CSE 411: Project 3

Replicated Lazy Linked-List

Due date: Friday December 3 (Groups of max 2)

In this project we will implement a replicated version of the lazy linked list we used before. The lazy linked list itself will not change (you can use the original lazy linked list code from synchrobench). This time, we will build on top of that a "simplified" state machine replication system, as follows:

- Requests to insert/remove/lookup a key will be sent (via sockets) by standalone **client** applications. A special 'terminate' request is used to shut down the server.
- Two **server** applications will be responsible for receiving and handling those requests. The two servers are **not identical**. One of them is a **primary** server, and the other is a **backup** server.
- Each client will send its requests to only one server.
    - All update/terminate requests are sent to the primary server (if a client sends an update request to the backup server, it will return an error message). This way, updates are ordered by definition and we don't need to implement an ordering protocol.
    - Lookup requests are sent to either one of them (in reality, this could be the one closer to the client geographically).

The details of how each server works are as follows:

Primary Server

- The primary server is a "single-threaded" application that listen to clients' requests in an infinite loop, and handles each of them as follows:
    - Terminate request: propagate request to the backup server and then exits.
    - Update (insert/remove) request: add the request to a "log file" (for crash recovery - as discussed later), send the request to the backup server, execute the operation on the linked list, and then respond to the client with the return value.
    - Read (contains) operation: execute operation and then respond to the client with the return value.

Backup Server

- The backup server is a "multi-threaded" application with two threads. The first is listening to the update/terminate requests propagated from the primary server, and the second is listening to lookup requests from the clients. In both cases, the backup server just executes requests without logging them in a file.

Additionally, we will implement a simple recovery mechanism as follows:

- If the primary server crashes and restarts, the first thing it executes (before it resumes listening to client requests) is a recovery method that reads the entries in the "log file", sends a special "do_recover" request to the backup server (with the log file attached), and rebuilds the list. The backup server will similarly rebuild the list before receiving any new requests.

- If the backup server crashes and restarts, it will send to the primary server a “request_recover” message, asking for the log file (you should think of a neat way for the primary server to listen for this special request without interfering with its normal execution).

  This assignment is intentionally under-specified. All details like format of request/response messages, format of log file entries, …, etc is part of your responsibilities and will be checked while grading. You should ask questions on Piazza to get more information about exactly what I expect of your program.

Bonus:
The proposed protocol is complete and efficient (we will discuss why in lectures) However, it is too simple and can be improved in so many ways. Any optimization can be considered for Bonus credits (up to 20% of project grade). Examples are:

- Support of “per-server” concurrent reads
  - To do so, you should think of each server as a multi-threaded application where n worker threads are handling lookup requests, and one thread handles update requests.
- Allow both servers to handle update requests
  - This is less trivial because writes must be executed in the same order in both replicas and this order must be the same as what is written in the log file.
- Implement a better recovery mechanism
  - You may think of ideas like “checkpointing” and “compacting the log”.
- Allowing a backup server to take over if the main server permanently crashes
  - Again, not trivial. What if we have more than one backup server and all of them try to take over at the same time. What if the main server was seemingly down but eventually it recovers from the crash and resumes receiving client requests.
- Enabling a linearizable lookup operations
  - Again, not trivial. To do so, the backup server has to make sure that it does not miss stale updates from the server.