What’s it for?

• Temporal ordering of events produced by concurrent processes

• Synchronization between senders and receivers of messages

• Coordination of joint activity

• Serialization of concurrent access for shared objects
Physical clocks
Logical vs. physical clocks

• Logical clock keeps track of event ordering
  – among related (causal) events

• Physical clocks keep time of day
  – Consistent across systems
Quartz clocks

1880: Piezoelectric effect
– Curie brothers
– Squeeze a quartz crystal & it generates an electric field
– Apply an electric field and it bends

1929: Quartz crystal clock
– Resonator shaped like tuning fork
– Laser-trimmed to vibrate at 32,768 Hz
– Standard resonators accurate to 6 parts per million at 31° C
– Watch will gain/lose < ½ sec/day
– Stability > accuracy: stable to 2 sec/month
– Good resonator can have accuracy of 1 second in 10 years
  • But … frequency changes with age, temperature, and acceleration
Atomic clocks

• Second is defined as 9,192,631,770 periods of radiation corresponding to the transition between two hyperfine levels of cesium-133

• Accuracy:
  better than 1 second in six million years

• NIST standard since 1960
UTC

• UT0
  – Mean solar time on Greenwich meridian
  – Obtained from astronomical observation

• UT1
  – UT0 corrected for polar motion

• UT2
  – UT1 corrected for seasonal variations in Earth’s rotation

• TAI: International Atomic Time (Temps Atomique International)
  – Weighted average of ~200 atomic clocks: TAI-UT1 = 0 on Jan 1, 1958

• UTC: Coordinated Universal Time (Temps Universel Coordonné)
  – Civil time measured on an atomic time scale
  – Kept within 0.9 seconds of UT1; integral Δ from TAI
  – Atomic clocks cannot keep mean time (UT0)
    • Mean time is a measure of Earth’s rotation
Physical clocks in computers

- Real-time Clock: CMOS clock (counter) circuit driven by a quartz oscillator
  - Battery backup to continue measuring time when power is off

- OS generally programs a timer circuit to generate a periodic interrupt
  - Timer hardware
    - Programmable Interval Timer (PIT) – Intel 8253, 8254
    - High Precision Event Timer (HPET)
    - Advanced Programmable Interval Controller (APIC)
  - E.g., 60, 100, 250, 1000 interrupts per second
    (Linux 2.6+ adjustable up to 1000 Hz)
  - Interrupt service procedure increments a counter in memory
Problem

• Getting two systems to agree on time
  – Two clocks hardly ever agree
  – Quartz oscillators oscillate at slightly different frequencies

• Clocks tick at different rates
  – Create ever-widening gap in perceived time
  – Clock Drift

• Difference between two clocks at one point in time
  – Clock Skew
8:01:24
Skew = +84 seconds
+84 seconds/35 days
Drift = +2.4 sec/day

Oct 23  8:00:00

8:01:48
Skew = +108 seconds
+108 seconds/35 days
Drift = +3.1 sec/day
Perfect clock

Computer’s time, $C$

UTC time, $t$

\[
\frac{dC}{dt} = 1
\]
Drift with slow clock

Computer’s time, C

UTC time, t

\[ \frac{dC}{dt} < 1 \]

skew

perfect time
Drift with fast clock

\[ \frac{dC}{dt} > 1 \]

\( dC/dt > 1 \)
Dealing with drift

We want to set the computer to the time of day

Not good idea to set a clock back
- Illusion of time moving backwards can confuse message ordering and software development environments
Dealing with drift

Go for *gradual* clock correction

If fast:
   Make the clock run slower until it synchronizes

If slow:
   Make the clock run faster until it synchronizes
Dealing with drift

The OS can do this:

Change the rate at which it requests interrupts

  e.g.:

    if system requests interrupts every 17 ms but clock is too slow:
    request interrupts at (e.g.) 15 ms

Not always practical: we may not have enough precision

Easier (software-only) solution

  → adjust the rate at which the system time is advanced

Adjustment changes slope of system time:

  Linear compensation function
Compensating for a fast clock

UTC time, $t$

Computer’s time, $C$

$\frac{dC}{dt} > 1$

Linear compensation function applied
Compensating for a fast clock

Now we’re drifting again!

UTC time, $t$

Computer’s time, $C$

perfect time
After synchronization period is reached
   – Resynchronize periodically
   – Successive application of a second linear compensating function can bring us closer to true slope
   – Long-term stability is not guaranteed – the system clock can still drift based on changes in temperature, pressure, humidity, and age of the crystal

Keep track of adjustments and apply continuously
   – e.g., POSIX `adjtime` system call and `hwclock` command
Going to sleep

- RTC keeps on ticking when the system is off (or sleeping)
- OS cannot apply correction continually
- Estimate drift on wake-up and apply a correction factor
Getting accurate time

• Attach GPS receiver to each computer
  – ± 100 ns to 1 μsec of UTC

• Attach WWV radio receiver
  – Obtain time broadcasts from Boulder or DC
  – ± 3 ms of UTC (depending on distance)

• Not practical solution for every machine
  – Cost, power, convenience, environment
Getting accurate time

Synchronize from another machine
  – One with a more accurate clock

Machine/service that provides time information:

*Time server*
Remote Request/Response

Simplest synchronization technique
- Send a network request to obtain the time
- Set the time to the returned value

Does not account for network or processing latency
Cristian’s algorithm

Compensate for delays

– Note times:
  • request sent: $T_0$
  • reply received: $T_1$
– Assume network delays are symmetric
Cristian’s algorithm

Client sets time to:

\[ T_{\text{new}} = T_{\text{server}} + \frac{T_1 - T_0}{2} \]

estimated overhead in each direction
Error bounds

If the minimum message transit time ($T_{min}$) is known:

Place bounds on accuracy of result
Error bounds

Earliest time message arrives

Latest time message leaves

\[ \text{range} = T_1 - T_0 - 2T_{\text{min}} \]

accuracy of result = \( \pm \frac{T_1}{2} T_0 T_{\text{min}} \)

September 24, 2014
© 2014 Paul Krzyzanowski
Cristian’s algorithm: example

- Send request at 5:08:15.100 ($T_0$)
- Receive response at 5:08:15.900 ($T_1$)
  - Response contains 5:09:25.300 ($T_{server}$)

- Elapsed time is $T_1 - T_0$
  
  5:08:15.900 - 5:08:15.100 = 800 ms

- Best guess: timestamp was generated 400 ms ago

- Set time to $T_{server} +$ elapsed time
  
  5:09:25.300 + 400 = 5:09:25.700

Note:

1 000 ms = 1 s
1 000 000 µs = 1s
Cristian’s algorithm: example

If best-case message time = 200 ms

\[ T_0 = 5:08:15.100 \]
\[ T_1 = 5:08:15.900 \]
\[ T_s = 5:09:25:300 \]
\[ T_{\text{min}} = 200 \text{ ms} \]

\[
\text{Error} = \pm \frac{900}{2} = \pm 450 \quad 200 = \pm \frac{800}{2} = \pm 400
\]
Berkeley Algorithm

- Gusella & Zatti, 1989
- Assumes no machine has an accurate time source
- Obtains average from participating computers
- Synchronizes all clocks to average
Berkeley Algorithm

• Machines run time daemon
  – Process that implements protocol

• One machine is elected (or designated) as the server (master)
  – Others are slaves
Berkeley Algorithm

• Master polls each machine periodically
  – Ask each machine for time
    • Can use Cristian’s algorithm to compensate for network latency

• When results are in, compute average
  – Including master’s time

• We hope: an average cancels out individual clock’s tendencies to run fast or slow

• Send offset by which each clock needs adjustment to each slave
  – Avoids problems with network delays if we send a time stamp
Berkeley Algorithm

Algorithm has provisions for ignoring readings from clocks whose skew is too great
  – Compute a fault-tolerant average

If master fails
  – Any slave can take over via an election algorithm
Berkeley Algorithm: example

1. Request timestamps from all slaves
2. Compute fault-tolerant average: Suppose \( \max \delta = 0.45 \)

\[
\frac{3:25 + 2:50 + 3:00}{3} = 3:05
\]
3. Send offset to each client
Network Time Protocol, NTP

• 1991, 1992
  – Internet Standard, version 3: RFC 1305

• June 2010
  – Internet Standard, version 4: RFC 5905-5908
  – IPv6 support
  – Improve accuracy to tens of microseconds
  – Dynamic server discovery
NTP Goals

• Enable clients across Internet to be accurately synchronized to UTC despite message delays
  – Use statistical techniques to filter data and gauge quality of results

• Provide reliable service
  – Survive lengthy losses of connectivity
  – Redundant paths
  – Redundant servers

• Provide scalable service
  – Enable clients to synchronize frequently
  – Offset effects of clock drift

• Provide protection against interference
  – Authenticate source of data
NTP servers

Arranged in strata

- 1\textsuperscript{st} stratum: machines connected directly to accurate time source
- 2\textsuperscript{nd} stratum: machines synchronized from 1\textsuperscript{st} stratum machines
- …

Synchronization Subnet
NTP Synchronization Modes

Multicast mode
– for high speed LANS
– Lower accuracy but efficient

Procedure call mode
– Similar to Cristian’s algorithm

Symmetric mode
– Intended for master servers
  • A probes B; B probes A → A adjusts its clock only if A’s stratum > B’s
– Peer servers can synchronize with each other to provide mutual backup
  • Pair of servers retain data to improve synchronization over time

All messages are delivered unreliably with UDP
NTP Clock Quality

- **Precision**
  - Smallest increase of time that can be read from the clock

- **Jitter**
  - Difference in successive measurements
  - Due to network delays, OS delays, and *wander* – clock oscillator instability

- **Accuracy**
  - How close is the clock to UTC?
NTP messages

- Procedure call and symmetric mode
  - Messages exchanged in pairs: request & response

- Time encoded as a 64 bit value:
  - Divide by $2^{32}$ to get the number of seconds since Jan 1 1900 UTC

- NTP calculates:
  - Offset for each pair of messages ($\theta$)
    - Estimate of time offset between two clocks
  - Delay ($\delta$)
    - Travel time: $\frac{1}{2}$ of total delay minus remote processing time
  - Jitter/Dispersion
    - Maximum offset error

- Use this data to find preferred server:
  - Probe multiple servers
  - *Pick lowest total dispersion & lowest stratum*
NTP message structure

• Leap second indicator
  – Last minute has 59, 60, 61 seconds

• Version number

• Mode (symmetric, unicast, broadcast)

• Stratum (1=primary reference, 2-15)

• Poll interval
  – Maximum interval between 2 successive messages, nearest power of 2

• Precision of local clock
  – Nearest power of 2
NTP message structure

- Root delay
  - Total roundtrip delay to primary source
  - (16 bits seconds, 16 bits decimal)

- Root dispersion
  - Nominal error relative to primary source

- Reference clock ID
  - Atomic, NIST dial-up, radio, LORAN-C navigation system, GOES, GPS, …

- Reference timestamp
  - Time at which clock was last set (64 bit)

- Authenticator (key ID, digest)
  - Signature (ignored in SNTP)
NTP message structure

- $T_1$: originate timestamp
  - Time request departed client (client’s time)
- $T_2$: receive timestamp
  - Time request arrived at server (server’s time)
- $T_3$: transmit timestamp
  - Time request left server (server’s time)
NTP’s validation tests

• Timestamp provided ≠ last timestamp received
  – duplicate message?

• Originating timestamp in message consistent with sent data
  – Messages arriving in order?

• Timestamp within range?

• Originating and received timestamps ≠ 0?

• Authentication disabled? Else authenticate

• Peer clock is synchronized?

• Don’t sync with clock of higher stratum #

• Reasonable data for delay & dispersion
SNTP

Simple Network Time Protocol

– Based on Unicast mode of NTP
– Subset of NTP, not new protocol
– Operates in multicast or procedure call mode
– Recommended for environments where server is root node and client is leaf of synchronization subnet
– Root delay, root dispersion, reference timestamp ignored

v3 RFC 2030, October 1996
v4 RFC 5905, June 2010
SNTP Example

Round-trip delay:

\[ d = (T_4 - T_1) - (T_2 - T_3) \]

Time offset:

\[ t = \frac{(T_2 - T_1) + (T_3 - T_4)}{2} \]
SNTP example

Offset =

\[ \frac{((800 - 1100) + (850 - 1200))}{2} \]

= \[ \frac{((-300) + (-350))}{2} \]

= \[ -650 / 2 = -325 \]

Time offset:

\[ t = \frac{(T_2 - T_1) + (T_3 - T_4)}{2} \]

Set time to \( T_4 + t \)

= \[ 1200 - 325 = 875 \]
Cristian’s algorithm

Offset = \( \frac{1200 - 1100}{2} = 50 \)

Set time to \( T_s + \text{ offset} = 825 + 50 = 875 \)
Key Points: Physical Clocks

• Cristian’s algorithm & SNTP
  – Set clock from server
  – But account for network delays
  – Error: uncertainty due to network/processor latency
    • Errors are additive
    • Example: ±10 ms and ±20 ms = ±30 ms

• Adjust for local clock skew
  – Linear compensating function
Precision Time Protocol
PTP: IEEE 1588 Precision Time Protocol

- Designed to synchronize clocks on a LAN to sub-microsecond precision
  - Designed for LANs, not global: low jitter, low latency
  - Timestamps ideally generated at the MAC or PHY layers to minimize delay and jitter

- Determine master clock
  - Use Best Master Clock algorithm to determine which clock in the network is most precise
  - Other clocks become slaves

- Two phases in synchronization
  1. Offset correction
  2. Delay correction
PTP: Choose the “best” clock

Best Master Clock

- Distributed election based on properties of clocks
- Criteria from highest to lowest:
  - Priority 1 (admin-defined hint)
  - Clock class
  - Clock accuracy
  - Clock variance: estimate of stability based on past syncs
  - Priority 2 (admin-defined hint #2)
  - Unique ID (tie-breaker)
Master initiates the protocol by sending a sync message containing a timestamp

Slave timestamps arrival with a timestamp from its local clock

\[ \text{Offset} + \text{Delay} = T_2 - T_1 \]
Slave needs to figure out the network delay. Send a *delay request*

Note the time it was sent.
Master marks the time of arrival and returns it in a delay response

\[
\text{Delay response} = \text{Delay} - \text{Offset} = T_4 - T_3
\]
PTP: Slave computes offset

\[ T_2 - T_1 = \text{delay} + \text{offset} \]
\[ T_4 - T_3 = \text{delay} - \text{offset} \]
\[ T_2 - T_1 + T_4 - T_3 = 2 \times (\text{offset}) \]

offset = \( \frac{T_2 - T_1 + T_4 - T_3}{2} \)
NTP vs. PTP

• Range
  – NTP: nodes widely spread out on the Internet
  – PTP: local area networks

• Accuracy
  – NTP usually several milliseconds on WAN
  – PTP usually sub-microsecond on LAN
The End