Groupy: a group membership service

Mikael Karlsson

September 28, 2019

1 Introduction

In this assigment I implemented a group membership service that provides atomic multicast where each node in the group gets messages in the same order. The aim is to have several application layer proceses with a coordinated state. If a node wishes to perform a state change it has to mulitcast the change to the group so that all nodes can execute the change. In the assignment a state is represented by a colour.

2 Main problems and solutions

A network consists of nodes where each nodes have 2 layers, application layer and group layer. The application layer is responsible for printing the state (colour) to the screen and communicates with the group layer. The group layer can either be a leader or a slave in the network and it's main purpose is to forward messages from/to the application layer. If a node wishes to multicast a message must first send it to the leader and the leader will forward the message to the rest of the group. Each node has a "view" of the network, a list with active nodes.

The assignment is divided into 3 different part Gms1, Gms2 and Gms3.

2.1 Gms1

Gms1 is very basic. It can create a leader and some slave nodes can join the network, but there is no error handling if the leader dies. The network will simply die when the leader dies, because nobody is forwarding the messages through the network.

2.2 Gms2

Gms2 can handle the case when the leader dies, this is done by an election. Each slave node is monitoring the leader. If the Leader dies, the nodes selects a new leader through an election. The election is pretty straight

forward, the node that is in front of the slave-list will be chosen as the new leader and a new view is broadcasted to the rest of the group so they get informed that there is a new leader.

2.3 Gms3

The third implementation handles when a leader dies when broadcasting a message to the group and not all the members got the message. The leader is broadcasting messages by using a list with the nodes in the network. The same list is used when there is an election for a new leader, and as I described above, the node at index 0 in the list will be the new leader. This means that if a leader dies in the middle of broadcasting messages, the new leader did receive the last message before the leader died. Therefore we save the last message so that the new leader can re-broadcast it and all the nodes will get the message.

Unfortunely this solution creates another problem, there is a risk that a node receives the same message twice. This is solved by a sequence number that each message will be tagged with. The group process stores a sequence number, if it receives a message with a lower sequence number than the one it stores, it knows that the message is old. If the sequence number is equal to the stored one, it means that it is a valid message. The leader increments the sequence number every time it sends a message and the slave increments it when it receives a message.

3 Evaluation

I tested by Gms1 by starting a leader and 4 slaves. It is visible that the nodes is in the same state since all windows is changing colour simultaneously. If i exit one of the slaves the network will still work, but if i kill the leader the network will stop working as expected. The nodes will stop change colour because there is no leader that can forward the messages from the slaves.

I tested Gms2 in the same way as Gms1. As expected the network stilled worked when i killed the leader. This is because a new leader is selected through a election. But when the "crash-function" killed the leader in the middle of a broadcast i could see that the nodes didn't synchronize in it state change.

I tested Gms3 in the same way and i could see that the implementation handled a situation where the leader didn't broadcast a message to all the nodes before it was killed by the "crash-function".

4 Conclusions

I think it was a fun assignment that showed how a group membership service can be implemented with atomic multicast. It also showed how we can handle cases when a leader dies and still maintain the network.
As mentioned in the description of the assignment, we have build a system that is relying on reliable delivery of messages, something that is not guaranteed. So my answer to the question: "How would we have to change the implementation to handle the possibly lost messages? How would this impact performance?" is that we could implement some type of logical clock and a memory with old messages for each node, and the leader keeps track of each nodes clock. Then the network will notice that a message has been lost when it receives a message with number 10 but the last message it got from that node had number 8, then the leader will ask the node to resend message number 9 stored in the nodes memory. This method will take up much more memory because each node has to save old messages.

Another way to solve the issue is to implement acknowledgement messages back to the leader from the slaves. This method wont take up memory but it will slow down the speed because lot of messages will be sent over the network.