6.824 - Spring 2015

6.824 Lab 5: Persistence

Due: May 8 11:59pm

Introduction

In this lab you'll add persistence to your key/value server. The overall goal is to be able to recover after the crash and restart of one or more key/value servers. It's this capability that makes fault-tolerance really valuable! The specific properties you'll need to ensure are:

- If a key/value server crashes (halts cleanly with disk intact), and is re-started, it should re-join its replica group. The effect on availability of one or more such crashed key/value servers should be no worse than if the same servers had been temporarily disconnected from the network rather than crashing. This ability to re-start requires that each replica save its key/value database, Paxos state, and any other needed state to disk where it can find it after the re-start.
- If a key/value server crashes (halts cleanly) and loses its disk contents, and is re-started, it should acquire a key/value database and other needed state from the other replicas and re-join its replica group. If a majority of a replica group simultaneously loses disk contents, the group cannot ever continue. If a minority simultaneously lose their disk content, and re-start, the group must recover so that it can tolerate future crashes.

You do not need to design a high-performance format for the on-disk data. It is sufficient for a server to store each key/value pair in a separate file, and to use a few more files to store its other state.

You do not need to add persistence to the shardmaster. The tester uses your existing `shardmaster` package.

This lab requires more thought than you might think.

You may find Paxos Made Live useful, particularly Section 5.1. Harp may also be worth reviewing, since it pays special attention to recovery from various crash and disk-loss failures.

You should either do Lab 5, or a project, but not both.

Collaboration Policy

You must write all the code you hand in for 6.824, except for code that we give you as part of the assignment. You are not allowed to look at anyone else's solution, and you are not allowed to look at code from previous years. You may discuss the assignments with other students, but you may not look at or copy each others' code. Please do not publish your code or make it available to future 6.824 students -- for example, please do not make your code visible on github.

Software

Do a `git pull` to get the latest lab software. We supply you with new skeleton code and new tests in `src/diskv`.

```
$ add 6.824
$ cd ~/6.824
$ git pull
...
$ cd src/diskv
$
```

Getting Started

First merge a copy of your Lab 4 code into `diskv/server.go`, `common.go`, and `client.go`. Be careful when merging StartServer(), since it's a bit different from Lab 4. And don't copy `test_test.go`; Lab 5 has a new set of tests.

There are a few differences between the Lab 4 and Lab 5 frameworks. First, StartServer() takes an extra `dir` argument that tells a key/value server the directory in which it should store its state (key/value pairs, Paxos state, etc.). A server should only use files under that directory; it should not use any other files. The tests give a server the same directory name each time the tests re-start a given server. StartServer() can tell if it has been re-started (as opposed to started for the first time) by looking at its `restart` argument. The tests give each server a different directory.

The second big framework difference is that the Lab 5 tests run each key/value server as a separate UNIX process, rather than as a set of threads in a single process. `main/diskvd.go` is the `main()` routine for the key/value server process. The tester runs `diskvd.go` as a separate program, and `diskvd.go` calls StartServer().

After merging your Lab 4 code into `diskv`, you should be able to pass the tests that print (lab4). These are copies of Lab 4 tests.

Hints

If a server crashes, loses its disk, and re-starts, a potential problem is that it could participate in Paxos instances that it had participated in before crashing. Since the server has lost its Paxos state, it won't participate correctly in these instances. So you must find a way to ensure that servers that re-join after disk loss only participate in new instances.

`diskv/server.go` includes some functions that may be helpful to you when reading and writing files containing key/value data.

You may want to use Go's gob package to format and parse saved state. Here's an example. As with RPC, if you want to encode structs with gob, you must capitalize the field names.

The Lab 5 tester will kill key/value servers so that they stop executing at a random place, which was not the case in previous labs. One consequence is that, if your server is writing a file, the tester might kill it midway through writing the file (much as a real crash might occur while writing a file). A good way to cause replacement of a whole file to nevertheless be atomic is to write to a temporary file in the same directory, and then call `os.Rename(tempname,realname)`.

You'll probably have to modify your Paxos implementation, at least so that it's possible to save and restore a key/value server's Paxos state.

Don't run multiple instances of the Lab 5 tests at the same time on the same machine. They will remove each others' files.

Handin procedure

Submit your code via the class's submission website, located here:

https://6824.scripts.mit.edu:444/submit/handin.py/

You may use your MIT Certificate or request an API key via email to log in for the first time. Your API key (XXX) is displayed once you logged in, which can be used to upload lab4 from the console as follows.

```
$ cd ~/6.824
$ echo XXX > api.key
$ make lab5
```

You can check the submission website to check if your submission is successful.

You will receive full credit if your software passes the `test_test.go` tests when we run your software on our machines. We will use the timestamp of your **last** submission for the purpose of calculating late days.

Please post questions on Piazza.