

# PA160: Net-Centric Computing II.

## Time Synchronization

Luděk Matyska

Faculty of Informatics Masaryk University

Spring 2016

# Time on a single node

- Timer (clock)
  - Oscillating quartz crystal
  - Counter
    - Each oscillation decreases the counter value
    - Interrupt when zero

Each individual interrupt is a *tick*

  - Holding register
    - Counter initiation after interrupt
  - Stored time increased after each interrupt

# Distributed computer systems

- Each node has its own timer
  - Time is not automatically synchronized
  - Clock skew
- Relation to the absolute time
- Problems
  - Internode synchronization
  - Absolute time synchronization

# Absolute time

- Original (old) definition of the time unit
  - 1 second equals  $1/86\,400$  solar day
- Current definition of the time unit—atomic clock
  - Electronic transition frequency of the electromagnetic spectrum of atoms
  - Cesium-133 standard in 1955
  - Chip-scale atomic clock in 2004 (125 mW)
  - 1 second equals 9 192 631 770 cycles (transitions between two energy levels of Cs-133)
- International atomic clock
  - Average measurement of around 50 world laboratories

# Universal Coordinated Time, UTC

- Atomic and solar times are not synchronized
- Leap second needed
  - Compensation for the irregularities of Earth rotation
  - Added whenever difference between atomic clock and mean solar time gets 800 ms
- Result is the Universal Coordinated Time (UTC)
  - GMT replacement
- UTC globally synchronized

# Clock synchronization

- Basic assumptions
  - A set of nodes with their own clocks/timers
  - Interrupt frequency  $H$  Hz

$C_p(t)$  is the time measured by clock at node  $p$

- Ideally  $C_p(t) = t$  for all  $p$
- Real behavior: If there exists  $\rho$  such as

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

then clock  $C_p$  works with the specification  $\rho$

- $\rho$  is defined by the clock producer (it is the maximal time skew)
- If we are looking for a clock synchronization with the highest difference  $\delta$ , then the synchronization must occur at most every  $\delta/2\rho$  seconds

# Cristian's algorithms

- We have a *time server*
  - synchronized with UTC
- Each  $\delta/2\rho$  each node sends a request to the server
- Server replies with its own (UTC) time (as fast as possible)
- Naive solution: the node modifies its time accordingly

# Problems

- **Important**—delay compensation
  - Time stops to have linear course (shape)
    - Unexpected and undesirable effects
    - Time cannot move “backwards”
  - Solution
    - The absolute time is not increased at the interrupt
    - We “stop” the time
- **Small**—communication delay
  - Measure the time of sending ( $T_0$ ) and receiving ( $T_1$ ) the request
  - Add time of the transfer  $(T_0 + T_1)/2$
  - Correct for the time spent at the server (request processing), if known



# Berkeley algorithm

- Active time server
  - Periodically queries nodes for their absolute time
  - Averages node times
  - Sends this new time to all nodes
- Suitable if no access to UTC source exists
  - Analogy of UTC—time based on the agreement of the nodes

# Decentralized solutions

- Re-synchronization intervals
  - A starting point  $T_0$  globally agreed
  - Interval  $i$  starts at the time  $T_0 + iR$
  - Interval  $i$  ends at the time  $T_0 + (i + 1)R$
  - $R$  is an agreed system parameter
- All nodes broadcast their absolute time at the beginning of each interval
- Each node does the following
  - Receives  $S$  messages
  - Computes the average (with the removal of  $m$  outliers)
  - Improvement possible if the message propagation delay known

# NTP

- Network Time Protocol
  - Version 3 (RFC1305), version 2 (RFC1119), version 1 (RFC 1059)
  - S(imple)NTP: RFC 1769
- Hierarchical structure based on stratum
  - Stratum 1 directly connected to UTC (atomic clock, ...)
  - Stratum  $i + 1$  connects to server(s) at Stratum  $i$
  - Up to 16 levels (Stratum 16)
- Highly scalable
- More than one server
  - Tolerant to server fault of precision loss
- Servers `tik.cesnet.cz` and `tak.cesnet.cz`

# Logical time

- Absolute time is not always necessary
- *Relative time* (relation between events) often more important
- *Logical time*
  - No need for absolute synchronization
  - Need agreement on the order

# Lamport timestamps

- Relation “happens-before”
- $a \rightarrow b$  means that all processes agree that  $a$  happened before  $b$
- If  $a$  represents an event of sending a particular message and  $b$  represents the receipt of the same message, then  $a \rightarrow b$  holds
- Properties
  - $a \rightarrow b$  is transitive
  - Events  $x$  and  $y$  are *concurrent* iff nor  $x \rightarrow y$  nor  $y \rightarrow x$  holds

# Implementation

- Each process does have its own logical clock
- For events within any particular process the relation “happens-before” is trivially fulfilled
- Interprocess synchronization is the result of message passing
  - Each message contains time stamp  $T_s$  of the sender (its time of sending the message)
  - If internal receiver time  $T_r$  is lower (i.e. “younger”) than the time of sending the message ( $T_s$ ), we put  $T_r = T_s + 1$
- Additional condition
  - No two events happen at the same time

# Summary

- Lamport's algorithm is sufficient to define and keep a global (logical) time in a distributed system
- Properties
  - If  $a$  happens before  $b$  in the same process,  $C(a) < C(b)$
  - If  $a$  is sending and  $b$  receipt of the same message,  $C(a) < C(b)$
  - For all events  $a, b$  such that  $a \neq b$ ,  $C(a) \neq C(b)$  holds
- The algorithms provides global (partial) order on events in a distributed system
- First published in 1978 in CACM; one of the most cited articles in Computer Science of all time