Isabel Ghourchian CINTE14

Homework 4 Groupy - a group membership service

1 Introduction

The assignment was to implement a group membership service that provides atomic multicast with the aim of having several application layer processes with a coordinated state. This means that they should perform the same sequence of state changes. Each node's state is visualized in a colour and when a node wishes to change state/colour it multicasts it to all members in the group. All the nodes in the group should be synchronized and therefore every node in the system should display the same colour.

2 Main problems and solutions

A group membership service is implemented that provides atomic multicast in view synchrony. The communication is divided into views, and messages will be said to be delivered in a view.

2.1 First implementation

The first implementation of the system is a program without support for fault-tolerance. A group consist of several nodes who play the role of either a leader or a slave. Slaves receives messages from the application layer and forwards them to the leader who multicasts it to the rest of the group. The nodes will then receive the message and update their colors.

The application layer also known as the master has a group process that it communicates with also known as a slave. The application layer will send multicast messages to the slave and receive multicast messages from it and decide if a new node should be allowed to enter the group and what initial state it will have.

2.2 Failure handling

We then added an election procedure whose purpose is to elect a new leader. All slaves have the same list of nodes and they select the first node as the leader. A slave that detects that it is the first node in the list will adopt the role as a leader. The problem is if a leader dies before it has sent a message the message will be lost. Some nodes will therefore not receive their messages and will no longer be synchronized.

2.3 Reliable multicast

To solve the problem of a leader who dies before multicasting a message to the group, a new version was implemented. The messages now have a sequence number and a copy of the last message received from the leader.

The newly elected leader should resend the last message seen. This is needed to cover the cases where a previous leader crashed and did not manage to send the message to all nodes in the group. This could cause duplicate messages being sent, if a node received a message just before the old leader dies and then again receives it when the new leader is selected.
To avoid duplicate messages, the sequence number of a message is checked and if it is lower than the number of the last seen message it means that means the message has already been received and we discard it.

2.4 Optional task - Reliable delivery

The optional task was to change the implementation to handle possibly lost messages. In Erlang the only guarantee is that messages are delivered in FIFO order and no guarantee that they actually do arrive.
To solve this the slave should send an acknowledgement message to the leader each time it receives a message from the leader. The leader waits for the acknowledgment message and if it has not arrived within a certain amount of time the message is resent.

```
receive
      {ack, Id} -> ok
  after 100 ->
      sendMsg(Node, Msg)
end.
```

3 Evaluation

The program is started with a benchmark program called test which starts some workers. In the first implementation there is no election of a new leader, so if the leader crashes there is no new one chosen. All the node's colours will then stop changing colours since no more messages will be received from any leader.

In the second implementation a new leader will be elected but when a leader dies, the other nodes will be out of synchronization. They start blinking in different colours.

When running the last implementation of the program, the synchronization between the nodes remains even though a leader crashes.

The optional part shows printouts of messages that has been lost but the nodes are still synchronized because the leader resends the messages. The nodes would otherwise become out of synchronization if there was no acknowledgement message. The disadvantage with using acknowledgement messages is the effect it has on performance since the number of messages now has increased.
If we have a congested network where there is delay of messages and the monitor might think that a leader has died when in reality it is still alive. And then if a new leader is selected there will be one extra leader.

4 Conclusions

From this assignment I learned the importance of node synchronization and the things that might happen during crashes. I also learned how Erlang delivers messages in FIFO order and that they are not always delivered correctly.