

# Seminar Report: Groupy

May 1, 2019

**Carles Tornel**  
**Jesus Alfaro**  
**Ricard Abril**

## 1 Introduction

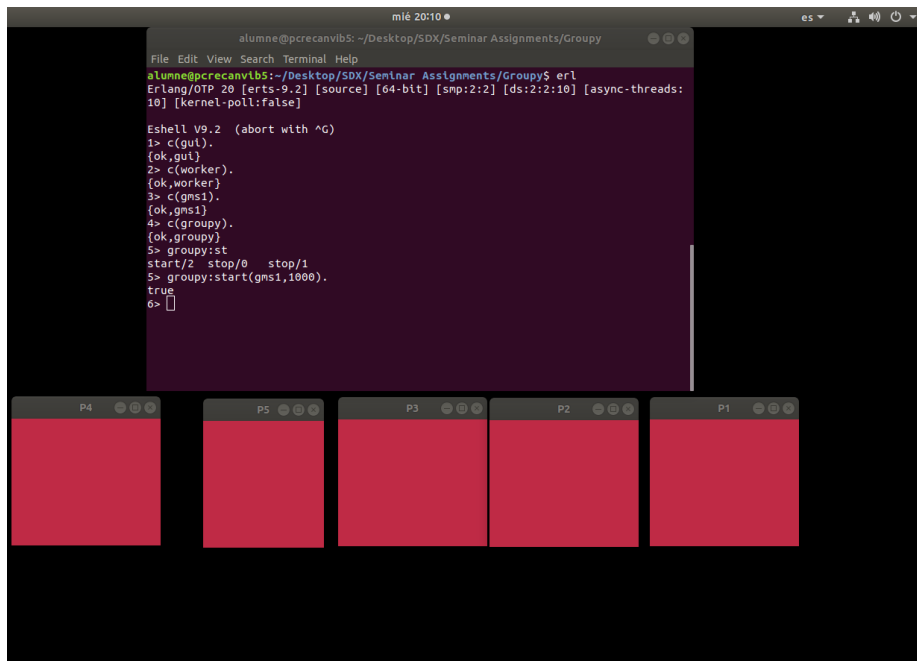
En aquesta pràctica haurem d'implementar un servei multicast on un únic procés serà el líder del sistema i al qual els altres processos enviaran un missatge cada cop que vulguin enviar missatges multicast. La presencia del líder fa que el sistema tingui un ordre, també serà el que tingui constància dels processos que estan actius al sistema i l'únic que fa el multicast.

## 2 Experiments

### 2.1 First Implementation

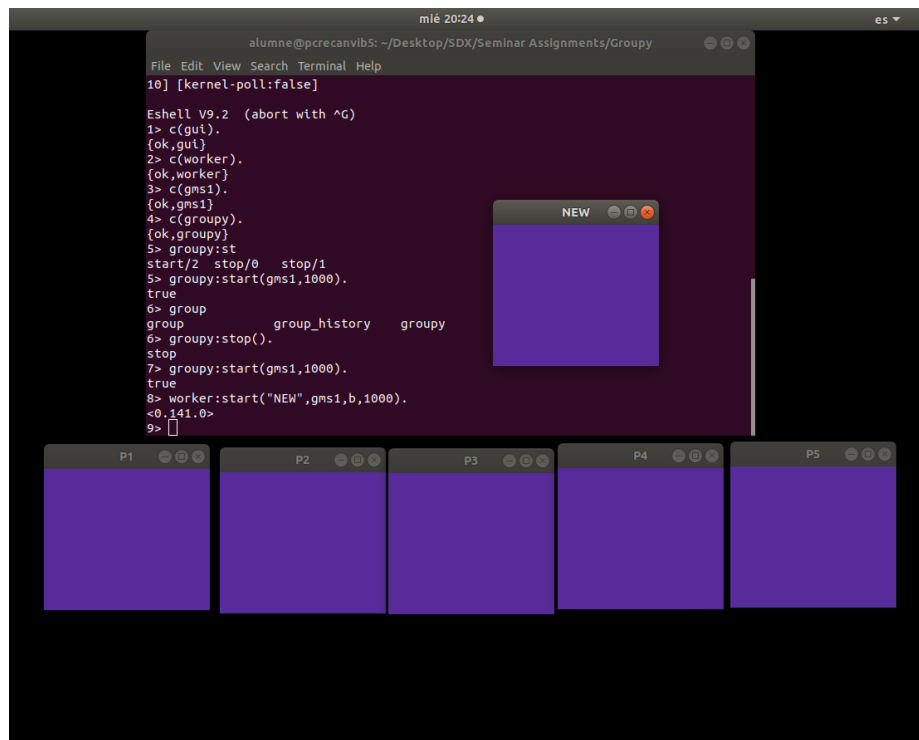
1. *Do some experiments to see that you can create a group, add some peers and keep their state coordinated.*

Crearem 4 esclaus i un líder, que mitjançant comunicació multicast es comunicaran, en aquesta comunicació, utilitzarem FIFO i Total Order, la seqüencialització del líder i el tipus de comunicació que utilitzarem garantirà que l'estat es preservi en tots el grup. Tal com podem veure a la figura següent:



Tal com observem a la figura, podem crear un grup i preservar el seu estat.

El segon objectiu d'aquest experiment és el de poder afegir nous nodes, que estiguin coordinats amb la resta. Per a tal fi, crearem un nou worker, que farà una request to join a un Slave, aquest reencaminara la petició al Lider del grup i so no hi ha cap fallada, el nou node passarà a formar part del grup com a Slave. Tal com podem veure a la figura següent:



- Per adaptar Groupy per tal de tenir cada worker en una instància erlang diferent, hem creat el fitxer groupyrem.erl, on tenim la implementació necessària per dur-ho a terme.

Activities **alunne** File Edit View Search Terminal Help

```

Tr0.0.0.1 -setcookie secret
Erlang/OTP 20 [erts-9.2] [source] [64-bit]
[smp:2:2] [ds2:2:10] [async-threads:
10] [kernel-poll:false]

sEshell V9.2. (abort with ^C)
(node@127.0.0.1)>

```

P1

P2

P3

P5

P4

alunne@pcrecavib5: ~/Desktop/SDX/Seminar

File Edit View Search Terminal Help

```

aLunne@pcrecavib5:~$ cd Desktop/SDX/Seminar\ Assignments/Group5/
alunne@pcrecavib5:~/Desktop/SDX/Seminar Assignments/Group5$ erl -name node@127.0.0.1 -setcookie secret
Erlang/OTP 20 [erts-9.2] [source] [64-bit] [smp:2:2] [ds2:2:10] [async-threads:10] [kernel-poll:false]

Eshell V9.2. (abort with ^C)
(node@127.0.0.1)>

```

alunne@pcrecavib5: ~/Desktop/SDX/Seminar Assignments/Group5

File Edit View Search Terminal Help

```

aLunne@pcrecavib5:~$ cd Desktop/SDX/Seminar\ Assignments/Group5/
alunne@pcrecavib5:~/Desktop/SDX/Seminar Assignments/Group5$ erl -name node@127.0.0.1 -setcookie secret
Erlang/OTP 20 [erts-9.2] [source] [64-bit] [smp:2:2] [ds2:2:10] [async-threads:10] [kernel-poll:false]

Eshell V9.2. (abort with ^C)
(node@127.0.0.1)> c(gul).
{ok,gul}
(node@127.0.0.1)> c(worker).
{ok,worker}
(node@127.0.0.1)> c(gns1).
{ok,gns1}
(node@127.0.0.1)> c(groupyren).
{ok,groupyren}
(node@127.0.0.1)> groupyren:start(gns1,1000).
14670.71.0x
(node@127.0.0.1)>

```

alunne@pcrecavib5: ~/Desktop/SDX/Seminar Assignments/Group5

File Edit View Search Terminal Help

```

aLunne@pcrecavib5:~$ cd Desktop/SDX/Seminar\ Assignments/Group5/
alunne@pcrecavib5:~/Desktop/SDX/Seminar Assignments/Group5$ erl -name node@127.0.0.1 -setcookie secret
Erlang/OTP 20 [erts-9.2] [source] [64-bit] [smp:2:2] [ds2:2:10] [async-threads:10] [kernel-poll:false]

Eshell V9.2. (abort with ^C)
(node@127.0.0.1)>

```

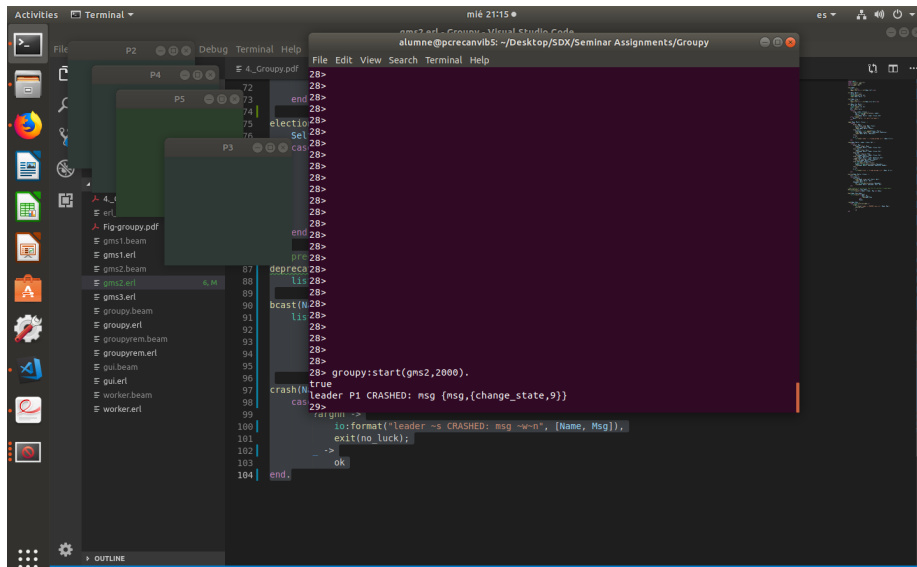
## 2.2 Handling Failures

### 2.3 Failure Detector

1. *Do some experiments to see if the peers can keep their state coordinated even if nodes crash.*

Ara el nostre sistema distribuït hauria de ser capaç de suportar fallades. Per provar-ho, experimentarem amb les diferents parts crítiques, com per exemple quan un Líder cau i s'ha de seleccionar un de nou.

Erlang ens permet monitorar processos, per la qual cosa quan els Slaves detecten que el Líder ha caigut, entraran en un estat d'elecció, del qual el primer Slave de la cua serà escollit com a nou Líder. En cas que els Slaves no hagin detectat que el Líder ha caigut, el nou Líder enviarà un multicast amb la nova vista. Tal com podem veure en la figura següent:



```
File Edit View Search Terminal Help
groupy_ert.erl - Groupy - Visual Studio Code
alume@pcprecanib5: ~/Desktop/SDX/Seminar Assignments/Groupy

P4 72
P5 73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104

end.

groupy: start(gms2, 2000).
true
leader P1 CRASHED: msg {msg, {change_state, 9}}
cas
29>
    iio:format("leader -s CRASHED: msg {msg, {Name, Msg}}",
    exit(no_luck);
    ->
    ok;
end.
```

Tal com es veu, el crash de Líder, no afecta la coordinació del grup.

Així i tot, segueix existint una manera de què la coordinació del grup es pugui veure afectada, aquesta situació es donarà quan un Líder cau mentre està fent multicast, en elegir a un nou Líder, els Slaves poden quedar descoordinaats.

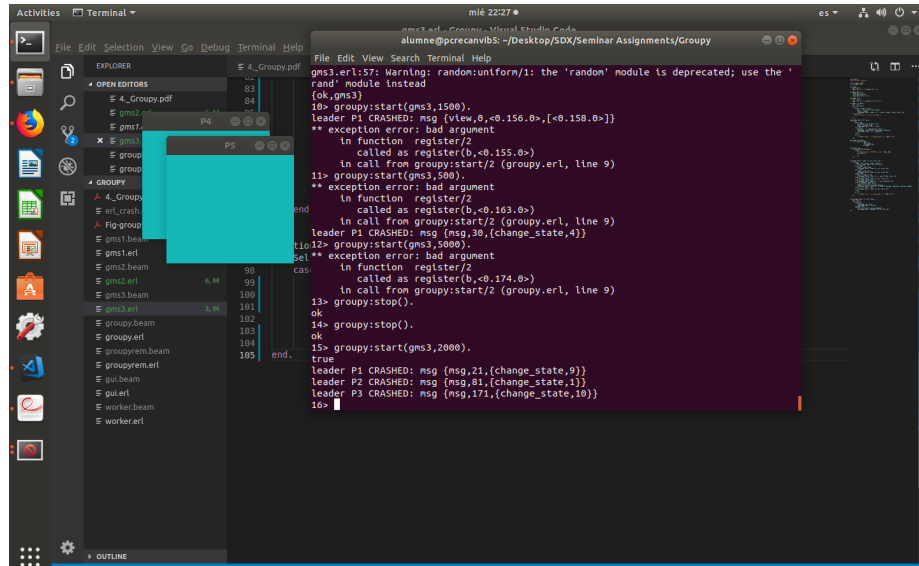
Per altra banda, si el que cau és un Slave, no hauríem de tenir cap problema de coordinació, ja que simplement aquest no rebirà els missatges que li siguin enviats i tampoc enviarà cap missatge al Líder.



## 2.5 Reliable Multicast

1. Repeat the experiments to see if now the peers can keep their state coordinated even if nodes crash.

Ara el nostre sistema sí que serà completament tolerant a fallades, ja que no existeix la possibilitat que a l'elegir un nou Lider es descoordini el grup. Tal com podem observar ala figura següent:



The screenshot shows a terminal window with the following content:

```
gms3.erl:57: Warning: random:uniform/1: the 'random' module is deprecated; use the 'rand' module instead
10> group:start(gms3,1500).
{ok,gms3}
leader P1 CRASHED: msg {view,0,<0.156.0>,<0.158.0>}
** exception error: bad argument
   in function register/2
   called as register(b,<0.155.0>)
   in call from group:start/2 (groupy.erl, line 9)
11> group:start(gms3,500).
** exception error: bad argument
   in function register/2
   called as register(b,<0.163.0>)
   in call from group:start/2 (groupy.erl, line 9)
leader P1 CRASHED: msg {msg,30,[change_state,4]}
12> group:start(gms3,5000).
** exception error: bad argument
   in function register/2
   called as register(b,<0.174.0>)
   in call from group:start/2 (groupy.erl, line 9)
13> group:stop().
ok
14> group:stop().
ok
15> group:start(gms3,2000).
true
leader P1 CRASHED: msg {msg,21,[change_state,9]}
leader P2 CRASHED: msg {msg,81,[change_state,1]}
leader P3 CRASHED: msg {msg,171,[change_state,10]}
16>
```

2. Try to keep a group rolling by adding more nodes as existing nodes die. En aquest experiment fem que quan el Lider cau, afegim un nou node al grup, de forma que mai ens quedarem amb un grup d'un sol worker.

### 3 Open Questions

1. *Why do the workers desynchronize?*

Això el causa la fallada del leader abans d'acabar d'enviar els missatges multicast, ja que només una porció dels slaves rebran aquest últim missatge, donant lloc a que el worker respectiu no tindrà disponible el canvi de vista/color correcte. I quedarà dessincronitzat.

2. *How would we have to change the implementation to handle the possibly lost messages?*

Per resoldre el problema dels missatges perduts, haurem d'utilitzar algun mecanisme de reliable multicast. Com basic reliable multicast, enviant ACK al líder per tal de mantenir un control dels missatges enviats i així que el líder no canvisi d'estat fins que rebi tots els ACK corresponents, d'aquesta manera minimitzariem la dessincronització a un únic color. En el cas de canvi de líder, no podem estar segurs que el següent de la llista, per convertir-se en líder, hagués rebut tots els missatges i tornar a reenviar als slaves en cas necessari. Per aquest motiu una possible solució seria implementar un procés d'elecció que prioritzés nodes que tinguin disponibles missatges amb el màxim número de seqüència, per tal que el reliable multicast funcioni.



3. *How would this impact performance?*

La nostra solució de basic reliable multicast haurà de doblar el nombre de missatges, ja que cada slave haurà d'enviar un ACK, a més el líder haurà d'esperar un RTT sencer en cas de no rebre tots el ACK i fer el canvi d'estat.

4. *What would happen if we wrongly suspect the leader to have crashed?*

Amb la implementació del sistema actual, ja tindrem present un nou Leader. Els Slaves envien peticions multicast a aquest nou Leader però l'antic continuarà gestionant les peticions de canvi d'estat del seu worker, amb qui continuarà fent multicast.

La resta de nodes del sistema (Leader + Slaves) processarà aquests canvis d'estat ja que no es pot saber que el missatge de canvi d'estat sigui emès pel Leader. Els Workers estaran sincronitzats ja que rebran els mateixos missatges. L'únic que no funcionarà d'acord a la nova implementació és el Worker associat a l'anterior Leader. Aquest no hi serà a la llista de la resta i només respondrà a les seves propies peticions.

## 4 Personal opinion

Aquest laboratori ha ajudat a clarificar un dels conceptes més comuns dels sistemes distribuïts en xarxa: La comunicació multicast. Mitjançant la pràctica de l'implementació de nodes com poden ser els Workers, Leaders i Slaves, es té una visió més exemplificadora del problema que s'ha tractat.