# Seminar Report: Chatty

**Albert Bausili, Noa Yu Ventura, Pol Verdura**

March 8, 2023

Upload your report in PDF format.
Use this LaTeX template to format the report.

## 1 Open questions

*Try to answer all the open questions in the assignment. When asked to do so, perform experiments to support your answers.*

**a) Does this solution scale when the number of users increase?**
No, it does not, because one server needs to send a message to N clients, and if N is very large this server will saturate, especially when receiving messages from all the clients.

**b) What happens if the server disconnects?**
The users no longer receive messages sent by other users or updates of the client list, even though they can still send messages.

**c) Are the messages from a single client guaranteed to be delivered to any other client in the order they were issued? (hint: search for the 'order of message reception' in Erlang FAQ)**
They are guaranteed to be delivered in the order they were sent only if it is within one process. Ex: in the same process message A is sent and then message B is sent. If message B is received, it is guaranteed that message A has arrived first.
If it's not within the same process, it is not guaranteed at all.

**d) Are the messages sent concurrently by several clients guaranteed to be delivered to any other client in the order they were issued?**
No, because the server might receive the message before or later due to latency, we can not guarantee that the two messages will arrive at the same time to the server. Therefore, we cannot guarantee that they will be delivered in the order they were issued.

**e) Is it possible that a client receives a response to a message from another client before receiving the original message from a third client?**
Yes, an example would be the following: the third client message can be stuck in the buffer of a router while the response may not get stuck in any router, as it does not necessarily need to pass through the same path as the third client.

**f) If a user joins or leaves the chat while the server is broadcasting a message, will he/she receive that message?**

Firstly, the server finishes broadcasting the message, and sends it to all clients, including the one that left. It is because when the server processes this message, it reads the client list before it is updated. It then reads the leave request from the client and updates the clients list.

On the client's end, it leaves the chat locally and sends to the server a message that it's leaving the chat. Then the client receives the broadcasted message from the server, but since the user left the chat (exited the process), the user does not receive or read the message, it is ignored by the client machine.

**g) What happens if a server disconnects?**

When a server disconnects the other servers receive a message informing them of the new Server list. This works independently of the server that is shut down.

The clients connected to that server are no longer able to communicate through the chat, it stops working and a message is shown that the client has left. The other clients using the service can still communicate as if nothing happened.

**h) Do your answers to previous questions c), d), and e) still hold in this implementation?**

Our answer to the c) question still stands as there are multiple processes involved and erlang only preserves the message order inside the same process.

Our answer to the d) question is the same as the latencies in the connections between client-server and also between server-server may have different latencies, thus the messages may be received by the servers in different orders than they were issued by the clients.

Our answer to the e) question still is the same as different clients and servers may be in different networks with different topologies and different routes between them, and the same example still works for this new implementation.

**i) What might happen with the list of servers if there are concurrent requests from servers to join or leave the system?**

Servers may get their list of servers overwritten, because when they receive a request they broadcast their server list once updated. Doing so, it might cause an inconsistency over the servers of the system overwriting their past updates and losing their lists. This is, having two events 1 and 2, and one server having its list updated for event 1 but also getting an update from an outdated server which has its list updated for event 2, without knowing that event 1 happened.

**j) What are the advantages and disadvantages of this implementation regarding the previous one? Compare their scalability, fault tolerance, and message latency (measured as the number of hops).**

1. **Scalability**: there's a lot more scalability thanks to load balancing, because now the load can be distributed among all servers, a large number of clients can connect to different servers, not only one. However, as we lack a distribution system or any kind of load balancing, we depend on the users connecting evenly among all the servers and we cannot assure that there will be a good load balance.
2. **Fault tolerance**: there's robustness against failures because if a server is disconnected, the other servers can still provide the service the clients ask for, disconnecting a server only disconnects the clients that were connected to it, it doesn't disconnect all clients using the service. It is not perfect because the disconnected server's clients lose the service, but it could be easily be solved by redirecting them to another server.
3. **Message latency**: the message a client has sent needs to first arrive at the server the client is connected to, and then it is distributed to all the servers, which will send this message to their own clients. The message now has to do more hops compared to the previous implementation, where the maximum number of hops was at most 2. Now, the time it takes to be sent and received also depends on the number of hops and the latency between clients-servers and between servers-servers.

# 2 Personal opinion

*Give your opinion of the seminar assignment, indicating whether it should be included in next year's course or not.*

**Albert**: From my point of view, this laboratory has been a great start point into the distributed systems and I really do appreciate that the Erlang language is used just as transport and we are not forced to learn the language to its core, as in the future we may end up using other programming languages.

**Pol**: Personally, it's a nice first contact with Erlang. This seminar lets you see how Erlang works, start to feel comfortable with it and think about how many things you can do with it. Moreover, it's a really fun laboratory to start with and not that complicated.

**Noa**: I think that this laboratory is very easy, but ideal in order to understand the basic concepts of a distributed system and how it works. It is very satisfying because you can see what you do and what it is exactly doing, and it is very fun to talk with friends through a chat you made yourself.