Ryan Oman
Brad Bicknell
Abhishek Choudhary

CSE 490h – Project 5 – Facebook

We were able to add the application code for Facebook without changing much of our existing code. Here is a list of things we did change in the existing codebase:

- If an error is returned from the server on any command in a transaction, that transaction is aborted. Since each Facebook operation requires several reads and writes, this was necessary to guarantee that a Facebook operation completes the entire transaction or not at all.
- We separated our classes into packages instead of having a bunch of classes in one folder. We made the necessary changes to the compile.sh file. To run, you can't just specify the node class, you have to specify the package also. For example, to run the DistNode class, you would pass nodes.DistNode as an argument to the execute.pl script.

Implementation Details

Our Facebook code works directly on top of the existing transactional code. Each Facebook operation is implemented by a separate class. Each class inherits from the FacebookOperation abstract class, which contains shared functionality by all operations. Each sub-class contains a static final field that specifies the commands to be executed by that Facebook operation. A FacebookOperation is instantiated in the onCommand method and stored as a global field in the DistNode class. When an operation finishes, the field is set to null, and if there is currently an operation running, on one node, no other operation is allowed to start. When an operation is instantiated, it loads its predefined commands into a queue, which it then passes to onFacebookCommand, which is the former onCommand method in the DistNode class. Each FacebookOperation is also tied to the transaction that it started. If the transaction fails, an error message is printed out and the user must retry. A running FacebookOperation receives a callback when a command finishes, or when a transaction commits or aborts. When a command finishes, the FacebookOperation executes the next command in the queue. When a transaction commits, the FacebookOperation prints out a success message. Each operation is defined in the following sections. Here is a list of the support files used for Facebook:

- users – this file stores all users created, one per line. Each line has the following format: "[username] [password]"
- [username]_requests – this file stores all friend requests for the given username, one request per line. Each line has the following format: "[requester_username]"
- [username]_friends – this file stores all friends of the given username, one friend per line. Each line has the following format: "[friend_username]"
- [username]_wall – this file stores all messages posted on the given user's wall. The file is printed out as is, with later posts at the bottom of the file.
- logged_in – this file stores all currently logged in user and the nodes that they are logged in on. Each line has the following format: "[username] [node_addr]"
- .cur_user – this file explained in more detail under the login command

Create a User

Command: "create [username] [password]"

Execution:

- txstart
- get users – check to see if the user exists already, if it does, print an error message and exit
- append users "[username] [password]\n" – add user to users file
- create [username]_requests – create file to write requests to for new user

- create [username]_friends – create file to write friends to for new user
- create [username]_wall – create file to write messages to for new user
- txcommit

Login as a User

Logging in and our works a little bit differently than the other commands. On every client node, there is a temporary file called .cur_user. This file contains the current logged in username if it thinks a user is logged in, and doesn't exist if there is no user logged in. This logged in state is persisted across client restarts, because the server keeps track of clients that are logged in on in the logged_in file. Essentially the .cur_user file is like a session cookie. This cur_user file is passed to all of the other commands to keep track of the currently logged in user. Every FacebookOperation starts by checking that the current user is logged in on the server, since a client could think it is logged in, even though it isn't. In that case, the client would remove the .cur_user file and print an error message.

Command: "login [username] [password]"

Execution:

- txstart
- get logged_in – check to see if the user is logged in already, if they are, print error and exit
- get users – check to see if user exists, if it doesn't, print error and exit
- append logged_in "[contents]" – add user to list of logged in users, a user can be logged in to multiple nodes at the same time, so it also includes the node's address
- txcommit

Logout as a User

Command: "logout"

Execution:

- txstart
- get logged_in – check to see if user is logged in on this node
- put logged_in "[contents]" – replace the old contents of the file without the user/node combo that is logging out.
- txcommit

Request a Friend

Command: "request [username]"

Execution:

- txstart
- get logged_in – check to see if user is logged in
- get users – check to see that requested friend's username exists
- get [username]_requests – check to see that this user hasn't already requested to be friends
- get [friend]_requests – check to see that the friend hasn't already requested to be friends with the current user
- get [current_username]_friends – check to see that current user isn't already friends with the requested friend
- append [friend]_requests "requester_username\n" – append the requester's username to the requestee's request file
- txcommit

Ryan Oman
Brad Bicknell
Abhishek Choudhary

Accept a Friend
Command: "accept [username]"
Execution:

- txstart
- get logged_in – check to see if user is logged in
- get users – check to see that requested friend's username exists
- get [friend]_requests – check to see that the friend has requested to be friends with the current user
- append [current_username]_friends "[requester_username]\n" – add requester to current user's friend list
- append [requester_username]_friends "[current _username]\n" – add current user to requester's friend list
- put [current_username]_requests "[contents]" – remove friend request from current user's request file
- txcommit

Post a Message
Command: "post "[message]"" or "post [message]" if there isn't whitespace in the message
Execution:

- txstart
- get logged_in – check to see if user is logged in
- get [username]_friends – get all current user's friends
- append [friend]_wall "[message]\n\n" – append the message on every of the current user's friend's walls, the message is surrounded with a box of stars and accompanied with who the message came from
- txcommit

Read All Messages
Command: "read"
Execution:

- txstart
- get logged_in – check to see if user is logged in
- get [username]_wall – get the current user's wall
- txcommit

Known Issues/Assumptions

- The system doesn't recover from/handle a corrupted file state. For example, if a friends file has contains a user that doesn't exist/doesn't have all of its supporting files.
- Only one Facebook operation can run on any node at one time. Allowing multiples to run at the same time severely complicated our code.