15-749/15-449: Engineering Distributed Systems

Project 3: Coda Server Lookaside using CAS

Key Dates

Assigned	Monday March 17, 2014
Checkpoint 1	Monday March 24, 2014
Due	Monday March 31, 2014

Instructions

- All projects should be implemented individually, not in groups.
- You are free to give and receive help from anyone in class on high-level design issues, clarification on how something works, tracking down hard-to-find bugs, use of Linux tools, structure of the code base, and so on.
- It is *not* acceptable to copy code from/to someone else or tell/ask someone where lines of code have to be added, deleted or modified. If in doubt ask one of the instructors to help you, or check whether what you propose to do is acceptable.

Goal of the Project

By now everyone should be familiar with the Coda client code. In this project we are going to add the server in to the mix and make a substantial change to the way files are written back to the server.

The goal of this project is to use the SHA1 checksum of a file's content to avoid the overhead of sending a file to the server if the server already has a file with identical content. To do this we will send a SHA1 checksum of the file contents when we want to store a new file on the server. The server will then use this SHA1 checksum to check if it already has some other file with the same contents and, if so, avoid the data transfer.

In the course of this project we will:

- Build a lookaside database for the server that can be used to find existing copies for files.
- Add a new RPC2 call which uses lookaside to avoid data transfers during trickle-reintegration.
- Preserve interoperability with older clients and servers.

Setting up a Coda server

The server should already be built and installed from the source we have been working on and we only have to configure the server. But first we have to trick the server into allowing us to run it locally.

```
codaconfedit server.conf ipaddress 127.0.0.1
```

Next we run vice-setup as root. vice-setup will ask several questions where various files should be placed. Here is an example session (with example answers).

```
Do you want the file /etc/coda/server.conf created? [yes] yes

What is the root directory for your coda server(s)? [/vice] /vice

Is this the master server, aka the SCM machine? (y/n) y

Enter a random token for update authentication : something_random

Enter a random token for auth2 authentication : something_random

Enter a random token for volutil authentication : something_random

Enter an id for the SCM server. (hostname localhost.localdomain)
serverid: 1

You need to give me a uid (not 0 or 1) and username (not root)
for a Coda System:Administrator member on this server,
Enter the uid of this user: 1000

Enter the username of this user: admin

Are you ready to set up RVM? [yes/no] yes

What will be your log file (or partition)? /vice/LOG

What is your log size? (enter as e.g. '20M') 2M

Where is your data file (or partition)? /vice/DATA

What is the size of you data partition (or file)
[32M, 64M, 128M, 256M, 512M, 1G]: 32M

Proceed, and wipe out old data? [y/n] y

Where shall we store your file data [/vicepa]? /vicepa

Shall I set up a vicetab entry for /vicepa (y/n) y

Select the maximum number of files for the server.
[256K, 1M, 2M, 16M]: 256K

Congratulations: your configuration is ready...
Shall I try to get things started? (y/n) y
```

At this point the setup script will start the various daemons that are used by the system and create an initial volume which will be mounted at the root of your new Coda realm. Along the way we also created a single user with administrative rights. The username is **admin** and the password is set to the default value **changeme**.

If you now start your Coda client you should be able to access the new server as a new realm at `/coda/localhost.localdomain/`. To authenticate as the admin user you can use

```
clog admin@localhost.localdomain
Password: changeme
```

The clean way to stop the server is to use `volutil shutdown`, which sends an RPC2 call to the server to initiate a shutdown. You can check the log in `/vice/srv/SrvLog`, it should be clear when the shutdown has completed.

To restart the server, run `startserver`, which is a wrapper script that does some additional things like log rotation, or `codasrv`, the actual server binary. After a reboot you also need to run `auth2` as root to restart the authentication daemon, otherwise `clog` will not work.

Add a lookaside database to the server

The server will need to construct and maintain a lookup table of known SHA1 checksums for existing container files in `/vicepa`. It can then use this table to local copies of files with a matching checksum. If the server does not have a local copy and falls back to fetching the file across the network it should add the SHA1 checksum of the newly stored file to the lookaside database. The server-side lookaside database can also be constructed with the `mklka` command. The rwcdb database, which is the format in which `mklka` writes out the sha1-to-name mappings, is very efficient for lookups and will automatically reopen the database whenever it is changed on disk. So this database can, for instance, be updated by a separate process, such as running `mklka` from a cronjob.

Note: Because the CDB database format was originally designed to be read-only, our read-write CDB variant actually keeps updates in memory and rebuilds the complete file only when updates are explicitly synced back to disk. We can also open the database read-only and any additions will only exist in memory. If, at any point in time, `mklka` is used to build a new version of the database based on files in `/vicepa`, the new version is automatically opened and any in-memory updates are dropped.

Add a new RPC2 operation

When we want to send file data to a server, we obtain a handle to a temporary container file. Using this handle we push pieces of the local file to a server which reassembles them. When all pieces have been sent we reintegrate the store referring to the reintegration handle. As the data now only exists on one replica we follow this by explicitly triggering server resolution to propagate the new file data to any other replicas. In the code we tend to refer to this as partial-, or trickle-reintegration.

We want to add the following RPC2 call to `coda-src/vicedep/vice.rpc2`:

```
54: ViceOpenReintHandlePlusSHA (IN ViceFid Fid, IN RPC2_CountedBS SHAval,
                                IN OUT RPC2_Unsigned Length,
                                OUT ViceReintHandle Rhandle);
```

This call will be used as an improved replacement of the `ViceOpenReintHandle` RPC2 call.

We send two extra parameters to the server, the SHA1 checksum and length of the file we intend to store in the temporary container. The server uses this SHA1 checksum to look for a copy in the lookaside database and returns both the reintegration handle as well as the correct length as a confirmation that it found a match. If the server didn't find a match it will still return a reintegration handle, but should return a length of 0 to indicate that the container has not been filled.

The client can use the returned length to initialize the local offset, the end result is that if the server found a hit in the lookaside database, the client will not send any pieces of the file and immediately commit the store on the server by sending a `ViceCloseReintHandle` RPC2 call.

Checkpoint 1: Show that your initial implementation is working

- Show that you can copy a file from `/coda` to `/coda` without actually sending any file data back to the server. You may assume that *somefile* was already present on the server and that `mklka` has been run so that the checksum is present in the lookaside database.

Add support for lookaside for new files

When a new file is written to your server, add it's hash and the path to the container file to the lookaside database so that any additional copies of the same file will no longer require a data transfer.

Interoperate with others

For the final you are expected to show that your client and server are able to interoperate with the clients and servers of other students.

It is difficult to test a server bound to `localhost`. When someone else's client asks your server for the volume location it will respond with `you can find it at 127.0.0.1` and their client will then proceed and connect to whichever Coda server it can find on their local machine.

On top of that the Redhat Enterprise configuration is set up with a firewall that blocks incoming connections.

We will provide everyone with a virtual machine running in the Amazon EC2 cloud on Friday before the final demo. You will be able to SSH to this virtual machine, build your code, and deploy a Coda server that can be accessed via a public IP address.

We will still use the client running locally.

Final: Demo the full implementation

- Show that your client and server avoid unnecessary data transfers when you copy a recently created file.
- Show that you can read and write files on other students' Coda server implementations.
- Submit your code:
  - Make sure any newly added files have been added to the repository with `git add <filename>` and that all changes have been committed with `git commit`.
  - You can check what your commit history looks like by running `gitk` and for any uncommitted state with `git status`.
  - Push your changes back to your gitlab.cmusatyalab.org repository.
  - Submit your work by creating a pull request for the upstream Coda repository at `https://gitlab.cmusatyalab.org/15749/coda.git`.

Postscript

You may notice in the code that there are still code paths that used to send file contents to the server. When we had very good connectivity to the servers there was a synchronous `ViceStore` RPC2 call or we would reintegrate the `CML_STORE` operation to all available servers which then requested the missing data by sending a `CallBackFetch` RPC2 operation back to the client. Although this mode is still supported by both clients and server, it is disabled in recent Coda clients for several reasons. The callback connection uses a null key when setting up the encrypted connection which is shared by all users on the client as it is needed even if no local users have a token. Also because there is only a single shared connection from the server to the client, a large file back fetch blocks callback break RPCs.