Seminar Report: Chatty

Roger Oriol Pérez, Jordi Vergaray Moran, Pau Alòs Mairal

February 23, 2020

1 Open questions

a) Does this solution scale when the number of users increase?

No, this solution doesn't scale because all the communications must be between one server.

b) What happens if the server fails?

The clients don't notice that the server has failed until they try to send a message and realize that the message is not sent.

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 FAQ2)

Yes, since the Erlang FAQs state that: "Yes, but only within one process. If there is a live process and you send it message A and then message B, it's guaranteed that if message B arrived, message A arrived before it.".

d) Are the messages sent concurrently by several clients guaranteed to be delivered to any other client in the order they were issued?

No, since the order that the messages are issued does not guarantee that they get to the server in the same order.

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?

With just one server it's not possible since when the second message gets to the server, it's guaranteed that the first message will already be in the queue, so the second will get second in the queue.

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

Since the join and leave events are also messages in the server, in the first case when the user sends the join message, the broadcast is already sent to all the clients that the server had, so the client won't receive the message. In the second case, the client execution will not stop until the server handles the leave message, so the broadcast message will be received by the client before exiting.

g) What happens if a server fails?

Only the clients connected to the server that has failed will lose service and therefore they will not be able to send or receive messages. The rest of the client will not be affected.

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

The answer to the question c will be the same for the same reason provided in the Erlang FAQs.

The answer to the question d will not the same. You cannot guarantee that the latency between servers and clients will keep the same order in which the message are issued.

The answer to the question e will not be the same. Like the previous question, you cannot guarantee that the latency between servers and clients will keep the same order in which the messages are issued, even with this causality, since the client that receives does not participate in this causality.

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

Since when a server joins or leaves the system we are updating the whole list of servers in every server, if two different servers issue a new update of the list (join or leave) the system can be left in a weird state where the list of servers is different in different servers of the system.

j) What are the advantages and disadvantages of this implementation re- garding the previous one? (compare their scalability, fault tolerance, and message latency)

This second solution is more scalable than the first, since we can have any number of servers in the system. It is also better in fault tolerance since if a server crashes, the rest of the system will keep working normally. With latency, the first solution is better since with only one server we can guarantee that the order of the messages will keep causality.

2 Personal opinion

In our opinion, this seminar assignment is a great introduction to the development of distributed systems with Erlang. It showcases the basic features and challanges of these kinds of systems with a very simple application. Therefore, since we believe that this seminar gives such a great introduction to the world of distributed systems with Erlang, it should keep being included in next year's course.