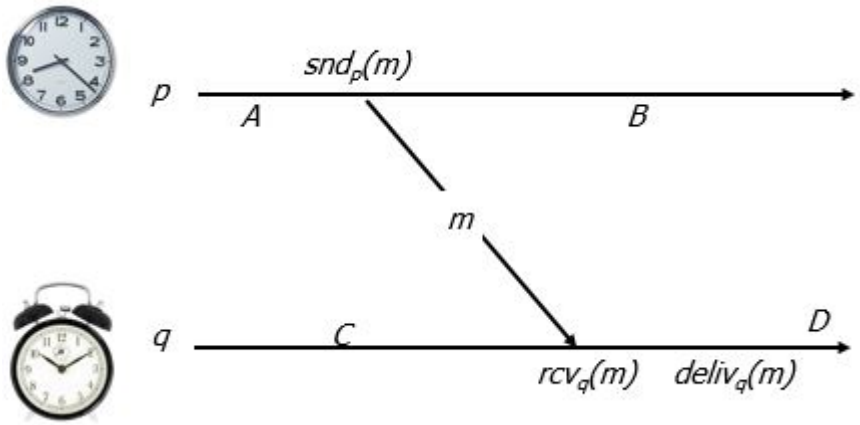
Logical Clocks - Disposition

Lukas Peter Jørgensen, 201206057, DA4

25. marts 2014

1 The happens-before relation

When you just need a well-defined ordering of events and not physical time. For example an online-forum, where we want all answers to be posted after the questions and we want all answers to be in the same order on all replicas of the forum.
That is, all we care about is whether event A *happened before* event B.



Local ordering

A happens before $snd_p(m)$ which happens before B, and C before $rcv_q(m)$ which happens before D.
This is the local ordering, at each individual process, denoted:

$$A \xrightarrow{p} snd_p(m) \xrightarrow{p} B$$

And

$$C \xrightarrow{q} rcv_q(m) \xrightarrow{q} D$$

Distributed ordering

$snd_p(m)$ happens before $rcv_q(m)$, this is the distributed ordering of the system, denoted:

$$snd_p(m) \xrightarrow{M} rcv_q(m)$$

Transitive ordering

Since $A \rightarrow snd_p(m)$ and $snd_p \rightarrow rcv_q(m)$ we get that $A \rightarrow rcv_q(m)$ and therefore $A \rightarrow D$.

Concurrent ordering

B and D are *concurrent*, even though it looks like B happens before D, there is know way of knowing since no information have flowed between the two processes.

1.1 Lamport's logical clocks

Each process holds a local counter C_p (integer) and everytime an event that matters to p happens at a process p, the process increments C_p.
When p sends m it sets:

$$C_m = C_p$$

When q receives m set:

$$C_q = max(C_q, C_m) + 1$$

From previous image:

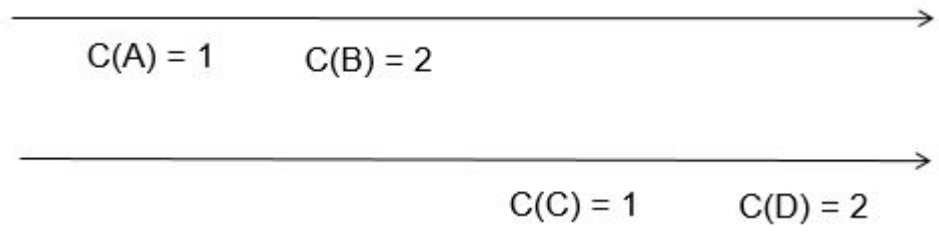
$$C(A) = 1, C(snd_p(m)) = 2, C(m) = 2, C(B) = 3$$

$$C(C) = 1, C(rcv_q(m)) = max(1, 2) + 1 = 3, C(deliver_q(m)) = 4, C(D) = 5$$

Comparison with Happens-Before

If $A \xrightarrow{p} B$ then $C(A) < C(B)$
If $A \xrightarrow{M} B$ then $C(B) = C(A) + 1$
If not $A \xrightarrow{p} B$ or $A \xrightarrow{M} B$, and there exist C such that $A \rightarrow C \rightarrow B$; induction says $C(A) < C(C)$ and $C(C) < C(B)$, so $C(A) < C(B)$.

But, $C(A) < C(B) \not\Longrightarrow A \rightarrow B$.
Because, processes that don't communicate still assign timestamps and hence events will "seem"to have an order.
For example:



So $C(C) < C(B)$ but we don't have $C \rightarrow B$.

2 Totally Ordered Multicasting

In som situations we'd want an additional requirement, that no two events ever occur at exactly the same time. To achieve this goal, we can attach the ID of the process in which the event occurs to the low-order end of the time i.e. after the decimal point. For example an event at time 40 at process P_i will be timestamped $40.i$.
With this additional requirement, Lamport's logical clocks can be used to implement totally-ordered multicasts.
P_1 writes 'a' at time 1, P_2 writes 'b' at time 1, both add them to a local queue:
$Q(P_1) = \{('a', 1.1)\}$, $Q(P_2) = \{'b'.1.2\}$
P_1 recieves 'b' which is timestamped 1.2 and P_2 recieves 'a' which is timestamped 1.1 both insert them into the queue:
$Q(P_1) = \{('a', 1.1), ('b', 1.2)\}$, $Q(P_2) = \{('a', 1.1), ('b', 1.2)\}$
They now end up with same copies of the queues, they will now send out acknowledgements for the recieved messages and they can then be sent to the application.

3 Vector clocks

Can capture causality.

Let each process P_i maintain a vector VC_i with the following two properties:
$VC_i[i]$ is the number of events that have occured so far at P_i in other words, $VC_i[i]$ is the local logical clock at process P_i. If $VC_i[j] = k$, then P_i knows that k events have occured at P_j. It is thus P_i'S knowledge of the local time at P_j

The two properties are maintained by following the three following rules, similar to lamport clocks:
When event happens at p, increment $VC_p[index_p]$.
When sending a message, set $VC(M) = VC_p$.
When receiving, set $VC_q = max(VC_q, VC(M))$ where max is max on each component of the vector.

We now know how many events at other processes have taken place before P_i sent message m, we can now make enforce causal ordering by delaying any recieved message recieved by P_j from P_i with timestamp $ts(m)$ until the following two conditions are met:

$$1.\ ts(m)[i] = VC_j[i] + 1$$

$$2.\ ts(m)[k] \leq VC_j[k] \text{ for all } k \neq i$$

1. states that m is the next message that P_j was expecting from process P_i, the second condition states that P_j has seen all the messages that have been seen by P_i.