Script **generated by TTT**

Title: Distributed_Applications (26.05.2014)

Date: Mon May 26 09:16:19 CEST 2014

Duration: 47:41 min

Pages: 12

physical clocks are used to compute the current time in order to timestamp events, such as

- modification date of a file
- time of an e-commerce transaction for auditing purposes

[External - internal synchronization](#)
[Clock correctness](#)
[Synchronization in a synchronous system](#)
[Cristian's method for an asynchronous system](#)
[Network Time Protocol (NTP)](#)
[Precision Time Protocol (PTP)](#)

Precision Time Protocol (PTP)

PTP is a protocol to synchronize clocks throughout a computer network

NTP is typically used over the Internet handling large amounts of nondeterministic delays; accuracy in the ms-range.

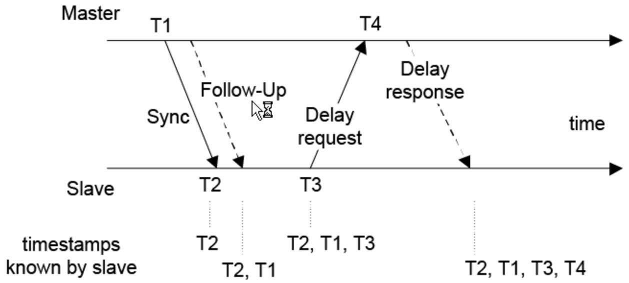
PTP is designed for LANs achieving clock accuracy in the sub-microsecond range.

[Synchronization Message Exchange](#)

- PTP supports an algorithm to perform a distributed selection of the best candidate clock.

Synchronization Message Exchange

PTP defines a master-slave hierarchy.



- master periodically transmits a Sync message using UDP multicast.
- the follow-up message includes the actual time the Sync message left the master.
- slave initiates exchange with master to determine round-trip delay: delay-request and delay-response messages.
- if d is the transit time for the Sync message and o the constant offset between master and slave clocks
  - $T2 - T1 = o + d$ and $T4 - T3 = -o + d$
  - $o = (T2 - T1 - T4 + T3)/2$

Time is an important and interesting issue in distributed systems

- We need to measure time accurately:
  - to know the time an event occurred at a computer
  - to do this we need to synchronize its clock with an authoritative external clock
- Algorithms for clock synchronization useful for
  - concurrency control based on timestamp ordering
  - authenticity of requests e.g. in Kerberos

Three notions of time:

- time seen by an external observer ⇒ global clock of perfect accuracy.
  - However, there is ***no global clock in a distributed system***
- time seen on clocks of individual processes.
- logical notion of time: event a occurs before event b.

In order to guarantee consistent states among the communicating components, the messages must be delivered in the correct order. The happened-before relation after Lamport may help to determine a message sequence for a distributed application.

- The following rules apply:
  - Events within a component are ordered with respect to the before-relation , i.e. $a \rightarrow b$
  - if "a" is a send event of component TK1, and "b" the respective receive event of component TK2, then $a \rightarrow b$;
  - if $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$;
  - if $\neg (a \rightarrow b)$ and $\neg (b \rightarrow a)$, then $a \parallel b$; i.e. a and b are concurrent, i.e. they are not ordered.
- Utilization of logical clocks to determine the event sequence.
- Let
  - T: a set of timestamps
  - $C: E \rightarrow T$ a mapping which assigns a timestamp to each event
  - $a \rightarrow b \Rightarrow C(a) < C(b)$
- If the reverse deduction is valid, too (⇔), then the clock is called strictly consistent.

Each component manages the following information:

- its local logical clock lc; lc determines the local progress with respect to occuring events.
- its view on the global logical clock gc; the value of the local clock is determined according to the value of the global clock.

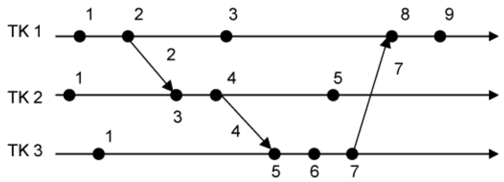
There exist functions for updating logical clocks in order to maintain consistency; the following two rules apply.

Rules

- Rule R1 specifies the update of the local clock lc when events occur.
- Rule R2 specifies the update of the global clock gc.
  1. ***Sending event*** : determine the current value of the local clock and attach it to the message.
  2. ***Receiving event*** : the received clock value (attached to the message) is used to update the view on the global clock.

Distributed execution model

Example



The scalar clock mechanism defines a partial ordering on the occurring events.

scalar clocks are ***not strictly consistent*** , i.e.

the following is not true: $C(a) < C(b) \Leftrightarrow a \rightarrow b$

Description

The time is represented by n-dimensional vectors with positive integers. Each component TK_i manages its own vector vt_i [1....n]. The dimension n is determined by the number of components of the distributed application.

- vt_i [i] is the local logical clock of TK_i .
- vt_i [k] is the view of TK_i on the local logical clock of TK_k ; it determines what TK_i knows about the progress of TK_k
- Example: vt_i [k] = y, i.e. according to the view of TK_i , TK_k has advanced to the state y, i.e. up to the event y.
- the vector vt_i [1....n] represents the view of TK_i on the global time (i.e. the global execution progress for all components).

Execution of R1

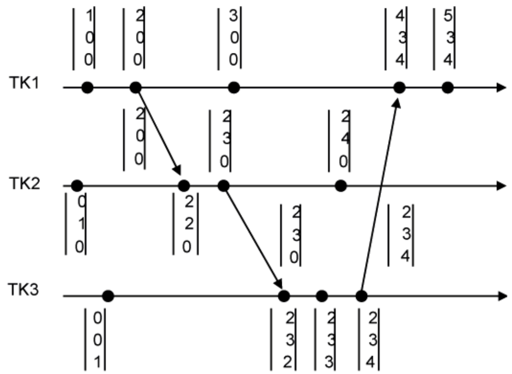
$vt_i [i] := vt_i [i] + d$

Execution of R2

After receiving a message with vector vt from another component, the following actions are performed at the component TK_i

- update the logical global time: $1 \leq k \leq n$: $vt_i [k] := \max (vt_i [k], vt[k])$.
- execute R1
- deliver message to the application process of component TK_i

Example for vector clocks



optimization: omit vector timestamps when sending a burst of multicasts

$\Rightarrow$ missing timestamp means: use values of previous vector timestamp and increment the sender's field only.

Comparison of two vector clocks (timestamps) vh[1..n] and vk[1..n]:

$vh \leq vk$	$\Leftrightarrow$	$\forall x$: $vh[x] \leq vk[x]$
$vh < vk$	$\Leftrightarrow$	$vh \leq vk$ ***and*** $\exists x$: $vh[x] < vk[x]$
$vh \parallel vk$	$\Leftrightarrow$	$\neg (vh < vk)$ ***and*** $\neg (vk < vh)$

Let a and b be events with timestamps (vector clocks) va and vb, then the following is true

$a \rightarrow b$	$\Leftrightarrow$	$va < vb$
$a \parallel b$	$\Leftrightarrow$	$va \parallel vb$

If a of Tk_i and b of Tk_j have been triggered, then the following is true

$a \rightarrow b$	$\Rightarrow$	$va[i] < vb[i]$ ***and*** $va[j] < vb[j]$
$a \parallel b$	$\Leftrightarrow$	$va[i] > vb[i]$ ***and*** $va[j] < vb[j]$

Vector clocks are strictly consistent.