Report : Groupy: a group membership service

Zhanbo Cui

October 6, 2021

1 Introduction

In this assignment, I implemented a small group service which involves the concept of group communication, leader election and failure detection in distributed systems. In this report, I will explain the principle and key implementation of this system. Finally, the issues mentioned in optional task also be discussed.

2 Main problems and solutions

2.1 Groupy architecture

There are many nodes in the group, each of them contain application layer, group layer and network layer. The *application layer processes* are responsible for sending or receiving state change message from *group layer processes*, meanwhile it also maintains a GUI. As for the *group layer processes*, they are responsible for forwarding message to other *group layer processes.* There are two different roles in the system: *leader node* and *slave node*, *slave node* can only communicate with *leader node*, and *leader node* can multicast message to all nodes in the group. Finally, the Erlang virtual machine can help us complete the work of the network layer fortunately.

2.2 Handling failure

As mentioned in the tutorial, all nodes use the Erlang built in support to detect and report that processes have crashed.

```
init(Id, Group, Master, Rnd)->
    ...
 %% this node begain to monitor the Leader
```

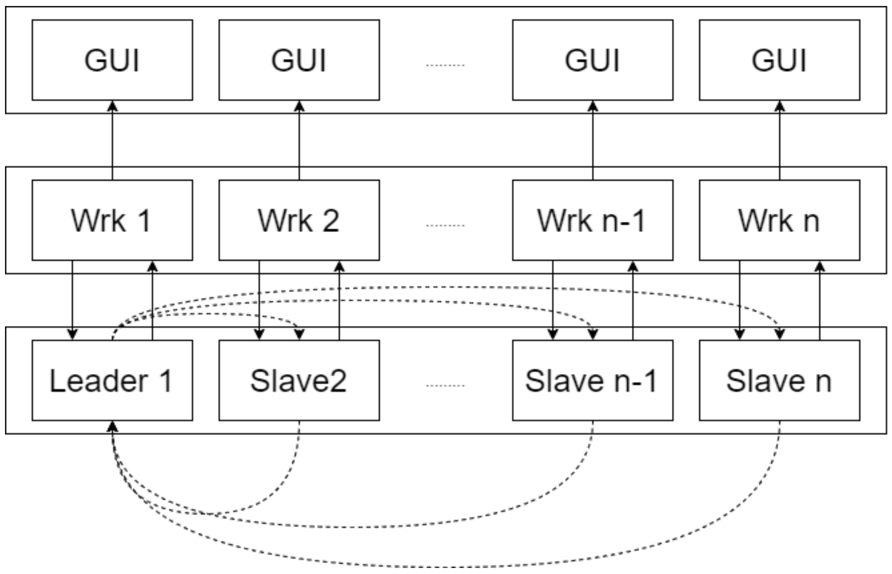


Figure 1: Groupy architecture

```
    erlang:monitor(process, Leader),
%% enter receive mode
    slave(Id, Master, Leader, N+1, Last, NewSlaves, NewGroup);
    ...
```

When a leader crashed(receive the DOWN message), the first node in the ordered slave list is selected as the new leader and multicast the new view to all slave nodes, then the *group layer processes* will deliver the new view to *application layer processes.*

2.3 Reliable multicast

If a leader crashes while doing a multicast, only some of the nodes will receive the message they all should have received. For more reliable multicast, the new leader need to re-multicast the message to all nodes in the group. Old leader send the message by following the order of node in the list, so the new leader as the second one in the list defiantly has received the message. New leader only need to multicast the last message to all the nodes when it was appointed to leader. Meanwhile, each message is attached with a serial number, so nodes don't care the message which had been received.

```
election(Id, Master, N, Last, Slaves, [_|NewGroup])->
    ...
```

```
%%if the first candidate is itself
  [Self|Rest]->
    %%resend the last message to all nodes in the group
      bcast(Id, Last, Rest),
          ...
    %%update the sequence number
      leader(Id, Master, N+1, Rest, NewGroup);
  ...
```

2.4 Optional task

We design system under the condition that message is not lost when forwarding between the nodes. It depends on the Erlang system to guarantee all of them. However, in a real world environment, messages can be lost in delivery.

2.4.1 Strong synchronization

We can ask an acknowledgement when the message delivered between nodes or even between *application layer processes* and *group layer processes.* If the *ack* message is not received over timeout, the message will be resent.

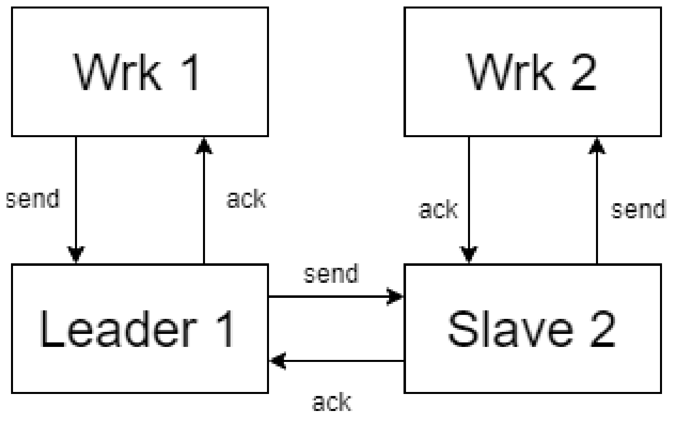


Figure 2: Strong synchronization

We can use this method during the connection establishment phase, because a lost message during connection establishment will cause the connection to fail.

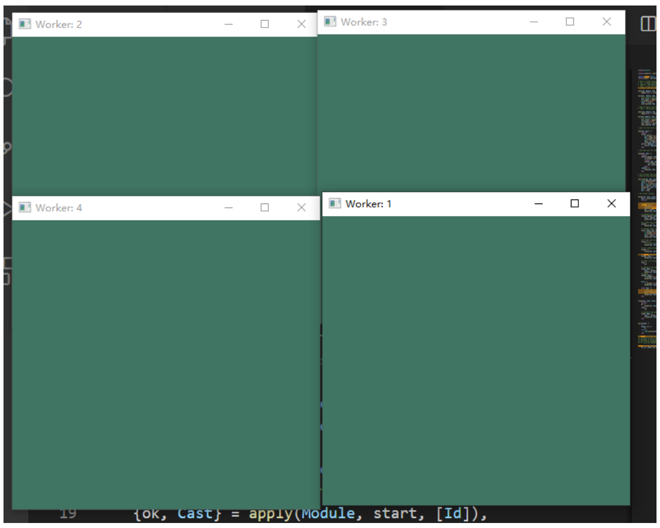
2.4.2 Weak synchronization

If use *Strong synchronization* during the normal message exchange phase ensures that each node receives a consistent history of messages. But I don't think it's necessary, because we won't always receive no message, we just need to make sure that we receive new information as much as possible. In previous pattern matching in system, node receive the the message which sequence number is equal to the sequence number kept by node and than update the sequence number. However, if some message lost, the node will receive a message with a larger sequence number, at which point node should receive this new message and update its sequence number based on the sequence number of this message.

```
%%old message
    {msg, I, _} when I < N->
        slave(Id, Master, Leader, N, Last, Slaves, Group);
%%new message
    {msg, I, Msg} ->
        ...
    slave(Id, Master, Leader, I + 1, NewLast, Slaves, Group);
```

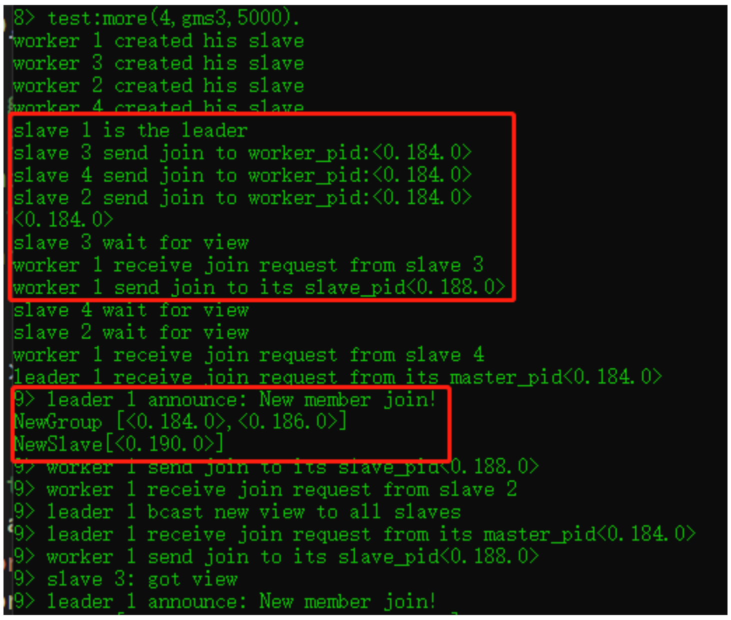
3 Evaluation

We start with four nodes and output their connection establishment and message passing process as far as possible.



```
{ok, Cast} = apply(Module, start, [Id]),
```

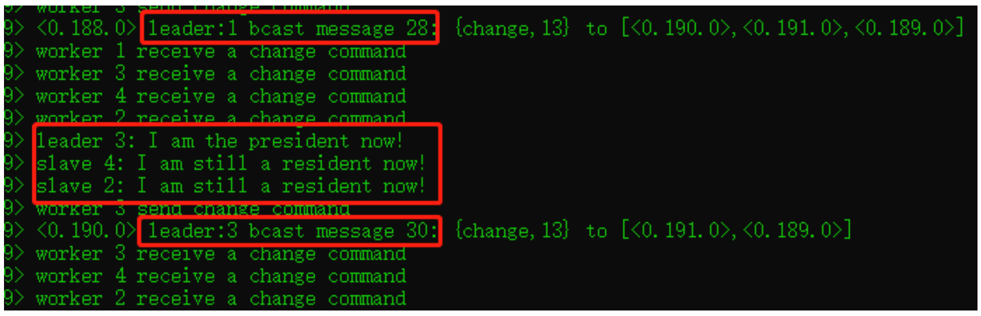
Figure 3: Nodes boost



```
8> test:more(4,gms3,5000).
worker 1 created his slave
worker 3 created his slave
worker 2 created his slave
worker 4 created his slave
slave 1 is the leader
slave 3 send join to worker_pid:<0.184.0>
slave 4 send join to worker_pid:<0.184.0>
slave 2 send join to worker_pid:<0.184.0>
<0.184.0>
slave 3 wait for view
worker 1 receive join request from slave 3
worker 1 send join to its slave_pid<0.188.0>
slave 4 wait for view
slave 2 wait for view
worker 1 receive join request from slave 4
leader 1 receive join request from its master_pid<0.184.0>
9> leader 1 announce: New member join!
NewGroup [<0.184.0>,<0.186.0>]
NewSlave[<0.190.0>]
9> worker 1 send join to its slave_pid<0.188.0>
9> worker 1 receive join request from slave 2
9> leader 1 bcast new view to all slaves
9> leader 1 receive join request from its master_pid<0.184.0>
9> worker 1 send join to its slave_pid<0.188.0>
9> slave 3: got view
9> leader 1 announce: New member join!
```

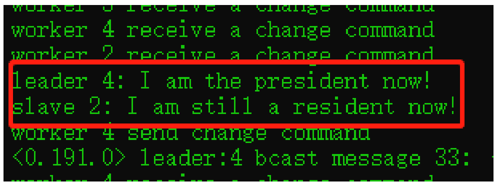
Figure 4: How a node join a existing group

Close the node and output the election process. The previous leader is node 1, now node 3 is elected as a leader. Node 3 Begin to ”bcast”.



```
9> worker 3 send change command
9> <0.188.0> leader:1 bcast message 28: {change,13} to [<0.190.0>,<0.191.0>,<0.189.0>]
9> worker 1 receive a change command
9> worker 3 receive a change command
9> worker 4 receive a change command
9> worker 2 receive a change command
9> leader 3: I am the president now!
9> slave 4: I am still a resident now!
9> slave 2: I am still a resident now!
9> worker 3 send change command
9> <0.190.0> leader:3 bcast message 30: {change,13} to [<0.191.0>,<0.189.0>]
9> worker 3 receive a change command
9> worker 4 receive a change command
9> worker 2 receive a change command
```

Figure 5: Election



```
worker 3 receive a change command
worker 4 receive a change command
worker 2 receive a change command
leader 4: I am the president now!
slave 2: I am still a resident now!
worker 4 send change command
<0.191.0> leader:4 bcast message 33:
worker 4 receive a change command
```

Figure 6: Another election

4 Conclusion

Through this assignment we can see that with a feasible election method we can ensure that a system based on group communication will work in the event of some errors. But group communication is a kind of indirect communication, it is very dependent on the message, so it is necessary to devise some mechanisms to ensure that message can be delivered correctly.