Charlie Su
Saranyu Phusit
Mark Yang
CS490-DS0

DISTRIBUTED SYSTEMS PROJECT 1

SingleThreadedChatServer {Attempts 1,2,3,4}

SingleThreadedChatServer was surprisingly more difficult to implement than the MultiThreadedChatServer in that it was supposedly only allowed to run on a single thread, but that is seriously impossible when you have to keep a ServerSocket's accept() active to receive clients. The major problems we encountered originally were that BufferedReader's readLine() function was *blocking,* which ruined our implementation of heartbeat checking (we defined a heartbeat as System.currentTimeMillis(), and at the end of every loop check if the difference between the actual current System.currentTimeMillis() and the last heartbeat was over the heartbeat_rate value), because it meant that they won't be able to get the heartbeat until after the other clients' heartbeats have arrived. We resolved it with BufferReader's ready() function and a lot of other tweaks. There are still issues when the server is accepting *too many* users concurrently, with ArrayLists not being updated and indices going over bounds.

MultiThreadedChatServer: 4,8,16,32 {Attempts 1,2,3,4}

MultiThreadedChatServer was a lot more intuitive to implement because we were allowed to create new threads for each Clients' heartbeats *and* messages, meaning we didn't have to worry about BufferedReader's readLine() function blocking, worry about the "get" command taking too much time and delaying the heartbeats' readings of other clients, and worry about the ArrayLists not registering on time and getting IndexOutOfBoundsExceptions. It's the first approach almost anyone would take to implement a chat server!

RMIChatServer (Tested with 2K users for each Dummy Client = 8k in total)

Throughput: (number of responses received per second)

Client A: 331.3452617627568
Client B: 260.2811035918792
Client C: 204.7711682195147
Client D: 210.21652301870927

Average: 251.65351
Standard Deviation: 58.70885

Latency: (in milliseconds)

Client A: 2.6805
Client B: 3.543
Client 3: 4.5565
Client 4: 4.4245

Average: 3.80113
Standard Deviation: 0.8720

RMIChatServer is easier to implement because every actions for chatting are centralized in the remote object. It was pretty hard to learn how to work with it at first, but as I grew accustomed to it, it eventually became intuitive.

Though with more than 2.5k users per Dummy Client, we got some Exception as following:

```
error during JRMP connection establishment; nested exception is:
java.net.SocketException: Connection reset....
```

And with 10k and 100k users, 10-seconds heartbeatrate couldn't handle them all, causing the false disconnection broadcasts. Also, we got the above exception as well.