Report HW4: Groupy - a group membership service

Artem Sliusarenko

October 4, 2023

1 Introduction

The purpose of this assignment is to create a group of processes with coordinated state. Group behaviour should be synchronized using atomic multicast approach. Each state change has to be communicated to all group members before it is executed. Keeping processes synchronized is the main challenge since processes can crash and be replaced by a new group member. The role of multicasting is assumed by the group leader and all state change requests must go through the group leader. An external node can requests to become a memeber through any group member, however group leader will decide when to include new group member.

2 Main problems and solutions

3 Evaluation

One of the cornerstones of synchronized processes is failure handling. Section 3.1 of the assignment introduces failure detectors. Failure detectors help the group recover if its' leader has crashed. New leader is elected and processes continue operating in synchronized manner. Section 3.1 raises a question. *We must pay attention to what we should do if, as a slave, we receive the view message from the new leader before we notice that the old leader is dead. Should we refuse to handle view messages unless we have seen the DOWN message from the leader, or should we happily receive and accept the new view and then ignore trailing DOWN messages?*

Slaves should not refuse to handle view message from new leader. All group members have to have same state. Eventually, slave will receive the DOWN message from old leader process and realise that new leader was elected.

Section 3.2 discusses the unreliability of current multicast functionality. We can observe some nodes behaving out of sync with the rest of the group.

My experiment revealed that nodes that have send join request to the group whoes leaderis about to crash behave out of sync with the rest of the group.

Reliable multicast, section 3.3, utilizes message identifiers passed to the group nodes in addition to the last message group members have received. That way we can make sure to deliver latest message to all nodes if leader has died during the multicast procedure. The latest imlementation of the group multicast module, gms3, does allow us to keep rolling as existing nodes die by adding new nodes to the group.

We are operating on the assumption that all messages will be delivered, when in reality it is not guaranteed. To handle lost messages we weould need to add a persistant store to keep all our messages with unique identifiers. A message is lost if Nodes receives message with identifier N+1 while it is expecting message with identifier N. In that case we could roll the state of the group back one step and multicast message with identifier N, continue until there are no more messages in the persistant store. Then clients can take over. Replaying some messages from the persistant store comes with decrease in performance.

4 Conclusions

Our implementation of group multicast module works. However, it only works given we made some assumptions about how Erlang environment works and ideal world functionality of erlang monitor. In gms3 we have addressed issues of reliable message sending assuming they will arrive, however in reality we are not guaranteed that messages will arrive.