Seminar Report: Chordy

Marc Ortiz
Omair Iqbal
Víctor González

June 4, 2017

1 Introduction

Aquesta pràctica consisteix en implementar una distributed hash table seguint l'esquema de Chord; en el qual es podran afegir nodes, tenir un storage i controlar tant la caiguda d'un node com la recuperació del seu storage.

2 Work done

Els fitxers font es troben en la carpeta codes i les imatges en la carpets figures per si no es veuen prou bé.

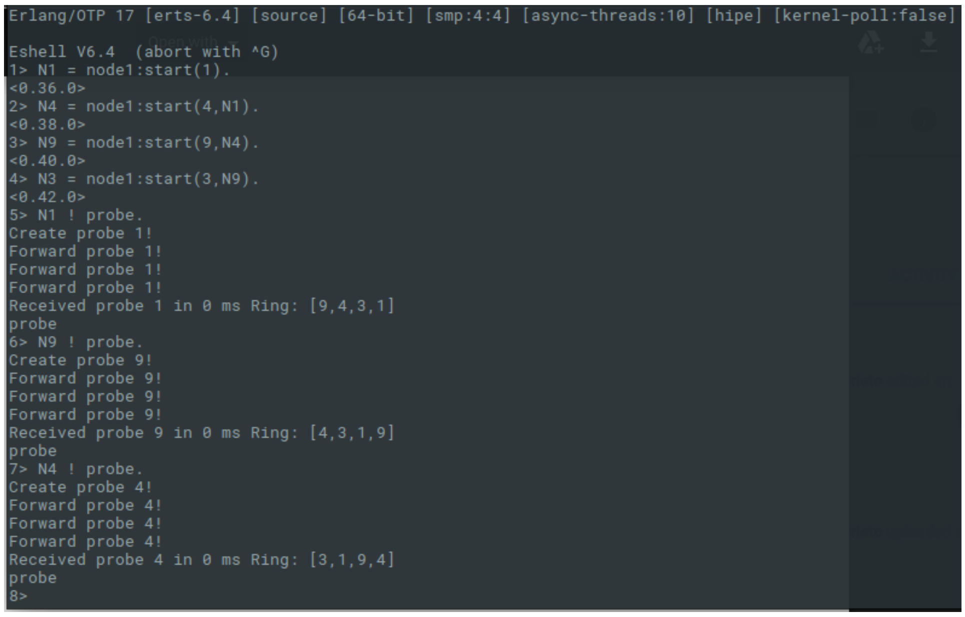
3 Experiments

3.1 Building a ring

En aquesta primera implementació bàsica tenim nomes l'estructura del ring, on poden afegir nodes pero de moment no poden afegir storage.

1. *Do some small experiments, to start with in one Erlang machine but then in a network of machines. When connecting nodes on different platforms remember to start Erlang in distributed mode (giving a -name argument) and make sure that you use the same cookie (-setcookie).*

   Creem 4 nodes (N1, N3, N4, N9) per testejar y després enviem missatges probe als nodes per veure si el ring esta connectat o no y el temps que tarda el missatge en enviar-se per tot el ring.



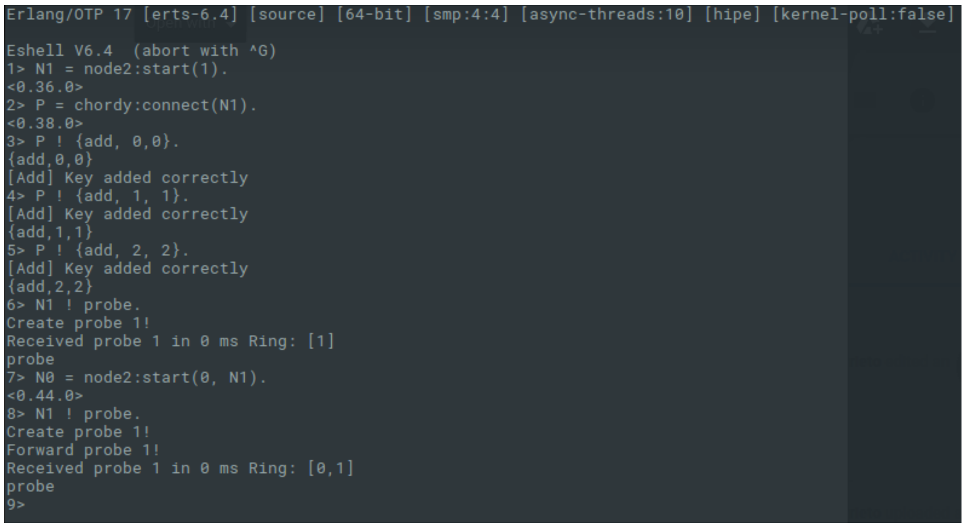
```
Erlang/OTP 17 [erts-6.4] [source] [64-bit] [smp:4:4] [async-threads:10] [hipe] [kernel-poll:false]

Eshell V6.4  (abort with ^G)
1> N1 = node1:start(1).
<0.36.0>
2> N4 = node1:start(4,N1).
<0.38.0>
3> N9 = node1:start(9,N4).
<0.40.0>
4> N3 = node1:start(3,N9).
<0.42.0>
5> N1 ! probe.
Create probe 1!
Forward probe 1!
Forward probe 1!
Forward probe 1!
Received probe 1 in 0 ms Ring: [9,4,3,1]
probe
6> N9 ! probe.
Create probe 9!
Forward probe 9!
Forward probe 9!
Forward probe 9!
Received probe 9 in 0 ms Ring: [4,3,1,9]
probe
7> N4 ! probe.
Create probe 4!
Forward probe 4!
Forward probe 4!
Forward probe 4!
Received probe 4 in 0 ms Ring: [3,1,9,4]
probe
8>
```

Figure 1: S'envien 3 missatge probe per veure si el ring segueix connectat

3.2 Adding store

En aquesta segona implementació afegirem un storage a cada node i amb la posibilitat d'afegir i buscar per key-value pairs.

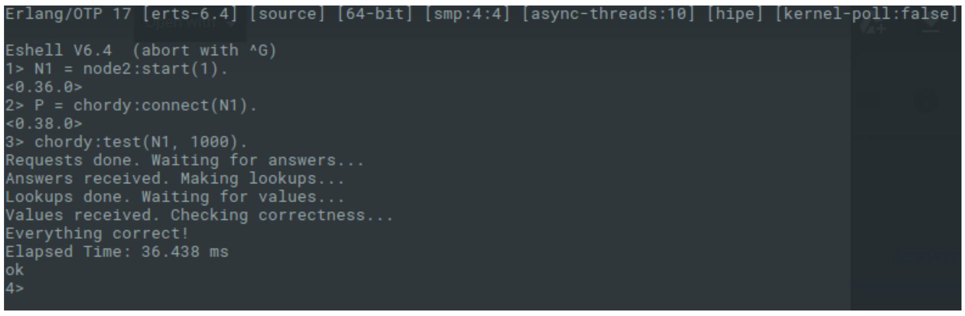
1. *If we now have a distributed store that can handle new nodes that are added to the ring we might try some performance testing. You need several machines to do this. Assume that we have eight machines and that we will use four in building the ring and four in testing the performance.*



```
Erlang/OTP 17 [erts-6.4] [source] [64-bit] [smp:4:4] [async-threads:10] [hipe] [kernel-poll:false]

Eshell V6.4  (abort with ^G)
1> N1 = node2:start(1).
<0.36.0>
2> P = chordy:connect(N1).
<0.38.0>
3> P ! {add, 0,0}.
{add,0,0}
[Add] Key added correctly
4> P ! {add, 1, 1}.
[Add] Key added correctly
{add,1,1}
5> P ! {add, 2, 2}.
[Add] Key added correctly
{add,2,2}
6> N1 ! probe.
Create probe 1!
Received probe 1 in 0 ms Ring: [1]
probe
7> N0 = node2:start(0, N1).
<0.44.0>
8> N1 ! probe.
Create probe 1!
Forward probe 1!
Received probe 1 in 0 ms Ring: [0,1]
probe
9>
```

Figure 2: 2 nodes en el ring amb keys

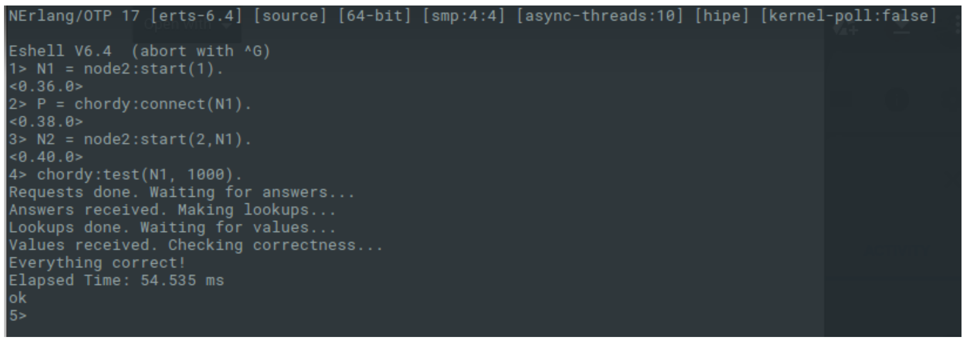
2. *As a first test, we have one node only in the ring and let each of the four test machines add a number (e.g., 1000) of random elements (key-value pairs) to the ring and then do a lookup of the elements, measuring the time it takes. You can use the test procedure given in 'Appendix A', which should be given with the ID of the node to contact.*



```
Erlang/OTP 17 [erts-6.4] [source] [64-bit] [smp:4:4] [async-threads:10] [hipe] [kernel-poll:false]

Eshell V6.4  (abort with ^G)
1> N1 = node2:start(1).
<0.36.0>
2> P = chordy:connect(N1).
<0.38.0>
3> chordy:test(N1, 1000).
Requests done. Waiting for answers...
Answers received. Making lookups...
Lookups done. Waiting for values...
Values received. Checking correctness...
Everything correct!
Elapsed Time: 36.438 ms
ok
4>
```

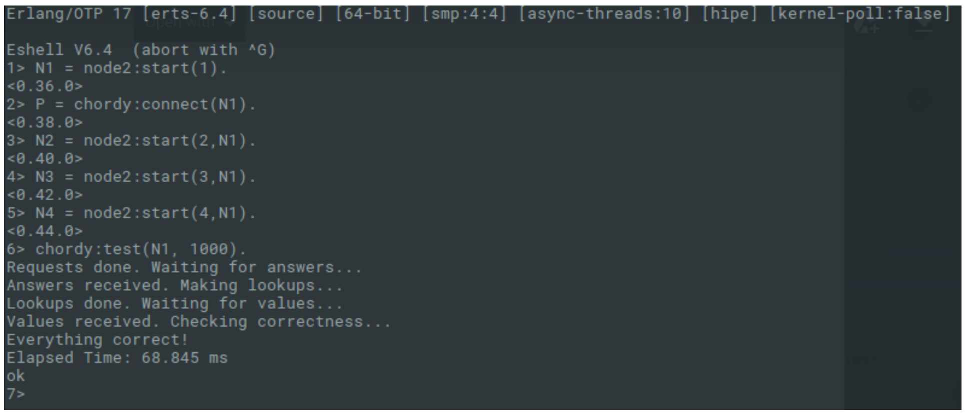
Figure 3: Testing nomes un node

3. *How does the performance change if we add another node to the ring? Add two more nodes to the ring, any changes? Does it matter if all the test machines access the same node or different nodes in the ring?*



```
NErlang/OTP 17 [erts-6.4] [source] [64-bit] [smp:4:4] [async-threads:10] [hipe] [kernel-poll:false]

Eshell V6.4  (abort with ^G)
1> N1 = node2:start(1).
<0.36.0>
2> P = chordy:connect(N1).
<0.38.0>
3> N2 = node2:start(2,N1).
<0.40.0>
4> chordy:test(N1, 1000).
Requests done. Waiting for answers...
Answers received. Making lookups...
Lookups done. Waiting for values...
Values received. Checking correctness...
Everything correct!
Elapsed Time: 54.535 ms
ok
5>
```

Figure 4: Experiment amb 2 nodes



```
Erlang/OTP 17 [erts-6.4] [source] [64-bit] [smp:4:4] [async-threads:10] [hipe] [kernel-poll:false]

Eshell V6.4  (abort with ^G)
1> N1 = node2:start(1).
<0.36.0>
2> P = chordy:connect(N1).
<0.38.0>
3> N2 = node2:start(2,N1).
<0.40.0>
4> N3 = node2:start(3,N1).
<0.42.0>
5> N4 = node2:start(4,N1).
<0.44.0>
6> chordy:test(N1, 1000).
Requests done. Waiting for answers...
Answers received. Making lookups...
Lookups done. Waiting for values...
Values received. Checking correctness...
Everything correct!
Elapsed Time: 68.845 ms
ok
7>
```

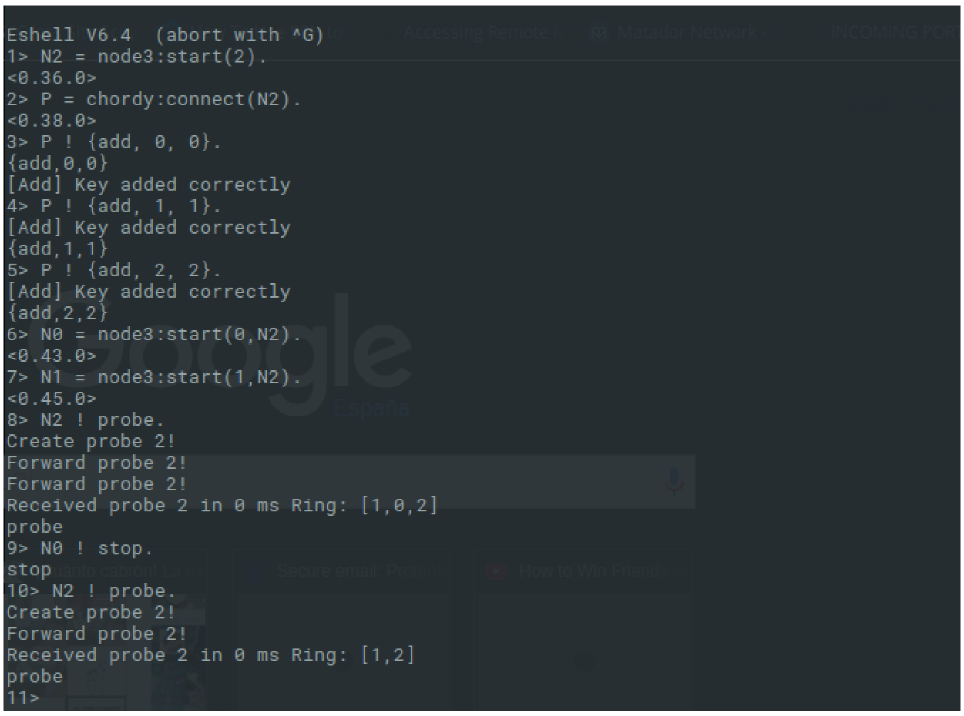
Figure 5: Experiment amb 4 nodes

Com podem veure a mes nodes, mes temps.

3.3 Handling failure

En aquesta tercera implementacio tindrem un sistema tolerant a fallades.

1. *Do some experiments to evaluate the behavior of your implementation when nodes can fail.*



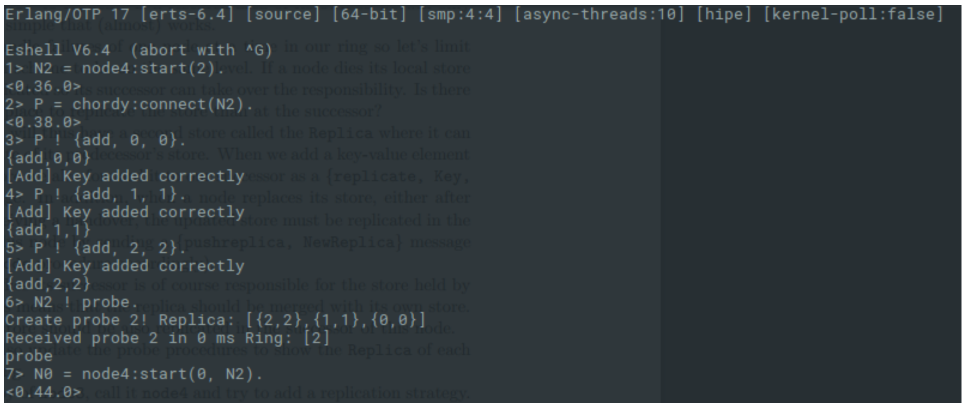
```
Eshell V6.4  (abort with ^G)
1> N2 = node3:start(2).
<0.36.0>
2> P = chordy:connect(N2).
<0.38.0>
3> P ! {add, 0, 0}.
{add,0,0}
[Add] Key added correctly
4> P ! {add, 1, 1}.
[Add] Key added correctly
{add,1,1}
5> P ! {add, 2, 2}.
[Add] Key added correctly
{add,2,2}
6> N0 = node3:start(0,N2).
<0.43.0>
7> N1 = node3:start(1,N2).
<0.45.0>
8> N2 ! probe.
Create probe 2!
Forward probe 2!
Forward probe 2!
Received probe 2 in 0 ms Ring: [1,0,2]
probe
9> N0 ! stop.
stop
10> N2 ! probe.
Create probe 2!
Forward probe 2!
Received probe 2 in 0 ms Ring: [1,2]
probe
11>
```

Figure 6: Sistema tolerant a fallades

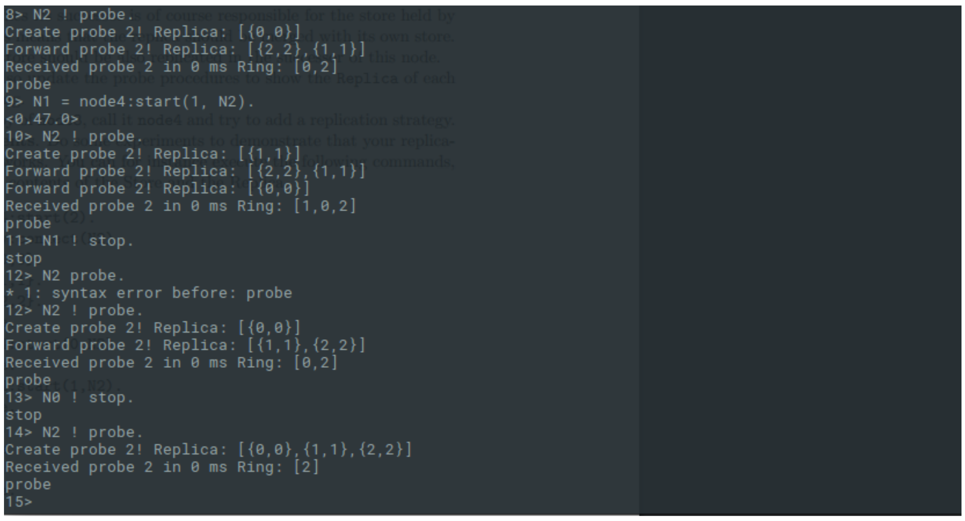
3.4 Replication

En aquesta ultima implementacio tenim un sistema que permet replicar la informacio contenida als nodes.

1. *Do some experiments to demonstrate that your replication strategy works. You can for instance execute the following commands, and check the contents of the Store and the Replica*



```
Erlang/OTP 17 [erts-6.4] [source] [64-bit] [smp:4:4] [async-threads:10] [hipe] [kernel-poll:false]

Eshell V6.4  (abort with ^G)
1> N2 = node4:start(2).
<0.36.0>
2> P = chordy:connect(N2).
<0.38.0>
3> P ! {add, 0, 0}.
{add,0,0}
[Add] Key added correctly
4> P ! {add, 1, 1}.
[Add] Key added correctly
{add,1,1}
5> P ! {add, 2, 2}.
[Add] Key added correctly
{add,2,2}
6> N2 ! probe.
Create probe 2! Replica: [{2,2},{1,1},{0,0}]
Received probe 2 in 0 ms Ring: [2]
probe
7> N0 = node4:start(0, N2).
<0.44.0>
```



```
8> N2 ! probe.
Create probe 2! Replica: [{0,0}]
Forward probe 2! Replica: [{2,2},{1,1}]
Received probe 2 in 0 ms Ring: [0,2]
probe
9> N1 = node4:start(1, N2).
<0.47.0>
10> N2 ! probe.
Create probe 2! Replica: [{1,1}]
Forward probe 2! Replica: [{2,2},{1,1}]
Forward probe 2! Replica: [{0,0}]
Received probe 2 in 0 ms Ring: [1,0,2]
probe
11> N1 ! stop.
stop
12> N2 probe.
* 1: syntax error before: probe
12> N2 ! probe.
Create probe 2! Replica: [{0,0}]
Forward probe 2! Replica: [{1,1},{2,2}]
Received probe 2 in 0 ms Ring: [0,2]
probe
13> N0 ! stop.
stop
14> N2 ! probe.
Create probe 2! Replica: [{0,0},{1,1},{2,2}]
Received probe 2 in 0 ms Ring: [2]
probe
15>
```

Figure 7: Sistema almost amb replicacio

3.5 Open questions

3.5.1 Building a ring

1. *What are the pros and cons of a more frequent stabilizing procedure?*

   Un procediment d'estabilització més freqüènt fará que els nodes que no estiguin estabilitzats s'enllacin molt més ràpid al ring, a costa d'enviar molt més missatges. Caldria trobar un trade-off entre l'overhead creat pel missatges extres enviats i la velocitats dels nodes quan s'enllacen amb el ring, ja que a més freqüència d'estabilització no sempre comporta a una major velocitat d'enllaçament dels nodes al ring.

2. *Do we have to inform the new node about our decision? How will it know if we have discarded its friendly proposal?*

   Hem de informar-li ja que probablement tingui menys informació que nosaltres i per tant hem de assenyalar la direcció correcta per tal que trobi el seu successor adequat. Ha de ser així per que nosaltres que tenim mes informació com per exemple sabem el nostre propi successor, el podem guiar millor.

3. *What would happen if we didn't schedule the stabilize procedure? Would things still work?*

   Si no féssim el proces d'estabilització, aquest nomes es faria quan es rep un status i això nomes passa quan s'afegeix al ring. Si cauen nodes i no estabilitzem, tenim incoherencia.

3.5.2 Handling failures

4. *What will happen if a node is falsely detected of being dead (e.g. it has only been temporally unavailable)?*

   Sera descartat i el ring cambiara fins que torni a detectar el node, moment en el qual tornara a formar part de la topologia.

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.