CSE 490h – Project 2: Cache Coherence Writeup

Wayne Gerard - wayger
Zachary Stein - steinz

January 27, 2011

Description:

In this project we implemented write-back cache coherence with single copy semantics for use by a small number of clients sharing a single address space coordinated by a single manager. The manger keeps track of the permissions for any given file in the system and assumes that the address space is empty at initialization time. Since the implementations of Transactions and PAXOS in future projects will provide fault tolerance and recovery, we assumed no errors for this project.

Implementation:

We used the standard Ivy cache coherence protocol, creating separate message types for each command (e.g. one for WQ, one for RQ). The manager is implemented on top of a regular client so that it can use the functionality implemented in the first project. Create and delete are implemented via RPCs.

Clients:

When a client calls a high-level command, that client checks its cache to see if it has permissions on that file. If it lacks the permissions it needs to perform the desired operation, it asks the manager for read and possibly write permissions, and queues the action it was trying to take. After receiving the appropriate permission from the manager the action is completed. If an error occurs (e.g. the file requested does not exist) the operation is thrown out.

Managers:

The manager keeps track of client cache statuses using a couple data structures. We could have had the manager poll clients on every request to determine their status and determine who needs what request sent out, but this method is much faster and reduces the number of packets sent. Unless some client has ownership of a file, it is assumed that the manager has the latest revision of any file.

Outstanding issues:

Our deletion and creation mechanism(s) are not as smooth as they could be.

Because of problems differentiating blank files from created and deleted files, we implement create and delete using RPCs. However, this change was made Friday afternoon, and although it will fix the issues in the long run, it has not been fully debugged. The client and manager locking data and cache data structures are not yet updating correctly. Fortunately, we are confident that we will fix this soon using the test code built with the intention of generating Synoptic graphs.

Additional steps:

One thing that is necessary before each run is to clean the local storage system for each node. Without cleaning the local storage system, the user will likely encounter errors related to file existence. For convenience, we have written a script "clean.sh" that should be run before using CacheCoherenceTester. We do not consider this to be an error, since it is assumed that there are no faults, the client and server should have no shared state between runs.

While the user doesnt need to explicitly make WQ, RQ, etc. requests in order to use our cache coherent client, there is an additional setup step that the user needs to take before using our implementation. In addition to our previous fault tolerance mechanism of handshaking, it is also necessary to explicitly define who the server/manager is for each node. This is done by calling up the command $< NODE >$ manageris $< SERVER >$. So, for example, for three nodes, each script should start with the following set of commands:

```
START 0
START 1
START 2
time
0 manager // establishing that 0 is the manager
1 manageris 0 // inform all clients
2 manageris 0
0 noop 1 // handshaking
2 noop 1
time
time
time
time // allow 3-4 rounds for handshaking to take place to avoid dropping commands
```

Alternatively, if no more than 4 nodes are desired, the PerfectInitializedClient can be used to bypass handshaking and manager establishment. Node 0 will always be the manager in this case. Note that you still have to start the nodes you would like to use.

State Machines:

See writeups/project2/490.png for a few basic diagrams of IVY CC.
See output/*/myproj.dot.png for Synoptic graphs of a single client running CC.
The version with multiple clients running CC is yet to be completed, due to time constraints.

Synoptic:

We were successful in generating Synoptic traces of our Protocol running partitioned by node. Unfortunately, these graphs are tall and unwieldy since they maintain the arbitrary high-level operation order imposed by the specific script run as invariants in the final graph.

We are still in the process of generating the ideal Synoptic graph of our protocol partitioned by high-level operation. The CacheCoherenceTester has been written to this end, but hasn't been fully tested. It works by performing many random high-level operations (create, delete, get, put, append)

at random on random files (the random number generator is currently seeded with a constant value for debugging purposes). It currently works if no attempts are made to perform operations on files that don't exist, create files that do exist, and do other non-sensible things with a single client (and a single manager). It uses callbacks and user defined Synoptic events to partition the Synoptic log into the desired traces. It is yet to run successfully with multiple clients.

We also considered generating several random scripts, each containing a single high-level operation to be run, and partitioning by log-file generated. Unfortunately, this technique would fail to capture several important characteristics of our protocol since cache data structures would be cleared between each run and we currently have no way to recover them between rounds. The easiest fix for this would be for the server to assume it has the newest version of every file it knows about upon initialization and for the clients to invalidate all of the local files in their cache. However, the CacheCoherenceTester is likely to be more extensible and robust, so we will concentrate our time trying to get it working instead.

You can run what we have so far by running ./compile.sh, ./clean.sh to clear out storage, ./ccTest1Client.sh to generate the synoptic log, and ./synoptic.sh -i -f -c synoptic_args/490h.args total.log to generate the synoptic graph output/myprot.dot.png. Running ./ccTest2Client.sh will run the multi-client version, but this is currently broken.