Clock Synchronization

Pedro F. Souto (pfs@fe.up.pt)

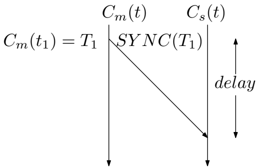
April 5, 2019

Roadmap

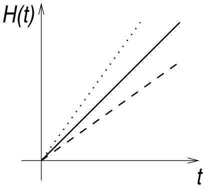
Roadmap

Distributed System Model

- A set of sequential processes that execute the **steps of a distributed algorithm**
  - DS are inherently concurrent, with real parallelism
- Processes communicate and synchronize by exchanging messages
  - The communication is not instantaneous, but suffers delays



- Processes may have access to a local clock
  - But local clocks may drift wrt real time



- DS may have partial failure modes
  - Some components may fail while others may continue to operate correctly

Fundamental Models

Synchronism characterizes the system according to the temporal behavior of its components:

- processes
- local clocks
- communication channels

Failure characterizes the system according to the types of failures its components may exhibit

Models of Synchronism

Synchronous iff:

1. there are known bounds on the time a process takes to execute a step
2. there are known bounds on the time drift of the local clocks
3. there are known bounds on message delays

Asynchronous No assumptions are made regarding the temporal behavior of a distributed system

- These 2 models are the extremes of a range of models of synchronism

Dilemma

- It is relatively simple to solve problems in the synchronous model, but these systems are very hard to build

Roadmap

Clock Synchronization

Observation

- In our every-day lives we use time to coordinate our activities
  - Distributed applications can do the same, as long as each process has access to a "sufficiently" **synchronized clock**

Synchronized Clocks

- Are essential for implementing:

  some basic services:

    Synchronous distributed algorithms i.e. rounds
    Communication protocols e.g. TDMA

  some distributed applications:

    Data collection and event correlation
    Control systems
    GPS

- Enable to reduce communication and therefore improve performance (Liskov 93)

Applications (according to David Mills)

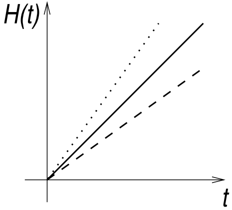
- Distributed database transaction journaling and logging
- Stock market buy and sell orders
- Secure document timestamps
- Aviation traffic control and position reporting
- Radio and TV programming launch
- Real-time teleconferencing
- Network monitoring, measurement and control
- Distributed network gaming and training

- Take it with a grain of salt

src: David Mills, Network Time Protocol (NTP) General Overview

Local Clocks

- Computers have hardware clocks based on quartz crystal oscillators, which provide a local measure of time, $H(t)$. Often:
  - These clocks generate periodic interrupts
  - The OS uses these interrupts to keep the local time



- These clocks have a **drift** $\left(\frac{dH(t)}{dt} - 1\right)$ wrt a perfect clock
  - Quartz-based oscillators have drifts in the order of 10^{-6}, i.e. they may run faster/slower a few μs per second
- Even if this drift is **bounded**, unless clocks **synchronize**, their values may diverge forever, i.e. their **offset/skew** $(H_i(t) - H_j(t))$, may grow unbounded

What is Clock Synchronization?

External Clock Synchronization (CS)

I.e. synchronization wrt an external time reference (R)

$$|C_i(t) - R(t)| < \alpha, \forall i$$

where α is known as the CS **accuracy**

Internal Clock Synchronization

I.e. synchronization among the local clocks in the system – needs not be related to **real time**

$$|C_i(t) - C_j(t)| < \pi, \forall i, j$$

where π is known as the CS **precision**

Note that external clock synchronization implies internal clock synchronization.

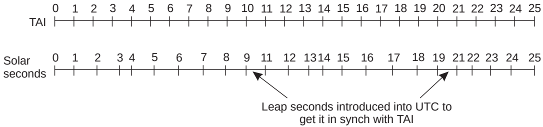
- What's the precision?

Time Scales/Standards

International Atomic Time (TAI) This is a weighted average of the time kept by around 300 atomic clocks in over 50 national labs

- More stable clocks are given larger weights
- It uses GPS satellites for clock comparisons that provide the data for the calculation of TAI

Coordinated Universal Time (UTC) It is derived from the TAI, by adding leap seconds to compensate for variations in Earth's rotation speed



GPS provides time with an accuracy better than 100 nanoseconds

- Each satellite has its own atomic clocks (3 or 4), which are kept synchronized with the Master Clock at US Naval Observatory
- The MC@USNO is a set of more than 40 atomic clocks that produces UTC(USNO), which is used for computing the UTC

Roadmap

Clock Synchronization: Centralized Algorithm

Assumptions

Each process has a local clock

- Master/server clock provides the time reference
- Slave/client clock synchronize with the master/server clock

Local clock drifts are bounded

$$(1+\rho)^{-1} \leq dC_i(t)/dt \leq 1+\rho, \text{for all correct processes } i$$

System is synchronous

- There are known bounds (both lower and upper) for communication delays
- The time a process requires to take a step is negligible with respect to the communication delays (and therefore bounded)

Clock Synchronization: Specification

Accuracy

$|C_i(t) - C_m(t)| \leq \alpha$, for all correct processes i

where $C_m(t)$ is the master's clock

Centralized Clock Synchronization: Idea

Master

- Periodically
  - read the local time
  - broadcast it in a `SYNC` message

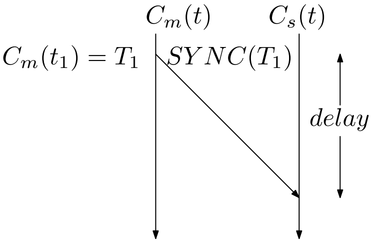
Slave

- Upon reception of a `SYNC` message,
  - **update** the local clock

Master Time Estimation

Issue There is a *delay* between the reading of the time @ the master, and its processing @ the slaves

$$C_s(t) = C_m(t) + delay$$



Problem What is the value of *delay*?

Response Do not know, but can be estimated:

$$min < delay < max$$

max is known because the system is synchronous

- To minimize the error, should use $[C_m(t) + min, C_m(t) + max]$ midpoint:

$$C_s(t) = C_m(t) + \frac{min + max}{2}$$

SYNC Message Period

To ensure **accuracy**, it must be;

$$|C_m(t) - C_s(t)| < \alpha$$

Because the clock drifts are bounded:

$$\left| \frac{dC_s(t)}{dt} - \frac{dC_m(t)}{dt} \right| < 2\rho$$

Hence,

$$\frac{max - min}{2} + 2\rho T < \alpha$$

Observation The clock skew lower bound is determined by the **delay jitter** (($max - min$))

- If T is large, the second term may dominate
    - NTP sometimes uses a T of 15 days, and $\rho \sim 10^{-6}$

Master Time Estimation in Practical Systems (1/3)

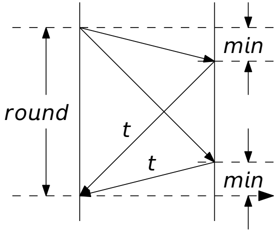
Issue Often, there is no known upper bound for the communications delay

Approach (Christian) Estimate it based on the round-trip-delay, *round*

Let *min* be the lower delay bound

Then, when the slave receives the master's clock reading, time at the master will be in the interval:

$$[t + min, t + round - min]$$



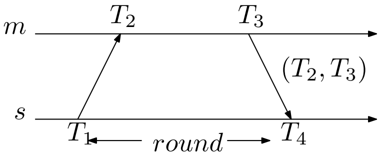
By choosing the **midpoint** in this interval, we **bound the error** to:

$$\frac{round}{2} - min$$

Master Time Estimation in Practical Systems (2/3)

Issue The server may take some time to process the request.

Solution Timestamp the times at which messages are sent or received, using **local clocks**



Let $C_s(t) = C_m(t) + O(t)$
Then,
$T_2 = T_1 - O(t_1) + d_1$
$T_4 = T_3 + O(t_3) + d_2$

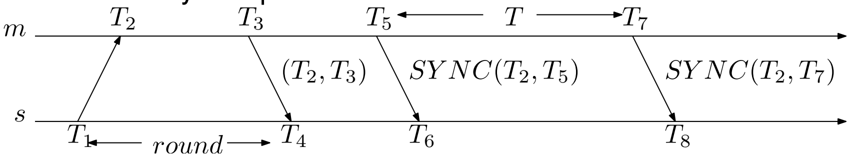
Note: Upper-case T's are **clock times**, lower case are **real-times**

Assume $O = O(t_1) = O(t_3)$
Then $O = \tilde{O} + (d_1 - d_2)/2$
Where $\tilde{O} = ((T_4 - T_3) + (T_1 - T_2))/2$
Because $d_1, d_2 \geq 0$, then $d_1 - d_2 \leq d_1 + d_2$
Let: $del = (d_1 + d_2)/2$
Then: $\tilde{O} - del \leq O \leq \tilde{O} + del$, and:

$\tilde{O}$ can be seen as an estimate of the offset
del can be seen as an estimate of $\tilde{O}$'s error bound

Master Time Estimation in Practical Systems (3/3)

- By including the appropriate timestamps in the SYNC messages $\tilde{O}$ and *del* may be updated



- The accuracy of these timestamps is critical to achieve high accuracy CS
  - The lower in the protocol stack they are generated the better
  - In some cases, the timestamps are sent in later messages
- $\tilde{O}$ can be used later to estimate the value of the master clock at the **time of reception** of a SYNC message with the master's time:

$$Time(SYNC) - \tilde{O}$$

Local Clock Correction

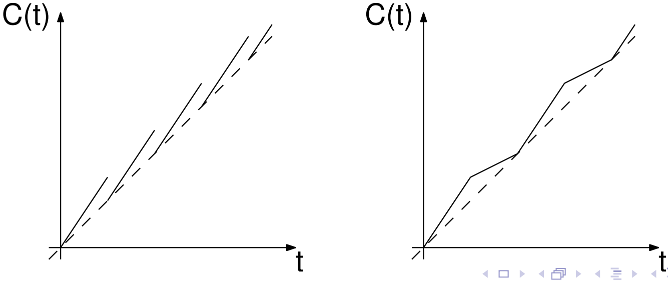
Instantaneous correct offset only at every synchronization point

- May lead to non-monotonic clocks

Amortized over the synchronization period, i.e. adjust both a and b

$$C_s(t) = at + b$$

- So as to ensure that:
  - Local time is continuous
  - Minimize the error at the end of the synchronization period, possibly subject to the constraint that the change in the rate may be bounded



Roadmap

Network Time Protocol (NTP)

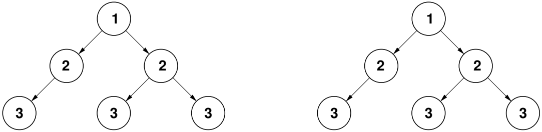
- Clock synchronization protocol designed for the Internet
  - tolerates communication delay jitter
  - it is robust against loss of connectivity
  - scales to a large number of clients
  - it is robust against interferences, whether malicious or accidental

Mills, *Internet time synchronization: the Network Time Pro tocol*, IEEE Trans. on Communications, October 1991, 1482-93.

Mills, *Improved Algorithms for Synchronizing Computer Network Clocks*, IEEE Trans. on Networks, June 1995, pp. 245-54.

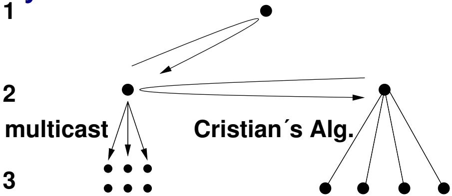
NTP: Architecture

- NTP servers are organized hierarchically in a synchronization subnet



- Servers at the top (**primary**) are connected to a UTC source
- Secondary servers synchronize with the primary and so on
- The leaves of the trees are clients
- Each level of the hierarchy is known as a **stratum**, the lowest stratum comprises the primary servers
- The synchronization subnet may reconfigure in the sequence of failures. E.g.:
  - If a UTC source becomes unreachable, a primary server may become secondary
  - If a primary server becomes unreachable, a secondary server may switch to another primary server

NTP: Synchronization Modes



Multicast a server multicasts periodically its time to clients

- The precision is relatively low, but is OK for LANs (why?)

Request-reply similar to Cristian's algorithm.

Symmetrical in which servers swap their times. Used by servers at the lowest strata. (Ensures the highest precision.)

- I.e. at this level, NTP uses a **decentralized** algorithm

- Besides that, NTP uses lots of statistics to compensate for the the jitter in communication delay
  - It is this variability that limits the precision
  - The OS itself also introduces some jitter
- NTP uses UDP always

Roadmap

Synchronized Clocks: Practical Usage

- On the Internet, it is possible to have synchronized clocks with:
  - A small **enough** **skew** (δ)
  - A high **enough** probability
- We can take advantage of this to reduce:
  - communication
  - state (stored in stable storage)

  and therefore improve performance
- What if clocks get ouf of sync? Depends:

  Compensate at a higher level
  Do nothing there are different reasons:
  1. Correctness is not affected
  2. Domination of other failures
  3. Engineering trade-off: performance benefits outweight failure costs

Synchronized Clocks: At-most-once Messaging (1/3)

Simplistic solution

- Remember all the messages received
- Discard duplicated messages

Issue: Cannot remember ALL messages received

Solution Forget **far away past**

Session-based

- Node has to execute some handshake before sending the first message after a while
  - In response, the receiver generates a **conn(ection) id**;
  - The sender must tag subsequent messages with the conn id
  - The handshake may be too high overhead

Synchronized clocks

- All nodes have a synchronized clocks with accuracy α

Synchronized Clocks: At-most-once Messaging (2/3)

Each message has:

A connection id

- selected by the sender - no need for handshake
- must be unique (also among all senders)

A timestamp

Receivers keep:

A connection table (CT) with, for each connection:

- The timestamp, *ts*, of the last message received
  - If not replaced by that of more recent message, it is kept at least for the life-time, λ, of a message in the network, i.e. as long as $ts > time - \lambda - \alpha$ (Why subtracting α?)

An upper bound, *upper*, on the value of all timestamps removed from the connection table.

Upon reception of a message it is discarded if its timestamp is:

- either smaller than that of the entry of the connection table with the corresponding connection id
- or smaller than *upper* (if there is no entry in the CT)

Synchronized Clocks: At-most-once Messaging (3/3)

Issue: What if the receiver recovers after a crash?

- Must ensure that messages delivered before the crash are not delivered again

Solution

Naïve store in **stable storage** the largest timestamp of all messages received, and discard any message with a smaller ...

Efficient Periodicallly save in stable storage an upper-bound, $latest = time + \beta$, of the time-stamp of all received timestamps

1. Upon reception of a message with a timestamp larger than *latest* either delay its delivery or discard it
2. Upon recovery from a crash:
   - Set *upper* to *latest*, thus discarding messages whose timestamps are older than the saved upper-bound.

Trade-off

- Synchronized clocks improve performance
- At the cost of occasionally rejecting a message

Synchronized Clocks: Leases

Originally were just like a **timed locks**

- Locks are difficult to manage in an environment where lock owners may crash
- It is even possible to have read/write leases

Generally They can be used to ensure some property during a limited time, i.e. the **lease duration**

Lease duration may be

Absolute requires **synchronized clocks**
Relative requires **synchronized rates**

Renewals A lease is valid for its duration

- Unless its owner **renews** the lease

Question How long should the lease duration be?

Synchronized Clocks: Preventing Replay-Attacks

Kerberos uses Needham and Schroeder's shared key authentication protocol

- But instead of using their solution to prevent replay attacks, they use synchronized clocks
  - And, more generally, to prevent the use of keys for too long, i.e. for key expiration
- Check the details in Liskov's paper

Synchronized Clocks vs. Synchronized (Clock) Rates

- In most applications, synchronized rates are enough
  - In computer networking, retransmission mechanisms usually assume clock rates synchronized with real time rates
  - TCP also assumes that to prevent segments sent in the scope of a connection to be delivered in the scope of another connection (when a connection is shutdown abruptly)
- This is a reasonable assumption for computer systems with quartz clocks
  - Nowadays, these clocks have a drift rate lower than 10^{-6}
- Whenever possible **synchronized rates** should be used rather than **synchronized** clocks
  - Synchronized clocks depend on synchronized rates, and on occasional communication
  - But the use of synchronized rates also requires communication, so that the instant of the relevant event can be bounded
- Synchronized clocks are more powerful than synchronized rates
  - They are able to support other algorithms for which synchronized rates are not enough
  - They can provide warnings when the clocks get out of synch

Roadmap

Further Reading

- Tanenbaum and van Steen, Section 6.1 of *Distributed Systems*, 2nd Ed.
- NTP Project Page
- Barbara Liskov, *Practical Uses of Synchronized Clocks*, in Distributed Computing 6(4): 211-219 (1993)
- Time at the Bureau International des Poids e Mesure, keeper of the TAI
- Time and Frequency Division of the NIST Physical Measurement Lab
- Precise time @ US Naval Observatory, owner of the Master Clock used in GPS