Quorum Based Replicated Distributed File Server

Team Members:

- Danh Nguyen (nguy1952)
- Wen Chuan Lee (leex7095)

For a quick overview on how to run the project and test, please go to the **How to Run the Project section**, as well reading the **Client** section regarding the UI commands.

Overall Design

We designed the coordinator to be the entry point for all requests, in which all servers must first contact the coordinator in order to queue requests. The coordinator also acts as a server for any client, this is accomplished with implementing the same Thrift interfaces. The clients perform reads and writes into the distributed file system, through servers. In the coordinator there is are 2 synchronized data structures, a Queue of Requests, as well as a subscription set.

1. Requests made by a client are first passed to a Server, in which a connection is opened to the Coordinator, and the request is passed over.
2. The coordinator then puts the request in a syncronized queue called `requests`.

3. The Server then waits for the request to be approved by checking on the `subscriptions` set.
4. A QueueWatcher thread will then process the requests by removing them from the queue and putting them into `subscriptions`.
5. The Server will now see their request in the subscription queue and remove the request from the queue and process the request.

Concurrent reads and blocking writes are acheived using these sets, as Coordinator threads (each servicing a server request) are constantly checking the subscription set for their request. Upon receiving their request in the subscriptions set, they are given the green light to go ahead and assemble a quorum for read and write.

Reads are allowed into the subscription queue concurrently, while a write request is not put into the subscription queue until all reads are processed. Upon receiving a write request, the QueueWatcher waits for the subscription set to be empty (all requests have been processed). After which the write request is then put into the subscription set (the write request will be the **only** request in the set), and the QueueWatcher avoids approving other requests until the write request is complete and removed from the subscription queue.

We have now achieved concurrent reads and blocking writes across the entire DFS.

The Coordinator

The coordinator has a few primary responsibilities:

- Acts as the entry point for queuing all Server requests.
- Puts all read and write requests to the synchronized queue.
- Starts one QueueWatcher thread and one ServerSync thread
- Ensures blocking writes (no server threads can read when a write request is being processed)
- Also acts as a server (by implementing the same Thrift interfaces as the Server)

Servers

The server is a multithreaded Java application that services all client requests, mainly reading and writing. Reads and writes on the servers are done by first connecting to the well-known coordinator to make a request to read or write a file to the DFS. The call to the coordinator is necessary as the requests are all put into a synchronized request queue.

For added convenience when testing, file versions are output into respective directories of the server with the format `FILE_VERSIONS_`*hashcode* which contains the FILENAME and version number of each server filesystem. This file is updated upon every sync, which takes occurs every 5 seconds by default.

Client

The Client is a terminal to the File Server.

The Client will establish a connection to any server.

If the Client is successful, an simple interactive terminal asking for user input is then launched.

If the Client is unsuccessful, the Client will go to sleep for one second before retrying indefinitely.

The terminal contains a few simple commands to interact with the File Server.

- `read <filename>` - requests a file which will be the most updated file from the File Server.
- `write <filename>` - reads a file stored in `uploads` and uploads it to the File Server and replicating it across NW machines.
- `ls` - lists all files in the `uploads` directory
- `load-test <number of reads> <number of writes> <number of files>` - performs a batch upload operation to put all files stored in `uploads` into the File Server.
- `stats` - Lists the stats of the session. This displays the read and write of all requests.
- `exit` - closes the connection to the Server and quits the interactive terminal

`load-test` will start up a batch operation of randomized read and write requests. Testing and data collection was done using this command.

In the command, files from the `upload/` directory are randomly chosen

to be either written or read, in which some files not in the DFS may be attempted to be read. This testing would ensure that the distributed file system did not only correctly handle perfect scenarios.

Sync

Sync between servers happens after a set amount of time. Operations on the request queue is blocked until this sync is finished. The steps to how the servers are synced goes as follows:

- The coordinator is on a timer and calls sync after a set amount of time.
- When the sync occurs, the coordinator collects all the versions of all the files in the File Server.
- The coordinator uses the global view of the File Server to mapping of machines with the most updated versions of the files.
- This global view of the File Server is then sent around the network across all servers.
- The servers uses the view to check which file they need to download from across the network.
- The sync is finished when all servers have been checked in with the global view.

Life of a Request

For reads and writes from a client to a server, the following occurs:

- First a client issues Read/Write to the server.

- The server makes a RPC to the coordinator to start a quorum and waits for the response.
- On the Coordinator side, the request is put in a queue and waits till that request is ready to be served.
- When the request reaches the front of the queue the QueueWatcher thread notifies the coordinator to start the quorum process
- Based on NR/NW that many random machines are gathered.
- The coordinator pings all machines in the quorum for the version numbers for file.
- Depending on if it is a read or write the coordinator will be able to perform direct reads/writes bypassing the quorum procedure after it determines what files need to be read/updated.
- The result of the read/write is then returned back to the server that asked it.

Performance Results

Please refer to data.pdf

How to Run the Project

The File Server was built with the help of Ant. A prerequisite to run this project requires Ant. Additionally, the generated thrift files were also kept in a version control, but the `Thrift` compiler is also needed if Thrift files need to be generated (else only the Java Thrift library is needed).

Ant was used to handle automatic building and generation of class files (into a separate `bin` directory) as well as creating short targets for rapid developement.

To start the Coordinator, Servers, and Client on localhost

(This starts all processes on one terminal using forked processes and will cause all processes to print to the same `System.out`, which would cause very messy output as all threads attempt to print to `stdout` in an unsynchronized way.)

```
ant start-all
```

To start the File Server across multiple machines

```
1. ssh username@x32-05.cselabs.umn.edu
2. cd into project directory
3. ant start-coordinator
4. ssh username@x32-XX.cselabs.umn.edu (open another termin
al)
5. cd into project directory
6. ant start-server
7. Repeat steps 4 - 6 for the other Nodes
8. ssh username@x32-XX.cselabs.umn.edu (or open another ter
```

```
minal)
9. cd into project directory
10. ant start-client
```

Note

The servers connect to the coordinator to enroll into the File Server located at `localhost` on port `9090`, **to override this default, on Step 6 do**:

```
ant -Dcoordinator.address='x32-XX' -Dcoordinator.port='XXXX
' start-server
```

And on **Step 10 do** to change which server to connect to (the default connects to the coordinator):

```
ant -Dcoordinator.address='x32-XX' -Dcoordinator.port='XXXX
' start-client
```

Please see Ant Targets and Overriding Properties for more information.

Ant Targets

The fastest way to test out the entire DHT is to create 1 SuperNode and multiple Nodes on the same machine (but with a different port). Multiple

targets have been provided for automating the process of starting everything.

Note: Run commands from the project directory, where `build.xml` is localed.

- `ant start-all`
    - Create 1 Coordinator (port 9090) and 10 Nodes and a Client on the same machine.
    - All processes run in their own Java VM and as a forked process.
    - Client waits for 4 seconds before deciding to joining the cluster.
    - Note: see Ant Properties on how to start targets with different values.
- `ant start-coordinator`
    - Starts the coordinator on the current machine on the port specified by `coordinator.port`
- `ant start-server`
    - Starts a node that will connect to the coordinator located at `coordinator.address` and `coordinator.port`.
- `ant start client`
    - Starts the client that connects to a server specified under `coordinator.address` and `coordinator.port`

    - *After the client has joined, no new nodes should be allowed to join the DHT.*

- `ant thrift`

    - Fires up the thrift compiler to generates Java versions of the `.thrift` files located in `src` and saves them to `src/gen-java`.
    - This target should be called everytime the new *thrift* interfaces have been defined pr to update to a newer version of *thrift*.
    - Targets must be modified if additional *thrift* files are created, as I could not find a way for *Ant* to pass files individually to the *Thrift* compiler without introducing Ant plugins or Maven.

- `ant build`

    - Build but don't run the project. Note all targets that run depend on this target, so there is no need to run `ant build` before running the target.

- `ant clean`

    - Deletes all compiled bytecode in `bin`

- `ant diagnostics`

    - Prints out the Java Class Path defined in *master-classpath*

Ant properties (and overriding them)

Many of the above targets depend on some known information about the SuperNode and the ports to connect to. Sometimes these constants should be overridden (for example when starting multiple nodes/clients on the same machine). **To Override these constants**, use the format:

```
-D(propertyName)=(value)
```

To start the SuperNode with a minimum of `10` nodes:

```
ant -Dcoordinator.port=5050 start-coordinator
```

With a different port number `5555`:

```
ant -Dcoordinator.port=8 -DsuperNode.port=5555 start-supern
ode
```

Alternatively open up `build.xml` and edit the values:

```
 <!--Default Coordinator and Server properties used unless
provided-->

 <property name="coordinator.address" value="localhost" />
 <property name="coordinator.port" value="9090"/>
 <property name="server.port" value="5000" />

 <property name="src.dir" value="src" />
```

```
<property name="build.dir" value = "bin"/>
<property name="thrift.install" value="/usr/local/Thrift/"
/>

<!-- CHANGE THESE VALUES -->
<property name="NR" value="5"/>
<property name="NW" value="6"/>
```