Lecture notes: Studying distributed systems – The notion of time

M2 MOSIG: Large-Scale Data Management and Distributed Systems

Thomas Ropars

2023

These notes discuss the notion of time in distributed systems[1].

1 Asynchronous systems

A distributed system can be seen as an asynchronous system. It means that we make no timing assumptions about processes and links. In an asynchronous system:

- there is no bound on the transmission delay of messages;
- there is no bound on the relative speed of processes.

Such a system is used to model unpredictable load on the network and on the CPU.

We will see later in the course that some problems cannot be solved if you do not make any additional assumptions about time. But for now, we can start by wondering if, even without any synchronized physical clocks and no assumption on time, we can have a measure of the progression of time?

2 Logical time

In the previous lecture, we introduced the *happened-before* relation to capture causal relations between events in a distributed system. We rephrase the question above as follows: Is it possible to time-stamp the events of a distributed computation such that the happened-before relation can be inferred? In other words, if $\mathrm{TS}(e)$ denotes the time-stamp of some event e, is it possible to satisfy the following property:

$$e \rightarrow e' \iff \mathrm{TS}(e) < \mathrm{TS}(e').$$

We first introduce time-stamps that satisfy only $e \rightarrow e' \implies \mathrm{TS}(e) < \mathrm{TS}(e')$. These time-stamps are called *logical (scalar) clocks* or *Lamport clocks*. Then we introduce time-stamps that satisfy $e \rightarrow e' \iff \mathrm{TS}(e) < \mathrm{TS}(e')$. These time-stamps are called *(logical) vector clocks*. Logical (scalar) clocks and vector clocks are used, either explicitly of implicitly,[2] in several distributed algorithms.

[1] Acknowledgments: Parts of these notes are strongly inspired by the lectures notes of Andre Schiper on *Distributed Algorithms*.

[2] The basic mechanism of their implementations is used.

2.1 Logical scalar clocks (Lamport clocks)

The property $e \to e' \implies \mathrm{TS}(e) < \mathrm{TS}(e')$ can be ensured with the logical clocks defined by Lamport [1]. The time-stamps of event e will be denoted by $LC(e)$, and the logical clock of process p_i will be denoted by LC_i. The events on p_i are time-stamped using LC_i according to the following rules:

- The initial value of LC_i is 0 for all processes
- For any internal event on process p_i, $LC_i = LC_i + 1$
- When process p_i sends message m, $LC_i = LC_i{+}1$, and the value of the logical clock is attached[3] to message m. It means that if $ts(m)$ is the time-stamp on message m, $ts(m) = \mathrm{LC}(e_i^k)$, where $e_i^k \equiv \mathrm{send}(m)$
- When process p_j receives message m, $LC_j = \max(\mathrm{LC}_j, ts(m)) + 1$

It can be shown that the following property holds: $\forall$ events e, e': $\quad e \to e' \Rightarrow \mathrm{LC}(e) < \mathrm{LC}(e')$.

Figure 1 presents an example of execution where all events are timestamped using Lamport clocks.

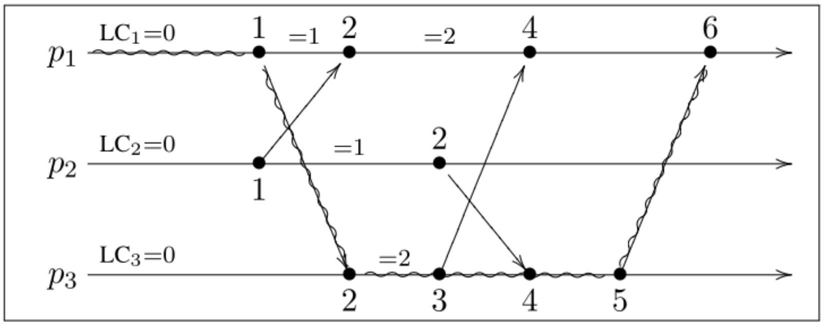

Figure 1: Example of execution with Lamport Clocks

After occurrence of event e_i^j on p_i, the logical clock of p_i is updated: $\mathrm{LC}_i := \mathrm{LC}(e_i^j)$.

Note that $\quad \mathrm{LC}(e) < \mathrm{LC}(e') \not\Rightarrow e \to e'$. Take for example the event on p_1 with time-stamp 2 and the event on p_3 with time-stamp 3 in Figure 1.

Remark LC(e) is equal to the length of the *longest causal chain* ending at event e.
Example: $LC(e_1^4) = 6$. Longest causal chain: $e_1^1 \to e_3^1 \to e_3^2 \to e_3^3 \to e_3^4 \to e_1^4$.

2.2 Logical vector clocks

Vector clocks, proposed independently by Mattern and by Fidge in 1988, satisfy the property $e \to e' \iff \mathrm{TS}(e) < \mathrm{TS}(e')$. The time-stamp of event e will be denoted by VC(e), and the vector clock of process p_i will be denoted by VC_i.

[3] We also say "*piggybacked*".

VC(e) is a vector of size n. For some event e_i occurring at process p_i, the time-stamping rules ensure the following property:

- For $i = j$, $\mathrm{VC}(e_i)[j]$ = number of events on p_i up to and including e_i.
- For $i \neq j$, $\mathrm{VC}(e_i)[j]$ = number of events on p_j that *happened before* e_i.

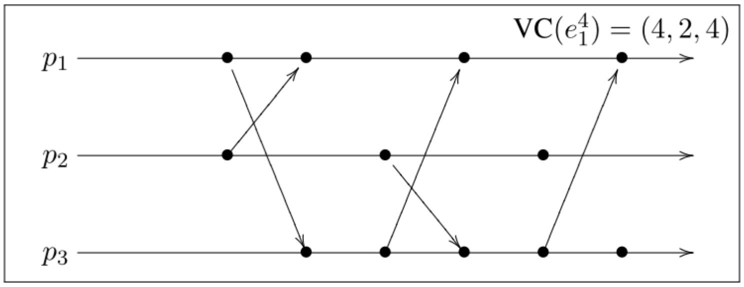

Figure 2: Illustration about Vector Clocks

Consider e_1^4 in Figure 2, with time-stamp $(4, 2, 4)$. We can say that:

- 4: e_1^4 is the fourth event on p_1;
- 2: Two events on p_2 happened before e_1^4;
- 4: Four events on p_3 happened before e_1^4.

The events on p_i are time-stamped using VC_i according to the following rules:

```
if $e_i$ is an internal event or send($m$) then
    $\forall j \neq i, \quad \mathrm{VC}(e_i)[j] = \mathrm{VC}_i[j]$
    $\mathrm{VC}(e_i)[i] = \mathrm{VC}_i[i] + 1$
else
    {$e_i$ is a receive event: message $m$ with timestamp $ts(m)$}
    $\mathrm{VC}(e_i) = \max(\mathrm{VC}_i,\ ts(m))$ [4]
    $\mathrm{VC}(e_i)[i] = \mathrm{VC}(e_i)[i] + 1$
end if
```

Similarly to Lamport clocks, $ts(m)$, the time-stamp piggy-backed on message m, is defined as the time-stamp of the *send*(m) event: $\mathrm{TS}(m) = \mathrm{VC}(e_i^j)$, where $e_i^j \equiv \mathrm{send}(m)$.

Similarly to Lamport clocks, after occurrence of event e_i^j on p_i, the vector clock of p_i is updated: $\mathrm{VC}_i := \mathrm{VC}(e_i^j)$.

We consider the relation $<$ on vectors, defined as usual:

$$\begin{array}{lll} \mathrm{VC}(e) < \mathrm{VC}(e') & \Leftrightarrow & \forall i \quad \mathrm{VC}(e)[i] \leq \mathrm{VC}(e')[i] \quad \textit{and} \\ & & \exists j \quad \mathrm{VC}(e)[j] < \mathrm{VC}(e')[j]. \end{array}$$

It can be shown that vector clocks indeed ensure the following property:

$$\mathrm{VC}(e) < \mathrm{VC}(e') \Leftrightarrow e \rightarrow e'.$$

[4] The *max* of two vectors is computed element by element.

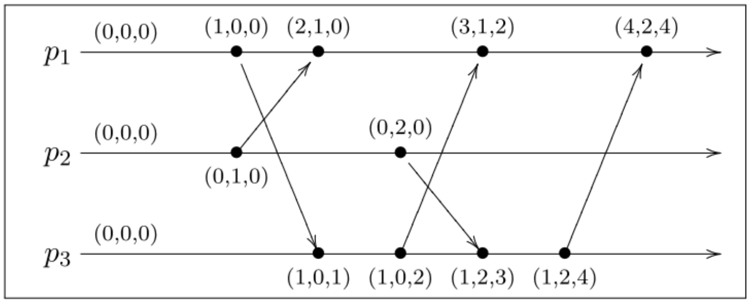

Figure 3: Example of execution with Vector Clocks

References

[1] L. Lamport. Time, clocks, and the ordering of events in a distributed system. In *Concurrency: the Works of Leslie Lamport*, pages 179–196. 2019.