

Seminar Report: Ordy

Víctor Sendino García, Marc Mèndez Roca, Adrián Manco Sánchez

May 2, 2018

1 Introduction

L'objectiu d'aquesta pràctica és desenvolupar un servei de missatges per grup que implementi atomic multicast. Es vol tenir diversos processos a nivell d'aplicació que trasicionin d'estat, de manera que tots siguin al mateix estat, és a dir, coordinats. Quan un node vol canviar d'estat, ho comunica mitjançant multicast, i el sistema ha de sincronitzar tots els nodes per a fer efectiu el canvi, respectant l'ordenació total que ha de garantir l'atomic multicast.

La dificultat radica en que els processos poden patir fallades de crash, però el sistema ha de garantir que la sincronia és manté en tot moment.

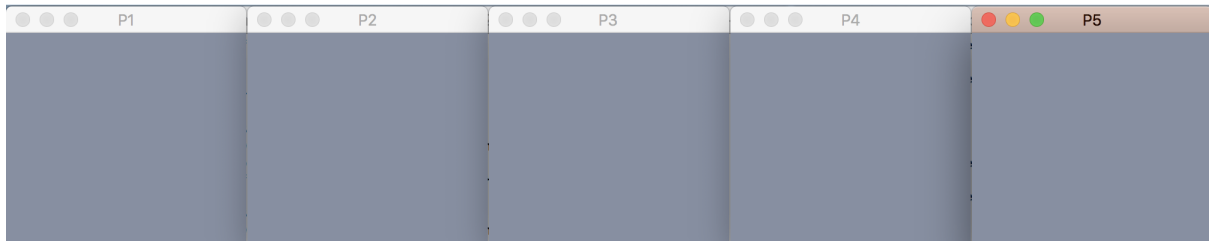
2 Work done

Com als anteriors seminaris, el treball a fer a consistit bàsicament per omplir els forats que hi havia als codis proporcionats. Hem implementat 3 versions diferents (gms1,gms2,gms3), on cadascuna d'elles afegia més complexitat. En el primer cas, gms1, només permetia començar amb un node i anar afegint més nodes mantenint-los sincronitzats segons els canvis d'estat. En aquesta primera versió no es tenen en compte les fallades. El primer procés es convertia directament en el líder, i si cau no es torna a escollir a ningú. En gms2 afegim més tolerància a fallades. Els slaves detecten les fallades del leader i s'escollix un de nou. Els slaves no poden fallar.

En la tercera versió (gms3) afegim números de seqüència i intentem implementar reliable multicast. On el nou leader reenvia l'últim missatge per si de cas no tothom l'ha rebut. Els codis estan inclosos en el tar.gz adjunt.

3 Experiments

- 1. Do some experiments to see that you can create a group, add some peers and keep their state coordinated. You can use the following code to start and stop the whole system. Note that we are using the name of the module (i.e. gms1) as the parameter Module to the start procedure. All the workers but the first one need to know a member of the group in order to join. Sleep stands for up to how many milliseconds the workers should wait until the next message is sent.**
- 2. Adapt the groupy module to create each worker in a different Erlang instance. Remember how processes are created remotely, how names registered in remote nodes are referred, and how Erlang runtime should be started to run distributed programs.**



Aquí tenim la primera versió sense fallades, en la qual el líder és sempre el mateix.
L'objectiu d'aquesta versió és comprovar que no es perd l'estat i que al llarg del temps tots els "Workers" mantenen l'estat.
El codi s'ha modificat per que cada mòdul es trobi en una instància Erlang diferent.

3.1 Failure Detectors

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

```
* 1: syntax error before: ')'
(p1@127.0.0.1)1> c(groupy).
{ok,groupy}
(p1@127.0.0.1)2> c(worker).
{ok,worker}
(p1@127.0.0.1)3> groupy:start(gms2, 1000).
<8126.91.0>
leader P1 CRASHED: msg {msg,{change_state,5}}
leader P2 CRASHED: msg {msg,{change_state,16}}
leader P3 CRASHED: msg {msg,{change_state,11}}
leader P4 CRASHED: msg {msg,{change_state,18}}
(p1@127.0.0.1)4>
BREAK: (a)bort (c)ontinue (p)roc info (i)nfo (l)oaded
(v)ersion (k)ill (D)b-tables (d)istribution
a
MacBook-Pro-de-Marc:groupy marcmenendez$ erl -name p1@127.0.0.1 -setcookie secret -kernel inet
_dist_listen_min 1025 -kernel inet_dist_listen_max 2000
Erlang/OTP 20 [erts-9.2.1] [source] [64-bit] [smp:4:4] [ds:4:4:10] [async-threads:10] [hipe]
[kernel-poll:false] [dtrace]

Eshell V9.2.1 (abort with ^G)
(p1@127.0.0.1)1> groupy:start(gms2, 1000).
<7990.71.0>
leader P1 CRASHED: msg {msg,{set_state,#Ref<7989.546143922.801898499.8802>,{0,0,0}}}
leader P2 CRASHED: msg {msg,{change_state,12}}
(p1@127.0.0.1)2>
a
MacBook-Pro-de-Marc:groupy marcmenendez$ erl -name p3@127.0.0.1 -setcookie secret -kernel inet
_dist_listen_min 1025 -kernel inet_dist_listen_max 2000
Erlang/OTP 20 [erts-9.2.1] [source] [64-bit] [smp:4:4] [ds:4:4:10] [async-threads:10] [hipe]
[kernel-poll:false] [dtrace]

Eshell V9.2.1 (abort with ^G)
(p3@127.0.0.1)1> groupy:start(gms2, 1000).
<7990.71.0>
leader P1 CRASHED: msg {msg,{set_state,#Ref<7989.546143922.801898499.8802>,{0,0,0}}}
leader P2 CRASHED: msg {msg,{change_state,12}}
(p3@127.0.0.1)2>
a
MacBook-Pro-de-Marc:groupy marcmenendez$ erl -name p5@127.0.0.1 -setcookie secret -kernel inet
_dist_listen_min 1025 -kernel inet_dist_listen_max 2000
Erlang/OTP 20 [erts-9.2.1] [source] [64-bit] [smp:4:4] [ds:4:4:10] [async-threads:10] [hipe]
[kernel-poll:false] [dtrace]

Eshell V9.2.1 (abort with ^G)
(p5@127.0.0.1)1> groupy:start(gms2, 1000).
<7990.71.0>
leader P1 CRASHED: msg {msg,{set_state,#Ref<7989.546143922.801898499.8802>,{0,0,0}}}
leader P2 CRASHED: msg {msg,{change_state,12}}
(p5@127.0.0.1)2>
a
MacBook-Pro-de-Marc:groupy marcmenendez$ erl -name p4@127.0.0.1 -setcookie secret -kernel inet
_dist_listen_min 1025 -kernel inet_dist_listen_max 2000
Erlang/OTP 20 [erts-9.2.1] [source] [64-bit] [smp:4:4] [ds:4:4:10] [async-threads:10] [hipe]
[kernel-poll:false] [dtrace]

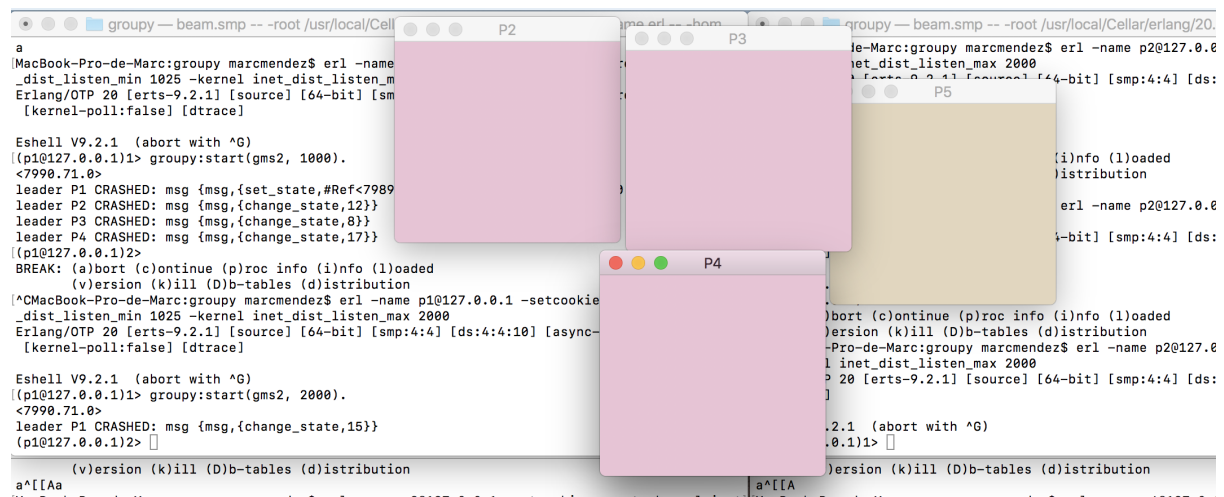
Eshell V9.2.1 (abort with ^G)
(p4@127.0.0.1)1> groupy:start(gms2, 1000).
<7990.71.0>
leader P1 CRASHED: msg {msg,{set_state,#Ref<7989.546143922.801898499.8802>,{0,0,0}}}
leader P2 CRASHED: msg {msg,{change_state,12}}
(p4@127.0.0.1)2>
```

En aquest cas hi ha tolerància a fallades doncs quan es produeix una fallada de líder es llença un missatge per tal de reelegir el líder i poder seguir mantenint l'estat.

3.2 Missing messages

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

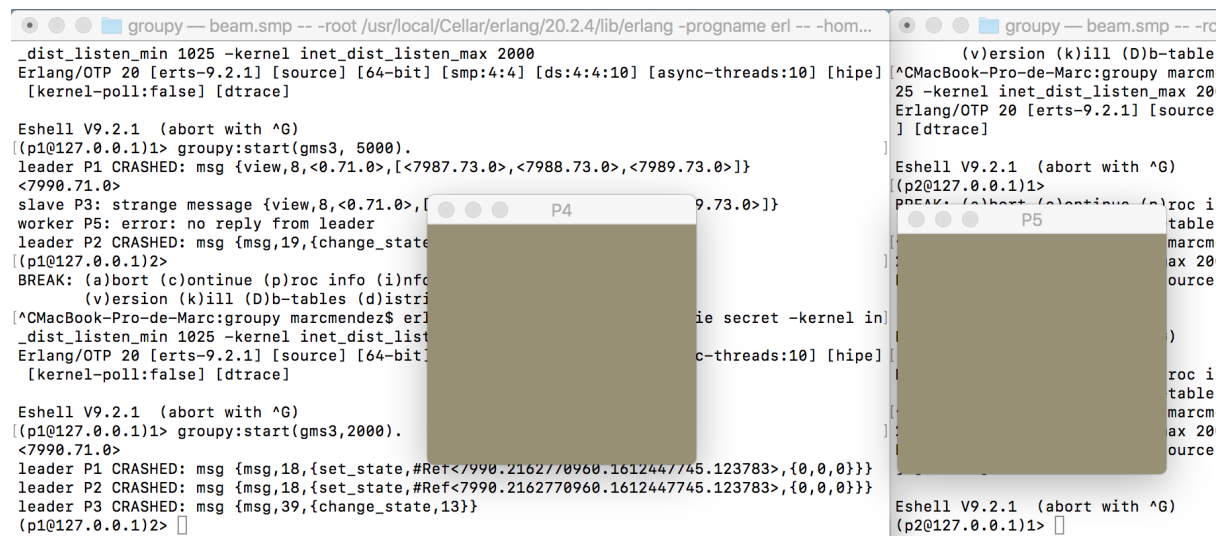
En el cas anterior tot va bé però quan el líder cau al mig d'un missatge de canvi d'estat pot donar-se el cas que alguns nodes quedin desincronitzats, i per tant ja no comparteixin el mateix estat.



Aquí es veuria el cas en que líder llença un missatge de canvi d'estat però no tots els nodes el reben i per tant hi ha un o més que no estan en sintonia amb el líder.

3.3 Reliable Multicast

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



Com es pot veure, en aquesta versió si que manté el estat al morir i per tant, no hi ha el problema que hi havia abans que quan moria un node al mig d'un missatge. Per tant, tot i haver-hi fallades, el control que fan del estat és correcte.

ii) Try to keep a group rolling by adding more nodes as existing nodes die.

Not completed.

4 Open questions

gms2

1. Why could the state of the workers go out of sync?
 - a. Si el Leader cau mentre fa un multicast, no tots el Slaves rebran el missatge, i ja que el canvi de color es basa en l'addició d'un color al color existent, els Slaves que no hagin rebut el missatge durarn a terme l'addició, i apareixeran diferents de la resta.

gms3

1. How would we have to change the implementation to handle the possibly lost messages?
 - a. S'hauria d'implementar un Scalable Multicast, o un Basic Reliable Multicast. En el primer cas, com que un Slave no demanaria (mitjançant NACK), els missatges que no li han arribat, durant un temps es podria apreciar la desincronització causada per la pèrdua de missatges. A més, el Leader necessitaria guardar els missatges enviats, per poder reenviar-los donat el cas. En el segon cas com que els Slaves podrien enviar ACKs al Leader, podríem acotar una hipotètica desincronització a un missatge, si fem que sigui condició haver rebut ACK de tots els Slaves abans del següent canvi d'estat.
2. How would this impact performance?
 - a. Com ja he comentat, Scalable necessitaria que el Leader recordés els missatges que ha enviat, i és una càrrega extra de treball. En Basic, el sistema hauria de suportar una quantitat major de missatges a causa dels ACK dels Slaves cap el Leader.
3. What would happen if we wrongly suspect the leader to have crashed?
 - a. Un dels slaves sortiria escollit com a nou leader i la resta menys l'antic leader estaran inclosos a la nova llista de Slaves. Per tant els Slaves de la llista enviarien les noves peticions de multicast a aquest nou leader mentre que l'antic continuarà pensant que és el leader i per tant al rebre les peticions del seu worker les distribuirà a la seva antiga llista de Slaves. Aleshores ens trobem en la situació en que si el worker del líder antic fa una petició aquesta arribarà a tots els antics slaves, que estaran tots sincronitzats entre ells. Però si la petició prové d'un altre worker aleshores com el nou leader no té a la llista de slaves a l'antic aquesta no li arribarà mai, per tant serà l'únic que estarà desincronitzat mentre la resta si que funcionarà correctament.

5 Personal opinion

En general, ha estat un seminari útil, ja que hem fet diversos abans, i estem més familiaritzats amb el mètode de realització d'aquests, i amb Erlang. Com és costum, els canvis de color a les finestres fan que sigui fàcil comprovar el progrés o detectar errors al sistema, a més de ser agradable a la vista.