SINF 2345

Afraid to Chat Strangers :

Laurent Lamouline - 3597-05-00
Vincent Nuttin - 5772-05-00

May 13, 2010

Contents

1 Part 1

1.1 Architecture

The main goal of the first part of this project is to implement a single chat room where users can join and leave whenever they want, exchanging messages with each other.

To realize this system, we choose the architecture shown below:

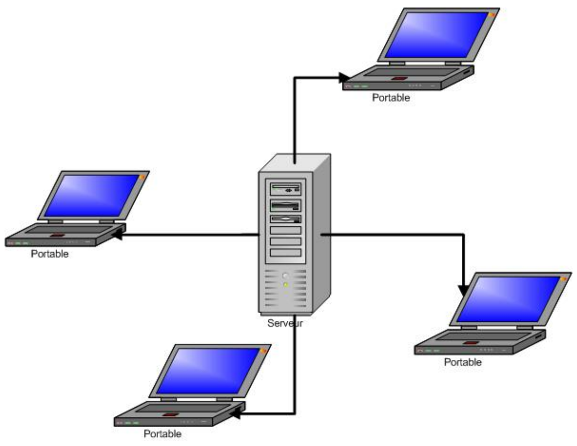

Figure 1: Architecture model of the chat room

We choose the architecture where each client of the chat is connected to a server that processes all the requests and broadcasts them. The server is initialized when a client decided, this creates a ticket indicating how other participants can join the server to participate to the chat. So thanks to this ticket (stored in a file), the client who initiate the chat can invite some other people just by giving them the ticket identifying the server. In our structure, each client is connected via its proper link to the server and can send messages to it. These messages are received by the server and then broadcasted, by using the *Best Effort Broadcast*, to all chat nodes that are "alive". To manage node failure, we have added an *Eventually Perfect Failure Detector* (EPFD) for each client connected to the server. As we will explain in the next part, this system allows us to know if a node is active or not (suspected) and so taking decision consequently. The internal model of our chat is given below.

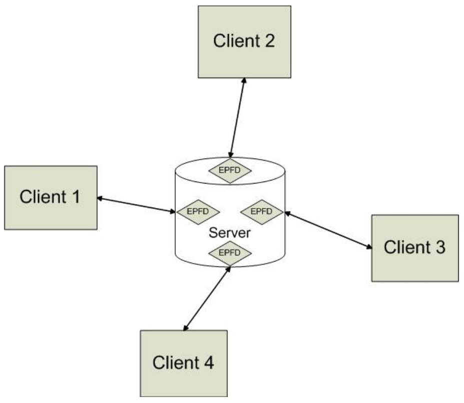

Figure 2: Internal model of the chat room

1.2 Some scenarios

If a client fails

If a client fails, it will not be able to send its `alive` messages any more. So, the EPFD will detect that this client is not connected to the chat any more and will advertise it to the other (alive or suspected to be dead) clients. The entire application still work normally.

If the server fails

It's clear that if the server fails, he does not receive the `alive` messages from the other clients. Due to this, the EPFD will not be able to say if clients are alive or not. The server is not able any more to broadcast the messages too. So, if the server fails, the entire application fails.

If someone try to connect with an already used username

At the beginning of the connection, the client send a `ReqUser` request with its chosen username to the server. The server looks to the table of all users and reply to the client if it is ok or not. Then, if it's ok, the client can send a `join` message to effectively join the chat and participate to the conversation.

2 Part 2

2.1 Architecture

The main goal of the second part of this project is to implement a chat where users can join and leave whenever they want, exchanging messages with each other without any kind of centralized stuff.

To realize this system, we choose the architecture shown below:

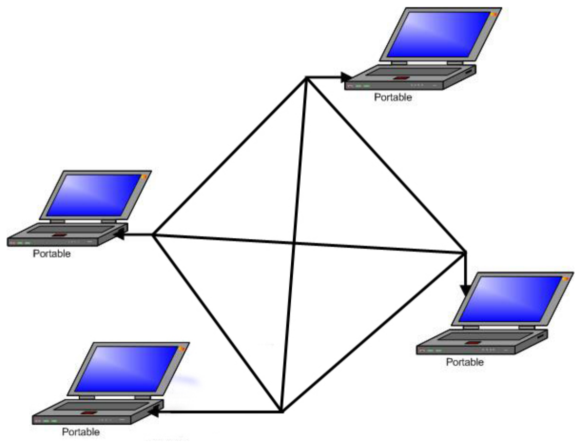

Figure 3: Architecture model of the chat room

We choose the architecture where each client of the chat is able to communicate directly with every other participant without having to go through any server. Every participant creates a ticket that can be use by others to join the chat. Each generated ticket is an entry point of the chat.

Concerning the broadcast of messages, each client has different modules to perform a `Fail-silent Causal Reliable Broadcast`. This module manages the causality and uses a `Eager Reliable Broadcast` module. This one uses a `Best Effort Broadcast` module (already implemented in the first part of the project).

To manage node failures, each client has its own list of *Eventually Perfect Failure Detectors* (EPFD) (one per participant). As we will explain in the next part, this system allows us to know if a node is active or not (suspected).

2.2 Some scenarios

If a client fails

If a client fails, it will not be able to send its `alive` messages any more. So, the EPFD of all other participants will detect that this client is not connected to the chat any more and this event will be displayed on the screen. The entire application still work normally without this client.

If someone try to connect with an already used username

At the beginning of the connection, the client send a `ReqUser` request to the client behind the chosen ticket. This message contains its chosen username. The contacted person looks to the table of all users it knows and reply to the client if it is ok or not. If it's ok, this person also sends a message to all people it knows to advertise them of the new participant's address. Then, the client can send a `join` message to effectively join the chat and participate to the conversation.

3 Conclusion

To conclude, we can say that the major differences between the two architectures are

- In the first case, if the initiator of the chat leaves or fails, the entire application fails. In the second case, it is not the same because each participant is autonomous.
- In the first case, the room can always continue working if and only if the server is up. In the second architecture, the only condition to make the room always working is to have at least one active client in the room. That permits us to join the room via its ticket.

It seems to be clear that the second application is more robust than the first one and will better fits the real needs on Internet.