Seminar Report: Chatty

Xavier Caballero & Albert Capdevila

March 6, 2022

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

1 Open questions

1.1 One Server

a) Does this solution scale when the number of users increase?
No, the users scale as there is a client for each of them but the server is the bottleneck of the system. Once the server reaches its max throughput it does not scale anymore.

b) What happens if the server disconnects?
That the users remain "connected" but their messages are not sent to each other since we only have one server that manages all the clients.

c) Are the messages from a single client guaranteed to be delivered to any other client in the order they were issued?
In this case, as there is only one process, we can assure the order of the messages sent will be preserved.

d) Are the messages sent concurrently by several clients guaranteed to be delivered to any other client in the order they were issued?
For that other case we use multiple processes simultaneously sending messages, so the order can not be preserved.

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?
In this case it is not possible as the server first broadcasts the question to all clients before listening for the answer.

f) If a user joins or leaves the chat while the server is broadcasting a message, will he/she receive that message?
In the case of a join, it won't receive the message because the server will process the join after broadcasting the message.
When the client sends an exit, the server will first finish the broadcast and then process the exit. In this case, the client will first receive the message and then close.

1.2 Robust implementation

g) What happens if a server disconnects?
Clients that were connected to the server being disconnected, won't receive any message and their messages won't go anywhere. Clients connected to other servers will continue working fine.

h) Do your answers to previous questions c), d), and e) still hold in this implementation?
In the case of **"c"** the order will be preserved as the messages are sent from 1 process to another.
For **"d"** it will continue without preserving the order for the same reasons.
For **"e"** it could happen if we take into account the following situation:
Clients (C)
Servers (S)
Latency (t)

MESSAGE	**TIME**
C1 (Question) : S1 - S2 - C2 C2 (Answer) : S2 - S3 - C3	t1
C1 (Question) : S1 - S3 - C3	t2

If t1 < t2, C3 will first receive the answer and then the question. We know that the answer will also go from C2 to C1 but this case it is not important for the explanation.

i) What might happen with the list of servers if there are concurrent requests from servers to join or leave the system?
In this case, it could happen that there is no consistency between the lists of the servers. If we have two servers and each of them makes a request to join the system at the same time, each server will send an update and this will cause them to have outdated lists.

j) What are the advantages and disadvantages of this implementation regarding the previous one? (compare their scalability, fault tolerance, and message latency)
Scalability: In this case, since we can have more than one server, there is no bottleneck as in the first solution. This makes this implementation more scalable.
Fault tolerance: This case is very similar to the previous one. As we have several servers, if one fails, only the clients that were connected to that specific server will lose the connection to the chat. This makes this implementation more fault tolerant.
Message latency: If there are many servers and clients connected at the same time, it could happen that the time to receive a message increases considerably and consequently there would be a higher latency.

2 Personal opinion

We think this seminar is a good introduction to Erlang since it allows us to see a bit how the language works without too many complications. As we have the code almost done, this seminar is a good way to better understand functional programming. For the previous reasons, we think that this seminar it should be included in next year's course.