Groupy: a group membership service

Nikhil Dattatreya Nadig

October 4, 2017

1 Introduction

The objective of this exercise was to construct a group membership service that provides multicast. The aim is to have several application layer processes with a coordinated state i.e. they should all perform the same sequence of state changes.
The objective of this assignment is to:

- Understand the concept of coordination and agreement.
- Understand the importance of reliability in a distributed system.

2 Main problems and solutions

The problem is that all nodes need to be synchronized even though nodes may come and go (crash). This problem is handled incrementally.

2.1 First Version

In this version, several nodes are created and entered to a group. The first node is by default the leader and the other nodes are the slaves. If for any reason, the leader is lost or crashes the other nodes i.e. the slaves do not change their state. This happens because, there is no election of a new leader to direct the other slaves.

2.2 Second Version

In the second version of the solution, the leader is monitored so that we can elect a new leader if the existing leader crashes. To do this, Erlang's monitor function is used.

```
erlang:monitor(process, Leader)
```

Monitoring all the nodes in the group would have been a little tricky, but here, we are required to monitor only the leader. Once it is detected that the leader has crashed/no longer available, an election needs to be carried out to decide on the next leader. For the sake of simplicity of the assignment, the first node in the list of all the nodes is chosen as the next leader by all the nodes. This solution is however still not complete. There can be a case when the last message from the leader was not received by the slaves.

2.3 Third Version

In the third version, a more reliable multicast is created. This is done by storing the last message received from the leader. When the leader crashes, the new leader re sends the last message it received from the previous leader. This is to ensure that, all the nodes in the group are in the same state. There are certain edge cases that need to be handled. There can be instances when the last message that the new leader received from the last leader may not be the last message the old leader sent. In this case, the other nodes which received the actual last message will receive an older message from the leader. Here, the nodes need to reject these states and wait for the next message.

2.4 Further Improvements

In all the previous version, it is assumed that all the messages that are sent from the leader are sent to the slaves. But, this may not be the case, due to network failure or traffic there are chances for a message delivery failure. Messages sent need to be stored by the leader so that if any of the nodes do not receive the message, the message can be sent again.

3 Evaluation

In the first version of the solution, synchronization has been achieved. However, if the leader node fails, the rest of the nodes do not continue to function. In the second version, the election of a new leader is implemented so that on failure of the existing leader, the entire group does not stop functioning. In the third version, the membership service is more robust where last messages are stored by all slaves and the new leader broadcasts it when so as to maintain synchrony.

4 Conclusions

We have understood the importance of synchronization among several application layers that needs to have the same state at the same time.