Chat System

David Gong, Stephen Hamilton

Abstract—Our goal is to develop a distributed chat server client system that is fault tolerant.

1 Introduction

THIS system will be composed of a client and a server. Servers will communicate through spread and utilize multicast with agreed ordering. There will be 5 servers that will communicate with each other and maintain a total order for all chat rooms. Clients can connect to any of the five servers, login, and create chat rooms, like already seen messages, change names, and leave chat rooms. The system will be fault tolerant in that each server can go down, and the system will continue to work as long as 3 or more servers are running. A server can leave the network and come back, and as long as a majority is present, the service will continue to work.

2 Design

2.1 Assumptions

Below are our assumptions.

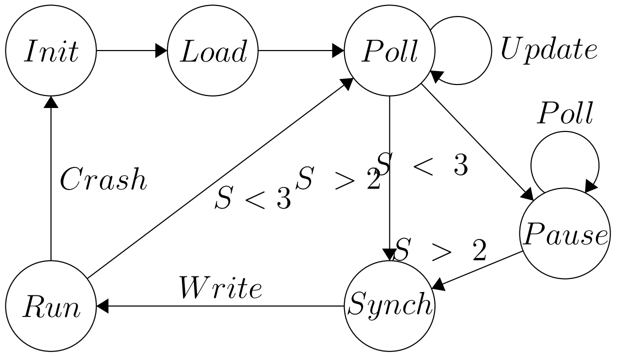
- It is valid to have a chat client not able to communicate if the server they connect to is not connected with the majority group of servers
- A chat client can go offline or crash, but will not produce malicious code and/or messaging that can cause the spread daemon to work incorrectly (i.e. spoof partitioning/join messaging).

2.2 Chat Message Packet

```
// Structure of chat packet.
int type;//Type of packet 0 for message,
   1 for like
int sequence;//Global ordering of message
char name[25]; //Text name of user
char group[25]; //Text name of chat room
char text[80]; //Text of chat message
int like_sequence; //Integer of sequence
   number of message liked
```

The message packet is utilized for both chat messages and "like" messages created by a user. The message packet will be saved to disk when received from spread, and the sequence number will be assigned upon receiving it from spread (it is null initially).

2.3 Server State Machine



2.4 Correctness

Servers will utilize Spread so that each message delivered by spread will be sequenced to ensure all servers maintain an agreed sequence over all messages. When a server receives a message from spread, it will append the message struct to the server logfile. This will only occur when in the run state. If there are not enough servers for a majority, the servers will tell the client that the system is down, and recommend they connect to another server. Once the server is in the majority partition, they will send the

latest agreed sequence number to the first available server to receive updates to the current sequence. Once they are synchronized, it will run like the other servers.
Upon a loss of a server, if the total number falls below 3, servers will update each other to the latest message, but no longer accept chat messages from users. Once the number is 3 or more, the servers will send their latest agreed number, and the highest one will update all others lower than that number. The servers that are not involved in this update will update to the agreed latest value once they join this majority of servers.

When clients send messages, they go to the chat server, and do not appear in the chat room until it is received by the spread daemon and ordered. When this occurs, the client is updated with their own message in the chat room. This is important, because it allows the spread daemon to give an agreed ordering to all chat messages, and this ordering cannot happen unless there is a majority of chat servers communicating.