Project 2: Part 6: Caching Extents

Due: 11:59PM Thursday, March 24, 2011

1 Introduction

In this lab you will build a server and client that cache extents at the client, reducing the load on the server and improving client performance. The main challenge is to ensure consistency of extents cached at different clients. To achieve consistency we will use the caching lock service from Part 5.

First you'll add a local write-back extent cache to each extent client. The extent client will serve all extent operations from this cache; the extent client will contact the extent server only to fetch an extent that is not present on the client. Then you'll make the caches at different clients consistent by forcing write-back of the dirty cached extent associated with a filehandle (and deletion of the clean extent) when you release the lock on that filehandle.

Your client will be a success if it manages to operate out of its local extent and lock cache when reading/writing files and directories that other hosts aren't looking at, but maintains correctness when the same files and directories are concurrently read and updated on multiple hosts.

2 Getting started

There are no additional files or changes for this part.

2.1 Testing Performance

Our measure of performance is the number of put and get RPCs that your extent server receives. You can tell the extent server to print out a line every 25 RPCs telling you the current totals as you did for the lock server in Part 5, by setting RPC_COUNT to 25.

Then you can start the servers, run the test-lab-4-c script, and look in extent_server.log to see how many RPCs have been received.

```
% export RPC_COUNT=25
% ./start.sh
% ./test-lab-4-c ./yfs1 ./yfs2
Create/delete in separate directories: tests completed OK
% grep "RPC STATS" extent_server.log
...
```

```
RPC STATS: 6001:801 6002:1402 6003:797
% ./stop.sh
```

The RPC STATS line indicates the number of put, get and getattr RPCs received by the extent server. The above line is the output of our solution for Part 5. Your goal is to reduce those numbers to about a dozen puts and at most a few hundred gets.

3 Step One: Extent Cache

In Step One you'll add caching to your extent client, without cache consistency. This cache will make your server fast but incorrect. (You can simply modify extent_client.cc and extent_client.h, or if you'd like, you can add the code to a sub-class in a separate file. Remember to svn add any new files you create.)

get() should check if the extent is cached, and if so return the cached copy. Otherwise get() should fetch the extent from the extent server, put it in the local cache, and then return it to the YFS client. put() should just replace the cached copy, and not send it to the extent server. You'll find it helpful for the next section if you keep track of which cached extents have been modified by put() (i.e., are "dirty"). remove() should delete the extent from the local cache.

When you're done, set RPC_COUNT and run test-lab-4-c giving the same directory twice, and watch the statistics printed by the extent server. You should see zero puts and somewhere between zero and a few hundred gets (or perhaps no numbers at all, if the value of RPC_COUNT is more than the number of gets). Your server should pass test-lab-4-a.pl and test-lab-4-b if you give it the same directory twice, but it will probably fail test-lab-4-b with two different directories because it has no cache consistency.

4 Step Two: Lock Client and Server, and Testing with RPC_LOSSY=0

In Step Two you'll ensure that each get() sees the latest put(), even when the get() and put() are from different YFS clients. You'll arrange this by ensuring that your extent client writes a file's modified (dirty) cached extents back to the extent server before the client releases the lock on that file. Similarly, your server should delete extents from its cache when it releases the lock on the relevant file.

You will need to add a method to the extent client to eliminate an extent from the cache. This flush() method should first check whether the extent is dirty in the cache, in which case it sends it to the extent server. Extents that your client has removed (with the extent client's remove() method) should also be removed from the extent server (if the extent server knows about them).

Your client will need to call flush() just before releasing a lock back to the lock server. You could just add flush() calls to yfs_client.cc before each release(). However, now that your lock client handles the caching of locks, flushing the extents after each release is overkill; what you really want is to flush the extents only once the client is forced to give the lock back to the lock server.

We provide an interface for this in the form of the lock_release_user class, defined in lock_client_cache.h. This is a virtual class supporting only one method: dorelease(std::string lockname). Your job is to

subclass lock_release_user and implement that subclass's dorelease method to call flush() on your extent client for whatever data is about to lose its lock. Then, create an instance of this class and pass it into the lock_client_cache object constructed in yfs_client.cc. Finally, your lock_client_cache must call the dorelease() method of its lu object before it releases a lock back to the lock server. (Note that lu was defined and initialized in the code we provided you for Part 5.) Overall, this will ensure that any dirty extents are flushed back to the cache before the lock is released, so that when the next client gets the lock and fetches the extent, it will see consistent data.

You should also keep extent meta-data cached along with the extents, and flush dirty meta-data back to the extent server along with the extents. If an extent is cached, then any calls that set attributes should change the meta-data in the cache, and need not propagate to the extent server until flush() is called.

When you're done with Step Two your server should pass all the correctness tests (test-lab-4-a.pl, test-lab-4-b, and test-lab-4-c should execute correctly with two separate YFS directories), and you should see a dramatic drop in the number of puts and gets received by the extent server.

5 Evaluation Criteria

We will test that your code passes the tests from part 4, as mentioned above, and we will check that there is a dramatic drop in the number of puts and gets received by the extent server.

6 Handin

Please submit all the files necessary for running Part 6, including the Makefile to:

`/afs/andrew/course/15/440-sp11/handin/proj2/your_andrew_id/part6/`

Please follow the same guidelines outlined in Part 1 for multiple submissions.

7 C++ Tutorials and Resources

- C++ Tutorial
  `http://www.cplusplus.com/doc/tutorial/`
- C++ Reference
  `http://www.cppreference.com/wiki/start`