Chatterbox Messenger

Documentation and Updates from Original Proposal

Soumya Kethu and Aniruddh Chaturvedi

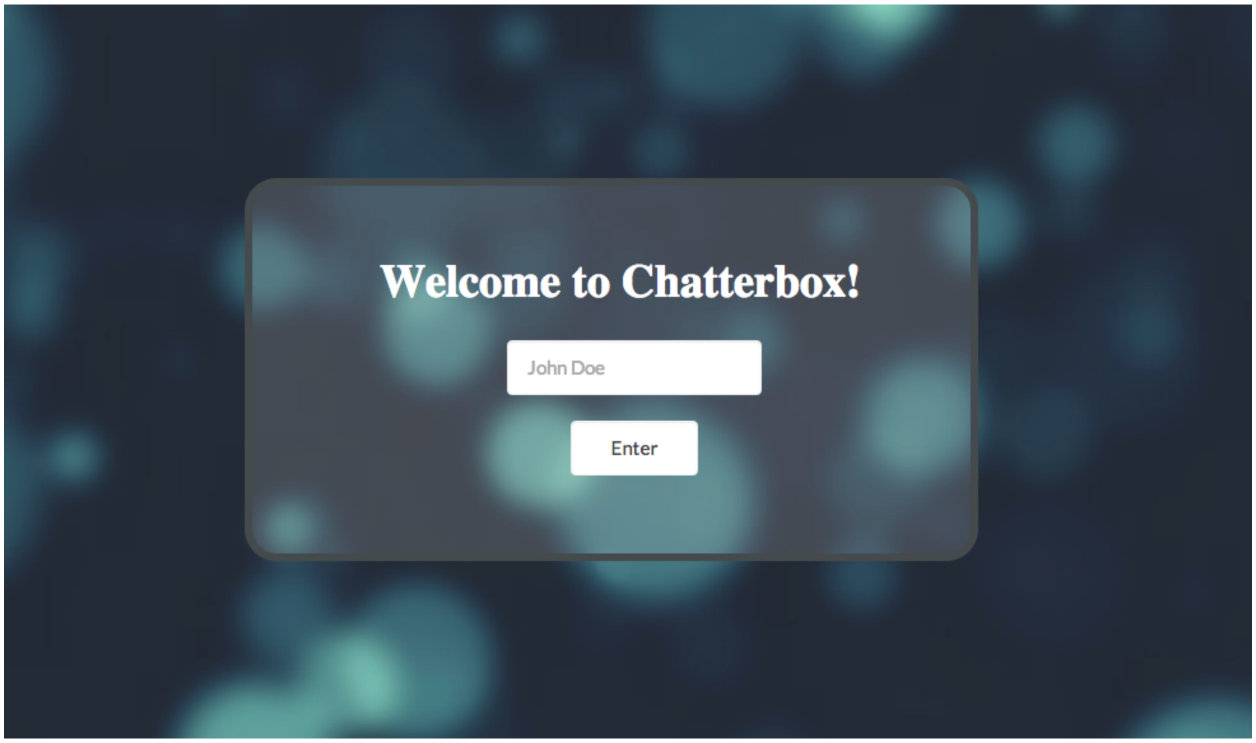
****__Update from our presentation with Professor Srini, to our final submission__ ****
Test basic 9 is the dead node test. We had misinterpreted what the test meant and didn't show this version to the professor. More information can be found in the last section.

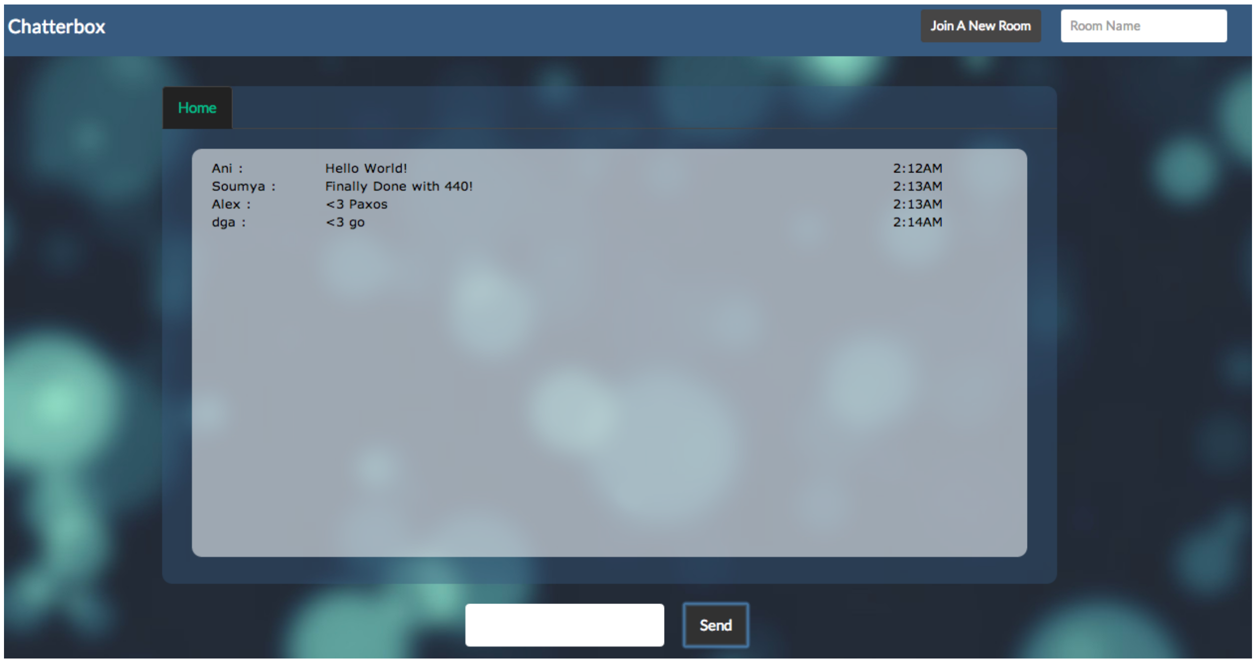
Front-End

For the front-end implementation of our web application we used HTML, CSS, Javascript and Bootstrap (what else?).
Instead of using Martini to connect our front end with our paxos implementation we decided to simply use websockets from "code.google.com/p/go.net/websocket", which is Golang's implementation of web sockets. All the interaction between the front end and the chat client server is through websockets, and from the chat client server to the paxos servers the interactions are through rpc. In addition a change from our original proposal, is that we decided to focus on our paxos implementation and therefore couldn't implement joining new rooms for users. Ignore the "Join Room" button at the top of the web page.

Files used for front-end:

File Name	**Description**
`bootstrap.min.css`	needed for bootstrap
`bootstrap.min.js`	needed for bootstrap
`bubbleBlur2.jpg`	background image for app
`chatclient.css`	all css used on top of bootstrap for app
`chatclient.js`	javascript used for chat client -> contains tl websockets
`startPage.html`	only html file required for using chatterbox





```
Ani :        Hello World!                 2:12AM
Soumya :     Finally Done with 440!       2:13AM
Alex :       <3 Paxos                     2:13AM
dga :        <3 go                        2:14AM
```

Paxos Implementation

Our implementation of Paxos is as originally planned.
We focused on creating an extremely robust Paxos implementation and a simple webapp that demonstrates the use of this algorithm rather than the opposite.
Our Paxos implementation handles failure, delay, logging, recovery, replication of logs, and recovery after a server dying.

Files for Paxos Implementation (in package multi-paxos)

File Name	**Description**
`paxos-server.go`	has functions related to all phases of paxos
`rpc.go`	contains the wrapper function for the paxos server when it registers to rpc
`args.go`	contains all structs that are used to implement paxos
`logger_impl.go`	contains the code for the paxos server to maintain their logs

Chat Client Implementation

Files for Chat client implementation (in package chatclient)

`chat-client.go`	contains all code required to connect with front end and connect with paxos servers
`chat-message.go`	contains the code for the chat message struct
`rpc.go`	contains wrapper function for the chat client server when it registers to rpc

Additional Files:

`purge.py`	A simple Python script to get rid of log files during testing.

Using Chatterbox Messenger

Running chatterbox messenger, involves the followings steps.

1. Run this command `./src/github.com/cmu440/chatterbox/tests/startChatterbox.sh` from the root directory (i.e. from `~/p3-impl`)
2. The main file corresponding to this script is chat-main.go
3. Note there is another file called main.txt (more on that below)
4. Run the command above, and then go on `http://localhost:1050` on your favorite web browser.
5. To join as another user, just go on `http://localhost:1050` in another window
6. Some notes:
    a. Send a message by pressing Send, not enter/return.
    b. When running the command sometimes rpc fails and the nodes don't join, just run the command again

Running Tests

To run the tests do the following:

1. First change `chat-main.go` to `chat-main.txt`
2. Next change `main.txt` to `main.go`
    a. The reason for doing this is because there are two main files -> one to start the webapp and one to run the tests, and we weren't able to figure out how to decide to run which one
3. now to run the tests you can call the following command: `/src/github.com/cmu440/chatterbox/tests/startservers.sh`
4. The command above will automatically run test 0 to change the test you want to run in the shell script startservers.sh change `-testNum` in the last line above the kill statements
5. If an error occurs with the command line flags, **this is a known bug** with golang. (See http://golang.org/src/pkg/flag/flag.go Line 747 (it even says `//BUG`, lol)).
    a. To fix it simply change the order of the flags. For example this is what is currently in the file.

```
isMaster := flag.Bool("isMaster", false, "to check if its a master server or not ")
N := flag.Int("N", 0, "to specify the number of servers")
port := flag.Int("port", 1111, "to specify the port number to start the master server
on")
testNum := flag.Int("testNum", 0, "the number of the test we want to run")
registerAll := flag.Bool("registerAll", false, "start test cases once all servers have
been registered")
```
`Simply change the order of the lines to or something similar`

```
N := flag.Int("N", 0, "to specify the number of servers")
port := flag.Int("port", 1111, "to specify the port number to start the master server
on")
testNum := flag.Int("testNum", 0, "the number of the test we want to run")
registerAll := flag.Bool("registerAll", false, "start test cases once all servers have
been registered")
isMaster := flag.Bool("isMaster", false, "to check if its a master server or not ")
```

6. Run the command in step 3 again.

Description of Tests

- `TestGetServers:` To check if `GetServers()` returns the correct number of servers once all servers have joined the paxos
- `TestBasic1:` To check that after the proposer sends a propose request, the acceptor replies OK if it hasn't seen anything before. Does one iteration of paxos without any failure of nodes
- `TestBasic2:` Does one iteration of paxos with the failure of first proposer after sending accept messages. Test that it commits the right message (the first one should be committed)
- `TestBasic3:` This is basically Shannon's scenario from her presentation about Paxos
- `TestBasic4:` Failure of a node, which then revives, check for consistency after recovery
- `TestBasic5:` Five iterations of paxos without failure to check consistency
- `TestBasic6:` Simple majority check, if out of 5 servers 3 are dead the message should not get committed!
- `TestBasic7:` Check recovery of a node that dies before it sends any accept messages
- `TestBasic8:` Check recovery of a node that dies after sending accept messages
- `TestBasic9:` **The dead node test.**
    a. Kill a node before it is able to propose anything.
    b. Commit one message through all of the remaining 4 servers.
    c. Now "quisce" as it says in piazza
    d. **You need to** Run the following command after it starts printing "WAITING FOR REGISTER ALL" ./src/github.com/cmu440/chatterbox/replacementServer.sh from the root directory (i.e. inside p3-impl)
    e. Check that the logs are up to date, except for 5. Then replace 5 with a new node, and recovers state of the new node
    f. Commit one more message, and check that all logs are consistent