Vector clock

From Wikipedia, the free encyclopedia
(Redirected from Vector clocks)

Vector clocks is an algorithm for generating a partial ordering of events in a distributed system and detecting causality violations. Just as in Lamport timestamps, interprocess messages contain the state of the sending process's logical clock. A vector clock of a system of *N* processes is an array/vector of *N* logical clocks, one clock per process; a local "smallest possible values" copy of the global clock-array is kept in each process, with the following rules for clock updates:

- Initially all clocks are zero.
- Each time a process experiences an internal event, it increments its own logical clock in the vector by one.
- Each time a process prepares to send a message, it sends its entire vector along with the message being sent.
- Each time a process receives a message, it increments its own logical clock in the vector by one and updates each element in its vector by taking the maximum of the value in its own vector clock and the value in the vector in the received message (for every element).

Example of a system of vector clocks. Events in the blue region are the causes leading to event B4, whereas those in the red region are the effects of event B4

The **vector clocks** algorithm was independently developed by Colin Fidge and Friedemann Mattern in 1988.[1][2]

Contents [hide]

Partial ordering property   [edit]

Vector clocks allow for the partial causal ordering of events. Defining the following:

- $VC(x)$ denotes the vector clock of event x, and $VC(x)_z$ denotes the component of that clock for process z.
- $VC(x) < VC(y) \iff \forall z[VC(x)_z \le VC(y)_z] \land \exists z'[VC(x)_{z'} < VC(y)_{z'}]$
  - In English: $VC(x)$ is less than $VC(y)$, if and only if $VC(x)_z$ is less than or equal to $VC(y)_z$ for all process indices z, and at least one of those relationships is strictly smaller (that is, $VC(x)_{z'} < VC(y)_{z'}$).
- $x \rightarrow y$ denotes that event x happened before event y. It is defined as: if $x \rightarrow y$, then $VC(x) < VC(y)$

Properties:

- If $VC(a) < VC(b)$, then $a \rightarrow b$
- Antisymmetry: if $VC(a) < VC(b)$, then $\neg VC(b) < VC(a)$

- **Transitivity**: if $VC(a) < VC(b)$ and $VC(b) < VC(c)$, then $VC(a) < VC(c)$ or if $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$

Relation with other orders:

- Let $RT(x)$ be the real time when event x occurs. If $VC(a) < VC(b)$, then $RT(a) < RT(b)$
- Let $C(x)$ be the Lamport timestamp of event x. If $VC(a) < VC(b)$, then $C(a) < C(b)$

Other mechanisms [edit]

- Almeida *et al.*, introduced in 2008 Interval Tree Clocks.[3] This mechanism generalizes Vector Clocks and allows operation in dynamic environments when the identities and number of processes in the computation is not known in advance. You can find an implementation of ITC named itc4j here .
- Torres-Rojas and Ahamad, developed in 1999 Plausible Clocks,[4] a mechanism that takes less space than vector clocks but that, in some cases, will totally order events that are causally concurrent.

See also [edit]

- Lamport timestamps
- Matrix clocks
- Version vector

References [edit]

1. ^ Colin J. Fidge (February 1988). "Timestamps in Message-Passing Systems That Preserve the Partial Ordering" (PDF). In K. Raymond (Ed.). *Proc. of the 11th Australian Computer Science Conference (ACSC'88)*. pp. 56–66. Retrieved 2009-02-13.
2. ^ Mattern, F. (October 1988), "Virtual Time and Global States of Distributed Systems", in Cosnard, M., *Proc. Workshop on Parallel and Distributed Algorithms*, Chateau de Bonas, France: Elsevier, pp. 215–226
3. ^ Almeida, Paulo; Baquero, Carlos; Fonte, Victor (2008), "Interval Tree Clocks: A Logical Clock for Dynamic Systems", in Baker, Theodore P.; Bui, Alain; Tixeuil, Sébastien, *Principles of Distributed Systems* (PDF), Lecture Notes in Computer Science **5401**, Springer-Verlag, Lecture Notes in Computer Science, pp. 259–274, doi:10.1007/978-3-540-92221-6 , ISBN 978-3-540-92220-9
4. ^ Torres-Rojas, Francisco; Ahamad, Mustaque (1999), "Plausible clocks: constant size logical clocks for distributed systems", *Distributed Computing* (Springer Verlag) **12** (4): 179–195, doi:10.1007/s004460050065

External links   [edit]

- Explanation of Vector clocks
- Timestamp-based vector clock implementation in Erlang
- Vector clock implementation in Objective-C
- Vector clock implementation in Erlang
- Why Vector Clocks are Hard
- Riak Vector Clocks

Categories: Distributed algorithms