# About logical clocks for distributed systems

Michel RAYNAL
IRISA
Campus de Beaulieu
35042 Rennes-Cédex, FRANCE
raynal@irisa.fr

**Abstract**

Memory space and processor time are basic resources when executing a program. But beside this implementation aspect (this time resource is necessary but does not belong to the program semantics), the concept of time presents a more fundamental facet in distributed systems namely causality relation between events. Put forward by Lamport in 1978, the logical nature of time is of primary importance when designing or analyzing distributed systems. This paper reviews three ways (linear time, vector time and matrix time) which have been proposed to capture causality between events of a distributed computation and which consequently allow to define logical time.

Key words : distributed systems, causality, logical time, happened before, linear time, vector time, matrix time.

## 1 Introduction

To be executed a program needs some memory space and some processor time. But time cannot be restricted to this resource aspect. As put forward by Lamport in a famous paper [9], time establishes causal dependencies on the events produced by a program execution. So in a distributed system composed of $n$ sites connected by communication channels, first : events on each site are totally ordered (events are sendings of messages, receipts of messages or internal events i.e. events not involving messages) ; second : for each message the sending event precedes the corresponding receiving event. The transitive closure of these precedence relations (sometimes called "happened before") defines a causal dependence relation : "$\rightarrow$" on the set of events produced by a distributed execution ; this relation is a partial order. In the figure 1 (a point represents an event, and an arrow a message transfer) for example we have $a \rightarrow b$. The set of all events $x$ such that for a given event $b$ we have $x \rightarrow b$ is called the causality cone of $b$, in short *cone(b)*. Finally two events $x$ and $y$, such that neither $x \rightarrow y$ nor $y \rightarrow x$, are said to be independent or concurrent, in short $x \parallel y$ (see figure 1).

In this paper we review timestamping mechanisms that allow to associate dates to relevant events. More precisely these dates must rely on a logical global time in order to be able to compare related events produced by distinct sites and must be consistent that is to say obey the monotony property : if $a \rightarrow b$ then the date associated to $b$ must be, with respect to the logical global time, after the date associated to $a$.

This review presents 3 timestamping mechanisms. The first one is the well known linear time, proposed by Lamport, that uses ordinary integers to represent time ; the second one uses $n$-dimensionnal vectors of integers and the third one uses $n \times n$ matrices. In order to ensure the monotony property all the timestamping mechanisms, that build a representation of time, obey a common pattern made of data structures and of a protocol (rules to manage these data structures).

i) Data structures to represent logical time.
Each site is endowed with local variables that allow it :

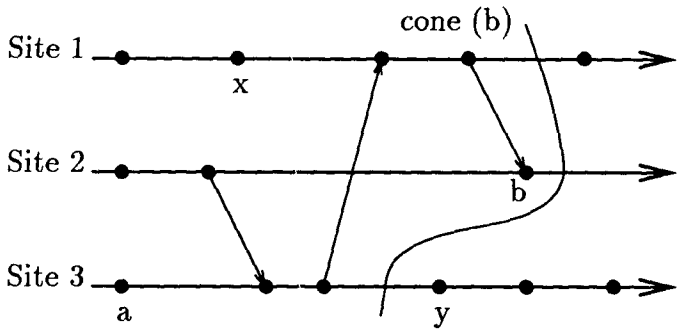


Figure 1: A distributed execution

- on the one hand to measure its own progress ; that is done with the help of a *logical local clock* (updated by rule *R1*).
- on the other hand to have a good representation of the *logical global time* ; this representation (updated by rule *R2*) allows it to timestamp events ; that is a local view of the global time.

ii) A protocol ensuring that the logical local clock and the local view of the global time of each site are managed consistently with the causality relation "$\rightarrow$". That is done by the two following rules.

- *R1* : before producing an event (sending, receiving or internal) a site has to increase its logical local clock (because it is progressing).
- *R2* : for the date (that is to say a timestamp with respect to the logical global time) of a receiving event be after the date of the corresponding sending event, every message $m$ piggybacks the value of the logical global time as perceived by the sender at sending time ; this allows the receiver to update its view of the global time. Then it execute *R1* and can timestamp the receiving event.

For each of these timestamping systems we first show how the fundamental monotony property is ensured (i.e. implementation of rules *R1* and *R2*), and then some properties of the associated time representations are given. (Actually properties attached to each of these timestamping mechanisms are immediate consequence of the monotony property on the way they represent time with integer, vector or matrix).

As this paper is essentially a survey, we are faced to the problem to quote original proposals. This is a very difficult task ; the references used are the ones known by the author ; if they are not the right ones, please let him know. However -as events in a distributed computation !- very similar proposals can be indepedent.

# 2 The linear time

## 2.1 The timestamping mechanism

This time representation is the well-known one, proposed by Lamport in 1978 in his seminal paper [9]. Time domain is the set of integers. Each site $S_i$ is associated an integer variable $h_i$ holding increasing values. The logical local clock of $S_i$ and its local view of the global time are here mixed up and represented by the only variable $h_i$. Rules *R1* and *R2* defining the consistency protocol are the following ones :

- *R1* : before producing an event (sending, receiving, internal) :

$$h_i := h_i + d \qquad (d > 0)$$

(each time *R1* is executed $d$ can have a different value).

- *R2* : when it receives the timestamped message *(m,h)* the site $S_i$ executes first the update :

$$h_i := max(h_i, h)$$

and then *R1*, before delivering the message $m$.

## 2.2 Properties

In addition to the monotony property it is possible to use this timestamping mechanism to build a total order "*t-before*" on the set of events, consistent with the causality relation "$\longrightarrow$". The timestamp of an event is then composed of its occurrence date and of the identity of the site that produced it. So if we consider two events $x$ and $y$ timestamped respectively by *(h,i)* and *(k,j)* the total order is defined by :

$$x \text{ } t\text{-}before \text{ } b \Longleftrightarrow (h < k \text{ or } (h = k \text{ and } i < j))$$

This total order is due to Lamport [9] ; it is generally used to ensure liveness properties in distributed algorithms.

If we consider that the increment value $d$ is always 1, we have the following very interesting property. Let $e$ be an event timestamped $h$. Then *h-1* represents the minimum logical duration, counted in units of events, required before producing the event $e$ [5] ; we call it the height of the causality cone associated to the event $e$, in short *height(e)*. In other words *h-1* events have been produced sequentially before the event $e$ regardless of the processes that produced these events. (In figure 2, 6 events precedes $b$ on the longest causal path ending in $b$).

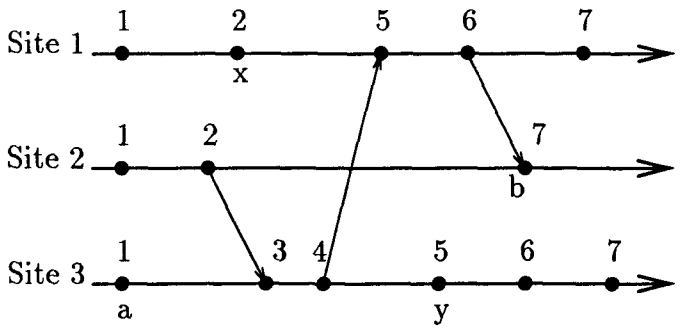


Figure 2: Lamport's clocks progress

# 3 Vector time

## 3.1 Vector clocks

Here the logical global time is represented by an $n$-dimensionnal vector. Each site $S_i$ is endowed with such a vector $vt_i[1..n]$. The idea embedded in such a vector is the following one, on a site $S_i$ :

- $vt_i[i]$ describes the logical time progress of the site $S_i$, considered alone ; that is the logical local clock of $S_i$. This variable holds increasing values locally generated. (Such a local variable can only be increased by rule *R1*).
- $vt_i[j]$ represents site $S_i$ knowledge of site $S_j$ local time progress. It is a local image of the value of $vt_j[j]$ ; it is updated by rule *R2*.
- the whole vector $vt_i$ constitutes the $S_i$ local view of the logical global time used to timestamp events.

The two rules *R1* and *R2* are the following ones for each site $S_i$ :

- *R1* : before producing an event :

$$vt_i[i] := vt_i[i] + d \qquad (d > 0)$$

- *R2* : each message $m$ piggybacks the vector clock $vh$ of the sending site at sending time. When receiving such a message *(m,vh)*, the site $S_i$ first updates its knowledge of the local times progress :

$$1 \leq k \leq n \ : \ vt_i[k] := max(vt_i[k], vh[k])$$

and then it executes *R1*.

The date associated to an event is now the value of the vector clock of the producing site at the time the event is produced. Figure 3 shows an example of vector clocks progress with the increment value *d=1*.

Such clocks have been introduced and used by several authors. Parker *et al.* used in 1983 a very rudimentary vector clocks system to detect inconsistencies of duplicated data due to partitionning [13]. Liskov and Ladin proposed a vector clock system to define highly available distributed services [10]. But the theory associated to these vector clocks has been developped in 1988 independently by Fidge [5,24], Mattern [11] and Schmuck [20]. Similar clocks systems have also been proposed and used by Strom and Yemini [21] to implement an optimistic recovery mechanism, and by Raynal to prevent drift between logical clocks [15].

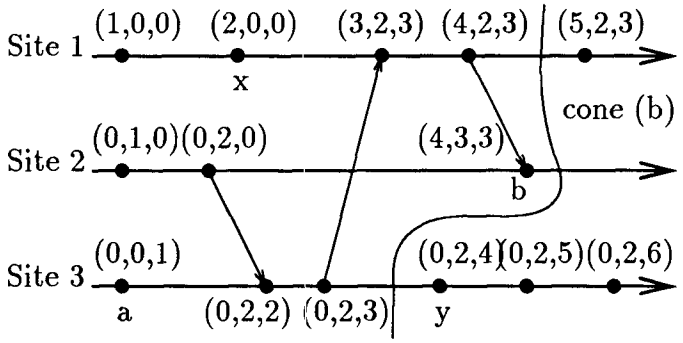


Figure 3: Vector clocks progress

## 3.2 Properties

These properties have been established in [5,11,20]. Moreover it has been shown in [23] that dimension of vectors cannot be less than $n$.

**An interesting isomorphism.**
Let us define the following tests on vectors :

$$\begin{aligned} vh \leq vk &\iff \forall\, x : vh[x] \leq vk[x] \\ vh < vk &\iff vh \leq vk \textbf{ and } \exists\, x : vh[x] < vk[x] \\ vh \parallel vk &\iff \textbf{not } (vh < vk) \textbf{ and not } (vk < vh) \end{aligned}$$

If we consider the partially ordered by "$\rightarrow$" set of events, that are produced by a distributed execution and timestamped by the vector clock system we have the following property. Let two events $x$ and $y$ timestamped respectively by $vh$ and $vk$, then :

$$\begin{aligned} x \rightarrow y &\iff vh < vk \\ x \parallel y &\iff vh \parallel vk \end{aligned}$$

In others words there is an isomorphism between the set of partially ordered events produced by a distributed computation and their timestamps. If we consider occurrence sites of events, the independence test can be simplified. So if $x$ and $y$ are timestamped respectively by *(vh,i)* and *(vk,j)* we have :

$$\begin{aligned} x \rightarrow y &\iff vh[i] < vk[i] \\ x \parallel y &\iff vh[i] > vk[i] \textbf{ and } vh[j] < vk[j] \end{aligned}$$

These clocks have a wide variety of applications. The reader can consult the following references. They are used to implement distributed debugging [5], causal ordering communication [19], causal distributed shared memory [1] and definition of global breakpoints [6]. Similar ideas have been used [8,21] to define consistent checkpoints for optimistic recovery.

**Event counting vector clock.**
If in the rule *R1* we considerer always *d=1*, then we have the following result : $vt_i[i]$ counts the number of events produced by the site $S_i$.

So if we consider an event $e$ timestamped $vh$ we have :

| | |
|---|---|
| $vh[j]$ | = number of events produced by the site $S_j$ that causally precede $e$ |
| $\sum vh[j] - 1$ | = total number of events that causally precede $e$. We define this number to be the weight of the causality cone of the event $e$. |

In the example of figure 3, the timestamp *(4,3,3)* associated to the event $b$ indicate that 4 events located on $S_1$ precede $b$ and that the weight of the cone associated to $b$ is 9. The weight of *cone(e)* is the minimum number of events that must have occured before $e$.

## 3.3 Towards a concurrency measure for distributed computations

A simple and easily computable concurrency measure can be defined in the following way. Let $e$ be an event. We define the concurrency measure associated to $e$ as (the denominator is only used to obtain a value ranging between 0 and 1) :

$$cm(e) = \frac{n * height(e) - weight\ of\ cone(e)}{(n-1)\ height(e)}$$

This measure claims that the computation needed to produce an event $e$ is maximally concurrent (balanced and parallel) if *cm(e)=0* ; on the opposite if *cm(e)=1* the computation is entirely sequential (of course to measure the concurrency of a complete execution of a distributed program we can add a fictitious event that follows causally the last events produced by each site).

Such a measure is easily computed if we equip the underlying system with a Lamport's linear clock and a vector clock mechanisms. The height associated to an event is obtained from its Lamport's timestamp and the weight of the associated causality cone from its vector timestamp. Others measures based on vectors can be found in [25].

# 4 Matrix time

## 4.1 Matrix clock

In this case the logical global time is represented by an $n \times n$ matrix. So each site $S_i$ is endowed with a matrix $mt_i[1..n, 1..n]$ whose entries have the following meaning.

- $mt_i[i, i]$ is the logical local clock of $S_i$, increasing as the computation of the site $S_i$ progresses.
- $mt_i[k, l]$ represents the view (or knowledge) the site $S_i$ has about the knowledge by $S_k$ about the logical local clock of $S_l$. The whole matrix $mt_i$ constitutes the $S_i$ local view of the logical global time.

In fact the row $mt_i[i, .]$ is nothing else than the vector clock $vt_i[.]$ ; so this row inherits the properties of the vector clock system.

Rules *R1* and *R2* are similar to the preceding ones for each site $S_i$ :

- *R1* : before producing an event :

$$mt_i[i, i] := mt_i[i, i] + d \qquad (d > 0)$$

- *R2* : each message $m$ piggybacks a matrix time $mt$. When it receives such a message *(m,mt)* from a site $S_j$, the site $S_i$ executes (before *R1*) :

$$1 \leq k \leq n : mt_i[i, k] := max(mt_i[i, k], mt[j, k])$$
$$1 \leq k, l \leq n : mt_i[k, l] := max(mt_i[k, l], mt[k, l])$$

Such a clock system has been proposed in 1984 by Wuu and Bernstein [22] ; joined to a log system it allows to discard obsolete information (see the properties). A similar mechanism has also been used by Lynch and Sarin in 1987 for a similar purpose [18].

## 4.2 Properties

In addition to the properties of the vector clocks (when considering $mt_i[i, .]$) we have the following one :

$$\min_k(mt_i[k, i]) \geq t \Rightarrow \quad \text{site } S_i \text{ knows that every other site knows its progress till its local time } t$$

It is this property that can allow a site to no longer send an information with a local time $\leq t$ or to discard obsolete information ; to exploit this property, as said previously, the matrix time mechanism has to be used jointly with a log mechanism.

# 5 Other logical times

In [2] Awerbuch presents the *synchronizer* concept ; such a device allow to run a synchronous distributed algorithm on an asynchronous distributed system. In other words a synchronizer is an interpreter for synchronous distributed programs. Synchronous means here that the distributed program progresses logically step by step (for sites and channels) ; this progress relies on a global time assumption. From the point of view of synchronous distributed programs such a global time pre-exists and participates in their semantics. Developments about synchronizers can be found in [14, chapter 3]).

In distributed discrete event simulation a virtual time (the so-called simulation or model time) does exist and the semantics of a simulation program relies on such a time : its progress ensure that the simulation program has the liveness property. Designing a distributed simulation run-time consists in ensuring that the virtual time progresses (liveness) in such a way that causality relations of the simulation program are never violated (safety). Several implementations are possible for such run-times [7,12,17]. The logical time built by a synchronizer or by a distributed simulation run-time drives the underlying program (a synchronous or a simulation program). It has not to be confused with logical times presented previously. With the previous representations of logical time (linear, vector or matrix time) the aim is to be able to timestamp consistently events in order to ensure some properties such as liveness, consistency, fairness, etc ; so in this case logical time is only one means among others to ensure some properties. For example Lamport's logical clocks are used in the Ricart-Agrawala's mutual exclusion algorithm [16] to ensure liveness ; this time does not belong to the mutual exclusion semantics. In fact other means can exist to ensure properties such as liveness ; for example instead of a logical time the Chandy and Misra's mutual exclusion algorithm manages a directed acyclic graph to ensure liveness [4]. On the other hand the time provided by a synchronizer or a distributed simulation run-time does belong to the underlying program semantics ; this latter logical time is nothing else than the logical counterpart of the physical time offered by the environment and used in real-time applications [3].

# 6 References


[1] AHAMAD M., HUTTO Ph. W., JOHN R. *Implementing and programming causal distributed shared memory.* Proc. 11th IEEE Int. Conf. on Dist. Comp. Systems, Arlington USA, (May 1991), pp. 274-281

[2] AWERBUCH B. *Complexity of network synchronization.* Journal of the ACM, vol.32,4, (1985), pp. 804-823

[3] BERRY G. *Real time programming : special purpose or general purpose languages.* IFIP Congress, Invited talk, San Francisco, (1989)

[4] CHANDY K.M., MISRA J. *The drinking philosophers problem.* ACM Toplas, vol.6,4, (1984), pp. 632-646

[5] FIDGE L.J. *Timestamp in message passing systems that preserves partial ordering.* Proc. 11th Australian Comp. Conf., (Feb. 1988), pp. 56-66

[6] HABAN D., WEIGEL W. *Global events and global breakpoints in distributed systems.* Proc 21th Hawai ACM-IEEE Int. Conf. on System Sciences, (1988), pp. 166-175

[7] JEFFERSON D. *Virtual time.* ACM Toplas, vol.7,3, (1985), pp. 404-425

[8] JOHNSON D.B., ZWAENEPOEL W. *Recovery in distributed systems using optimistic message logging and checkpointing.* In processing 7th ACM Symposium on PODC, (1988), pp. 171-181

[9] LAMPORT L. *Time, clocks and the ordering of events in a distributed system.* Comm. ACM, vol.21, (July 1978), pp. 558-564

[10] LISKOV B., LADIN R. *Highly available distributed services and fault-tolerant distributed garbage collection.* Proc. 5th ACM Symposium on PODC, (1986), pp. 29-39

[11] MATTERN F. *Virtual time and global states of distributed systems.* Proc. "Parallel and distributed algorithms" Conf., (Cosnard, Quinton, Raynal, Robert Eds), North-Holland, (1988), pp. 215-226

[12] MISRA J. *Distributed discrete event simulation.* ACM Computing Surveys, vol.18,1, (1986), pp. 39-65

[13] PARKER D.S. *et al. Detection of mutual inconsistency in distributed systems.* IEEE Trans. on Soft. Eng., vol.SE 9,3, (May 1983), pp. 240-246

[14] RAYNAL M., HELARY J.M. *Synchronization and control of distributed systems and programs.* Wiley & sons, (1990), 124 p.

[15] RAYNAL M. *A distributed algorithm to prevent mutual drift between n logical clocks.* Inf. Processing Letters, vol.24, (1987), pp. 199-202

[16] RICART G., AGRAWALA A.K. *An optimal algorithm for mutual exclusion in computer networks.* Comm. ACM, vol.24,1, (Jan. 1981), pp. 9-17

[17] RIGHTER R., WALRAND J.C. *Distributed simulation of discrete event systems.* Proc. of the IEEE, (Jan. 1988), pp. 99-113

[18] SARIN S.K., LYNCH L. *Discarding obsolete information in a replicated data base system.* IEEE Trans. on Soft. Eng., vol.SE 13,1, (Jan. 1987), pp. 39-46

[19] SCHIPER A., EGGLI J., SANDOZ A. *A new algorithm to implement causal ordering.* Proc 3rd Int. Workshop on Dist. Algorithms, Nice, Springer-Verlag 392, (Bermond, Raynal Eds), (1988), pp. 219-232

[20] SCHMUCK F. *The use of efficient broadcast in asynchronous distributed systems.* Ph. D. Thesis, Cornell University, TR88-928, (1988), 124 p.

[21] STROM R.E., YEMINI S. *Optimistic recovery in distributed systems.* ACM TOCS, vol.3,3, (August 1985), pp. 204-226

[22] WUU G.T.J., BERNSTEIN A.J. *Efficient solutions to the replicated log and dictionnary problems.* Proc. 3rd ACM Symposium on PODC, (1984), pp. 233-242

[23] CHARRON-BOST B. *Concerning the size of logical clocks in distributed systems.* Inf. Proc. Letters, vol.39, (1991), pp. 11-16

[24] FIDGE C. *Logical time in distributed computing systems.* IEEE Computers, (August 1991), pp. 28-33

[25] RAYNAL M., MIZUNO M., NEILSEN M.L. *Synchronization and concurrency measures for distributed computations.* Research Report, IRISA-INRIA Rennes, (October 1991), 20 p.