

# Distributed Systems

## 05. Clock Synchronization

Paul Krzyzanowski  
Rutgers University  
Fall 2018

September 24, 2018 © 2014-2018 Paul Krzyzanowski 1

## Synchronization

Synchronization covers interactions among distributed processes

Clocks	Identifying <i>when</i> something happened
Mutual exclusion	Only one entity can do an operation at a time
Leader election	Who coordinates activity?
Message consistency	Does everyone have the same view of events?
Agreement	Can everyone agree on a proposed value?

All of these are trivial in non-distributed systems  
All of these are tricky in distributed systems

September 24, 2018 © 2014-2018 Paul Krzyzanowski 2

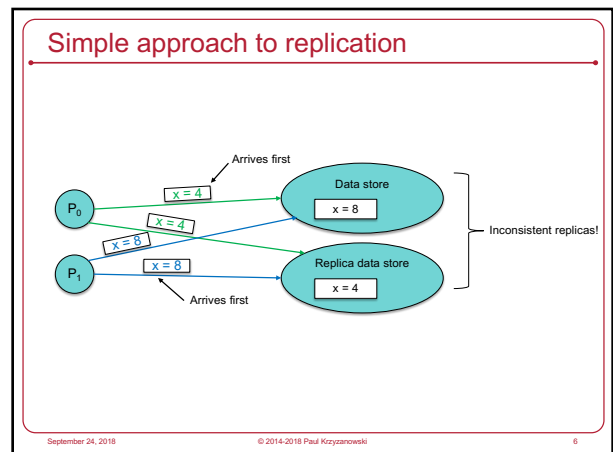
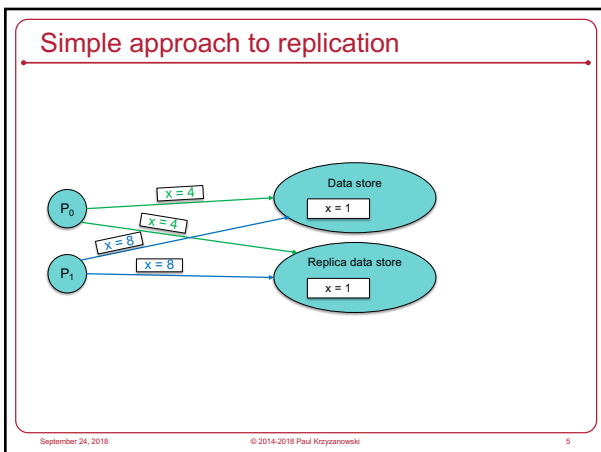
## Clock Synchronization

September 24, 2018 © 2014-2018 Paul Krzyzanowski 3

## Why?

- Allow a process to identify "now" in a way that's consistent with other processes on other systems
- Temporal ordering of events from concurrent processes
  - Example: replication & identifying latest versions
    - *Last writer wins* or *latest version wins*

September 24, 2018 © 2014-2018 Paul Krzyzanowski 4



### Simple approach to replication

Send a time stamp with each modification request  
Only newer timestamps can override older data

There are problems with this ... but physical clocks help this work most of the time

September 24, 2018 © 2014-2018 Paul Krzyzanowski 7

### Logical vs. physical clocks

- Physical clocks keep time of day
  - Consistent across systems
- Logical clock keeps track of event ordering
  - among related (causal) events

September 24, 2018 © 2014-2018 Paul Krzyzanowski 8

### Physical clocks

September 24, 2018 © 2014-2018 Paul Krzyzanowski 9

### Problem: Get two systems to agree on time

- Why is it hard?
  - 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
  - Relative offset**
- Short-term variation in frequency
  - Jitter**
- Also note: astronomical time vs. relative time
  - Time of day vs. count of seconds from **epoch**

September 24, 2018 © 2014-2018 Paul Krzyzanowski 10

### Dealing with drift

*Not good idea to set a clock back*

- Illusion of time moving backwards can confuse message ordering and software development environments

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

September 24, 2018 © 2014-2018 Paul Krzyzanowski 11

### Dealing with drift

The OS can do this:

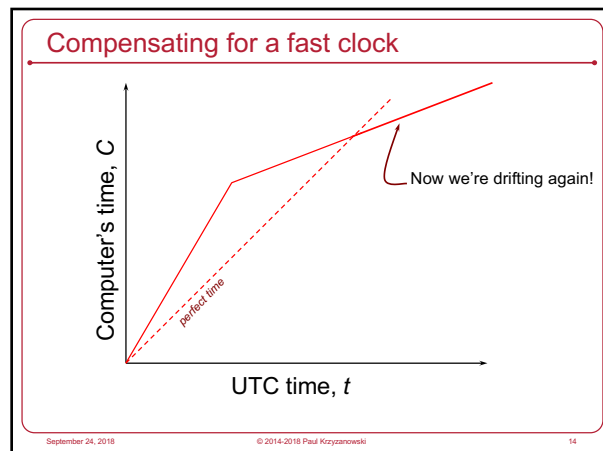
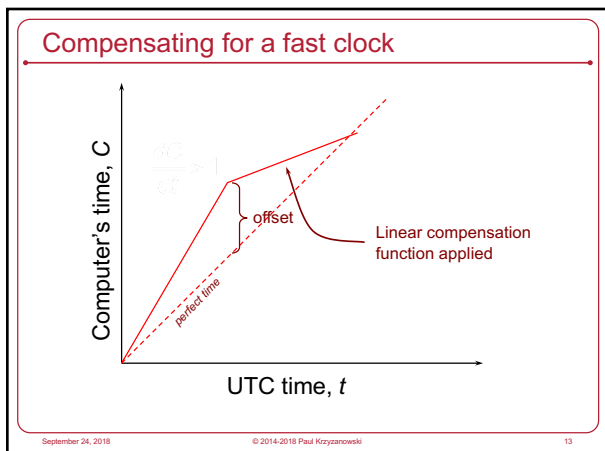
- Redefine the rate at which system time is advanced with each interrupt

or

- Read the counter but compensate for drift

Adjustment changes slope of system time:  
**Linear compensation function**

September 24, 2018 © 2014-2018 Paul Krzyzanowski 12



### Resynchronizing

After synchronization period is reached

- Resynchronize periodically
- Successive application of a second linear compensating function can bring us closer to true slope

**Long-term clock 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., BSD *adjtimex* & Linux *adjtimex* system calls and *hwclock* command

September 24, 2018 © 2014-2018 Paul Krzyzanowski 15

### Going to sleep

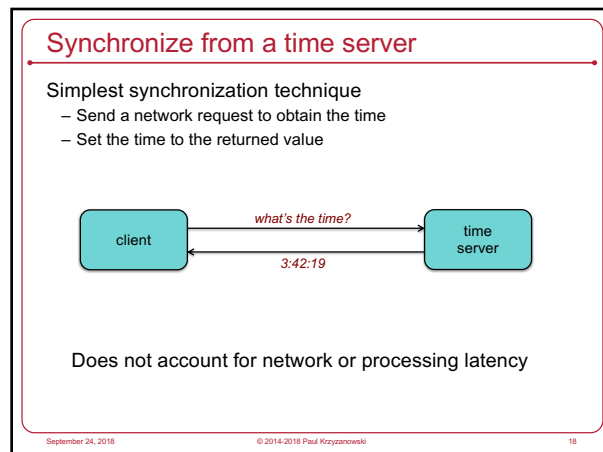
- RTC keeps on ticking when the system is off (or sleeping)
- OS cannot apply correction continually
- Estimate drift on wake-up and apply a correction factor

September 24, 2018 © 2014-2018 Paul Krzyzanowski 16

### Getting accurate time

- Attach GPS receiver to each computer
  - Accurate to ~ 40 ns
- Not practical solution for every machine
  - Cost, power, convenience, environment
  - Accuracy gets worse near buildings, bridges, trees, ...

September 24, 2018 © 2014-2018 Paul Krzyzanowski 17



### Cristian's algorithm

Compensate for delays

- Note times:
  - request sent:  $T_0$
  - reply received:  $T_1$
- Assume network delays are symmetric

server

client

$T_0$   $T_1$   $T_{server}$

request

reply

time

September 24, 2018 © 2014-2018 Paul Krzyzanowski 19

### Cristian's algorithm

Client sets time to:

server

client

$T_0$   $T_1$   $T_{server}$

request

reply

time

$\frac{T_1 - T_0}{2}$  = estimated overhead in each direction

$$T_{new} = T_{server} + \frac{T_1 - T_0}{2}$$

September 24, 2018 © 2014-2018 Paul Krzyzanowski 20

### Error bounds

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

Place bounds on accuracy of result

September 24, 2018 © 2014-2018 Paul Krzyzanowski 21

### Error bounds

server

client

$T_0$   $T_1$   $T_{min}$   $T_{min}$   $T_{server}$

request

reply

Earliest time message arrives

Latest time message leaves

range =  $T_1 - T_0 - 2T_{min}$

$$\text{accuracy of result} = \pm \frac{T_1 - T_0}{2} - T_{min}$$

September 24, 2018 © 2014-2018 Paul Krzyzanowski 22

### 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  $T_1 - T_0$ 

Note: 1 000 ms = 1 s  
1 000 000  $\mu$ s = 1s

$$5:08:15.900 - 5:08:15.100 = 800 \text{ ms}$$
- Best guess: timestamp was generated 400 ms ago
- Set time to  $T_{server} + \text{elapsed time}$ 

$$5:09:25.300 + 400 = 5:09:25.700$$

September 24, 2018 © 2014-2018 Paul Krzyzanowski 23

### Cristian's algorithm: example

If best-case message time=200 ms

$T_0 = 5:08:15.100$   
 $T_1 = 5:08:15.900$   
 $T_s = 5:09:25.300$   
 $T_{min} = 200 \text{ ms}$

server

client

$T_0$   $T_1$   $T_{server}$

request

reply

$$\text{Error} = \pm \frac{900 - 100}{2} - 200 = \pm \frac{800}{2} - 200 = \pm 200$$

September 24, 2018 © 2014-2018 Paul Krzyzanowski 24

## Berkeley Algorithm

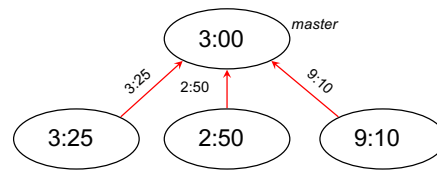
- Gusella & Zatti, 1989
- Assumes no machine has an accurate time source
- Obtains average from participating computers
- Synchronizes all clocks to a **fault-tolerant average**

September 24, 2018

© 2014-2018 Paul Krzyzanowski

25

## Berkeley Algorithm: example



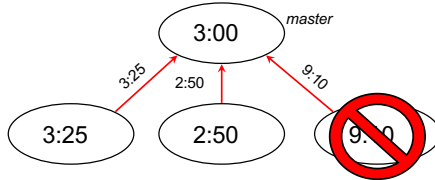
1. Request timestamps from all slaves

September 24, 2018

© 2014-2018 Paul Krzyzanowski

26

## Berkeley Algorithm: example



2. Compute fault-tolerant average: Suppose  $\max \delta = 0.45$

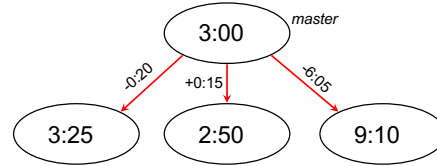
$$\frac{3:25 + 2:50 + 3:00}{3} = 3:05$$

September 24, 2018

© 2014-2018 Paul Krzyzanowski

27

## Berkeley Algorithm: example



3. Send offset to each client

September 24, 2018

© 2014-2018 Paul Krzyzanowski

28

## 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

September 24, 2018

© 2014-2018 Paul Krzyzanowski

29

## 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 huge numbers of clients to **synchronize frequently**
  - Offset effects of clock drift
- Provide **protection** against interference
  - Authenticate source of data

September 24, 2018

© 2014-2018 Paul Krzyzanowski

30

### NTP servers

Arranged in strata

- Stratum 0 = master clock
- 1<sup>st</sup> stratum: systems connected directly to accurate time source
- 2<sup>nd</sup> stratum: systems synchronized from 1<sup>st</sup> stratum systems
- ...
- 15<sup>th</sup> stratum: systems synchronized from 14<sup>th</sup> stratum systems

Synchronization Subnet

September 24, 2018 © 2014-2018 Paul Krzyzanowski 31

### NTP Synchronization Modes

- Multicast mode**
  - for high speed LANs
  - Lower accuracy but efficient
- Procedure call mode**
  - Cristian's algorithm
- Symmetric mode**
  - Peer servers can synchronize with each other to provide mutual backup
    - Usually used with stratum 1 & 2 servers
    - Pair of servers retain data to improve synchronization over time

All messages are delivered unreliably with UDP (port 123)

September 24, 2018 © 2014-2018 Paul Krzyzanowski 32

### 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 clock oscillator instability
- **Accuracy**
  - How close is the clock to UTC?

September 24, 2018 © 2014-2018 Paul Krzyzanowski 33

### NTP messages

- Procedure call and symmetric mode
  - Messages exchanged in pairs: request & response
- Time encoded as a 64 bit value:
  - Divide by 2<sup>32</sup> 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: 1/2 of total delay minus remote processing time
  - **Dispersion**
    - Maximum offset error relative to reference clock
- Use this data to find preferred server:
  - Probe multiple servers – each several times
  - *Pick lowest dispersion ... at the lowest stratum if tied*

September 24, 2018 © 2014-2018 Paul Krzyzanowski 34

### 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

September 24, 2018 © 2014-2018 Paul Krzyzanowski 35

### SNTP Example

Round-trip network delay:  $\delta = (T_4 - T_1) - (T_3 - T_2)$

Time offset:  $t = \frac{(T_2 - T_1) + (T_3 - T_4)}{2}$

September 24, 2018 © 2014-2018 Paul Krzyzanowski 36

### SNTP Example

Round-trip network delay:  $\delta = (T_4 - T_1) - (T_3 - T_2)$

Time offset:  $t = \frac{(T_2 - T_1) + (T_3 - T_4)}{2}$

September 24, 2018 © 2014-2018 Paul Krzyzanowski 37

### SNTP example

Offset = 
$$\begin{aligned} & ((800 - 1100) + (850 - 1200)) / 2 \\ & = ((-300) + (-350)) / 2 \\ & = -650 / 2 = -325 \end{aligned}$$

Time offset:  $t = \frac{(T_2 - T_1) + (T_3 - T_4)}{2}$

Set time to  $T_4 + t$   
 $= 1200 - 325 = 875$

September 24, 2018 © 2014-2018 Paul Krzyzanowski 38

### SNTP = Cristian's algorithm

Just define  $T_s = \frac{T_2 + T_3}{2}$

September 24, 2018 © 2014-2018 Paul Krzyzanowski 39

### 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:  $\pm 10$  ms and  $\pm 20$  ms =  $\pm 30$  ms
- Adjust for local clock drift
  - Linear compensating function

September 24, 2018 © 2014-2018 Paul Krzyzanowski 40

### Precision Time Protocol

September 24, 2018 © 2014-2018 Paul Krzyzanowski 41

### 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 a **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

September 24, 2018 © 2014-2018 Paul Krzyzanowski 42

### 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)

September 24, 2018 © 2014-2018 Paul Krzyzanowski 43

### PTP: Master initiates sync

Master initiates the protocol by sending a *sync* message containing a timestamp

Slave timestamps arrival with a timestamp from its local clock

$Offset + Delay = T_2 - T_1$

September 24, 2018 © 2014-2018 Paul Krzyzanowski 44

### PTP: Send delay request

Slave needs to figure out the network delay. Send a *delay request*

Note the time it was sent.

September 24, 2018 © 2014-2018 Paul Krzyzanowski 45

### PTP: Receive delay response

Master marks the time of arrival and returns it in a *delay response*

$Delay\ response = Delay - Offset = T_4 - T_3$

September 24, 2018 © 2014-2018 Paul Krzyzanowski 46

### PTP: Slave computes offset

$master\_slave\_difference = T_2 - T_1 = delay + offset$   
 $slave\_master\_difference = T_4 - T_3 = delay - offset$   
 $master\_slave\_difference - slave\_master\_difference = 2(offset)$   
 $T_2 - T_1 - T_4 + T_3 = 2(offset)$   
 $offset = (T_2 - T_1 - T_4 + T_3) / 2$

September 24, 2018 © 2014-2018 Paul Krzyzanowski 47

### 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

September 24, 2018 © 2014-2018 Paul Krzyzanowski 48



