6.824 - Spring 2012

6.824 Lab 5: Caching Extents

Due: Friday, March 30th, 5:00pm

Introduction

In this lab you will modify YFS to cache extents, reducing the load on the extent server and improving YFS performance. The main challenge is to ensure consistency of extents cached at different `yfs_client`s: to make sure that each `yfs_client` sees extent data that reflects the latest modifications by other clients. To achieve consistency we will use the caching lock service from Lab 4.

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 (or the attribute of the 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 by multiple clients.

Getting Started

Begin by initializing your Lab 5 branch with your implementation from Lab 4. There is no new code for this lab, except for an update to the makefile to build this lab.

```
% cd ~/lab
% git commit -am 'my solution to lab4'

Created commit ...
% git pull
remote: Generating pack...
...
% git checkout -b lab5 origin/lab5
Branch lab5 set up to track remote branch refs/remotes/origin/lab5.
Switched to a new branch "lab5"
% git merge lab4
```

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 Lab 4, by setting `RPC_COUNT` to 25.

Then you can start the servers, run the `test-lab-3-c` tester, and look in `extent_server.log` to see how many RPCs have been received.

```
% export RPC_COUNT=25
% ./start.sh
% ./test-lab-3-c ./yfs1 ./yfs2
Create/delete in separate directories: tests completed OK
% grep "RPC STATS" extent_server.log
...
RPC STATS: 1:2 6001:804 6002:2218 6003:1878 6004:198
% ./stop.sh
```

The `RPC STATS` line indicates the number of `bind`, `put`, `get`, `getattr`, and `remove` RPCs received by the extent server. The above line is the output of our solution for Lab 4. Your goal is to reduce those numbers to about a dozen `put`s and at most a few hundred `get`s.

Step One: Extent Cache

In Step One you'll add caching to your extent client, without cache consistency. This cache will make your `yfs_client` fast but inconsistent.

Your new `extent_client::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. `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. `extent_client` should maintain cached attributes for each extent so that `extent_client::getattr()` returns reasonable values.

When you're done, set `RPC_COUNT` and run `test-lab-3-c` giving the same directory twice, and watch the RPC STATS printed by the extent server. You should see zero `put`s and somewhere between zero and a few hundred `get`s (or perhaps no numbers at all, if the value of `RPC_COUNT` is more than the number of `get`s). Your server should pass `test-lab-3-a.pl` and `test-lab-3-b` if you give it the same directory twice, but it will probably fail `test-lab-3-b` with two different directories because it has no cache consistency.

You can 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 `git add` any new files you create.

Step Two: Cache Consistency

In Step Two you'll ensure that each `get()` sees the latest `put()`, even when the `get()` and `put()` are from different YFS clients. The extent client and lock client will cooperate to do this, helped by the `yfs_client` using the same ID for a file/directory i-number, lock, and extent. When your lock client releases a lock back to the lock server, you should ensure that the extent client first flush the extent to the server, and then remove it from local cache. A flush requires putting the corresponding extent (if dirty) to the extent server, or removing the extent (if locally removed) from the extent server.

Suppose client A acquires the lock on an i-number, get()s the file's data from the extent server and modifies it in its extent client cache, and keeps the lock and extent cached. Client B then asks for the lock, so that the lock server sends client A a revoke message. Client A will send the dirty data back to the extent server before releasing the lock. Then B will acquire the lock and get() the i-number's extent. Since B cannot have the extent in its cache (if it ever cached it, it must have deleted it when it gave up the lock), and A wrote the modified extent to the extent server before releasing the lock, B will see A's modifications.

You will now be using locks for two different purposes: first, to ensure that each file system operation is atomic; second, to drive the extent cache consistency scheme. This may mean that you have to add acquires and releases to `yfs_client` that were not necessary to pass the tests for the previous labs. Ensure that **every** `yfs_client` call to get(eid), put (eid), getattr(eid), and remove(eid) is wrapped in acquire(eid)/release(eid).

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 should send it to the extent server. If the extent has been removed (with the extent client's `remove()` method), `flush()` should send a remove RPC to the server. `extent_client` should cache attributes fetched from the server, and invalidate the relevant attributes in `flush()`. You need not explicitly send dirty attributes to the extent server; instead, it is enough to let the server update the attributes as a side-effect of `flush()` sending a put RPC for a dirty extent.

Your YFS server should 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 an extent in `lock_client::release()` is overkill; you only have to flush an extent when the lock server revokes the corresponding lock.

We provide you with a bit of glue to connect the lock client to the extent client 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(lock_protocol::lockid_t)`. 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 Lab 4.) This will ensure that `lock_client::flush()` is called before the lock is released.

When you're done with Step Two your server should pass all the correctness tests (`test-lab-3-a.pl`, `test-lab-3-b`, and `test-lab-3-c` should execute correctly with two separate YFS directories), and you should see a dramatic drop in the number of `put`s and `get`s received by the extent server.

Hints

- Make sure you use a pthreads mutex to protect the extent cache, in case multiple threads access it at once.
- Make sure that, if you only read an extent (or its attributes), you don't flush it back on a release. You should keep a flag in your cache that records if an extent (or its attributes) have been modified.
- You may be able to implement the caching in the extent client without modifying yfs_client.cc, except for instantiating (if you implemented your caching extent client as a separate class) and passing the object that implements the `lock_release_user` interface in `yfs_client::yfs_client`. However, as noted above, you may have to add acquire and release calls to `yfs_client`.

Collaboration policy

You must write all the code you hand in for the programming assignments, 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're not allowed to look at solutions from previous years). You may discuss the assignments with other students, but you may not look at or copy each others' code.

Handin procedure

E-mail your code as a gzipped tar file to 6.824-submit@pdos.csail.mit.edu by the deadline at the top of the page. To do this, execute these commands:

```
% cd ~/lab
% ./stop.sh
% make clean
% rm core*
% rm *log
% cd ..
```

```
% tar czvf `whoami`-lab5.tgz lab/
```

or

```
% cd ~/6.824/lab
% make handin
```

That should produce a file called `[your_user_name]-lab5.tgz` in your `lab/` directory. Attach that file to an email and send it to the 6.824 submit address.

Please post questions or comments on Piazza.
Back to 6.824 home.