations observed by clients.
- Discuss multiple aspects of the implementation of replication protocols, including update and read processing, load balancing, scalability, and handling of failures.

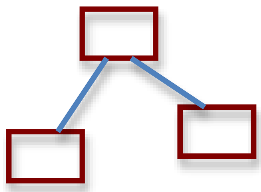
Exercises

Question 1: Reliability

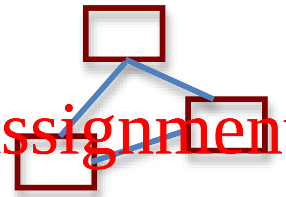
[Saltzer & Kaashoek] The town council wants to implement a municipal network to connect the local area networks in the library, the town hall, and the school. They want to minimize the chance that any building is completely disconnected from the others.

They are considering two network topologies:

1. Daisy Chain:



2. Fully connected:



Each link in the network has a failure probability of p. Now, answer the following questions:

1. What is the probability that the daisy chain network is connecting all the buildings?
2. What is the probability that the fully connected network is connecting all the buildings?
3. The town council has a limited budget, with which it can buy either a daisy chain network with two high reliability links (p = .000001), or a fully connected network with three low-reliability links (p = .0001). Which should they purchase?

Remember to explicitly state the assumptions you make for your reliability calculations.

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

Replication of a Certain Bookstore

The team of `acertainbookstore.com` have learnt about the bottleneck in the performance of their bookstore and would to improve the performance of the system by adding a number of server replicas and implementing a simple replication strategy. The system will consist of a single *master* bookstore server and multiple *slave* bookstore servers. The clients should be oblivious of this replication mechanism and this transparency on the client side is provided by a set of *proxies*, which route requests in a replication-aware way to an arbitrary replica as chosen by the server's load balancing policy. The architecture of the system is outlined in Figure 1.

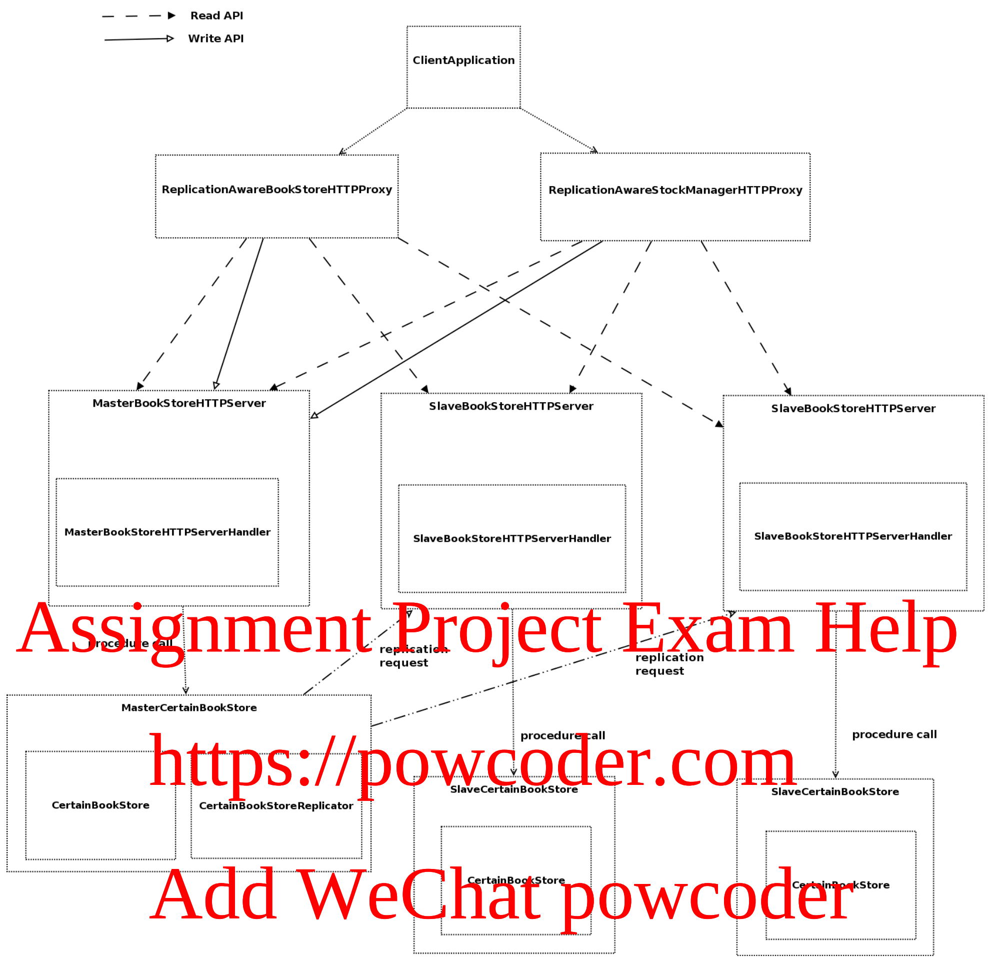


Figure 1: Architecture of replicated bookstore application

Basic Request Processing

In order to simplify the architecture, all *update requests* are sent to the single master bookstore server, which synchronously replicates all the updates to the slave bookstore servers. The slave bookstore servers service exclusively *read-only requests.* In case the master server receives read-only requests, it services them as well (in addition to update requests).

The proxies divide read-only request across all components according to a *load balancing policy.* A good policy ensures that only a small number of read-only requests are sent to the master when the system needs to process update requests, while the master participates more actively on servicing reads when there are less writes. In addition, a good policy ensures that slaves contribute equally to servicing reads.

One important issue is how to deal with failures during replication. In this assignment, you may assume a *fail-stop* model: Network partitions do not occur, and failures can reliably be detected. We model the situation of replicas hosted in a single datacenter, and a service configured so that network timeouts can be taken to imply that the replica being contacted indeed failed. In other words, the situations that the replica was just overloaded and could not respond in time, or that the replica was not reachable due to a network outage are just assumed not to occur in our scenario.

Proxies and Timestamps

Client session affinity to a proxy is assumed, so a client will use the same proxy for all its interactions with the bookstore in the same session. Even though the client employs the same proxy throughout the session, two reads directed by the proxy in sequence to different replicas may yield results that reflect the bookstore state after and then before the execution of an update request. In order to fix this problem and make sure that client reads are always monotonic,[1] the bookstores return a timestamp, the snapshot identifier, together with the result of every request. The snapshot identifier is logically a simple counter incremented every time the system processes an update request. A proxy only accepts a read result with snapshot identifier equal to or greater than the snapshot identifier received in the last non-erroneous result (either read or write). In other words, the proxy ensures that the timestamp of the result is equal to or later than its current timestamp. This guarantees that the proxy never yields results to clients that move back in time. In case a read result is rejected, the proxy attempts the read with different replicas until a result with an appropriate timestamp is returned (note that a read from the master will *always* return a result with the latest timestamp). The choice of replica for subsequent reads depends on the load-balancing policy in effect.

Managing the System Configuration

The proxies read a `proxy.properties` file to initialize the master and slave server mappings. The master server reads a `server.properties` file to initialize the slave server mappings. It is important to realize that these files might not contain the same mappings to allow flexible configurations. You must modify the filePath variable in the MasterCertainBookStore class and the proxy classes to point to the file.

For this assignment, you do not need to implement recovery from failures. If a slave fails, then the master which attempts to communicate with that slave will reliably detect the failure and flag the slave appropriately as failed in the configuration. No more operations will be then forwarded to that slave. Note that similar failure behavior can be implemented in the proxies (this has been left unimplemented and entirely optional for you to implement). However, note that the system must continue to operate and be available for reads and writes even in the case of slave failure. If the master fails, the system becomes unavailable for writes, but remains available for reads as long as there are slaves available. Finally, if all instances fail, the system becomes unavailable.

[1]The system never goes back in time to a client.

Summary of your Task

For this assignment, you are provided with a code handout that is similar in architecture to Assignment 1, but has been changed to integrate the master-slave replication mechanism into the architecture. Some parts of the implementation have been left for you to complete. *It is entirely optional if you want to additionally include the unimplemented methods of Assignment 1 in your solutions, i.e., `rateBooks`, `getTopRatedBooks`, and `getBooksInDemand`, or additionally include any of your implementation for Assignment 2. This assignment only asks for solutions 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 the following parts:

- Implement the `com.acertainbookstore.interfaces.Replicator` interface by filling out the `com.acertainbookstore.business.CertainBookStoreReplicator` and `com.acertainbookstore.business.CertainBookStoreReplicationTask` classes. The *Replicator* is present on the master bookstore server and is responsible for sending replication requests to the slave bookstore servers. Replication is achieved by re-execution of update requests at the slaves in the same order arbitrated by the master.[2] The replication on each slave server must occur *concurrently*. In other words, latency of replication should be hidden by overlapping replication to multiple slaves, and not executing replication requests serially. In addition, latency should be hidden by overlapping replication to slaves with processing of the update request at the master.

- Handle the replication requests on the slave bookstore servers by adding necessary code in `com.acertainbookstore.server.SlaveBookStoreHTTPMessageHandler` and `com.acertainbookstore.business.SlaveCertainBookStore` classes.

- Design and add a good load-balancing strategy for client requests in `com.acertainbookstore.client.ReplicationAwareBookStoreHTTPProxy` and `com.acertainbookstore.client.ReplicationAwareStockManagerHTTPProxy` classes. For example, one possible strategy is to simply pick replicas to process reads at random (including the master) and direct writes exclusively to the master. You should document and justify the strategy you design for load balancing.

- For the proxy classes mentioned above, note that the code handout already includes the logic to retry reads in case old timestamps are received. In your load-balancing strategy, you should design and document a mechanism by which the proxies try to answer client requests without waiting forever and failing if necessary.

- If you think you need extra variables or classes in order to implement the necessary changes, you are free to do so, as long as you document them. See the README file for instructions on launching servers.

[2]Note that we assume that update requests are deterministic.

- Recall that it is important that you explain your implementation and justify your decisions in your solution text (cf. questions for discussion below).

Implementation

Requirements on testing

In this assignment, we expect you to focus mostly on the implementation and discussion aspects. Nevertheless, you should test your code in at least the following ways:

- In principle, the replication architecture should be transparent to clients, which should experience the same semantics against the bookstore interfaces, as long as they interact with the same proxy. So you should be able to run standard tests against the bookstore interfaces while running your code with replication as an implementation.

- You should test failure conditions for both slaves and the master, and check that the system is able to mask these errors appropriately. In particular, failure of a slave should have no impact on functionality, while failure of the master should only make write operations unavailable.

In your solution text, it is acceptable to just state how you tested your implementation with the scenarios above. Of course, if you have time to perform any further testing, you can document that in your solution as well. However, *we will not consider extensive testing a focus point of this assignment, so as to limit the workload for providing a solution.*

Implement the Replicator on the master server

The `com.acertainbookstore.business.MasterCertainBookStore` class is a wrapper class over the *CertainBookStore* and runs on the master server. It contains a *Replicator* object which is used to propagate updates to the slave servers. You must implement the `com.acertainbookstore.interface.Replicator` interface. In order to do this, you must implement the following methods in the `com.acertainbookstore.business.CertainBookStoreReplicator` class:

- `replicate`: This method takes a *ReplicationRequest* that needs to be replicated to the set of active slave servers. This is an asynchronous method call so this method returns a list of future *ReplicationResult* objects. These futures can be used to obtain (and block on) the status of replication processing as is done in the MasterCertainBookStore. The replicate requests must be invoked concurrently on each slave server by the replicator.

The `com.acertainbookstore.business.CertainBookStoreReplicationTask` is used to model the concurrent replication task that is invoked by the *Replicator*. You must implement the following method in this class:

- `call`: This method must perform the replication request on a slave server. On completion it must return the result of processing, signaling either success or failure.
- Implement the constructor.

Implement the replication interface on the master server

The `com.acertainbookstore.server.ReplicationAwareServerHTTPProxy` is used to implement the client side rpc using HTTP of "Replication" interface. This method is invoked by the CertainBookStoreReplicator threads (in the call method of CertainBookStoreReplicationTask) in order to propagate replication requests to the slave. You must make the necessary implementations in the ReplicationAwareServerHTTPProxy class.

Implement the replication interface on the slave server

The `com.acertainbookstore.business.SlaveCertainBookStore` class is a wrapper class over the *CertainBookStore* and runs on the slave server. It must handle the replication requests sent to the slave server. You must make the necessary changes in the `com.acertainbookstore.server.SlaveBookStoreHTTPMessageHandler` class and `com.acertainbookstore.business.SlaveCertainBookStore` class in order to handle the replication requests, update the bookstore contents, and modify the snapshot identifier.

Implement the handling of replication on proxy servers

The `com.acertainbookstore.client.ReplicationAwareBookStoreHTTPProxy` and `com.acertainbookstore.client.ReplicationAwareStockManagerHTTPProxy` classes are used to provide the client side methods to the replicated bookstore. In order to implement load balancing on the proxy side, implement the following:

- `getReplicaAddress`: This method returns a server address from the list of slave server addresses and/or master address. A selection using randomness as a building block is an acceptable solution, as long as you discuss its advantages and limitations in your solution text.

Questions for Discussion on the Replication mechanism

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

1. Explain your implementation, making sure to document and justify your design choices and decisions. In particular, describe your strategy for load balancing, how you implemented latency hiding, and how you made use of the assumptions of the fail-stop model to handle failures of system components.

2. What are the advantages and disadvantages of the replication mechanism chosen? How do you think the replication solution would affect the performance of the system? Is there a bottleneck in the system now? Explain.

3. If a proxy fails and a client fails over to another proxy, then a client read may go back in time. What action(s) can the client take to ensure that this situation would not happen when it starts to interact with a different proxy? Explain.

4. Discuss what would happen to your implementation of the system if, in violation to the fail-stop failure model, a network partition would separate a subset of the replicas from the master, while still allowing those replicas to be contacted by the proxies.