CSE 490h – Project 1: Client-Server File Store Writeup

Wayne Gerard - wayger
Zachary Stein - steinz

January 24, 2011

Description:

For our implementation of single copy semantics for a small number of clients communicating with a server, we decided to have the server act as a manager, keeping track of who has what access to any given file, assuming that all clients are invalid for all files at initialization.

Implementation:

We used the standard Ivy cache coherence protocols, and created separate protocols for each cache coherency command (e.g. one for WQ, one for RQ). To both follow the specification, and to make things easier, we decided to separate functionality into clients and servers.

Clients:

When a client calls a read/write command, that client checks its current permissions on that file. If it lacks the appropriate permissions, then it asks the manager for read and/or write permissions, and queues the action it was trying to take. This action is then taken after receiving the appropriate permission from the manager, or not at all in the case of an error.

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.

Outstanding issues:

Our deletion mechanism is not as smooth as it could be.

First and foremost, we allow the client to delete files it has RW access to without informing the manager immediately. If another node requests access to this file, then the manager requests the client to send the data back, but the client does this by sending a blank file back to the manager, which the manager assumes is an implicit delete.

In the event that node 2 deletes file x, and then node 0 requests file x, node 2 will send a special command to the server indicating that it was deleted. The server will then return a FILE DOES NOT EXIST error to node 0, which node 0 should take as an indication that file x does not exist anymore. However, currently node 0 will keep whatever cached copy it has. It will always receive an error that the file does not exist if it asks the server for read and/or write permission, but will still have a cached copy of file x.

Additional steps:

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 manageris 1 // establishing the manager
1 manager
2 manageris 1
0 noop 1 // handshaking
2 noop 1
time
time
time
time (you should allow 3-4 rounds for handshaking to take place)
```