**A Peer-to-Peer based Chat Application**

**Team Members**

1. Akash Gupta - 1001122031
2. Harshitha Gowda - 1001098221
3. Meet Brahmbhatt - 1001119131
4. Shruthi Shanthaveerappa - 1001106474

**Introduction:**

Peer-To-Peer communication mechanisms are used in order to get away from the drawbacks a client/server model suffers from. The following points are a few of the drawbacks that the classic client/server model suffers from:

• It has a single point of failure. Essentially this means that if the server goes down the service is not able to function properly if it is even able to function at all.

• The server needs to serve a possibly high amount of work because all requests in the system are directed to the server, so the load of the server is high.

Peer-to-Peer communication is interesting and potentially effective mainly because it takes away the single point of failure drawback that the client/server application suffers from and it gives an application a distributed communication instead. This enables peers to communicate directly with each other instead of having a server in the middle which all communication passes through.

This report includes a detailed description of the peer-to-peer architecture used in the project to build the chat application. It also includes the instructions on how to compile and run the application. It also includes the limitations of the project.

Election Algorithms

* The coordinator election problem is to choose a process from among a group of processes on different processors in a distributed system to act as the central coordinator.
* An election algorithm is an algorithm for solving the coordinator election problem. By the nature of the coordinator election problem, any election algorithm must be a distributed algorithm.

A group of processes on different machines need to choose a coordinator

            -peer to peer communication: every process can send messages to every other process.

            -Assume that processes have unique IDs, such that one is highest

            -Assume that the priority of process Pi is i.

The ring architecture assumes that the processes are arranged in a logical ring and each process is knows the order of the ring of processes.

Processes are able to Skip faulty systems, that is, instead of sending to a certain process “j”, send to “j + 1”.

Faulty systems are those that don’t respond in a fixed amount of time

Election in a Ring: Ring Architecture.

Assume that processes form a ring: each process only sends messages to the next process in the ring.

Active list: its info on all other active processes.

Assumption: message continues around the ring even if a process along the way has crashed.

Background: Any process Pi sends a message to the current coordinator; if no response in T time units, Pi initiates an election

1. Initialize active list to empty.
2. Send an “Elect(i)” message to the right. + add i to active list.

If a process receives an “Elect (j)” message

            (a) This is the first message sent or seen

                        Initialize its active list to [i,j]; send “Elect(i)” + send “Elect(j)”

            (b) If i != j, add i to active list + forward “Elect(j)” message to active list

            (c) Otherwise (i = j), so process i has complete set of active processes in its active list.

                        => choose highest process ID + send “Elected (x)” message to neighbor

If a process receives “Elected(x)” message,

            Set coordinator to x

**Architecture:**

The Ring architecture is used in this application. It basically has a Main Server running on port number 9000 which contains the list of the chatrooms available as well as the available/online users. Whenever a client signs in, it receives the list of available chatrooms from the main server. This is done by the Main Server Thread. The information here is transmitted by the thread and received by the server. The client has a local server as well as a local client thread. The first client connected to the main server will act as a super peer whenever another client becomes to be a part of the ring. The newly connected client will contact the main server to get the ip address of the last connected client. The first client interacts with the newly connected client via the port number 9010. The newly connected client will now get the list of available chatrooms form the first client via its local client thread. The first client will then establish a connection with the second thread via its local client thread forming a ring. When the third client joins the connection, it will get the ip address of the second client from the main server. So it will establish a connection with the second client. Now the second client will forward the ip address of the third client to the first client. The first client will now form a connection with the third client forming a ring. This process continues whenever a new client joins the connection.

**Prerequisite:**

* Java version 8
* Net Beans (version 8.0.1) is needed to run this application.
* JDBC connector (version 5.1.33) is used for the database connectivity.
* MySQL Server.

**Steps to run the program:**

1. Run the Chatroom\_Server project.
2. Set Server’s IP address in Chatroom\_Client.java – line 45 (socket = new Socket("ip address", 9000)).

Run the Chatroom\_Client.

**Windows at Client side:**

* **First Window (LoginScreen.java)**
  + Sign-up button: To register as a new user.
  + Sign-in button: To login with existing credentials
* **Sign-up Window (Signup.java)**
  + Enter username, password and repeat password in to text fields.
  + If username is available, you will be presented with chat room window.
* **Sign-in Window (Singin.java)**
  + Enter correct credential for logging in.
  + If user name and passwords are correct then you will be presented with chat room window.
* **Chat room window (chatrooms.java)**
  + You can either join any chat room by writing name of available chat rooms.
  + You can also create new chat room by clicking Create button.
  + When you create a new chat room or successfully join any chat room, you can be directed to Chat Window of that.
  + In this window you can see your username on top of chat room list.
  + By clicking on logout button, all chat windows, which are currently open, will be dispose and after that chat room window will dispose.
* **Create Chat Room Window (CreateChatRoom.java)**
  + You will be asked to give new name for chat room. If chat room is already exists, you will be shown appropriate pop out. Otherwise, new chat room will be created.
* **Chat Window (ChatWindow.java)**
  + In this window, you can see text sent by other participants of the chat room.
  + You can see your name and chat room name on top part of the window.
  + You can write text in this window.
  + By clicking on disconnect button, you will leave chat room and appropriate message will be sent to other participant of the chat room.

**Functionalities implemented:**

1. Sign up and authenticating the user while logging into the application
2. GUI with the basic buttons and functionalities.
3. Application showing the available chat rooms.
4. Create new chat room functionality.
5. All three points in client-server protocols.
6. Chat server requirement is fully satisfied.
7. Chat client requirement is fully satisfied.
8. Extra credit:
   1. User will able to see the number of users joined to specific chat room.

**How the code is implemented?**

The code is implemented keeping in mind the ring architecture. As the execution starts, first we need to start the execution of the server. This server is used to authenticate the client and it gives the IP address of the first connected client. Now, as new clients are connected they send their authentication request to the main server. When new client evokes it sends a socket connection request to the main server, which in the response of this request sends a reply that the connection is open and also sends the IP address of the last connected client to the new client. Now this newly connected client replaces the previous IP address on the main server with the IP address of the newly connected client. This is done so that every new incoming client will get address of the lastly connected client in the link.

Now each client will have two IP addresses, one is the IP address of the client who joined in the network previously to that client which will become the local server to that client and the other will be the main server which is used for authentication. The new client sends a ‘join’ message in the network to other client which is ahead of it in the ring, which passes this message similarly till the first client who joined. Now the first client finds that the IP address currently it is connected to not the last client in the ring, and that a new client has joined the network so it updates the IP address in its record and in this way the ring is completed when a new user connects to the network.

When the client is authenticated and the ring is formed the client is redirected to page where the chatrooms are displayed. Here client can join any chatrooms or create one. Now when a new client joins the chatrooms, the local server of that client sends a message to the local server of the client that is ahead of present client in the ring. The local server of each client checks whether the current client is in the chatroom or not. If the client is in the chatroom the message is shown and if the client is not in chatroom then the local server just forwards the message till when the message is received back by the same client.

When a user sign off/leaves the network, the client sends a ‘leave’ message along with the IP address and that message is sent forward to all the other nodes until client having the leaving client as its superior node. The message consist of the sender and the receiver IP address, so when this message is received by the client who’s receiver IP address is equal to sender IP address, that receiver changes its receiver IP address to the receiver IP address of the ‘leave’ message. In this way the ring is maintained when a client leaves the network.

**The MySQL database should have the following tables:**

1. ‘userinfo’ table to store the user details like username and password.
2. ‘chatroom’ table to store the list of the available chatrooms.

**Limitations:**

The limitation of this project is that the first client which connects to the main server should always sign in due to implementation issue.

**References:**

<http://www.aisectuniversity.ac.in/Anusandhan3/16.pdf>

<http://www2.cs.uregina.ca/~hamilton/courses/330/notes/distributed/distributed.html>

[http://www.diva-portal.org/smash/get/diva2:524732/FULLTEXT01.pdf](http://www.diva-portal.org/smash/get/diva2:524732/FULLTEXT01.pdf" \t "_blank)

[http://www.codeproject.com/Articles/1297/Peer-to-Peer-Communicator-and-File-Transfer](http://www.codeproject.com/Articles/1297/Peer-to-Peer-Communicator-and-File-Transfer" \t "_blank)

<http://listoffreeware.com/list-of-best-free-p2p-chat-software/>

http://www.codeproject.com/Articles/17321/A-simple-peer-to-peer-chat-application-using-WCF-n