HW4 : Groupy - A Group Membership Service

[redacted]

October 1, 2024

1 Introduction

This assignment focuses on implementing a **group membership service** with **atomic multicast** and **leader election** in Erlang. Understanding these concepts is crucial for building distributed systems, where processes need to work together in a coordinated and fault-tolerant manner. It's common for nodes to join and leave the network, either intentionally or due to failures. Ensuring that all nodes have a consistent view of the group and can synchronize their state changes, even as nodes crash or are replaced, is essential for maintaining system reliability.

By learning how to manage node failures, elect leaders, and propagate state changes reliably, this assignment helps develop an understanding of fault tolerance and coordination—key challenges in distributed systems that ensure applications remain available and responsive even when parts of the system fail. The system ensures that **the leader is always present**. As long as the leader can manage the group and handle multicast, the system can operate smoothly. If a slave crashes, the group can technically continue functioning, since it's the leader that coordinates multicast and state synchronization.

2 Main problems and solutions

In the network, a node will always be acting as the leader. The leader constantly waits for messages from the nodes in the group, which it broadcasts to the group. The message can either be a message or a new node which wants to join the group. When a node wants to join the group, the list of slaves is updated by appending the existing slaves list with the new slave, which puts the new slave at the end of the list. The group is also updated, which appends the new pid to the existing group list. These changes are made by the leader, and are then broadcasted to all slaves in the list.

The slaves wait for messages from the leader, either these are messages, information of new joining slaves or a ‘DOWN’ message from the leader when it crashes. The slaves do not monitor their peers, only the leader, so any down message would indicate the leader crashing. This triggers an election function that elects a new leader. In the election function, the slave on the top of the list will assign itself as the leader, while all other slaves will also assign this slave as their new leader.

A slave which always monitors its leader, detects that their leader has crashed:

```
        {'DOWN', _Ref, process, Leader, _Reason} ->
            election(Id, Master, MessageNumber, LastMessage, Slaves, Group);
```

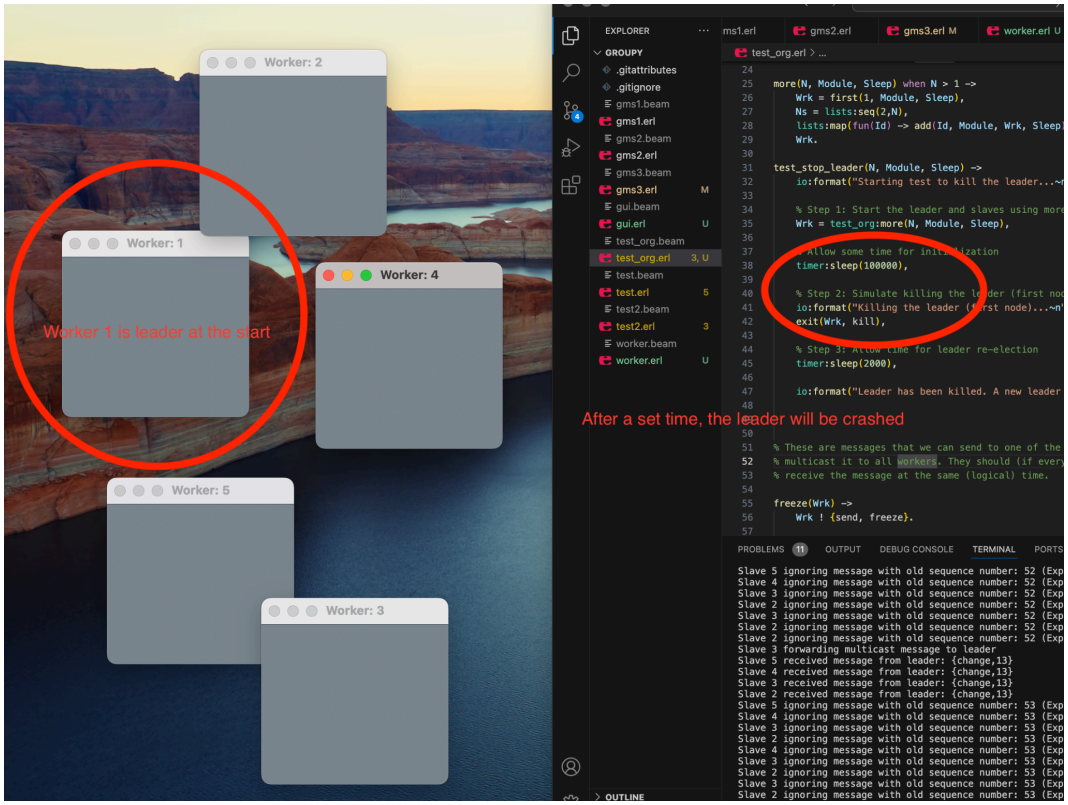
Each slave calls the election process, this happens within the election process:

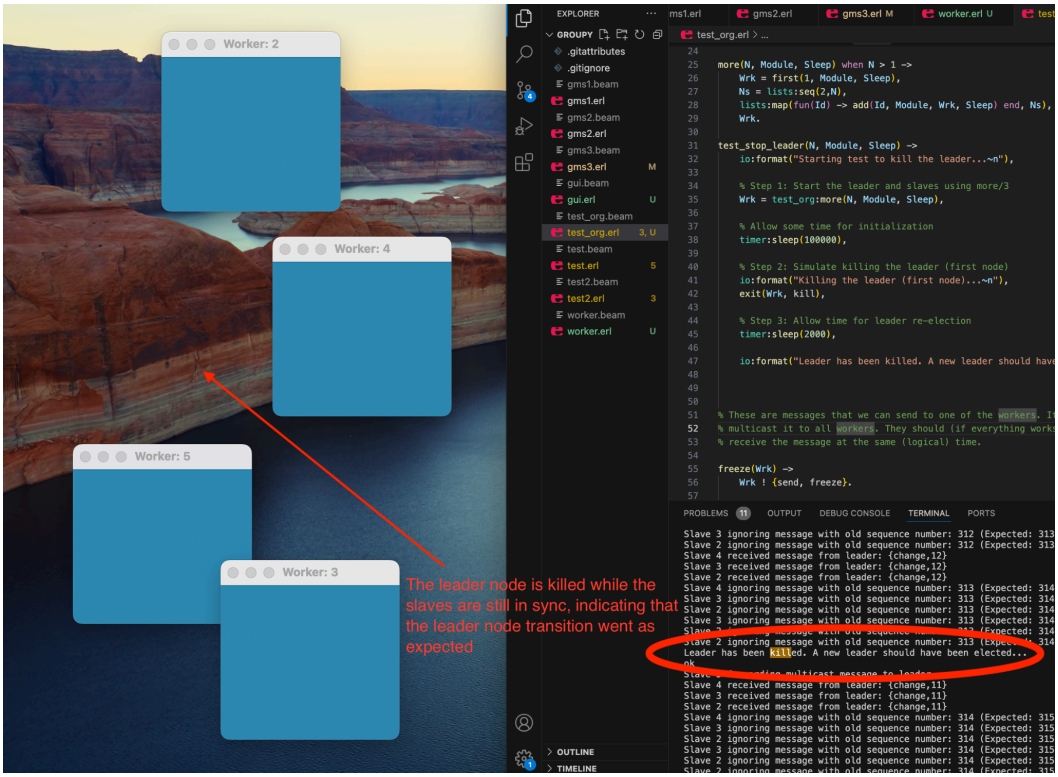
```
        [Self | Rest] ->
            broadcast(Id, {view, Slaves, Group}, Rest),
            Master ! {view, Group},
            leader(Id, Master, MessageNumber + 1, Rest, Group);

        [Leader | Rest] ->
            erlang:monitor(process, Leader),
            slave(Id, Master, MessageNumber, LastMessage, Leader, Rest, Group)
```

For each message being sent (not just in the election process in the code above), received and sent, a counter is being kept so that a node does not handle messages twice. The counter is passed around and kept for each slave, and incremented each time “something happens” i.e. the slave receives a message or sends a message of any sort.

If this message handling and election process work as intended, the slaves should be in synchronized color when we test the gms. The remaining slaves should also stay in sync once the leader crashes and a new slave is elected as the leader. We can see that gms3 handles this, as worker1 (the first leader) crashes, the first slave in the list takes over as leader and the group members stay in sync. See next page.


Worker: 2
Worker: 1
Worker: 4
Worker: 5
Worker: 3
Worker 1 is leader at the start
After a set time, the leader will be crashed


Worker: 2
Worker: 4
Worker: 5
Worker: 3
The leader node is killed while the slaves are still in sync, indicating that the leader node transition went as expected