Vector time

Definition

The system of vector clocks was developed independently by Fidge, Mattern, and Schmuck. In the system of vector clocks, the time domain is represented by a set of n-dimensional non-negative integer vectors.

Each process p_i maintains a vector $vt_i[1\ldots n]$, where $vt_i[i]$ is the local logical clock of p_i and describes the logical time progress at process p_i. $vt_i[j]$ represents process p_i's latest knowledge of process p_i local time. If $vt_i[j] = x$, then process p_i knows that local time at process p_i has progressed till x. The entire vector vt_i constitutes p_i's view of the global logical time and is used to timestamp events.

Process p_i uses the following two rules R1 and R2 to update its clock:

1. R1 Before executing an event, process p_i updates its local logical time as follows:

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

2. R2 Each message m is piggybacked with the vector clock vt of the sender process at sending time. On the receipt of such a message (m,vt), process p_i executes the following sequence of actions:
   1. update its global logical time as follows:

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

   2. execute R1;
   3. deliver the message m.

The timestamp associated with an event is the value of the vector clock of its process when the event is executed. Figure shows an example of vector clocks progress with the increment value d = 1. Initially, a vector clock is [0,0,0,….].

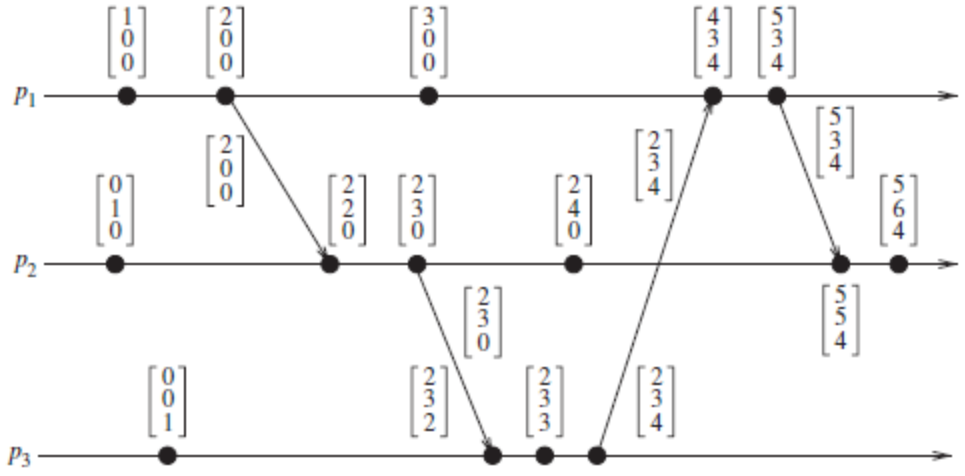

Figure: Evolution of vector time

The following relations are defined to compare two vector timestamps, *vh* and *vk*:

$$vh = vk \Leftrightarrow \forall x : vh[x] = vk[x]$$
$$vh \leq vk \Leftrightarrow \forall x : vh[x] \leq vk[x]$$
$$vh < vk \Leftrightarrow vh \leq vk \text{ and } \exists x : vh[x] < vk[x]$$
$$vh \parallel vk \Leftrightarrow \neg(vh < vk) \wedge \neg(vk < vh).$$

Basic Properties

Isomorphism

If events in a distributed system are timestamped using a system of vector clocks, we have the following property. If two events *x* and *y* have timestamps *vh* and *vk*, respectively, then

$$x \rightarrow y \Leftrightarrow vh < vk$$
$$x \parallel y \Leftrightarrow vh \parallel vk.$$

Thus, there is an isomorphism between the set of partially ordered events produced by a distributed computation and their vector timestamps. This is a very powerful, useful, and interesting property of vector clocks.

Strong consistency

The system of vector clocks is strongly consistent; thus, by examining the vector timestamp of two events, we can determine if the events are causally related.

Event counting

If *d* is always 1 in rule R1, then the *i*th component of vector clock at process p_i, $vt_i[i]$, denotes the number of events that have occurred at pi until that instant. So, if an event e has timestamp *vh*, *vh*[j] denotes the number of events executed by process p_j that causally precede *e*.

Applications

Since vector time tracks causal dependencies exactly, it finds a wide variety of applications. For example, they are used in distributed debugging, implementations of causal ordering communication and causal distributed shared memory, establishment of global breakpoints, and in determining the consistency of checkpoints in optimistic recovery.

Size of vector clocks

- A vector clock provides the latest known local time at each other process. If this information in the clock is to be used to explicitly track the progress at every other process, then a vector clock of size *n* is necessary.
- A popular use of vector clocks is to determine the causality between a pair of events. Given any events *e* and *f*, the test for $e \prec f$ if and only if $T(e) < T(f)$, which requires a comparison of the vector clocks of *e* and *f*. Although it appears that the clock of size *n* is necessary, that is not quite accurate. It can be shown that a size equal to the dimension of the partial order $(E, \prec)$ is necessary, where the upper bound on this dimension is *n*.

Physical clock synchronization: NTP

In distributed systems, there is no global clock or common memory. Each processor has its own internal clock and its own notion of time. In practice, these clocks can easily drift apart by several seconds per day, accumulating significant errors over time. Also, because different clocks tick at different rates, they may not remain always synchronized although they might be synchronized when they start.

Some practical examples that stress the need for synchronization are listed below:

- In database systems, the order in which processes perform updates on a database is important to ensure a consistent, correct view of the database. To ensure the right ordering of events, a common notion of time between co-operating processes becomes imperative.
- It is quite common that distributed applications and network protocols use timeouts, and their performance depends on how well physically dispersed processors are time-synchronized. Design of such applications is simplified when clocks are synchronized.

Clock synchronization is the process of ensuring that physically distributed processors have a common notion of time. It has a significant effect on many problems like secure systems, fault diagnosis and recovery, scheduled operations, database systems, and real-world clock values.

Due to different clocks rates, the clocks at various sites may diverge with time, and periodically a clock synchronization must be performed to correct this clock skew in distributed systems. Clocks are synchronized to an accurate real-time standard like UTC (Universal Coordinated Time). Clocks that must not only be synchronized with each other but also have to adhere to physical time are termed physical clocks.

Definitions and terminology

We provide the following definitions. C_a and C_b are any two clocks.

- **Time:** The time of a clock in a machine p is given by the function $C_p(t)$, where $C_p(t) = t$ for a perfect clock.
- **Frequency:** Frequency is the rate at which a clock progresses. The frequency at time t of clock C_a is $C_a'(t)$
- **Offset**: Clock offset is the difference between the time reported by a clock and the real time. The offset of the clock C_a is given by $C_a(t) - t$. The offset of clock C_a relative to C_b at time $t \geq 0$ is given by $C_a(t) - C_b(t)$.
- **Skew:** The skew of a clock is the difference in the frequencies of the clock and the perfect clock. The skew of a clock C_a relative to clock C_b at time t is $C_a'(t) - C_b'(t)$.

- **Drift (rate)**: The drift of clock C_a is the second derivative of the clock value with respect to time, namely, $C_a''(t)$. The drift of clock C_a relative to clock C_b at time t is $C_a''(t) - C_b''(t)$

Clock inaccuracies

Physical clocks are synchronized to an accurate real-time standard like UTC (Universal Coordinated Time).

However, due to the clock inaccuracy discussed above, a timer (clock) is said to be working within its specification if

$$1 - \rho \leq \frac{dC}{dt} \leq 1 + \rho,$$

where constant ρ is the maximum skew rate specified by the manufacturer.

Offset delay estimation method

The *Network Time Protocol (NTP)*, which is widely used for clock synchronization on the Internet, uses the the *offset delay estimation* method. The design of NTP involves a hierarchical tree of time servers. The primary server at the root synchronizes with the UTC. The next level contains secondary servers, which act as a backup to the primary server. At the lowest level is the synchronization subnet which has the clients.

Clock offset and delay estimation

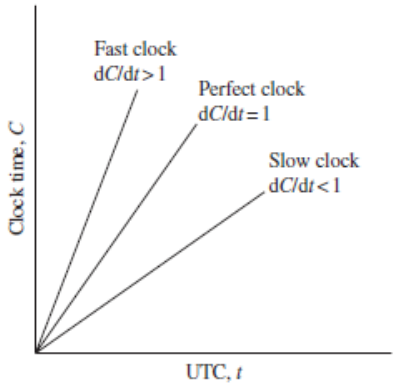

Figure: The behavior of fast, slow, and perfect clocks with respect to UTC

In practice, a source node cannot accurately estimate the local time on the target node due to varying message or network delays between the nodes. This protocol employs a very common practice of performing several trials and chooses the trial with the minimum delay. Recall that Cristian's remote clock reading method also relied on the same strategy to estimate message delay.

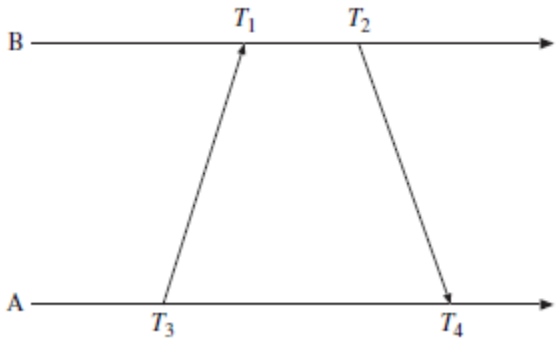

Figure: Offset and delay estimation

Figure shows how NTP timestamps are numbered and exchanged between peers A and B. Let T_1, T_2, T_3, T_4 be the values of the four most recent timestamps as shown. Assume that clocks A and B are stable and running at the same speed. Let $a = T_1 - T_3$ and $b = T_2 - T_4$. If the network delay difference from A to B and from B to A, called differential delay, is small, the clock offset θ and roundtrip delay δ of B relative to A at time T_4 are approximately given by the following:

$$\theta = \frac{a+b}{2}, \quad \delta = a - b.$$

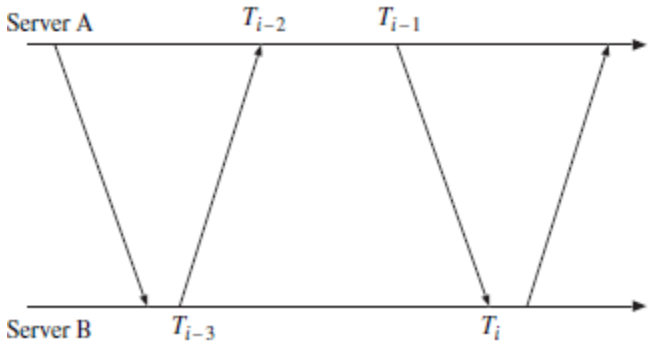

Figure: Timing diagram for the two servers

Each NTP message includes the latest three timestamps T_1, T_2, and T_3, while T_4 is determined upon arrival. Thus, both peers A and B can independently calculate delay and offset using a single bidirectional message stream as shown in Figure.