Synchronization of Physical Clocks

Alessio Bechini Dept. of Information Engineering, Univ. of Pisa

a.bechini@ing.unipi.it

Outline

Getting Computer Clocks Synchronized

- Generalities
1) Cristian's algorithm
2) Berkeley algorithm
3) Network Time Protocol

Generalities

We have seen that we cannot rely on a global clock in a distributed system, but often there is a necessity to have clocks synchronized within a certain tolerance interval.

Time References

Every node has an internal clock, whose value (il cui) has to be kept as close as possible to a reference time.

- **UTC** - coordinated universal time, primary reference for the scientific community, and based on International Atomic Time
- **UT1** - successor of GMT, "solar-based" reference time
- **Unix (POSIX) time** - nr. of seconds elapsed since 00:00:00 (UTC), Thursday, 1 January 1970, minus in-between *leap seconds*

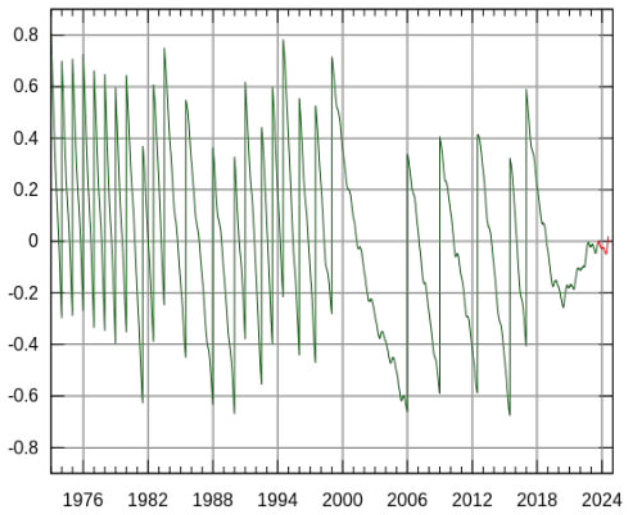
I leap seconds (secondi intercalari) sono aggiustamenti occasionali apportati all'ora ufficiale del Coordinated Universal Time (UTC) per tenere conto delle irregolarità nella rotazione della Terra.

Leap Seconds?

Earth's rotation slowdown and irregularities make UT1 deviate from UTC.

Thus, corrections are needed.

Chart aside: UT1-UTC vs UTC



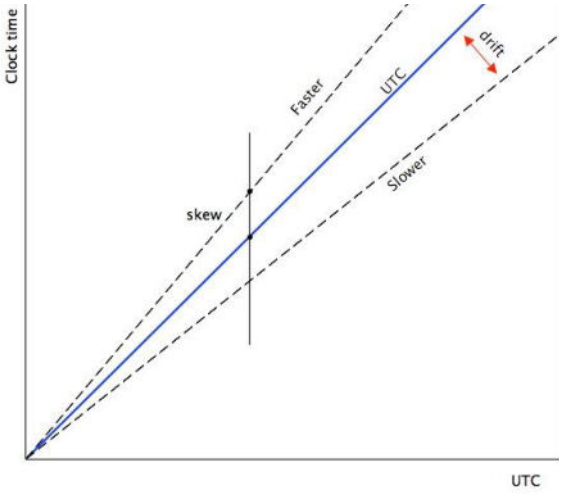
From Wikipedia

Non-Aligned Clocks

Skew - difference in instantaneous reads of two clocks

Drift - difference in the rate of clocks.

Keeping two clocks synchronized means imposing an upper bound D on any of their instantaneous reads.

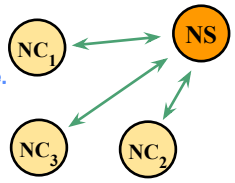


External vs Internal Synchronization

External: w.r.t. a trusted source of reference time $S(t)$, i.e.

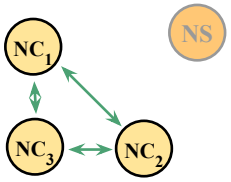
We want all the different nodes in our system to be synchronized with an external trusted reference.

$$| S(t) - C_i(t) | < D \quad \forall \; i$$



Internal: agreement within a group of nodes, i.e.

$$| C_i(t) - C_j(t) | < D \quad \forall \; i, j$$



If a system is externally synchronized with an accuracy D, then its internal clocks agree within a bound of $2D$

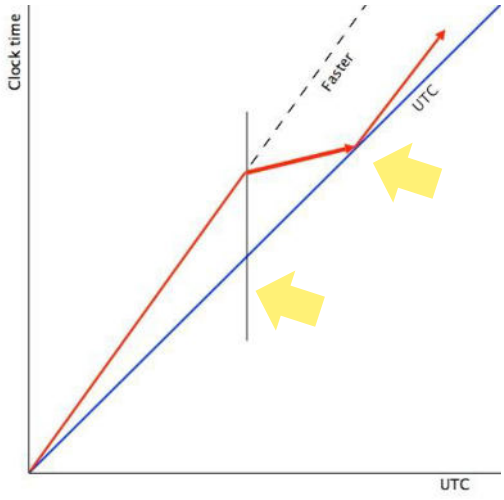
Resetting a Clock

(A clock will never go back; it can only increase.)

Important: clock *monotonicity* must be assured!

No way to set back a clock: instead, its pace (velocità) should be slowed down up to the point the correct value is reached.

Typical example: the “make” compilation utility.



Cristian's Algorithm

(External Synchronization)

Idea: Exploit Round Trip Time

Round Trip Time

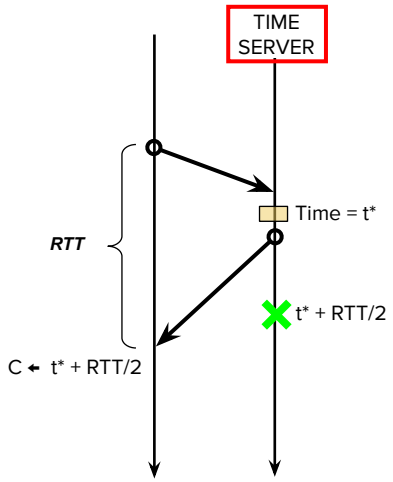
RTT can be measured by the client.

(is very small)

If RTT << 1, assuming req/reply latencies to be the same does not yield an excessive error, thus:

$$C \simeq t^* + RTT/2$$

In practice, several tries can be done, so to possibly get lower RTTs.



It's true, it's not very precise to simply add RTT/2. We should ideally add only the time the message took to return to us, but since we don't know it, there's not much we can do about it. However, we rely on the fact that the smaller the RTT you have, the more precise C will be with respect to the time of the time server.

Internal Synchronization: The Berkeley Algorithm, and Beyond

(Internal Synchronization)

A Centralized Solution

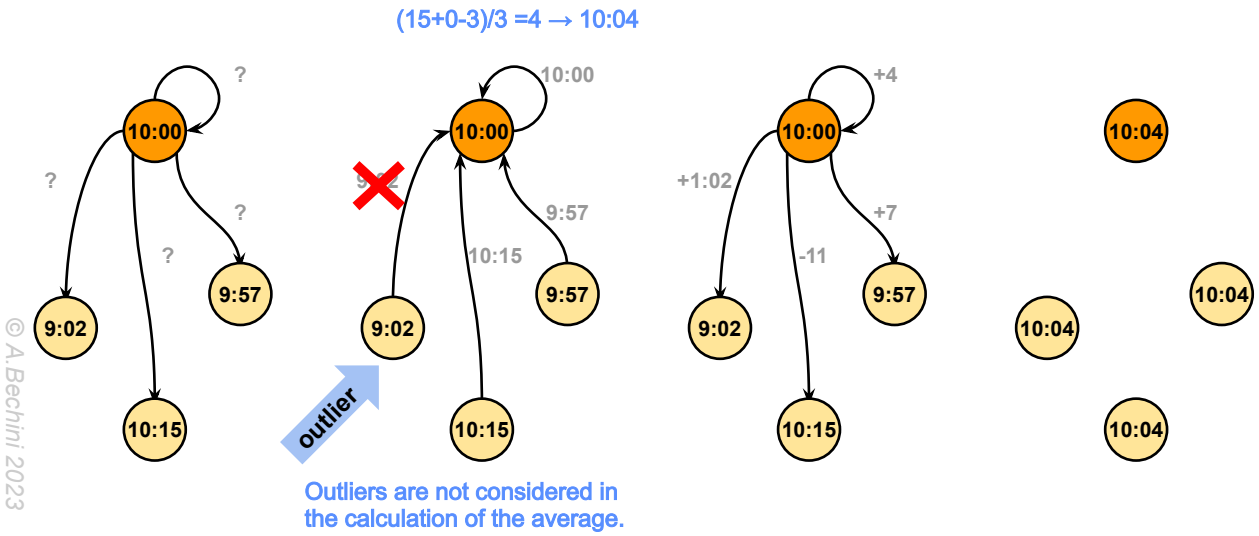
Developed for groups of UNIX computers.

One process/node acts as master (*time daemon*); Successive steps:

- The time daemon asks all the others for their clock values.
- Each node answers back its actual time value;
  the master annotate each RTT as well.
- The time daemon computes a *proper* average.
- The time daemon sends each node the ***clock correction value***

(why not broadcasting the time?) Because in this scenario, the message from the master to each client may experience varying latencies. These differences in latency can potentially cause a divergence in the clocks of the nodes.

Berkeley Algorithm: an Illustration



Outliers are not considered in the calculation of the average.

Decentralized Averaging Algorithm

Each node has a daemon, with approximated UTC.

- Periodically, each node broadcasts its own time.
- On each node, the new time value is obtained by averaging the local time and the received values.

NTP - Network Time Protocol

It is the protocol used nowadays to keep our devices (connected to the internet) synchronized.

NTP - Overview

To enable clients across Internet to get synchronized with UTC.

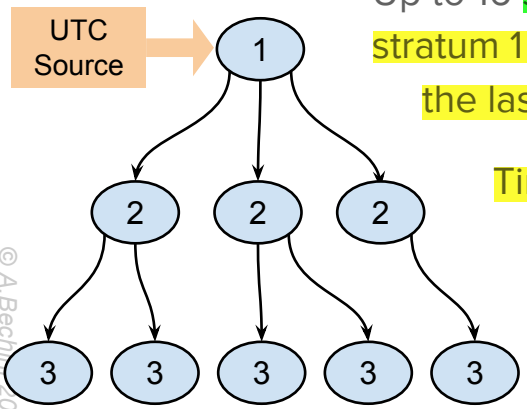
- Scalable to large networks
- Statistical techniques to filter data
- Authentication against interference
- Hierarchy of time servers, spread across the Internet
- Messages sent over UDP

NTP makes use of:

In the case of the Cristian algorithm, there was a single time server that clients would query for the time. However, this setup is impractical over the Internet because it's not feasible to have a single time server serving the entire Internet. Due to the vast scale and distribution of clients, multiple time servers are necessary, organized into different partitions of the network, to handle the load effectively.

What about the hierarchy containing all the different time servers? They are organized hierarchically in a tree structure. In the protocol's terminology, the levels of this tree are referred to as "strata".

Time Servers' Hierarchy - Strata



Up to 16 strata:

stratum 1 is connected to an UTC source;

the last one is "not synchronized".

Time servers synchronize in 3 modes:

Multicast mode

Procedure Call mode

Symmetric mode

As communication occurs between adjacent strata, the closer a node is to the UTC source, the more synchronized its time reference will be.

NTP Modes

- Multicast: 1+ servers multicast the time to the other nodes; suitable for LANs
- Procedure Call: more accurate because of latency compensation, using an approach based on Cristian's algorithm
- Symmetric: the most accurate and expensive, used between servers in the upper (most precise) strata; Pairs of servers exchange messages carrying the timestamp for the involved events

Symmetric Mode

From the exchanged messages, a process must be able to get an estimate o_i of the actual offset o to fix its clock

$$\begin{cases} T_{i-2} = T_{i-3} + t_m + o \\ T_i = T_{i-1} + t_{m'} - o \end{cases} \qquad \begin{cases} o = T_{i-2} - T_{i-3} - t_m \\ o = T_{i-1} - T_i + t_{m'} \end{cases}$$

"-o" perché 'o' deve rimanere concorde al timestamp del server A

$$RTT = t_m + t_{m'} = \Delta T_A - \Delta T_B = (T_i - T_{i-3}) - (T_{i-1} - T_{i-2})$$

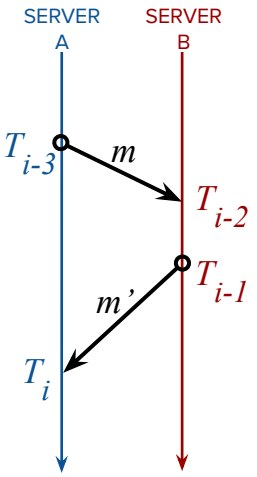
(ottenuta dal secondo sistema, sommando le due equazioni)

$$o = \tfrac{1}{2}(T_{i-2} - T_{i-3} + T_{i-1} - T_i - t_m + t_{m'}) = o_i + \tfrac{1}{2}(t_{m'} - t_m)$$

this relation can be used to obtain an estimate of 'o':

$$o_i - \tfrac{1}{2} RTT \le o \le o_i + \tfrac{1}{2} RTT$$

Having o_i, I will have an indication of the accuracy I achieve when using it to update my clock.



QUESTA SLIDE NON L'HO CAPITA MOLTO:
- non capisco perché da equazione si passi a disequazione (intervallo)

Symmetric Mode: How to Fix Time

Estimated offset: $o_i = \tfrac{1}{2}(T_{i-2} - T_{i-3} + T_{i-1} - T_i)$

Accuracy (upper bound): RTT

- To fix the time, several pairs $<o_i, RTT>$ are collected, and the most accurate value is used.

I will use the most accurate offsets to update the clock value on server A. This is how servers are kept synchronized.
Then, the time values are broadcasted to all other nodes that wish to obtain a rough estimate of the clock time.