Distributed Synchronization

- Communication between processes in a distributed system can have unpredictable delays, processes can fail, messages may be lost.
- Synchronization in distributed systems is harder than in centralized systems because the need for distributed algorithms.
- Properties of distributed algorithms:
  1. The relevant information is scattered among multiple machines.
  2. Processes make decisions based only on locally available information.
  3. A single point of failure in the system should be avoided.
  4. No common clock or other precise global time source exists.
- Challenge: How to design schemes so that multiple systems can coordinate/synchronize to solve problems efficiently?

The Myth of Simultaneity

"Event 1 and event 2 at same time"

Event 1

Event 2

Observer A: Event 2 is earlier than Event 1

Observer B: Event 2 is simultaneous to Event 1

Observer C: Event 1 is earlier than Event 2

\[ e_1 \parallel e_2 = e_1 \rightarrow e_2 \rightarrow e_1 \]

Event Timelines (Example of previous Slide)

Note: The arrows start from an event and end at an observation. The slope of the arrows depend on relative speed of propagation.
Causality

Observer A: Event 1 before Event 2
Observer B: Event 1 before Event 2
Observer C: Event 1 before Event 2

Requirement: We have to establish causality, i.e., each observer must see event 1 before event 2.

Event Timelines (Example of previous Slide)

Node 1
Node 2
Node 3
Node 4
Node 5

Note: In the timeline view, event 2 must be caused by some passage of information from event 1 if it is caused by event 1.

Why need to synchronize clocks?

foo.o created
Computer for compiling
Computer for editing
foo.c modified

Local clock time

Physical Time

Some systems really need quite accurate absolute times.

How to achieve high accuracy?
Which physical entity may deliver precise timing?

1. The sun
   Today: 1 sec ~ 1 day / 86400
   but rotation of earth is slowing down

2. An Atom
   State transitions in atoms (defined by BIH in Paris)
   1 sec = time a cesium atom needs for 9 192631 770
   state transitions*

BIH (Bureau International de l’Heure)
* TAI (International Atomic Time)
### Problem with Physical Time

A TAI-day is about 3 msec shorter than a day

=>

BHI inserts 1 sec, if the difference between a day and a TAI-day is more than 800msec

=>

UTC (Universal Time Coordinated) is the base of any international time measure.

### Physical Time

UTC-signals come from radio broadcasting stations or from satellites (GEOS, GPS) with an accuracy of:

- 1.0 msec (broadcasting station)
- 1.0 μsec (GPS)

GPS on all computers?

### Clock Skew Problem

**Clock skew (offset):** the difference between the times on two clocks

**Clock drift:** they count time at different rates

Ordinary quartz clocks drift by ~1 sec in 11-12 days. (10^{-6} secs/sec).

High precision quartz clocks drift rate is ~10^{-7} or 10^{-8} secs/sec

### Physical clock drift rate

- **Maximum drift rate**
  - One can determine how often they should be synchronized

Not all clock’s tick precisely at the current rate.
Clock Synchronization

Adjusting physical clocks:
• local clock behind reference clock
• local clock ahead of reference clock

Observation:
Clocks in DS tend to drift apart and need to be resynchronized periodically

A. If local clock is behind a reference clock:
• could be adjusted in one jump or
• could be adjusted in a series of small jumps

B. What to do if local clock is ahead of reference clock?
Monotonicity.

Computer clocks

- How a computer timer works?
  o A counter register and a holding register.
  o The counter is decremented by a quartz crystals oscillator. When it reaches zero, an interrupted is generated and the counter is reloaded from the holding register.
  o E.g., interrupt 60 times per second.
    • Use 61 to run the wall-clock time faster by making the value of the holding register smaller.

- The clock skew problem
  o logical clocks -- to provide consistent event ordering
  o physical clocks -- clocks whose values must not deviate from the real time by more than a certain amount.

Absolute Clock Synchronization (Cristian’s Algorithm)

- Initialize local:  \( t := t_{UTC} \)
  o (Problem: Message Transfer-Time)
- Estimate Message transfer-time, \( (t_1 - t_0) / 2 \)
  \( t := t_{UTC} + (t_1 - t_0) / 2 \)
  o (Problem: Time of the Request Message)
- Suppose: \( t_r \) is known, \( t := t_{UTC} + (t_1 - t_0 - t_r) \)
  o (Problem: Message transfer-times are load dependent)
- To improve accuracy: Multiple measurements \( (t_1 - t_0) \):
  o Throw away measurements above a threshold value
  o Take all others to get an average
- Assume \( (t_1 - t_0) \) ranges from [min, max]
  o What is the accuracy of \( t \)?
- Centralized time server
  o What if the server crashes?
  o What if the server gives the wrong time?
Relative Clock Synchronization (Berkeley Algorithm)

If you need a uniform time (without a UTC-receiver per computer), but you cannot establish a central time-server:

- Peers elect a master
- Master polls all nodes to give him their times by the clock
- The master estimates the local times of all nodes regarding the involved message transfer times.
- Master uses the estimated local times for building the arithmetic mean
  o Add fault tolerance
- The deviations from the mean are sent to the nodes
  o Is this better than sending the actual time?

The Berkeley Algorithm

- Averaging algorithm
  o The time daemon asks all the other machines for their clock values.
  o The machines answer.
  o The Time daemon tells everyone how to adjust their clock.