Groupy - a group membership service - Report HW4

Malte Berg

October 1, 2025

1 Introduction

In this homework assignment, the primary topics covered has been:

- Coordinating state changes
- Atomic multicasting
- Ensuring functionality, even with crashes

2 Main problems and solutions

The task started by downloading the supplied Erlang modules, containing a worker module, a GUI module, and a test module. These modules were looked over, but left for later as they would only be beneficial once a functioning version of the group membership service was implemented.

The first part of the task that was to be coded was the `gms1` module. The majority of the code was given in the instructions, along with explanations of the different parts of the module. The first revision of the task includes no failure handling, and will only function in the case of no crashes, and no missed messages.

The coded module functions, and a group can be created successfully with multiple peers.

For the `gms2` module, a new function for crash handling is implemented. Monitors are used to register when a leader dies, putting the slaves in a state of election, setting the first peer in the list as the new leader. As the leader can crash, a timeout is added when waiting for an answer from a dead leader to determine that an attempt to join the group through that peer is futile.

This state of the implemented module functions, and is ready to ship.

Randomness is then added to the same module to add the possibility for the leader to crash when trying to multicast a message. This introduces new challenges and shows that it is not ready to ship at all. With a crash, the peers can fall out of sync, due to non-reliable basic multicasting that can miss making sure that the same state is sent to all the peers before the crash. This is visible as the spawned windows shows different colors from each other.

The functions and messages in the module are all redone to include sequence numbers as well as saving the last message sent.

With all functions changed, the implementation functions, and with a created demo file, it is shown that it's possible to keep the group rolling by adding nodes as older nodes die.

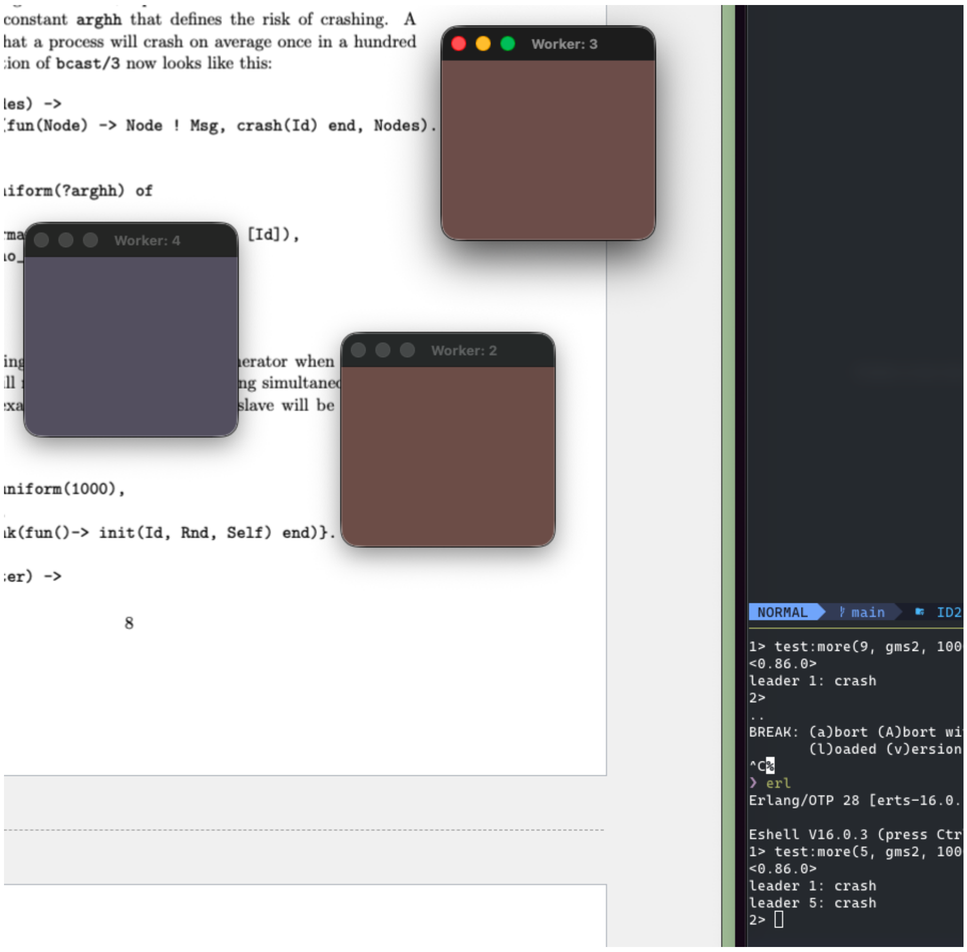
3 Evaluation

The finished program has ensured, in a basic way, that errors such as out-of-sync nodes do not appear, like can be seen in Figure 3.

While the implementation is functional, it is not optimal. The bonus task has not been implemented, meaning that the guarantee of delivery has not been fulfilled.

A way that this could be achieved would be to implement some sort of ACK that is to be delivered back to the sent node, ensuring that the message has been achieved. An example could be to tag the message with an additional value of what node it is sent from, so that the node can be sent back.

Looking at the effect this would have on performance, if these ACKs are to be multicasted as the remaining messages are, this would mean a lot of sent messages among the nodes not meant for the nodes that receive them.



constant **arghh** that defines the risk of crashing. A
hat a process will crash on average once in a hundred
:ion of **bcast/3** now looks like this:

```
les) ->
fun(Node) -> Node ! Msg, crash(Id) end, Nodes).

iiform(?arghh) of

ma                  [Id]),
io_

iniform(1000),

ik(fun()-> init(Id, Rnd, Self) end)}.

:er) ->
```

ing   erator when
ll   ng simultaneo
xa   slave will be

8

```
1> test:more(9, gms2, 100
<0.86.0>
leader 1: crash
2>
..
BREAK: (a)bort (A)bort wi
       (l)oaded (v)ersion
^C%
❯ erl
Erlang/OTP 28 [erts-16.0.

Eshell V16.0.3 (press Ctr
1> test:more(5, gms2, 100
<0.86.0>
leader 1: crash
leader 5: crash
2>
```

Figure 1: An example of gms2 becoming out of sync.