Workshop Report

CS 341 - Distributed Information Systems

Autumn 2012

Paper: Leslie Lamport;

"Time, Clocks, and the Ordering of Events in a Distributed System"

Author: Dominic Bosch

Table of Contents

Research Field & Setting

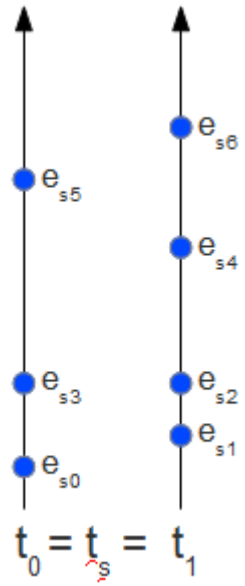
About the paper

The paper was published 1978 by Leslie Lamport in the “Communications of the ACM” (Association of Machinery). “Communications of the ACM” is still a large and prestigious platform for publishing papers. The paper was written nine years after the ARPA Net was manifested and its influence on Lamport's research is noticeable. Even though this influence might mostly have happened through the note he received and inspired him to write this paper.

By Lamport's own words[1], the origin of his paper was the RFC 677[2] from 1975, titled “The maintenance of Duplicate Databases” by Paul Johnson and Bob Thomas. They introduced message timestamps in a distributed algorithm. Lamport caught their intentions in terms of special relativity and pointed out that the algorithm they designed permitted anomalous behaviour that violated causality. He wrote down his findings, by building his logic up based on special relativity, saying that *“Thomas and Johnson didn't understand exactly what they were doing [...]“* [1](!), thus permitting anomalous behaviour in their implementation. He generalized his thoughts with the intention of providing means for any distributed algorithm. His findings moreover included the introduction of distributed state machines per process of the distributed algorithm in order to maintain synchronization.

The Basic Setup

Given a single process, total ordering of events happening in this process is trivial. If there are multiple processes running in parallel and require synchronization, it is necessary to have means to total order all events in terms of the system. Such local orderings are always only partial in terms of the whole system. And since two physical clocks are never running perfectly in parallel it is not possible to say that the measured physical times t_0 and t_1 of the two processes are always equal to the overall system time t_s. Lamport provides the means to ensure the equality in the picture shown on the right side holds, at least in the instants where it is of importance. When synchronization is required, messages with timestamps need to be exchanged in order for the processes to be total ordered in that instant.



Drawing 1: Sample time lines and several events in two processes

1 http://research.microsoft.com/en-us/um/people/lamport/pubs/pubs.html

2 http://tools.ietf.org/html/rfc677

To combine the ordering of events in processes and between processes, Lamport introduces the happened "before relation →". It is clear which events happened before the other in one process. For events in different processes it is only possible to say they happened before each other if one process was the sending of a message and the other one was the receiving of a message. And of course the happened before relation is transitive. Where ever it is not possible to say which event happened before the other, these events are considered to be concurrent.

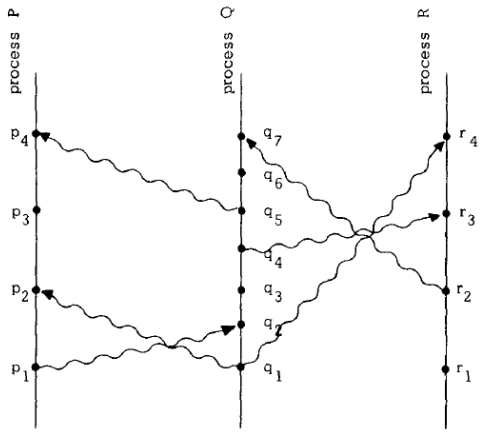
These rules for the "happened before" relation are written as shown below:

Definition: "→" is the smallest relation satisfying:

i. If events *a* and *b* are in the same process and *a* comes before *b*, then $a \rightarrow b$
ii. If *a* and *b* in different processes and *a* sends a message to *b*, then $a \rightarrow b$
iii. If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$

Two distinct events *a* and *b* are <u>concurrent</u> if $a \nrightarrow b$ and $b \nrightarrow a$

An example of how this ordering works is shown with helps of drawing 2. The curvy arrows denote messages exchanged between two processes, where the event at the start of the arrow is the event of sending the message and the event at the other end is the receiving of that message.

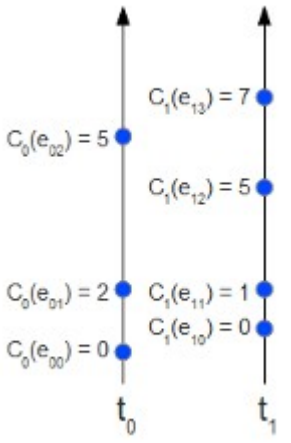


Drawing 2: Example communication between three processes

It is trivial to recognize p_1 happened before p_2. We also saw that the sending of the message p_1 to q_2 indicates $p_1 \rightarrow q_2$. Using the transitive rule of the happened before relation we are able to say that $p_1 \rightarrow r_4$ since $p_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow r_3 \rightarrow r_4$. But if we remove the message going from $q_4 \rightarrow r_3$, we broke an important link in this path and $p_1 \rightarrow r_4$ doesn't hold any longer. Moreover since $p_1 \nrightarrow r_4$ and $r_4 \nrightarrow p_1$ the two events are now considered concurrent, like p_3 and q_3. Events such as p_3 and q_3 are concurrent since no message was sent from or after one of these events, which arrived before or at the other event.

Logical Clocks

Lamport then introduces logical clocks in order to have a function that allows the comparison of timestamps between different events. There are only few requirements to logical clocks, thus there are numerous possible implementations. These logical clocks are used as a local instance, which apply a function to each event, happening in the process the clock sits. The function applied to the events returns the timestamp of the event. Now what Lamport's approach is going to lead to is a virtual overall system clock, that assigns different timestamps to every event happening in the system and thus provides a total ordering.



Drawing 3: Example implementation of increasing tick counters as local logical clocks

In the paper local clocks are identified by C_i and the overall system clock by C. What was just introduced, combined with the happened before relation leads to Lamport's clock condition:

<u>*Clock condition:*</u> if $a \rightarrow b$ then $C(a) < C(b)$ for any given a, b in the system. This condition is satisfied, if:

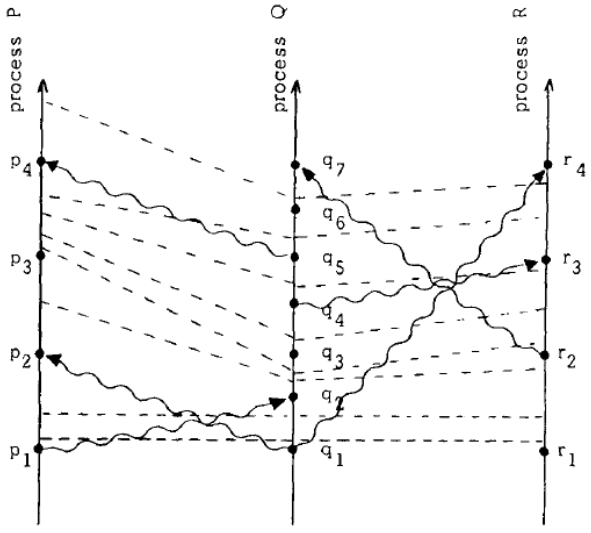
i. C1) a comes before b and both are in the same process P_i, then $C_i(a) < C_i(b)$
ii. C2) a is the sending of a message by P_i and b is the receipt of that message by P_j, then $C_i(a) < C_j(b)$

Looking at our example in Drawing 1, we already guess what two possible logical clocks could produce; something like in Drawing 3. We use an increasing tick counter, which means it is only allowed to increase, never to decrease, and between events it needs to increase at least one tick.

To visualize what the clock conditions mean in terms of a synchronized system, Lamport enhanced Drawing 2 with tick lines that reflect the required behaviour, which is shown in Drawing 4.

In other words the clock conditions mean between events within the same process there needs to be a tick line (C1) and every message needs to cross a tick line (C2).

Lamport provides implementation rules for such logical clocks, which were likely the improvements he suggested to Thomas and Johnson, though he didn't state exactly that.



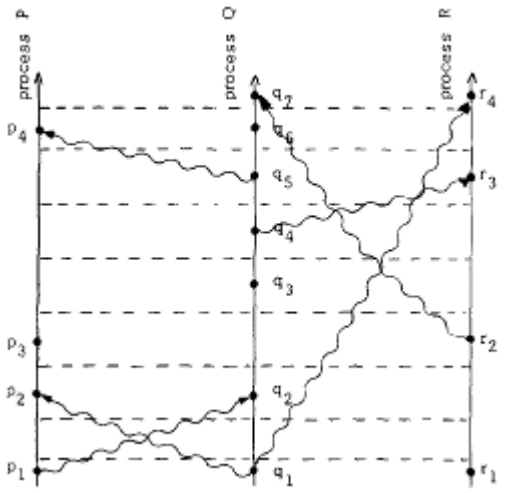
Drawing 4: Tick lines show where logical clocks are increased, thus fulfilling the clock conditions

Implementation rules for logical clocks:

i. Each local clock C_i increments between two events

ii. Messages from Process P_i to P_j need to contain a time stamp T_m. After receiving a message, the counter C_j is set bigger than T_m or is incremented if it is already bigger

These implementation rules lead to guaranteed logical clock systems in terms of the whole system. They are easier to be perceived if the tick lines in Drawing 4 are straightened, as shown in Drawing 5.



Drawing 5: Straightening the tick lines to get a logical view on the system and the occurring events and messages

Any system of clocks, satisfying the clock condition, can be used to place a total ordering on the set of all system events. Since there are numerous ways to implement logical clock systems, such a total ordering is not unique. This gets more clear as Lamport introduces an arbitrary ordering of the processes denoted by "<". He does that in order to account for the implementation rule number two, and thus completes the "happened before" relation into a total ordering "⇒" and the following notation:

If event *a* happens in process P_i and *b* in P_j, then $a \Rightarrow b$ if:

i. $C_i(a) < C_j(b)$, or

ii. $C_i(a) = C_j(b)$ and $P_i < P_j$

Mutual Exclusion Example

Having defined all the theoretical requirements in the paper up to this point, Lamport decided to come up with an example solution to the most basic problem in synchronization: mutual exclusion of a shared resource.

Lamport introduces a distributed algorithm that doesn't need a central scheduling process. Three conditions are needed for the solution to be correct:

i. Processes need to release the resource before access can be granted to another process

ii. Requests to the resource need to be granted in the order they arrived

iii. If every process which is granted the resource, eventually releases it, then every request is eventually granted the resource

In order to ensure every process has the same knowledge about the shared resource, Lamport introduces state machines in each of these processes. They maintain all resource request messages and thus hold the same total ordering like all other state

machines in the system. Requests to the resource need to be sent to all processes in the system in order to ensure the same total ordering in all state machines. This synchronized all other local clocks and since they all send an acknowledgement back, the initiating process is then also synchronized after receiving it. After this total synchronization all processes are able to determine which process is granted access to the resource. Then again after using the resource, the process needs to send a resource release message to all processes.

Anomalous Behaviour & Physical Clocks

As kind of an afterthought he decided to look what this logical attempt means for synchronization of real-time clocks. He was himself surprised as how difficult it was to prove the theorem he constructed thereafter in his paper.

Basically with anomalous behaviour he means the possibility of communication channels that are not within the system and thus bias the order of granting resources to processes. He makes the example of a guy calling his friend, telling him to issue a request on a shared resource. As they talk or shortly afterwards the first person issues such a request himself, before the other one was able to do so. In terms of the even bigger system, the person who was called by phone should be granted the resource. But since the caller was faster in issuing the request to the resource within the subsystem, he will be granted access by the total ordering of the subsystem.

He therefore introduces a so called “strong clock condition” which takes every event from the whole system into account.

A citation from the paper introduces a quite lengthy part of mathematical notation and proofs: *“One of the mysteries of the universe is that it is possible to construct a system of physical clocks which, running quite independently of one another, will satisfy the Strong Clock Condition”.*

Assuming $C_i(t)$ is a continuous differentiable function of t except for isolated jump discontinuities where the clock is reset. Since we require a true physical clock, it must run at approximately the correct rate, which means $dCi(t)/dt \approx 1$ for all t. This means we want following condition to be satisified:

(PC1): There exists a constant $\kappa \ll 1$, such thath for all i: $|dC_i(t)/dt - 1| < \kappa$

For typical crystal controlled clocks $\kappa \leq 10^{-6}$. Additionally since they need to be synchronized there must be a small constant ε such that

(PC2): for all i,j: $|C_i(t) - C_j(t)| < \varepsilon$.

This means that the variation in height of a single tick line is less than ε.

Let μ be a number such that if event a occurs at physical time t and event b in another

process happened after a ($a \rightarrow b$), then be occurs later than $t + \mu$. This means μ is less than the shortest transmission time for interprocess messages. It is always possible to choose μ as equal to the shortest distance between processes divided by the speed of light. Though normally μ can be significantly larger depending on the way of transmission of messages.

Together with above assumptions $C_i(t+ \mu) - C_j(t) > 0$, if $\varepsilon/(1-\kappa) \leq \mu$ means that it is impossible that anomalous behaviour happens.

If we then assume message m is sent at time t and received at time t', then the message delay is $v_m = t' - t$. v_m is not known to the receiving process but it is able to assume a minimum delay time $0 \leq \mu_m \leq v_m$. $\Xi_m = v_m - \mu_m$ is then the unpredictable delay of message m.

After those prerequisites we are able to define the earlier seen implementation rules for physical clocks, IR'1 and IR'2:

i. For each I, if Pi does not receive a message at time t, then C_i is differentiable at t and $dC_i(t)/dt > 0$.

ii. (a) If P_i sends a message m at time t, then m contains timestamp $T_m = C_i(t)$

    (b) Upon receiving m at time t', P_j sets $C_j(t')$ equal to $\max(C_j(t'), T_m + \mu_m)$

In order to maintain PC1 κ needs to be small enough so different events within the same process receive different timestamps.

We assume the system of processes is a directed graph. An arc from P_i to P_j represent a communication line between the two processes through which messages are sent directly. A message is sent over this arc every τ seconds if for any t, P_i sends at least one message to P_j between physical times t and $t + \tau$. The diameter of the directed graph is the smallest number d such that for any pair of distinct processes P_j, P_k, there is a path from P_j to P_k having at most d arcs.

Lamport then comes up with his theorem that bounds the length of time it can take the clocks to become synchronized when the system is first started:

Assume a strongly connected graph of processes with diameter d which always obeys rules IR'1 and IR'2. Assume that for any message m, $\mu_m \leq \mu$ for some constant μ, and that for all $t_0 \leq t$: (a) PC'1 holds.

(b) There are constants τ and Ξ such that every τ seconds a message with an unpredictable delay less than Ξ is sent over every arc. Then PC'2 is satisfied with $\varepsilon \approx d(2 \kappa \tau + \Xi)$ for all $\tau d + t_0 \leq t$, where the approximations assume $\mu + \Xi \ll \tau$.

This theorem goes beyond the important findings that were made earlier in the paper, but still it is interesting to follow his thoughts into the physical clocks. The proof of this theorem is in the appendix of the paper and will not be reproduced in this report.

Paper Discussion

Author

Lamport provides basic research on synchronization between arbitrary distributed processes through his paper. It is very conceptual and based on the theory of relativity which gives it an additional value, since the foundations of his conclusions derive from the physics of time. It is thus possible and even necessary to apply his findings to everything in the universe that requires synchronization, not only computer science. The research paper is well written and quite intuitive most of the time. There are some very simple examples included which makes it easy to grasp the essence of his points, without distracting from the very important value of the latter. Additionally there were some sentences in his paper and other resources which cheered the author. It also shows that he had quite some confidence in his own findings, which seems to be justified up to now.

After The Presentation

First Question

There was a confusion in the drawings from the paper about what are “happened before” relations and what are messages.

Referring to drawing 2, Lamport denoted messages as curvy lines between processes. According to the definition of the “happened before” relation, messages are “happened before” relations, but not only. There is also a “happened before” relation between two subsequent events in one process. And because of the transitive rule there are even many more “happened before” relations implicated in this drawing.

Second Question

“Is synchronization still a problem in times of very precise clocks which are also supported by GPS, regarding worldwide communication.”

It didn't loose importance since the essence of his findings is still present in all realizations of synchronizations. Taking precise physical clocks into account it might be the usual case that no equal time stamps are to be found anymore during synchronization in distributed systems, that communicate over large distances. But this only means that the total ordering of processes is unlikely to be required these days, it should though still be there as a backbone if it would happen. The total ordering of events in a geographically far distributed system is still required, it just became trivial.

If we take multi-cores or compute centres into account, Lamport's paper gets immediately important again, since the same problems arise there, just in smaller scale

than in the 1970ies.

Third Question

“Regarding slide 9, can we say if $C_i(a)<C_j(b)$ that *a* happened before *b*?”

Not in this example since here local logical clocks are used to return timestamps. If we would refer to the virtually constructed overall system clock *C*, then we could say that.