

# Seminar Report: Groupy

Marc Ortiz  
Omair Iqbal  
V ctor Gonz lez

April 26, 2017

## 1 Introduction

Aquesta practica consisteix en implementar un servei multicast per a tenir processos amb un estat coordinat. L'arquitectura consisteix en un conjunt de processos on només un d'ells és seleccionat com a Leader, i quan un dels nodes vol fer un multicast s'envia un missatge al Leader (el qual tindr  una visió dels processos actuals en up del sistema) i que s'encarregara de fer el multicast.

## 2 Work done

Els fitxers font es troben a la carpeta codes i les imatges la carpeta figures per si no es veuen prou b .

## 3 Experiments

### 3.1 The first implementation

En la primera implementaci  b sica (gms1) es comen a amb un node i es van afegint m s, no es tenen en compte les fallades.

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

En una mateixa m quina s'han creat 4 slaves i un Leader, que es comuniquen mitjan ant multicast FIFO + total order gr cies a la seq encialitzaci  del Leader que fa que l'estat de grup (color) es preservi.

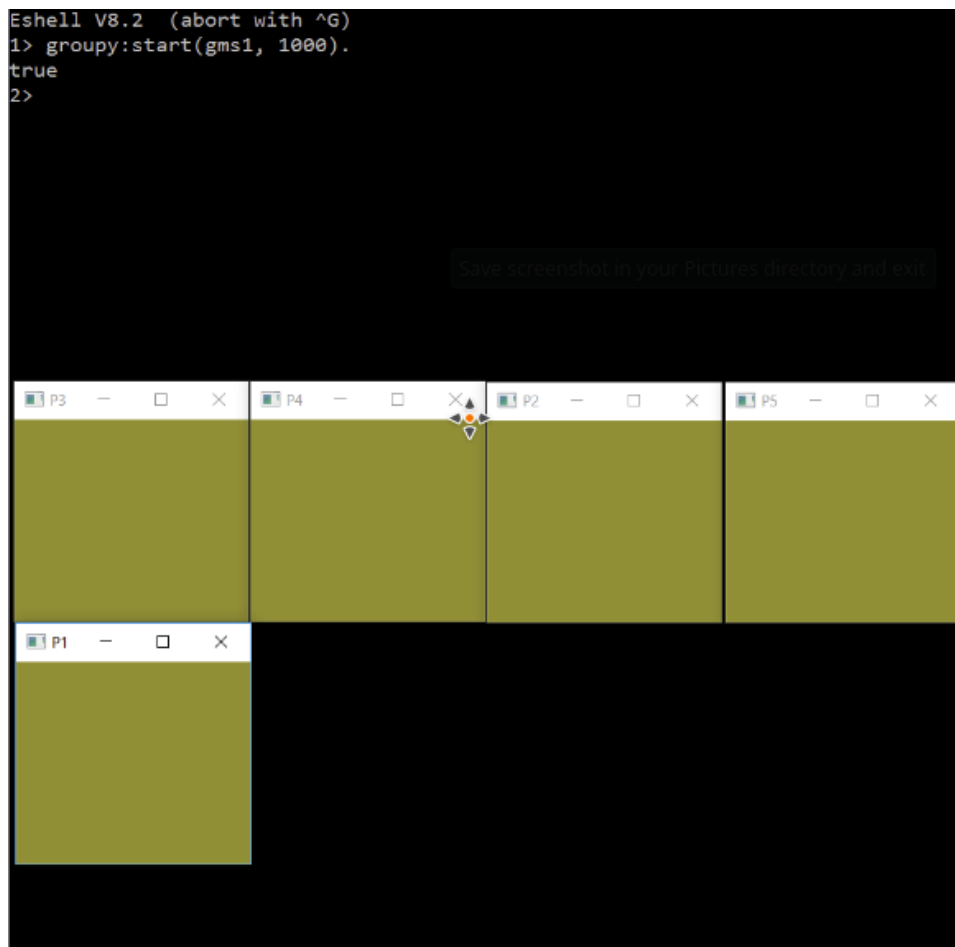


Figure 1: 4 slaves i 1 leader amb l'estat coordinat (mateix color)

L'objectiu d'aquesta primera implemetació es afegir nous nodes al grup i mantenir-los coordinats amb els altres. Per tant crearem un worker que pregunti per exemple al membre del grup "P2" per accedi-hi, P2 redireccionarà la petició (request to join) al Leader del grup i si no es perden missatges pel camí, el nou node formara part del grup i començara l'estat com a Slave. El mòdul gms1 assegura també que el node afegit estarà sincronitzat gràcies a una request del color del grup.

Podem veure aquí el resultat:

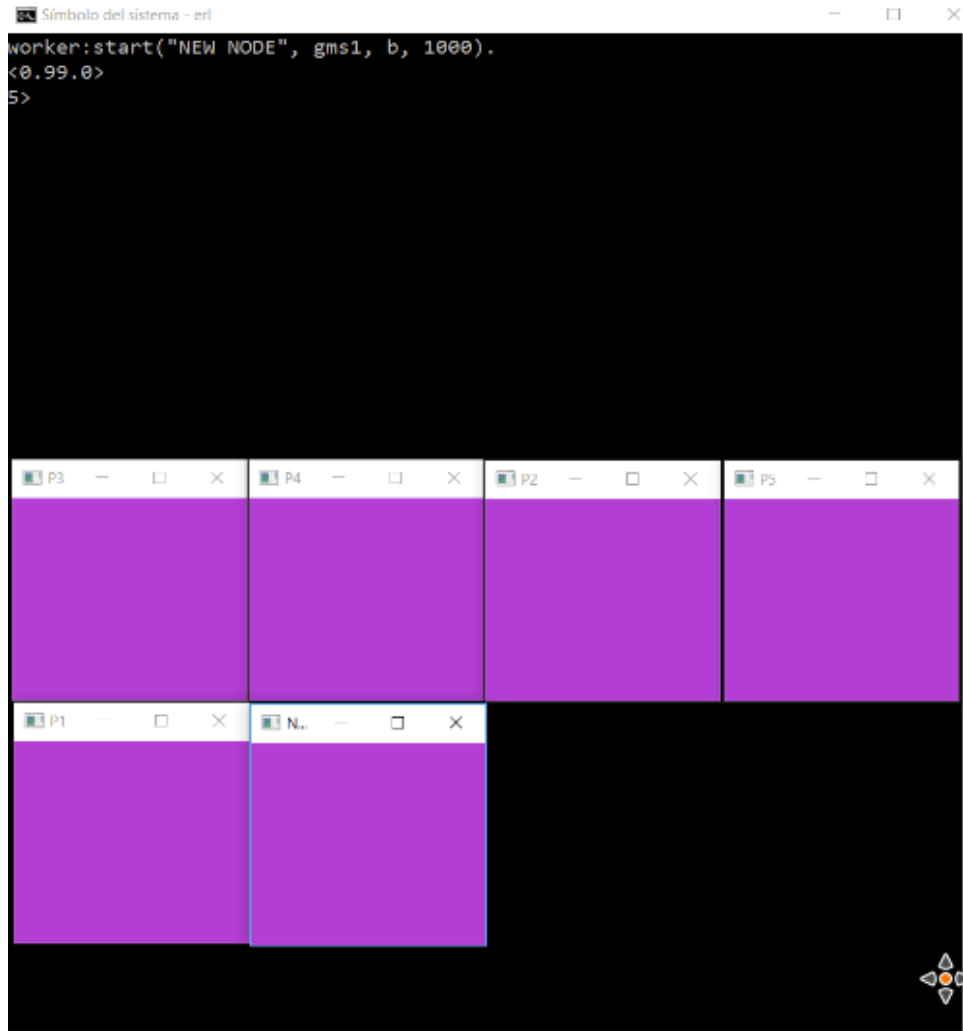


Figure 2: Nou node afegit.

2. *Adapt the groupy module to enable each worker to run in different machines*

Hem adaptat el codi del mòdul groupy per a que els nous nodes puguin executar-se en diferents màquines. Aquest codi es troba en el fitxer `groupy_remote.erl`

Execució d'atomic multicast amb possibilitat dels nodes per a ser executats des de diferents nodes:

```
(a@127.0.0.1)6> Nodes = ['a@127.0.0.1','a@127.0.0.1','a@127.0.0.1','a@127.0.0.1',
['a@127.0.0.1','a@127.0.0.1','a@127.0.0.1','a@127.0.0.1',
'a@127.0.0.1']
(a@127.0.0.1)7> groupy_remote:start(gms1,1000,Nodes).
<0.95.0>
(a@127.0.0.1)8>
```

Figure 3: Execució dels nodes en diferents màquines

Per simplicitat en el nostre exemple de la Figure 3 els nodes s’han executat en la mateixa màquina però modificant el paràmetre "Nodes" que permet l’execució dels processos en diferents màquines.

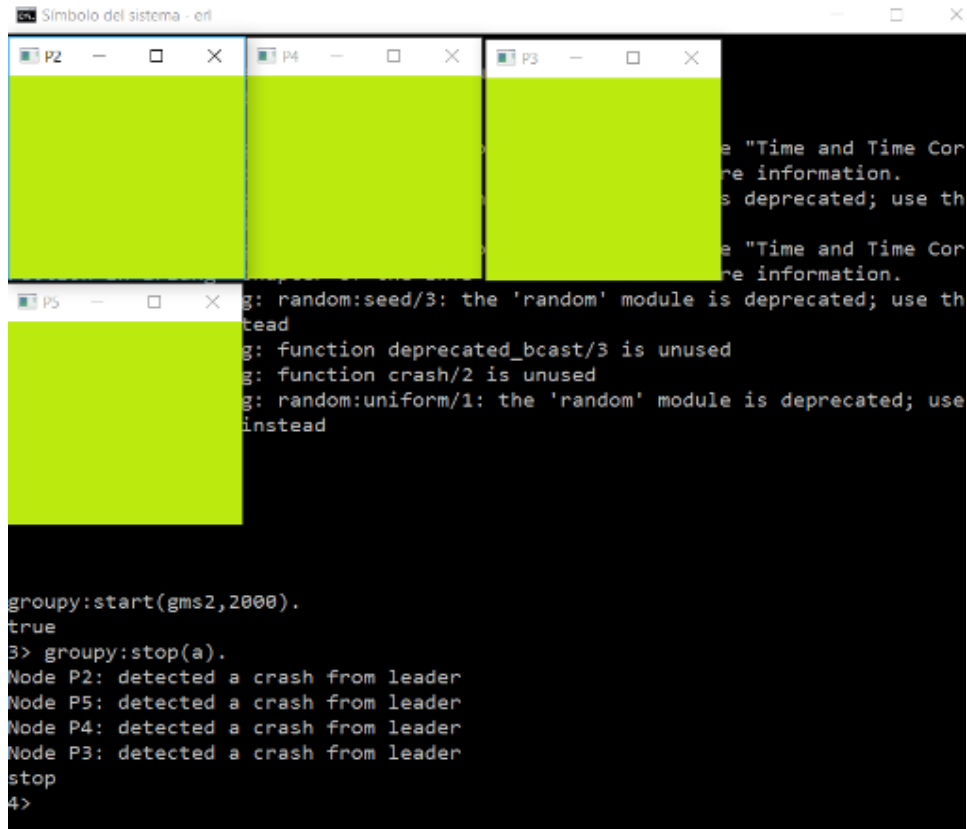
## 3.2 Handling failures

### 3.2.1 Failure detectors

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

En aquest segon experiment es vol observar la capacitat de suportar fallades del nostre sistema distribuït. Comencen experimentant la part més crítica que es quan un Leader del grup falla i s'ha de seleccionar un de nou. Gràcies a que Erlang ens permet monitoritzar processos, farem que els slaves monitoritzin al Leader i si veuen que ha caigut llavors entraran en un estat d'elecció on el primer node de la cua dels slaves sera escollit com a nou Leader. El nou Leader (si els slaves no se n'adonen que el antic Leader ha caigut) enviarà un multicast amb la nova vista (nou Leader i nous Slaves).

En la següent figura veiem l'execució d'aquest mòdul (gms2) i el comportament quan el Leader cau:



```
Símbolo del sistema - erl
P2 P4 P3
P5
g: random:seed/3: the 'random' module is deprecated; use th
thead
g: function deprecated_bcast/3 is unused
g: function crash/2 is unused
g: random:uniform/1: the 'random' module is deprecated; use
instead

groupy:start(gms2,2000).
true
3> groupy:stop(a).
Node P2: detected a crash from leader
Node P5: detected a crash from leader
Node P4: detected a crash from leader
Node P3: detected a crash from leader
stop
4>
```

Figure 4: Just després de la caiguda del Leader

Hem programat també un output en el programa com a referència de quins nodes han detectat el crash del Leader com a curiositat, tal i com es pot veure en la Figure 4.

Segons la nostra execució, el crash del Leader sembla que no influeix en la descoordinació del grup. Tot i així hem de reflexionar que en aquesta versió si tenim la mala sort de que el Leader fa crash mentre està fent un multicast del missatges, els processos si que poden quedar descoordinats quan s'escolleixi un nou Leader, i s'ha d'anar molt en compte amb aquesta execució.

Per veure-ho de manera mes clara, la següent figura mostra un exemple:

```
1>(gms2).
gms2
2> c(gms2).
gms2.erl:11: Warning: function deprecated_bcast/3 is unused
gms2.erl:12: Warning: random:uniform/1: the 'random' module is deprecated; use the 'rand' module instead
gms2.erl:20: Warning: function deprecated_bcast/3 is unused
gms2.erl:21: Warning: random:uniform/1: the 'random' module is deprecated; use the 'rand' module instead
gms2.erl:87: Warning: function deprecated_bcast/3 is unused
gms2.erl:98: Warning: random:uniform/1: the 'random' module is deprecated; use the 'rand' module instead
{ok,gms2}
3>

groupy:start(gms2,5000).
true
4> leader P1 CRASHED: msg {msg,{change_state,4}}
4> Node P2: detected a crash from leader
4> Node P5: detected a crash from leader
4> Node P4: detected a crash from leader
4> Node P3: detected a crash from leader
4> leader P2 CRASHED: msg {msg,{change_state,13}}
4> Node P5: detected a crash from leader
4> Node P3: detected a crash from leader
4> Node P4: detected a crash from leader
4>
```

Figure 5: Els Slaves queden descoordinats

Finalment també volem veure l'efecte del crash d'un Slave en el nostre sistema distribuït. Per simplicitat, suposarem que si un Slave fa crash, no cal modificar la vista i també que ho fa per sempre.

El crash del Slave per tant no hauria de ser un gran problema, simplement no rebrà els missatges que se li envien ni tampoc enviara missatges no influint així en la coordinació del grup.



Figure 6: Stop d'un slave

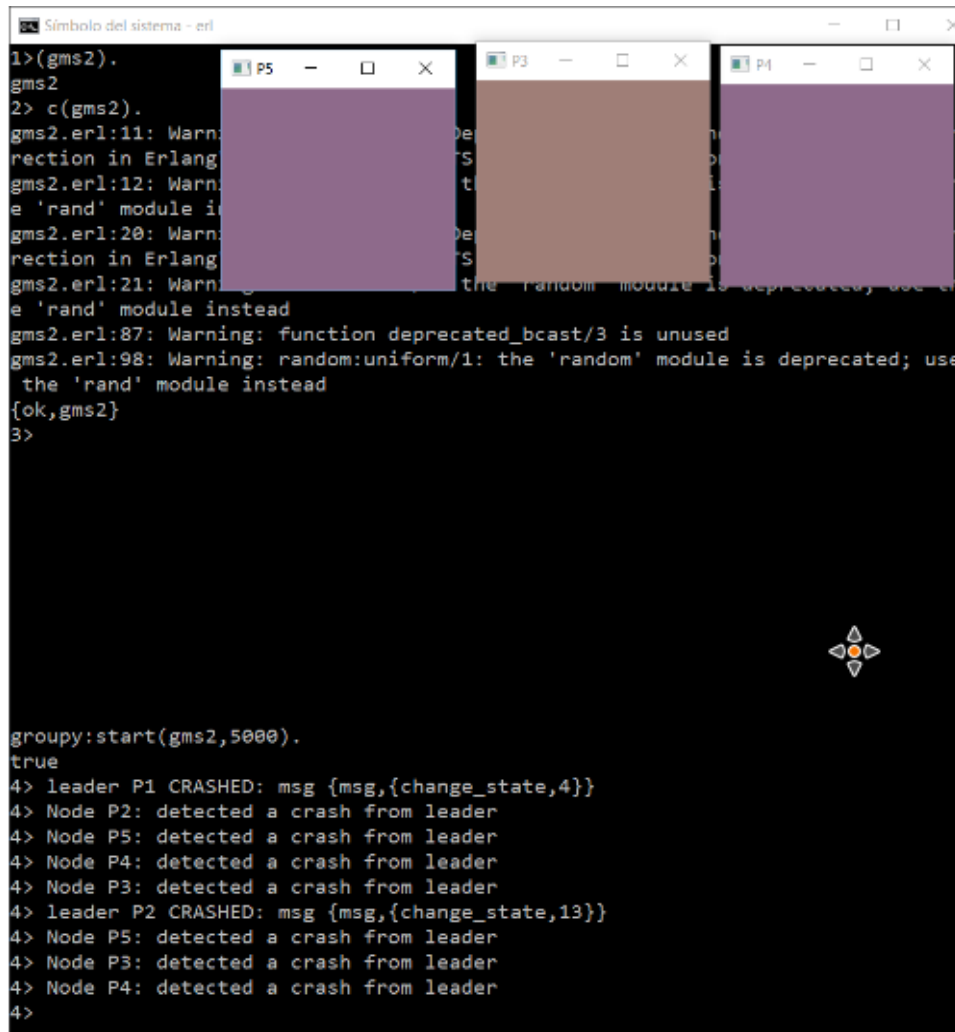
Obtenim com a conclusió que si volem un sistema coordinat en tot moment, el nostre sistema actualment no és tolerant a fallades.



### 3.2.2 Missing messages

1. Repeat the experiments and see if you can have the state of the workers become out of synch

Execució amb probabilitat de crash cada missatge:



```
1>(gms2).
gms2
2> c(gms2).
gms2.erl:11: Warning: Deprecation in Erlang: function deprecated_bcast/3 is unused
gms2.erl:12: Warning: Deprecation in Erlang: function deprecated_bcast/3 is unused
gms2.erl:20: Warning: Deprecation in Erlang: function deprecated_bcast/3 is unused
gms2.erl:21: Warning: Deprecation in Erlang: function deprecated_bcast/3 is unused
gms2.erl:87: Warning: function deprecated_bcast/3 is unused
gms2.erl:98: Warning: random:uniform/1: the 'random' module is deprecated; use the 'rand' module instead
{ok,gms2}
3>

groupy:start(gms2,5000).
true
4> leader P1 CRASHED: msg {msg,{change_state,4}}
4> Node P2: detected a crash from leader
4> Node P5: detected a crash from leader
4> Node P4: detected a crash from leader
4> Node P3: detected a crash from leader
4> leader P2 CRASHED: msg {msg,{change_state,13}}
4> Node P5: detected a crash from leader
4> Node P3: detected a crash from leader
4> Node P4: detected a crash from leader
4>
```

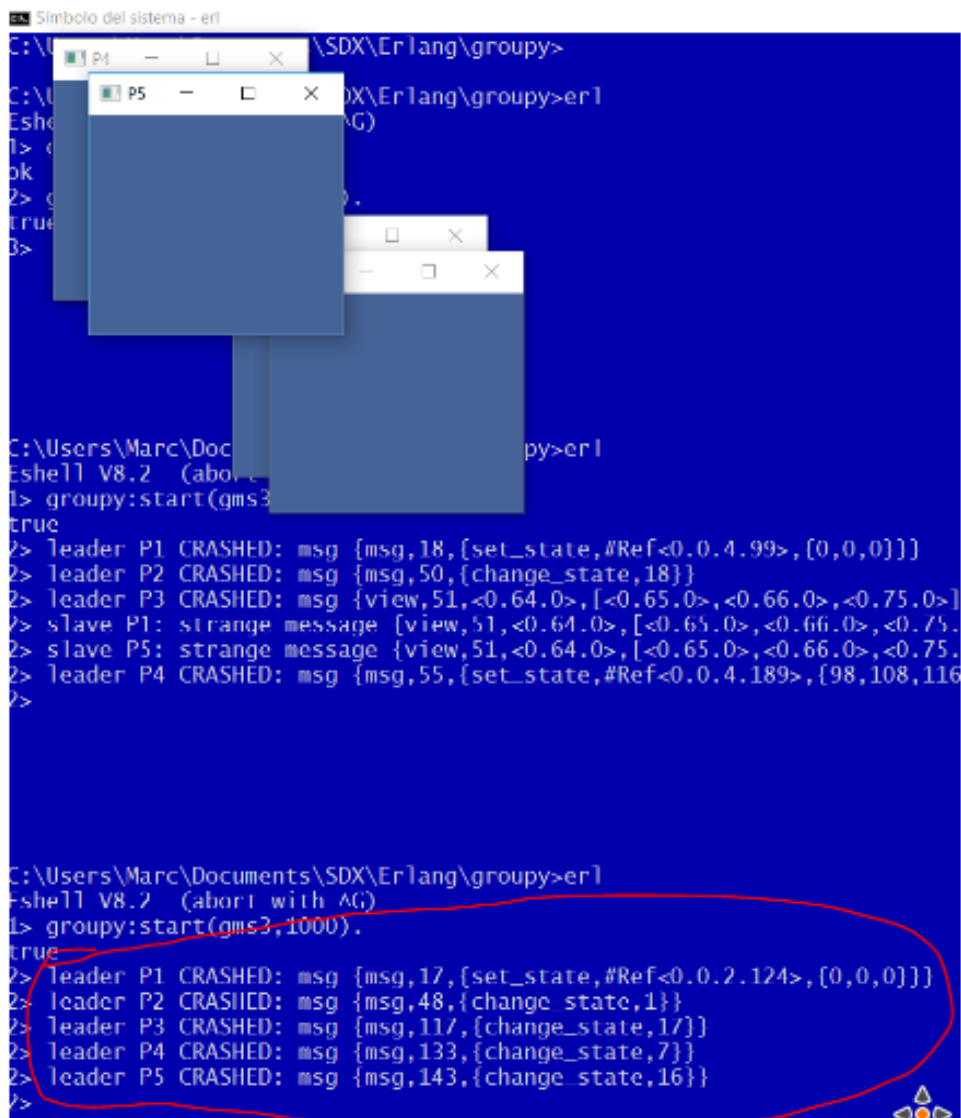
Figure 7: Slaves descoordinats

Amb aquest exemple podem concloure que si que es pot produir una descoordinació en el grup.

### 3.2.3 Reliable multicast

1. Repeat the experiments to see if now the peers can keep their state coordinated even if nodes crash and adding more nodes as existing nodes die

Amb aquest implementació si que tot i que per les múltiples fallades del Leaders, els processos continuen podent seguir coordinats tal i com es pot veure en la Figure de continuació.



```
C:\Users\Marc\Documents\SDX\Erlang\groupy>erl
Eshell V8.2 (abort with ^G)
1> groupy:start(gms3,1000).
true
2> leader P1 CRASHED: msg {msg,18,{set_state,#Ref<0.0.4.99>,[0,0,0]}}
2> leader P2 CRASHED: msg {msg,50,{change_state,18}}
2> leader P3 CRASHED: msg {view,51,<0.64.0>,[<0.65.0>,<0.66.0>,<0.75.0>]}
2> slave P1: strange message {view,51,<0.64.0>,[<0.65.0>,<0.66.0>,<0.75.0>]}
2> slave P5: strange message {view,51,<0.64.0>,[<0.65.0>,<0.66.0>,<0.75.0>]}
2> leader P4 CRASHED: msg {msg,55,{set_state,#Ref<0.0.4.189>,[98,108,116]}}
2>

C:\Users\Marc\Documents\SDX\Erlang\groupy>erl
Eshell V8.2 (abort with ^G)
1> groupy:start(gms3,1000).
true
2> leader P1 CRASHED: msg {msg,17,{set_state,#Ref<0.0.2.124>,[0,0,0]}}
2> leader P2 CRASHED: msg {msg,48,{change_state,1}}
2> leader P3 CRASHED: msg {msg,117,{change_state,17}}
2> leader P4 CRASHED: msg {msg,133,{change_state,7}}
2> leader P5 CRASHED: msg {msg,143,{change_state,16}}
2>
```

Figure 8: Els processos poden continuar coordinats a pesar de les fallades dels múltiples líders

A mes cada cop que un Leader cau, el nou Leader crea un nou Slave que demana ser afegit al grup, intentat així mantenir el mateix nombre de processos durant el temps.

### 3.3 Open questions

#### 3.3.1 Missing messages

1. *Why is this happening?*

Quan el Leader cau quan esta fent un multicast, només una fracció dels Slaves rebran l'últim missatge. Com que els canvis de vista/color en els workers son acumulatius (addició d'un color sobre l'actual), si un slave no ha rebut un missatge, al seu worker li falta una part del color actual. Per tant es desincronitzen.

#### 3.3.2 What possibly go wrong

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

Considerarem només la implementació de la part del líder, assumint que les peticions unicast en direcció al Leader representen molta menys complexitat. Per solucionar el problema dels missatges perduts s'ha d'implementar alguna versió del reliable multicast.

La versió scalable reliable multicast oferiria millor performance, però podria perjudicar la qualitat de la sincronització. Això és així perquè un node Slave que no rep un missatge enviarà el NACK quan el Leader li envii el següent. D'aquesta manera, el node endarrerit estarà un temps desincronitzat. Si el temps entre estats és prou gran, llavors aquesta descoordinació serà observable.

A més, podria donar-se la situació de que un Slave perdés més d'un missatge, i el Leader només ho sabria a partir del primer missatge que arribi. Més enllà de l'efecte que això tindria sobre la sincronització, el Leader hauria de guardar tota la sequencia de canvis d'estat, no només l'últim com fa ara. I per prevenir una fallada del Leader, tots els procesos haurien de tenir tota la sequencia de canvis d'estat.

Implementar basic reliable multicast ens donaria més control sobre la sincronització. En particular, podríem forçar que el leader no acceptés un canvi d'estat fins que tots els slaves hagin enviat l'ACK del canvi d'estat anterior. Així acotaríem la desincronització a un sol canvi d'estat (tant pel que fa al color com pel que fa al temps que els workers han estat descoordinats.)

A efectes pràctics, si assumim que els nodes slave no poden fer crash, aquesta segona solució és atomic multicast. Si els slaves poguessin fallar hauriem de canviar l'actual implementació de les vistes.

En les dues solucions no hem parlat de la situació de canvi de líder. En un sistema amb pèrdua de missatges, no podem estar segurs de que el següent de la llista haurà rebut l'últim missatge. Una possible solució seria implementar un procés d'elecció que prioritzés nodes que disposin del missatge amb el màxim número de seqüència.

Per finalitzar, afegir que la primera solució seria factible amb un parell de canvis. Primer, limitar el nombre de missatges que es guarden. És improbable que un node slave perdi més d'un cert nombre de missatges de manera consecutiva, és més eficient que considerar que no passarà.

Amb això, si un node s'endarrerís més missatges dels que caben en aquesta "caché", llavors el sistema no es podria recuperar. Però en aquest cas improbable, simplement fem un reset del node. De la mateixa manera que en l'operació de join, transmetre l'estat actual. Això és factible perquè el nostre estat, color, és poc complex.

## 2. *How would this impact performance?*

L'impacte sobre el rendiment de la primera solució (millorada) seria negligible. El cost d'un NACK (si es que és necessari) per multicast és mínim respecte al nombre de missatges del multicast, un per a cada node. En cas d'haver de recuperar missatges, el cost depèn del tamany de la caché. El temps de recuperació seria mínim si s'enviesin tots els missatges de cop. Fer reset de l'estat només costa un missatge.

La segona solució, basic reliable multicast, implica doblar el nombre de missatges. Un ACK per a cada missatge del multicast. A més, esperar els ACKS implica que el líder s'ha d'esperar  $\max(\text{RTT})$  abans d'enviar el següent canvi d'estat.

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

Amb la implementació actual, el sistema ja tindrà un nou Leader. Els Slaves enviaran les seves peticions de multicast al nou Leader. Però l'antic Leader encara gestionarà les peticions de canvi d'estat del seu worker, per les que sí farà multicast.

La resta de nodes, tant el nou Leader com els Slaves, processaran aquests canvis d'estat foranis perquè no controlem que el missatge de canvi d'estat provingui del Leader. Com tots els workers rebran els mateixos missatges, estaran sincronitzats. Excepte el worker associat a l'antic Leader, que ja no és a la llista dels demés i només respondrà a les seves propies peticions.

## 4 Personal opinion

Aquest lab ens ha ajudat ha comprendre millor els conceptes estudiats a teoria aixi com entendre com funciona i els problemes que te un dels protocols mes basics i utils en els sistemes distribuïts.