Homework 4 Report

Rafat Khan

October 6, 2021

1 Introduction

The goal of this exercise was to implement a group membership service, composed of several members. Each member has a state which is its color. Randomly, a member can generate a new color and multi-cast it to all other members in the group for updating. All the members have to be synchronized, that means all the members GUI have to display the same color at the same time.

2 Main problems and solutions

To ensure synchronisation between members, each message generated by a member is sent to all others members in the group, through the leader. But such a service is not very fault tolerant. For example, what if the multi-caster crashes after sending the message to only a subset of nodes belonging to a group. An important thing to understand is what guarantees the system actually gives on message sending. The specifications only guarantee that messages are delivered in FIFO order, not that they actually do arrive. We have built our system relying on reliable delivery of messages, something that is not guaranteed. This problem can be handled by keeping a number on each messages and each member should contain the last message it has seen in his state.

3 Evaluation

3.1 First implementation

First, an implementation without supporting fault-tolerance has been implemented. A multi-cast is often implemented via a layer on top of membership management and refers to a broadcast where not all receivers are intended receivers. In this assignment, a group is composed of a leader and several slaves. When a slave receives a message from the application layer, it transmits the state to the leader that multi-casts the message to all other members of the group. In this scenario all the members will receive the message for updating their colors and stay synchronized only if we imagine that there is no network failures.

To test that behaviour, we create a leader and three slaves. Their colors were synchronized which means that the implementation of the group membership service is correct. However, if the leader crashes, the two other slaves stop to displaying colors because the re-election of a new leader is not yet implemented.

3.2 Second implementation

In the second implementation, there is a error detector to monitoring the crash of leader. If the leader crashes, then a new leader will be generated by election process. In this way, the nodes can still be synchronized after the leader crashes. We added the detection of the leader crash for the slaves. When the leader crashes, there is a re-election procedure. When testing, we can see that when leader crashes, other members continue to display different colors, due to the re-election of a new leader.

If we insert a crash simulation when the leader multi-casts the message to other nodes, we can notice that when the leader dies, the slaves became out of synchronization.

With one leader and three slaves, we can notice than the second slave doesn't display the same color than the first slave. We can explain that problem by watching at the bcast/3 method. The leader have sent a message to slave 1, then crashes. So, the slave 2 didn't receive any message from the leader. As a consequence, the slave 1 has received a message than slave 2 didn't receive, so they went out of synchronization.

3.3 Third implementation

To avoid the synchronization problem, we implemented a third solution. Each message has a number and each member contains in his state, the last message he has seen. If the multi-cast of a message failed, the new leader will send again the message to all the members. To avoid duplicate messages, we check if the number of the message received is not inferior to the number of the last message seen. If so, the message is rejected.

With that implementation, we don't have the problems of synchronization when the leader crashes, and that's why all the colors are still synchronized between all members.

4 Conclusion

Through this assignment, we understood the techniques involved in building a reliable multi-cast service which is fault tolerant to nodes leaving/joining the group. Also, it gave us an opportunity to understand design goals of a reliable multi-cast system. We saw how we can manage a group membership service to deliver reliable messages to a group of nodes. The synchronization works, even if the leader crashes.