MANCHESTER 1824

The University of Manchester

Introduction to Basic Concepts: Synchronization

Synchronization

- The term synchronisation refers to one of two distinct but related concepts: data synchronisation and process synchronisation.
- Data synchronisation is about keeping multiple copies of a dataset in coherence with one another, given that the various copies are located in different nodes.
  - For example, when you ‘sync’ a mobile device to a laptop to copy photos back and forth, this is a form of data synchronisation.
- Process synchronisation is about multiple processes needing to act together to achieve a certain overall purpose.
- Synchronisation between processes requires fast and reliable communication between the processes.
- As, in distributed systems, none of these assumptions is safe, synchronisation behaviour is often challenging.

Synchronization is needed specially when …

- Multiple processes need to agree on the ordering of events, such as whether message *m1* from process *P* was sent before or after message *m2* from process *Q*.
- Multiple processes try to simultaneously access a shared resource, such as a printer, and should, instead, cooperate in granting each other temporary exclusive access.

Deadlock can result when multiple processes are sharing the same resources.
We will talk more about deadlock later.

Synchronization Challenges in Distributed Systems

- Since nodes in a distributed system are connected via a network, and networks are not always reliable, **coordination of actions that depend on communication over a network** is quite challenging.
  - For example, the same message sent to different nodes can take a different time interval to arrive at each node; if sent multiple times to the same node, it may take different times to arrive at the node, or sometimes it may not arrive at all.
- Sending a synchronisation message from one node to another could be considered, however, because there is no way of knowing exactly how long a message is going to take to arrive at its destination, this does not represent a good solution to this problem.
- Another alternative could be to timestamp a message as its sent out, but, as there is no way of knowing how long the message took to arrive at its destination, unless the recipient's clock is in sync with the sender's clock in the first place, which cannot be not guaranteed, as nodes are independent.
- In fact, about two nodes with independent clocks, the only thing that is certain is that **a message will not arrive at its destination before it is sent**.

Example 1: Obtaining the Ordering of Events

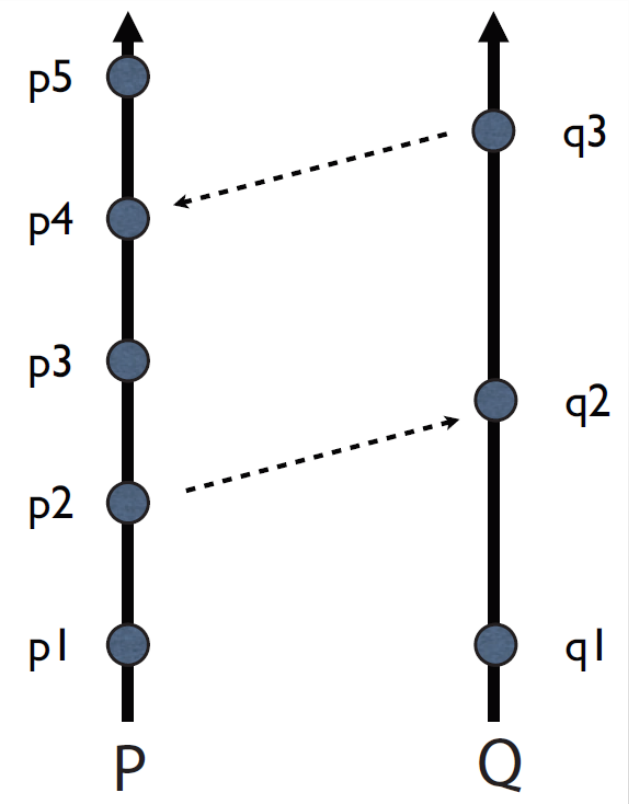
Assumptions:

1. P and Q are two distinct processes running on different nodes, each executing a sequence of instructions, p1 to p5 and q1 to q3, respectively.
2. The dotted lines represent the transmission of a message from one process to another.
3. P and Q do not have any shared notion of time.

What can we say about the ordering of events?

- p1 and q1 could happen in any order, i.e., between the two, we do not know which one happens first.
- We can be sure that q2 happens after p2, because q2 can only execute after receiving a message from p2. But, we cannot know how long after.
- We cannot say anything about p3 and q2. It could be that p3 is very fast, and happened immediately after p2, and before q2. Or it could be that there was a long delay after p2 and that it happened after q2. We do not know.
- In spite of the apparent 'visual' ordering, we know p4 actually happens after q3, because a message was sent from q3 and received at p4.
- We can safely say that p5 happens after q3, since it happens after p4, which we know happens after q3.

The sending and receiving of messages between components gives us an implicit partial ordering of events whether we intend it or not, because a message never arrives before being sent.



Beware of the following:

- If *a* and *b* are events in the same process, and *a* occurs before *b*, then *a -> b* is true.
- If *a* is the event of a message being sent by one process, and *b* is the event of the message being received by another process, then a -> b is also true. A message cannot be received before it is sent, or even at the same time it is sent, since it takes a finite, nonzero amount of time to arrive.
- *Happens-before (->)* is a transitive relation, so if *a -> b* and *b -> c*, then a -> c.
- If two events, *x* and *y*, happen in different processes that do not exchange messages (not even indirectly via third parties), then *x -> y* is not true, but neither is *y -> x*. These events are said to be concurrent, which simply means that nothing can be said (or need be said) about when the events happened or which event happened first.

This is read "event a happens before event b".

Logical Clocks

- Logical clocks take advantage of the fact that an implicit partial ordering of events can be obtained from the simple sending and receiving of messages between processes in a distributed system. As such, they do not measure "real time" but, instead, provide a distributed incremental pseudo time to events in a distributed system.
- A simple example is *Lamport's* logical clock. It assumes that each processor *i* has a Logical Clock, LCi.

Lamport's Logical Clock

1. When an event occurs on processor i, LCi is incremented by one.
2. When processor X sends a message to Y, it also sends its Logical Clock, LCx.
3. When Y receives the message, if its local Logical Clock is already in advance of the clock it has just received plus one timestep, it keeps its current Logical Clock, otherwise it sets its local Logical Clock to be the one it has just received plus one timestep, i.e.:

```
if LCy < (LCx + 1):
    LCy = LCx + 1
```


Notes on Lamport's Logical Clock

Note 1: the receipt of a message forces the recipient to move its clock forward so that the "happened after" relationship is preserved at that point.

Note 2: if a -> b then it is true that LCa < LCb. But it is not necessarily true that just because LCa < LCb then a -> b. This means that we cannot infer a causal relationship just by looking at timestamps.

Note 3: by using logical clocks, we can obtain a basic partial ordering of events, i.e., we can tell that one event happened after another one, but we do not obtain a perfectly synchronised global time.