EECS 591
Distributed Systems
Manos Kapritsos
Fall 2020
PREVIOUSLY ON DISTRIBUTED SYSTEMS
IMPLEMENTING STRONG CLOCKS
(the hard way)

Strong clock condition: \( p \rightarrow q \iff \theta(p) \subseteq \theta(q) \)
Vector clocks

Each process keeps a vector of natural numbers $VC$, one for each process

**Update rules**

If $e_i$ is a local or send event at process $i$:

$$VC(e_i)[i] := VC[i] + 1$$

If $e_i$ is a receive event of message $m$:

$$VC(e_i) := max\{VC, VC(m)\}$$
$$VC(e_i)[i] := VC[i] + 1$$
**Vector clocks**

![Diagram of vector clocks with labeled nodes and edges]

\[ VC(e_i)[j] = \text{number of events executed by process } j \text{ that causally precede } e_i \]
Comparing vector clocks

Equality
\[ V = V' \equiv \forall k : 1 \leq k \leq n : V[k] = V'[k] \]
(i.e. all elements are the same)

Inequality
\[ V < V' \equiv (V \neq V') \land (\forall k : 1 \leq k \leq n : V[k] \leq V'[k]) \]

Examples: \([2,0,0] < [2,0,1] < [3,0,1] < [4,1,1]\)

Strong clock condition: \( p \rightarrow q \iff VC(p) < VC(q) \)
Comparing vector clocks

Strong clock condition: $p \rightarrow q \Leftrightarrow VC(p) < VC(q)$
Causal delivery

A “monitor” process wants to record all messages (e.g. deadlock detection, system snapshot, etc)

- Processes send copies of their messages to the monitor
- Only increment the local component of $VC$ for send events
CAUSAL DELIVERY RULES

Monitor keeps an array $D$, where $D[i]$ is the number of messages delivered from process $i$

Monitor delivers message $m$ from process $j$ when:

$D[j] = VC(m)[j] - 1$

$D[k] \geq VC(m)[k], \forall k \neq j$
Causal delivery

\[ D \]

\begin{align*}
D[j] &= VC(m)[j] - 1 & (0, 0) & \checkmark & \checkmark & \checkmark \\
D[k] &\geq VC(m)[k], \forall k \neq j & (1, 0) (1, 1) & \checkmark & \checkmark & \checkmark
\end{align*}
ADMINISTRIVIA

- I've given a few more overrides. At capacity.
- I added an OH queue link to the class info.
- Eli’s OH are cancelled for tomorrow, as he is participating in the GEO strike.
Clock synchronization

What time is it?
Roman generals v2.0

Attack at midnight!

Chaaaaaarge!

12:00am

ZZZZZ

11:30pm
Clock drift

- **Bound on drift:** $\rho$

\[(1 - \rho)(t - t') \leq H(t) - H(t') \leq (1 + \rho)(t - t')\]

- **$\rho$ is typically small ($10^{-6}$)**

$\rho^2 \approx 0$

\[\frac{1}{1 - \rho} = 1 + \rho\]

\[\frac{1}{1 + \rho} = 1 - \rho\]
External vs internal synchronization

External Clock Synchronization:
keeps clock within some maximum deviation from an external time source.

- exchange of info about timing events of different systems
- can take actions at real-time deadlines

Internal Clock Synchronization:
keeps clocks within some maximum deviation from each other.

- can measure duration of distributed activities that start on one process and terminate on another
- can totally order events that occur in a distributed system
Probabilistic Clock Synchronization (Cristian)

- Master-Slave architecture
- Master can be connected to external time source
- Slaves read master's clock and adjust their own

How accurately can a slave read the master's clock?
Setup and assumptions

Goal: Synchronize the slave’s clock with the master

Assume that minimum delay is known
Assume that clock drifts are known ($\rho$ for both)
The protocol

$t$ (real time) \hspace{3cm} t = x$

slave $P(t)$ \hspace{3cm} “time=?”

master $Q(t)$ \hspace{3cm} “time=$T$”

$T$ \hspace{3cm} $Q(x)$

Question: what is $Q(x)$?
Ideal scenario

Assume no clock drift

\[ P(t) \]

\[ \text{slave} \]

\[ t = x \]

\[ Q(x) = T + \text{min} \rightarrow \text{Perfect synchronization!} \]

\[ \text{master} \]

\[ T \quad T + \text{min} \]

Assume no clock drift
Problem #1: message delay

\[ P(t) \rightarrow Q(t) \]

\[ t \]

\[ T \]

One extreme:

\[ Q(x) = T + 2d - \min \]

\[ \beta = 2d - 2\min \]

Another extreme:

\[ Q(x) = T + \min \]

\[ \beta = 0 \]
Problem #2: slave drift

\[2d(1 - \rho) \leq 2D \leq 2d(1 + \rho)\]
Problem #3: master drift

During the master’s clock drifts
Even if you know $\beta$, there is still some uncertainty!
Cristian's algorithm

\[ \text{slave} \quad P(t) \quad \text{master} \quad Q(t) \]

\[ \text{time=} \, ? \quad \text{time=} \, T \]

\[ t \quad 2d \quad \text{min} + \alpha \quad 2D \quad \text{min} + \beta \quad t = x \quad \alpha, \beta \geq 0 \]

\[ T \quad Q(x) = ? \]
Cristian’s algorithm

Naive estimation: \( Q(x) = T + (\text{min} + \beta) \)

(take master’s drift into account)

\[ Q(x) \in [T + (\text{min} + \beta)(1 - \rho), T + (\text{min} + \beta)(1 + \rho)] \]

\( 0 \leq \beta \leq 2d - 2\text{min} \) (take delay into account)

\[ Q(x) \in [T + (\text{min} + 0)(1 - \rho), T + (\text{min} + 2d - 2\text{min})(1 + \rho)] \]

\[ = [T + (\text{min})(1 - \rho), T + (2d - \text{min})(1 + \rho)] \]

(take slave’s drift into account)

\[ 2d \leq 2D(1 + \rho) \]

\[ Q(x) \in [T + (\text{min})(1 - \rho), T + (2D(1 + \rho) - \text{min})(1 + \rho)] \]

\[ = [T + (\text{min})(1 - \rho), T + 2D(1 + 2\rho) - \text{min}(1 + \rho)] \]
Slave's estimation and precision

Slave's best guess: \[ Q(x) = T + D(1 + 2\rho) - \min \cdot \rho \]

Maximum error: \[ e = D(1 + 2\rho) - \min \]

You can keep trying, until you achieve the required precision

(if that precision is reasonable)
Adjusting the clock

After synchronizing:

If slave simply sets $P(x) = Q(x)$, it could create time discontinuities.
Adjusting the clock

Logical clock: \[ C(t) = H(t) + A(t) \]

- Hardware clock
- Adjustment function
Network Time Protocol

- The oldest distributed protocol still running on the Internet
- Hierarchical architecture
- Latency-tolerant, jitter-tolerant, fault-tolerant.. very tolerant!
Hierarchical structure

Each level is called a “stratum”

- Stratum 0: atomic clocks
- Stratum 1: time servers with direct connections to stratum 0
- Stratum 2: Use stratum 1 as time sources and work as server to stratum 3
- etc....

Accuracy is loosely coupled with stratum level
Very tolerant. How?

- Tolerance to jitter, latency, faults: redundancy
- Each machine sends NTP requests to many other servers on the same or the previous stratum
- The synchronization protocol between two machines is similar to Cristian’s algorithm
- Each response defines an interval \([T_1, T_2]\)
- How to combine those intervals?
Marzullo’s algorithm

Given $M$ source intervals, find the largest interval that is contained in the largest number of source intervals

\[
\begin{align*}
[8, 12] \\
[11, 13] \\
[10, 12] \\
[11, 12]
\end{align*}
\]

\[
\begin{align*}
10 \pm 2 \\
12 \pm 1 \\
11 \pm 1 \\
11.5 \pm 0.5
\end{align*}
\]
Marzullo's algorithm

Given $M$ source intervals, find the largest interval that is contained in the largest number of source intervals.
The intuition

- Visit the endpoints left-to-right
- Count how many source intervals are active at each time
- Increase count at starting points, decrease at ending points