Time and logical clocks

Paulo Sérgio Almeida

February 2023

Universidade do Minho

Physical time

Logical time

Physical time

- Time reported by our physical clocks
  - mechanical clock, digital clock, atomic clock, GPS
- One could be tempted to use physical time to make decisions
  - compare events
  - compare file versions
  - decide who gets a lock
- But clocks cannot be fully synchronized
  - comparing clocks from different nodes is dangerous
  - even a small error may have serious consequences

It is worse: there is not really a global time

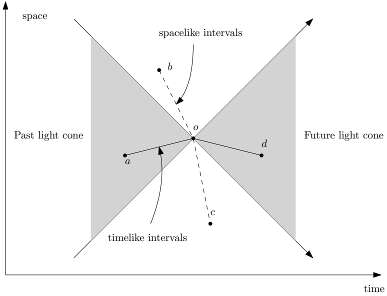
There is no global time

- From special relativity
  - time in different referentials appears to pass differently
  - each observer sees time at others passing slower
- From general relativity
  - time flows at different speeds according to gravitation
  - time passes slower near a dense object
- We could compute some *hyperplane of simultaneity*
  - to obtain comparable times at different points in space
  - like GPS does – but it is amazingly complex
  - even if very good approximation ...
  - ...we should not depend on it for correctness

Play with spacetime diagrams

`https://www.geogebra.org/m/HYD7hB9v`

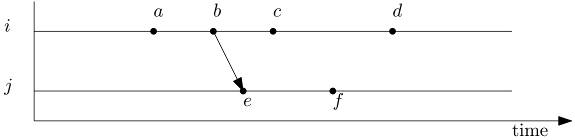
Past and future light cones



- Only events in the past light cone of o, like a, can influence o
- Only events in the future light cone of o, like d, can be influenced by o
- Events like b are *spacelike* relative to o
  - cannot influence or be influenced by o
  - comparing their times, as t_b and t_o, is meaningless
- Only comparing times of a *timelike* interval makes sense
  - i.e., compare an event with others from its past or future
  - $t_a < t_o$ and $t_o < t_d$ for all observers (reference frames)

Potential causality in distributed systems

- Light cones define the boundary of potential causality
- Distributed systems are limited to message passing
- Actual message passing defines potential causality
- Consider a single message from node i to j



- Events c and f cannot influence each other; same for d and f
- But $t_c < t_f$ and $t_d > t_f$ (assuming some reference frame)
- Event a can influence f, but like for c, $t_a < t_f$
- How can we distinguish the different roles of a and c with regards to f?

Physical time is not worth (much)

- There is really no global physical time
- Achieving aproximate global "background of time" is costly
- Comparing pysical times is not enough anyway
- We need something else, to know how causality propagates

Physical time

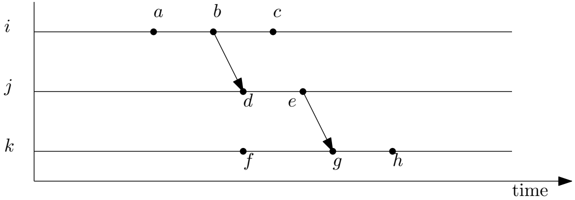
Logical time

Basic logical time properties

- Logical time is based on data updated by programs
  - such as a counter being incremented
  - without depending on any physical device
- It can use
  - local updates
  - propagation in messages
- It should mimic some properties of physical time
  - events from the past light cone should have smaller times
  - events from the future light cone should have greater times
  - comparison to other events can be meaningless

Happens-before

- Lamport defined a *happens-before* relation ($\rightarrow$), given by:
  (1) an event is related to future events from the same process
  (2) a send happens-before the respective receive
  (3) if $a \rightarrow b$ and $b \rightarrow c$, then $a < c$ (transitive closure)
- $e_1 \rightarrow e_2$ when there is a path of communication from e_1 to e_2
- In this example $a \rightarrow h$ because $a \rightarrow b$, $b \rightarrow d$, $d \rightarrow e$, ...



Two events related by happens-before define a timelike interval

Concurrent events

- Happens-before is a strict partial order
- Incomparable (unrelated) events are said to be concurrent

$$a \parallel b \Leftrightarrow a \not\rightarrow b \wedge b \not\rightarrow a$$

- Two concurrent events can define a spacelike interval
  - being physically impossible to influence each other
- Or they can define a timelike interval (but we don't know)
  - and they could influence each other by a *hidden channel*
- In a closed world assumption, where we know the full system
  - there are no hidden channels outside our algorithm
  - happens-before defined according to actual communication
  - concurrent events are as if spacelike
  - comparing their times can be meaningless

Logical clock condition

- Lamport defined a condition for any logical clock C

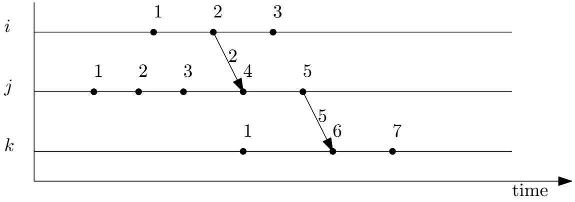
$$\text{if } a \rightarrow b \text{ then } C(a) < C(b)$$

- This is analogous to what we expect from physical clocks
  - any observer will compare timelike events in the same way
- Initially Lamport defined a clock value as a number
- It can be generalized to clock values in partially ordered sets

Lamport clocks

- A clock value is an integer, increasing along each causal path
- Each process i keeps a clock L_i
- An update (state transition) or send increments the clock
- A send attaches it to message m as L_m
- A receive of m stores one plus the maximum of clock values

$$L_i := \max(L_i, L_m) + 1$$



Advantages and limitations of Lamport clocks

- They have several advantages
  - cheap to compute, store and send in messages
  - can be used to define a total order of events

Defining a total order respecting happens-before

- assuming a total order of process identifiers
- for events a and b of processes i and j, respectively

$$a < b \Leftrightarrow L(a) < L(b) \vee (L(a) = L(b) \wedge i < j)$$

- Lamport clocks only respect, not characterize happens-before
  - comparing clocks doesn't inform whether events are concurrent
  - except when the clocks happen to have exactly the same value

Characterizing happens-before

- Many different logical clocks can respect the clock condition
- For any logical clock

$$a \to b \Rightarrow C(a) < C(b)$$

- But the reverse is not true in general

$$C(a) < C(b) \not\Rightarrow a \to b$$

- A clock characterizes happens-before when the reverse is true

$$a \to b \Leftrightarrow C(a) < C(b)$$

- Such clocks need to have values in a partially ordered set

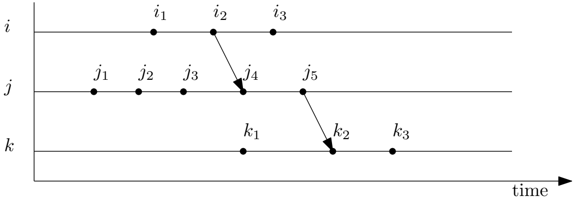
Causal history

- The causal history of an event is the set of all events that happened before it plus the event itself.

$$\mathbb{C}(e) = \{e\} \cup \{x \mid x \to e\}$$

- By definition

$$e \to e' \Leftrightarrow \mathbb{C}(e) \subset \mathbb{C}(e')$$



- Denoting i_n the n-th event of process i
- In the run, $\mathbb{C}(k_3) = \{i_1, i_2, j_1, j_2, j_3, j_4, j_5, k_1, k_2, k_3\}$

Vector clocks

- Causal histories are not to be used in practice
- But serve to reason about actual mechanisms (clocks)
- Looking at the previous example, to encode $\mathbb{C}$
  - if we number events sequentially in each process
  - it is enough to store the last event number from each process

$$V(k_3) = \{i \mapsto 2, j \mapsto 5, k \mapsto 3\}$$

  - or if we number processes, simply a vector $[2, 5, 3]$

> A *vector clock* is a vector (if processes are numbered) or map (in general) that maps process identifiers to their last event number present in a causal history.

Operations with vector clocks

- All events (update, send, receive) increment self component

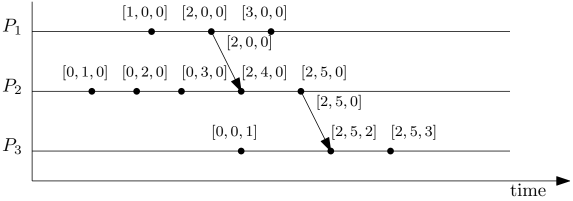
$$V_i[i] := V_i[i] + 1$$

- A send attaches the vector clock to message m as V_m
- A receive also performs the pointwise maximum:

$$V_i := \max(V_i, V_m)$$

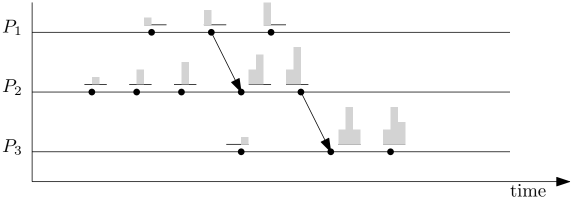
- Clocks compared by standard comparison of functions

$$V_i \leq V_j \Leftrightarrow \forall p \cdot V_i[p] \leq V_j[p]$$



Visual analogies for causal histories and logical clocks

- Over time many different logical clocks were designed
- Resorting to causal histories helps reasoning
- Clocks are different ways of representing/summarizing causal histories
- A visual analogy helps; for the same run with vector clocks:



- Vector clocks characterize happens-before:

$$e \rightarrow e' \Leftrightarrow \mathbb{C}(e) \subset \mathbb{C}(e') \Leftrightarrow V(e) < V(e')$$

Variable number of nodes

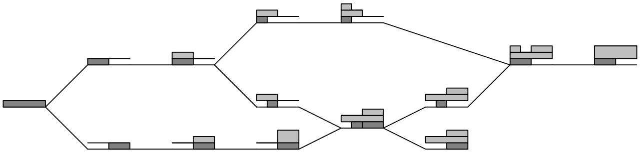
- A vector clock using a map adapts to variable number of nodes
- We need only use unique node identifiers
- Each id not mapped is implicitly mapped to 0
- But this makes vector clocks acquire more entries over time
- Node retirement is tricky

Dynamic systems with process creation and retirement

- One possible execution model allows
  - forking a new node from an existing one
  - joining two nodes into one, retiring one node
  - state transitions
  - sending messages
- A clock for such dynanic systems could/should
  - have dynamic (plastic) representations of identities
  - a way to manage (split, gather, retire) identities
  - have the notion of the identity owned by a node

Interval Tree Clocks – a logical clock for dynamic systems

- A clock is a function defined over a tree of intervals
- Reasonably complex, easy to understand visually
  - light-gray area represents causal history
  - dark-gray area represents owned identity
  - updates inflate some light-gray area over the dark-gray
  - forks split dark-gray area, keep light-gray area
  - joins merge both areas
  - partial order is according to light-gray area inclusion
  - buddy areas can be merged to simplify representation



Bibliography

Paulo Sérgio Almeida, Carlos Baquero, and Victor Fonte.
Interval tree clocks.
In Theodore P. Baker, Alain Bui, and Sébastien Tixeuil, editors, *Principles of Distributed Systems, 12th International Conference, OPODIS 2008, Luxor, Egypt, December 15-18, 2008. Proceedings*, volume 5401 of *Lecture Notes in Computer Science*, pages 259–274. Springer, 2008.

Leslie Lamport.
Time, clocks, and the ordering of events in a distributed system.
Commun. ACM, 21(7):558–565, 1978.

Friedemann Mattern.
Virtual time and global states of distributed systems.
In *Proceedings of the International Workshop on Parallel and Distributed Algorithms, Chateau de Bonas, France, October 1988*, pages 120–131. Elsevier Science Publishers, 1988.