Seminar Report: Ordy

Daniel García, Aitor Ardila, Oscar Hernández
April 17, 2016

1 - Introduction

In this seminar we are facing sorting problems using multicast messages between workers. Our job is to implement two algorithms including ‘causal’ ordering and ‘total’ ordering.

2 - Work done

In the first part of this seminar we had ‘basic’ multicast algorithm that sends to the rest of peer multicast messages with some Jitter achieving some interleave order.

(1) BASIC

```
(p1@127.0.0.1)13> ordy:start(basic,100,2000,10000).
```

p1 window

```
(p1@127.0.0.1)13> P1 SEND( 304) ; Subject: Re:Re:ukwvrflx
P1 RECV(1093) ; From: P1( 304) ; Subject: Re:Re:ukwvrflx
```

p3 window

```
P3 SEND( 304) ; Subject: Re:sbxdzfvu
P3 RECV(1062) ; From: P1( 262) ; Subject: Re:ukwvrflx
P3 RECV(1063) ; From: P2( 217) ; Subject: apkswxfl
P3 RECV(1064) ; From: P3( 304) ; Subject: Re:sbxdzfvu
```

Secondly, we needed to implement some causal sort algorithm in order to eliminate unsorted replies problems. We did this by using Vector Clocks and tracking VC of each process updating it if was necessary.

(2) CAUSAL

```
(p1@127.0.0.1)16> ordy:start(causal,100,2000,10000).
```

p1 Window

```
P1 RECV( 372) ; From: P3(  79) ; Subject: Re:afugisjr
P1 SEND( 139) ; Subject: Re:Re:afugisjr
P1 RECV( 373) ; From: P3(  80) ; Subject: Re:uxzvorkj
P1 SEND( 140) ; Subject: Re:Re:uxzvorkj
P1 RECV( 374) ; From: P3(  81) ; Subject: dsgbxyel
P1 RECV( 375) ; From: P3(  82) ; Subject: Re:Re:Re:jxyaic
P1 RECV( 376) ; From: P3(  83) ; Subject: mtzjpowb
P1 RECV( 377) ; From: P3(  84) ; Subject: Re:nzoiyqhs
P1 RECV( 378) ; From: P3(  85) ; Subject: rgjztyhw
P1 RECV( 379) ; From: P3(  86) ; Subject: tewgfjks
P1 RECV( 380) ; From: P1( 139) ; Subject: Re:Re:afuqisjr
```

Finally, we implemented total ordering which forces all messages to be received in the same order. We achieved this by using proposal messages.

(3) TOTAL

```
(p1@127.0.0.1)13> ordy:start(total,100,1000,5000).
```

p1 window

```
P1 RECV( 165) ; From: P3(  42) ; Subject: vufduoza
P1 RECV( 166) ; From: P1(  33) ; Subject: gwucpexj
P1 RECV( 167) ; From: P2(  48) ; Subject: tsqfrmwv
P1 SEND(  86) ; Subject: Re:tsqfrmwv
P1 RECV( 168) ; From: P2(  44) ; Subject: ytmcefri
P1 RECV( 169) ; From: P2(  60) ; Subject: Re:blsdgkvn
P1 RECV( 170) ; From: P3(  40) ; Subject: ntgrjoyf
P1 SEND(  87) ; Subject: Re:ntgrjoyf
P1 RECV( 171) ; From: P3(  52) ; Subject: jgupnhvr
```

p2 window

```
P2 RECV( 165) ; From: P3(  42) ; Subject: vufduoza
P2 RECV( 166) ; From: P1(  33) ; Subject: gwucpexj
P2 RECV( 167) ; From: P2(  48) ; Subject: tsqfrmwv
P2 RECV( 168) ; From: P2(  44) ; Subject: ytmcefri
P2 RECV( 169) ; From: P2(  60) ; Subject: Re:blsdgkvn
P2 RECV( 170) ; From: P3(  40) ; Subject: ntgrjoyf
```

p3 window

```
P3 RECV( 169) ; From: P2(  60) ; Subject: Re:blsdgkvn
P3 RECV( 170) ; From: P3(  40) ; Subject: ntgrjoyf
P3 SEND( 104) ; Subject: hluxrvpi
P3 SEND( 105) ; Subject: jdrvfsal
P3 SEND( 106) ; Subject: cxtnplhs
```

p4 window

```
P4 RECV( 167) ; From: P2(  48) ; Subject: tsqfrnwv
P4 RECV( 168) ; From: P2(  44) ; Subject: ytmcefri
P4 RECV( 169) ; From: P2(  60) ; Subject: Re:blsdgkvn
P4 RECV( 170) ; From: P3(  40) ; Subject: ntgrjoyf
P4 RECV( 171) ; From: P3(  52) ; Subject: jgupnhvr
P4 SEND(  96) ; Subject: Re:jgupnhvr
```

3 - Experiments

3.1 - Basic algorithm

Basic algorithm starts to unsort messages while Jitter increases as you can see in the figures below:

BASIC 100ms Sleep, 0 Jitter, 3 seconds executing

```
(p1@127.0.0.1)17> ordy:start(basic,100,0,3000).
(p1@127.0.0.1)18> P1 SEND(   1) ; Subject: enpuxdkg
P1 RECV(   1) ; From: P1(   1) ; Subject: enpuxdkg
P1 RECV(   2) ; From: P3(   1) ; Subject: qijwhzlp
P1 SEND(   2) ; Subject: efprthcl
P1 RECV(   3) ; From: P4(   1) ; Subject: Re:qijwhzlp
P1 RECV(   4) ; From: P4(   2) ; Subject: nskfamro
P1 RECV(   5) ; From: P2(   1) ; Subject: Re:Re:qijwhzlp
P1 RECV(   6) ; From: P2(   2) ; Subject: qdksgcpw
P1 RECV(   7) ; From: P1(   2) ; Subject: efprthcl
```

As you can see, everything is sorted in this environment

BASIC 300ms Sleep, 3 seconds Jitter, 20 seconds executing

```
(p1@127.0.0.1)23> ordy:start(basic,300,3000,20000).
```

```
P1 RECV( 875) ; From: P4( 213) ; Subject: Re:Re:ycarvepl
P1 SEND( 231) ; Subject: Re:Re:Re:ycarvepl
P1 RECV( 876) ; From: P1( 231) ; Subject: Re:Re:Re:ycarvepl
P1 SEND( 232) ; Subject: gaehbmzt
P1 RECV( 877) ; From: P1( 232) ; Subject: gaehbmzt
P1 RECV( 878) ; From: P2( 217) ; Subject: xdqwitck
P1 RECV( 879) ; From: P2( 239) ; Subject: Re:Re:qtcwpuea
P1 RECV( 880) ; From: P2( 209) ; Subject: Re:heoucpzn
P1 RECV( 881) ; From: P3( 204) ; Subject: Re:ycarvepl
P1 SEND( 233) ; Subject: Re:Re:ycarvepl
P1 RECV( 882) ; From: P1( 233) ; Subject: Re:Re:ycarvepl
```

As you can see, we are receiving replies from a message before the original message.

3.2 - Causal algorithm

CAUSAL 100ms Sleep, 0 Jitter, 3 seconds executing

```
(p1@127.0.0.1)23> ordy:start(causal,100,0,3000).
P1 RECV( 103) ; From: P3(  26) ; Subject: edzapjur
P1 RECV( 104) ; From: P3(  27) ; Subject: rcywjhzt
P1 SEND(  28) ; Subject: Re:rcywjhzt
P1 RECV( 105) ; From: P3(  28) ; Subject: Re:Re:Re:upbcfdjq
P1 RECV( 106) ; From: P3(  29) ; Subject: Re:Re:Re:ocgmftaq
P1 RECV( 107) ; From: P2(  29) ; Subject: Re:edzapjur
P1 RECV( 108) ; From: P4(  25) ; Subject: Re:Re:Re:upbcfdjq
P1 RECV( 109) ; From: P4(  26) ; Subject: aoxkzicl
P1 RECV( 110) ; From: P4(  27) ; Subject: Re:edzapjur
P1 RECV( 111) ; From: P2(  30) ; Subject: tzmrioaf
P1 RECV( 112) ; From: P2(  31) ; Subject: Re:Re:edzapjur
P1 SEND(  29) ; Subject: Re:Re:Re:edzapjur
P1 RECV( 113) ; From: P2(  32) ; Subject: liphvfzm
P1 RECV( 114) ; From: P4(  28) ; Subject: cugktibd
P1 SEND(  30) ; Subject: Re:cugktibd
P1 RECV( 115) ; From: P4(  29) ; Subject: Re:Re:Re:edzapjur
```

Messages still have causal order in this scenario

CAUSAL 300ms Sleep, 3 seconds Jitter, 20 seconds executing

```
(p1@127.0.0.1)25> ordy:start(causal,300,3000,20000).
417  P1 RECV( 456) ; From: P3( 100) ; Subject: Re:Re:vtaglkme
418  P1 RECV( 457) ; From: P3( 101) ; Subject: bdcwqonp
419  P1 RECV( 458) ; From: P3( 102) ; Subject: Re:chopywvi
420  P1 RECV( 459) ; From: P3( 103) ; Subject: riaodtbs
421  P1 RECV( 460) ; From: P3( 104) ; Subject: iuyvlwet
422  P1 RECV( 461) ; From: P3( 105) ; Subject: Re:vhvqkedf
423  P1 SEND( 144) ; Subject: Re:Re:vhvqkedf
424  P1 RECV( 462) ; From: P3( 106) ; Subject: Re:Re:jodzqvul
425  P1 RECV( 463) ; From: P3( 107) ; Subject: Re:Re:vtaglkme
426  P1 SEND( 145) ; Subject: Re:Re:Re:vtaglkme
427  P1 RECV( 464) ; From: P1( 144) ; Subject: Re:Re:vhvqkedf
428  P1 RECV( 465) ; From: P4( 111) ; Subject: Re:agtldhez
429  P1 RECV( 466) ; From: P4( 112) ; Subject: nbvpkqjw
430  P1 RECV( 467) ; From: P1( 145) ; Subject: Re:Re:Re:vtaglkme
```

Messages have now causal order in this scenario thanks to vector clocks, even if jitter is extremely high.

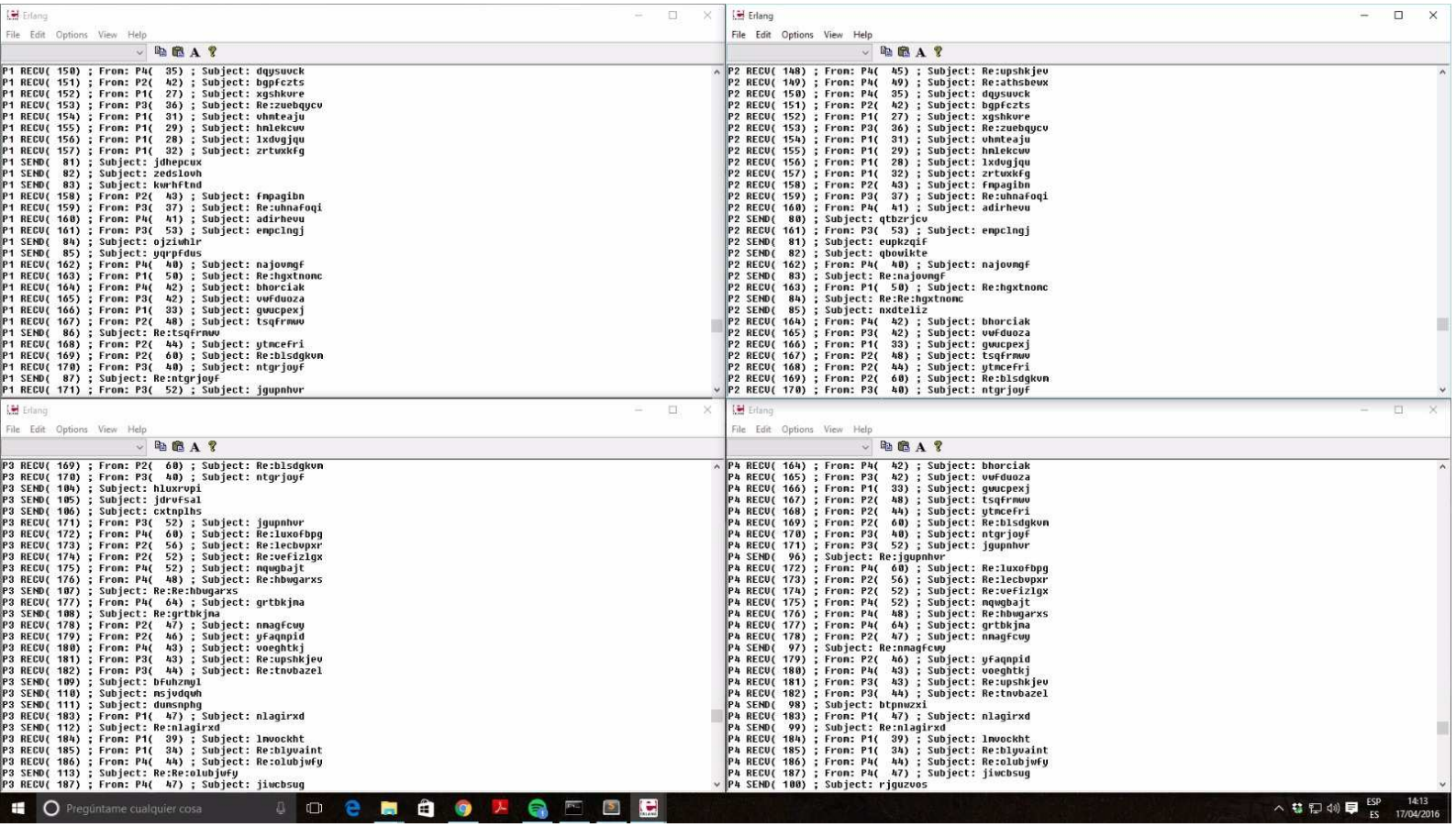
3.3 - Total algorithm

TOTAL 100ms Sleep, 0 Jitter, 3 seconds executing

```
(p1@127.0.0.1)15> P1 RECV( 461) ; From: P3( 123) ; Subject: uzpcxovs
(p1@127.0.0.1)15> P1 SEND( 111) ; Subject: Re:uzpcxovs
(p1@127.0.0.1)15> P1 RECV( 462) ; From: P1( 111) ; Subject: Re:uzpcxovs
(p1@127.0.0.1)15> P1 RECV( 463) ; From: P2( 122) ; Subject: xbinjlgk
(p1@127.0.0.1)15> P1 RECV( 464) ; From: P4( 108) ; Subject: lwjadohz
(p1@127.0.0.1)15> P1 RECV( 465) ; From: P3( 124) ; Subject: Re:lwjadohz
(p1@127.0.0.1)15> P1 RECV( 466) ; From: P2( 123) ; Subject: Re:Re:lwjadohz
(p1@127.0.0.1)15> P1 RECV( 467) ; From: P4( 109) ; Subject: Re:Re:Re:lwjadohz
(p1@127.0.0.1)15> P1 RECV( 468) ; From: P3( 125) ; Subject: bcmvihso
(p1@127.0.0.1)15> P1 RECV( 469) ; From: P2( 124) ; Subject: becfwlsq
(p1@127.0.0.1)15> P1 SEND( 112) ; Subject: Re:becfwlsq
(p1@127.0.0.1)15> P1 RECV( 470) ; From: P1( 112) ; Subject: Re:becfwlsq
(p1@127.0.0.1)15> P1 SEND( 113) ; Subject: ealnxdis
(p1@127.0.0.1)15> P1 RECV( 471) ; From: P4( 110) ; Subject: Re:Re:becfwlsq
```

As we can see, without jitter all messages are received ordered (same as with the other algorithms).

TOTAL 300ms Sleep, 3 seconds Jitter, 20 seconds executing



As we can see, message are received ordered (we can check this by the sequence number. In all 4 processes, the messages received are totally ordered.

4 - Open questions

Are the posts displayed in FIFO, causal, and total order? Justify why.

- ➔ For the BASIC implementation we can reach FIFO order if we set a small jitter. But when the jitter is high the messages will be delivered unsorted.

- ➔ For the CAUSAL implementation they are ordered as causal. With jitter, messages are delivered unsorted but reply messages come after original messages.

- ➔ For the TOTAL implementation they are ordered totally, even with jitter messages are displayed in the same order for all nodes.

5 - Personal opinion

In our opinion, this seminar is a good way to learn programming with Erlang sort algorithms in a distributed environment. We can force the system to sort the received messages even when there’s a high jitter. The output messages help to see clearly whether or not the implemented algorithm is working. We would recommend to use this seminar for future courses.