Distributed Systems Course Project: Two-Phase Commit

A simple transaction system

Summary

For this assignment, you will implement two-phase commit over a replicated, durable key-value store, using RPC (or RMI) to coordinate the replica processes. There are three major pieces you will need to implement. First, you will build a durable key/value store. Second, you will implement a two-phase commit protocol to coordinate writing values atomically to replicas of the key/value store. Third, you will expose an RPC (or RMI) interface to the two-phase commit coordinator for clients to communicate with.

Part A: Key/Value Store

In this part of the assignment, your job is to implement a durable key/value store library. Your library should expose three methods:

- put(key, value): stores the value "value" with the key "key".
- del(key): deletes any record associated with the key "key".
- value = get(key): retrieves and returns the value associated with the key "key".

The major requirement for your store is that it is durable; if your process exits (or is killed) and restarts, any values that were previously stored using a "put" must still be retrievable using a "get", and similarly, any value removed using a "del" must stay removed. It's up to you to decide on appropriate types for key and value, and to decide whether get/put/del can return failure in some cases.

I strongly recommend that you find existing code for this; you might, for example, consider using the sqlite database library. If you decide to roll your own library, this will take a lot of effort, particularly to get post-failure recovery correct.

Part B: Two-Phase Commit

Part B contains the bulk of this assignment. Your job is to implement a replicated key/value store, building on Part A. The architecture of your replicated key-value store is fairly simple; it consists of a single "master" process and multiple "replica" processes:

- **master:** the "master" process should expose an RPC interface to clients that contains three methods: get, put, and del. When the master process receives a state-changing operation (del or put), it uses two-phase commit to commit that state-changing operation to all replicas. When the master receives a "get" operation, it selects a replica at random to issue the request against.

  The master should be concurrent; it should permit multiple clients to connect to it and it should be able to process multiple operations concurrently.

- **replica:** each replica should wrap an instance of the key/value store you implemented in part a, and it should expose an RPC interface to the master to participate in two-phase commit. It's up to you to decide what specific RPC methods the replica should support.

  The replica should be concurrent: it should be able to process multiple commands concurrently. Thus, it needs to implement some form of concurrency control. For this assignment, the concurrency control scheme is very simple: if two concurrent operations manipulate different keys, they can safely proceed concurrently. If two concurrent operations manipulate the same key (e.g., two "put" operations to the same key show up at the same time), the first to arrive should be able to proceed while the second's two-phase commit should abort.

To implement two-phase commit, both the master and the replicas will need to do some form of logging to keep durable state associated with the two-phase commit. You will need to figure out how to implement logging, what to log, and how to replay your log on recovery.

Your implementation needs to be fault-tolerant. If either the master or one or more replicas fail, your two-phase commit implementation needs to do the right thing. Similarly, after a failed component recovers, your two-phase commit implementation needs to recover and proceed as appropriate.

It's fine for you to hard-code knowledge in the master and replicas about the configuration of the system, i.e., which processes exist and where they are running.

Part C: Clients

In part C, you should implement a client program that connects to the master and issues a stream of get, put, and del requests. You should be able to launch multiple clients in order to drive the master concurrently -- i.e., each separate client should be issuing its own stream of get/put/del operations.

Use your client programs to test your implementation of two-phase commit. Be sure to find a way to test as many corner cases as you can, such as the master failing during the period of uncertainty.

What to submit

You should submit the following:

- a report that contains the following:
    - describe the structure of your code, including any major interfaces that you implemented
    - describe and justify the RPC interface your replicas expose to the master.
    - how do you detect failures? if failures do occur, how are those reflected to clients via the RPC interface that the master exposes, if at all?
    - what interesting test cases did you explore, and why did you pick those?
- A tar ball, including your report, source code, a README file with intructions on how to compile and run your code, and the test programs.
- Put all above files into a directory, compress and email to the instructor.