

# Report 4: groupy

Xiaoying Sun

September 30, 2020

## 1 Introduction

In this assignment, I have implemented a group membership service that provides atomic multicast.

## 2 Main problems and solutions

### 2.1 Problems

- Can we keep a group rolling by adding more nodes as existing nodes die?
- Can we handle the possibly lost messages?
- What if Erlang failure detector is not perfect?
- How could one incorrect node delivers a message that would not be delivered to any correct node?

### 2.2 GUI

Honestly, I'm not familiar with GUI function. The idea of using color to present status from the senior student, Avneesh's code.

## 3 Evaluation

	add nodes	leader crushes	lost messages	detector failure
gms1	✓			
gms2	✓			
gms3	✓	✓		
gms4	✓	✓	✓	✓

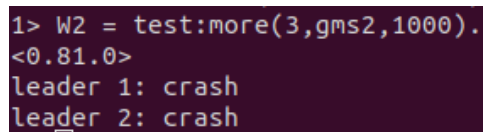
Table 1: Problems solved by different versions of *gms*

### 3.1 gms1

If the leader closed unexpectedly, the entire system would paralyze. No new leader will be selected, all the remaining workers will be **frozen**.

### 3.2 gms2

For *gms2*, as long as Erlang's failure detector detects the leader crashes, it will be closed immediately. So after running for a period of time, there is always **only one worker left**.



```
1> W2 = test:more(3,gms2,1000).  
<0.81.0>  
leader 1: crash  
leader 2: crash
```

Figure 1: Leader crashes in *gms2*

### 3.3 gms4

It's important to underline that the work above only guarantee that messages are delivered in FIFO order, not that they actually do arrive.

```
{msg, I, Msg} when I > N ->  
Ns = lists:seq(N,I),  
io:format("(Slave ~w): Msg with seq ~w lost!~n",[Id,Ns]),  
lists:map(fun(Missing) -> Leader ! {resendMsg, Missing, self()} end, Ns),  
Master ! Msg,  
slave(Id, Master, Leader, I+1, {msg,I,Msg}, Slaves, Group);
```

In order to avoid doublets of messages being received, we numbered all messages and only delivered new messages to the application layer.

- 1) Slave told Leader msg dropped.
- 2) Leader resent msg.
- 3) Slave receives the lost msg.

```

Msg dropped for node <0.99.0>
(Slave 3): Msg with seq [81,82] lost!
(Leader): Resending msgs
(Leader): Resending msgs
(Slave 3): Lost message seq 81 received again
(Slave 3): Lost message seq 82 received again
Msg dropped for node <0.99.0>
(Slave 3): Msg with seq [104,105] lost!
(Leader): Resending msgs
(Leader): Resending msgs
(Slave 3): Lost message seq 104 received again
(Slave 3): Lost message seq 105 received again

```

Figure 2: Leader resending message

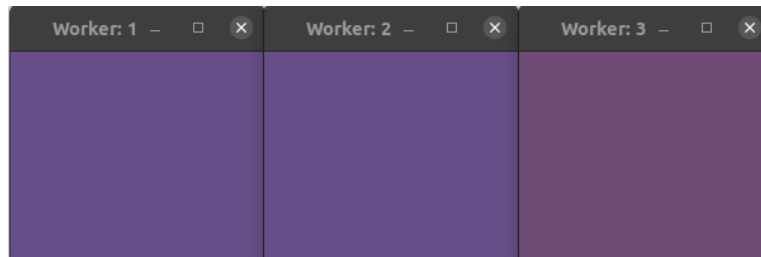


Figure 3: Color of *Slave3* out of sync

Then we shut down *Leader1* on purpose, terminal showed *Slave2* had become the new leader.

```

2: Electing new leader...
3: Electing new leader...
New leader: 2!

```

Figure 4: Electing new leader

Our implementation relies on Erlang's process failure detection, which is not perfect. For example, it may indicate failure for a leader process even though it is alive.

In such cases, the process can use a heartbeat message towards the leader's message and wait for a response. If it does not receive a response within a timeout period, it can assume that the leader process is dead.

## 4 Conclusions

From the above, we successfully implemented a reliable group membership service, which not only provided atomic multicast, but could also tolerate

various kinds of complicated errors.

We also found that when there are many nodes in one group or messages lost happens too frequently, simply resending message is not enough to guarantee sync.

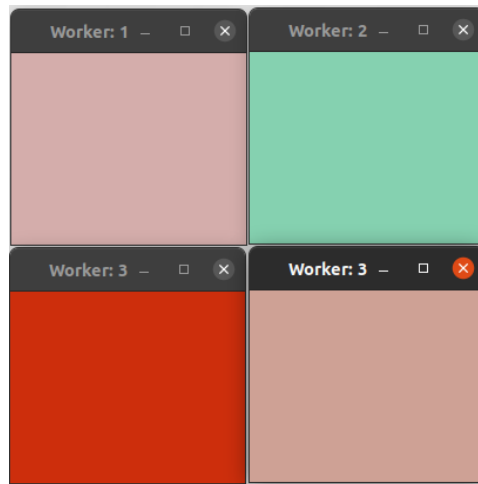


Figure 5: Msg lost too frequently

## 5 Links

- <https://www.kth.se/social/course/ID2201/page/groupy-a-group-membership-se/>
- <http://erlang.org/doc/man/erlang.html>
- <https://github.com/shobhanav/groupy>