Time

Lakshmi Ganesh
(slides borrowed from
Maya Haridasan,
Michael George)
The Problem

Given a collection of processes that can...
- only communicate with significant latency
- only measure time intervals approximately
- fail in various ways

... we want to construct a shared notion of time
The Problem

Given a collection of processes that can...
- only communicate with significant latency
- only measure time intervals approximately
- fail in various ways

... we want to construct a shared notion of time

But each process has a h/w clock, right??
What’s wrong with the clocks?
What’s wrong with the clocks?

Logical Clock = H/w clock + Adjustment factor
External Vs. Internal Clock Synchronization

- **External clock synchronization:**
  ‘Adjust’ clocks with respect to an external time reference

- **Accuracy:** how close logical time is to real time

- **Internal clock synchronization (ICS):**
  ‘Adjust’ clocks among themselves

- **Precision:** how close the clocks are to each other
Software Clock Synchronization

1. **Deterministic** → assumes an upper bound on transmission delays (which bounds accuracy) – guarantees some precision

2. **Statistical** → expectation and standard deviation of the delay distributions are known

3. **Probabilistic** → no assumptions about delay distributions (gives better accuracy)
Software Clock Synchronization

1. **Deterministic** → assumes an upper bound on transmission delays (which bounds accuracy) – guarantees some precision

2. **Statistical** → expectation and standard deviation of the delay distributions are known

3. **Probabilistic** → no assumptions about delay distributions (gives better accuracy)

Realistic?
Software Clock Synchronization

1. **Deterministic** \(\rightarrow\) assumes an upper bound on transmission delays (which bounds accuracy) – guarantees some precision

2. **Statistical** \(\rightarrow\) expectation and standard deviation of the delay distributions are known

3. **Probabilistic** \(\rightarrow\) no assumptions about delay distributions (gives better accuracy)
Software Clock Synchronization

1. **Deterministic** → assumes an upper bound on transmission delays (which bounds accuracy) – guarantees some precision

2. **Statistical** → expectation and standard deviation of the delay distributions are known

3. **Probabilistic** → no assumptions about delay distributions (gives better accuracy)

**Realistic?**

**Reliable?**

**Any guarantees?**
We will discuss two papers that solve ICS:

- **Optimal Clock Synchronization** [Srikanth and Toueg ’87]
  - Assume reliable network (deterministic)
  - Provide logical clock with optimal agreement
  - Also optimal with respect to failures

- **Probabilistic Internal Clock Synchronization** [Cristian and Fetzer ’03]
  - Drop requirements on network (probabilistic)
  - Provide very efficient logical clock
  - Only provide probabilistic guarantees
We assume...
Clock drift is bounded

\[(1 - \rho)(t - s) \leq H_p(t) - H_p(s) \leq (1 + \rho)(t - s)\]

Communication and processing are reliable

\[t_{recv} - t_{send} \leq t_{del}\]

Authenticated messages

will relax this later...
Paper 1: Our Goals

- **Property 1 (Agreement):**
  \[ |L_{pi}(t) - L_{pj}(t)| \leq \delta, \]
  \((\delta \text{ is the precision of the clock synchronization algorithm})\)

- **Property 2 (Accuracy):**
  \[ (1 - \rho_v)(t - s) + a \leq L_p(t) - L_p(s) \leq (1 + \rho_v)(t - s) + b \]
Paper 1: Our Goals

**Property 1 (Agreement):**

\[| L_{pi}(t) - L_{pj}(t) | \leq \delta,\]

(\(\delta\) is the precision of the clock synchronization algorithm)

**Property 2 (Accuracy):**

\[(1 - \rho_v)(t - s) + a \leq L_p(t) - L_p(s) \leq (1 + \rho_v)(t - s) + b\]

\(\rho_v \neq \rho\)

What is optimal accuracy?
Paper 1: Our Goals

Optimal Accuracy
- Drift rate of the synchronized clocks is bounded by the maximum drift rate of correct hardware clocks

\[ \rho_v = \rho \]

Fault-tolerant
- Up to \( f \) crash failures, performance failures, arbitrary (Byzantine) failures
Authenticated Algorithm

$k_{th}$ resynchronization - Waiting for time $kP$

real time $t$

logical time $kP$

$P$ – logical time between resynchronizations
Authenticated Algorithm

$k_{th}$ resynchronization - Waiting for time $kP$

logical time $kP$

$P$ – logical time between resynchronizations
Authenticated Algorithm

$k_{th}$ resynchronization - Waiting for time $kP$

$P$ – logical time between resynchronizations

logical time $kP$
Authenticated Algorithm

$k_{th}$ resynchronization - Waiting for time $kP$

$P$ – logical time between resynchronizations
Authenticated Algorithm

$k_{th}$ resynchronization - Waiting for time $kP$

Ready to synchronize

logical time $kP$

P – logical time between resynchronizations
Authenticated Algorithm

\(k_{th}\) resynchronization - Waiting for time \(kP\)

Logical time \(kP\)

\(P\) – logical time between resynchronizations
Authenticated Algorithm

\[ k_{th} \text{ resynchronization - Waiting for time } kP \]

P – logical time between resynchronizations
Authenticated Algorithm

$k_{th}$ resynchronization - Waiting for time $kP$

Ready to synchronize

logical time $kP$

$P$ – logical time between resynchronizations
Authenticated Algorithm

$k_{th}$ resynchronization - Waiting for time $kP$

$P$ – logical time between resynchronizations

$logical \ time \ kP$
Authenticated Algorithm

\( k_{th} \) resynchronization - Waiting for time \( kP \)

 logical time \( kP \)

P – logical time between resynchronizations
Authenticated Algorithm

$k_{th}$ resynchronization - Waiting for time $kP$

$P$ – logical time between resynchonizations
Authenticated Algorithm

$k_{th}$ resynchronization - Waiting for time $kP$

Synchronize!

$P$ – logical time between resynchronizations

logical time $kP$
Authenticated Algorithm

$k_{th}$ resynchronization - Waiting for time $kP$

Synchronize!

logical time $kP$

P – logical time between resynchronizations
Authenticated Algorithm

$k_{th}$ resynchronization - Waiting for time $kP$

$kP + \alpha$

P – logical time between resynchronizations

logical time $kP$
Authenticated Algorithm

\( k_{th} \) resynchronization - Waiting for time \( kP \)

Synchronize!

\( kP + \alpha \)

Logical time \( kP \)

\( P \) – logical time between resynchronizations
Achieving Optimal Accuracy

Uncertainty of $t_{delay}$ introduces a difference in the logical time between resynchronizations

$\rightarrow$ Reason for non-optimal accuracy

Solution:

- Slow down the logical clocks by a factor of

$$\frac{P}{(P - \alpha + \beta)}$$

where $\beta = \frac{t_{del}}{2(1 + \rho)}$
Authenticated Messages

- **Correctness:**
  If at least $f + 1$ correct processes broadcast messages by time $t$, then every correct process accepts the message by time $t + t_{del}$

- **Unforgeability:**
  If no correct process broadcasts a message by time $t$, then no correct process accepts the message by $t$ or earlier

- **Relay:**
  If a correct process accepts the message at time $t$, then every correct process does so by time $t + t_{del}$
Nonauthenticated Algorithm

- Replace signed communication with a broadcast primitive
  - Primitive relays messages automatically
  - Cost of $O(n^2)$ messages per resynchronization

- New limit on number of faulty processes allowed:
  - $n > 3f$
Broadcast Primitive

→ \((\text{echo, round } k)\)
Broadcast Primitive

Received \( f + 1 \) distinct (init, round \( k \))!
Broadcast Primitive

Received $f + 1$ distinct $(\text{echo, round } k)$!

$\rightarrow (\text{echo, round } k)$
Broadcast Primitive

1. Received $f + 1$ distinct (init, round $k$)!

2. Received $f + 1$ distinct (echo, round $k$)!

3. Received $2f + 1$ distinct (echo, round $k$)! Accept (round $k$)

→ (echo, round $k$)
Initialization and Integration

- Same algorithms can be used to achieve initial synchronization and integrate new processes into the network
  - A process independently starts clock $C^0$
  - On accepting a message at real time $t$, it sets $C^0(t) = \alpha$
- “Passive” scheme for integration of new processes
Paper 2: Why try another approach?

- Traditional deterministic fault-tolerant clock synchronization algorithms:
  - Assume bounded communication delays
  - Require the transmission of at least $N^2$ messages each time $N$ clocks are synchronized
  - Bursty exchange of messages within a narrow re-synchronization real-time interval
Probabilistic ICS

Claims:

- Proposes family of fault-tolerant internal clock synchronization (ICS) protocols
- Probabilistic reading achieves higher precisions than deterministic reading
- Doesn’t assume unbounded communication delays
- Use of convergence function $\Rightarrow$ optimal accuracy
Their approach

- Only requires to send a number of unreliable broadcast messages
- Staggers the message traffic in time
- Uses a new transitive remote clock reading method

Number of messages in the best case: $N + 1$

((N time server processes)
Probabilistic Clock Reading

Basic Idea:
Probabilistic Clock Reading

**Basic Idea:**

- $T_0$
- $T_2$
- $T_1$
- $m_1$
- $m_2$
- $p$
- $q$
Probabilistic Clock Reading

**Basic Idea:**

$$(T_2 - T_0)(1 + \rho) = \text{maximum bound (real time)}$$
Probabilistic Clock Reading

- **Basic Idea:**
Probabilistic Clock Reading

Basic Idea:

\[ \min \leq t(m_2) \leq (T_2 - T_0)(1 + \rho) - \min \]
Probabilistic Clock Reading

**Basic Idea:**

\[ \min \leq t(m_2) \leq (T_2 - T_0)(1 + \rho) - \min \]

\[ C_q = T_1 + \frac{\max(m_2)(1 + \rho) + \min(m_2)(1 - \rho)}{2} \]
Probabilistic Clock Reading

Basic Idea:

Is \( \text{error} \leq \Lambda \)?

Yes: Success
No? Try reading again (Limit: D)
Probabilistic Clock Reading

Basic Idea:

Is error $\leq \Lambda$?

Yes: Success

No? Try reading again (Limit: D)

Maximum acceptable clock reading error
Staggering Messages

$p$ slots per cycle
$k$ cycles per round
Transitive Remote Clock Reading

- Can reduce the number of messages per round to $N + 1$

\[ C_q(T,p) \]
\[ C_r(T,p) \]
\[ C_r(T,q) \]
Transitive Remote Clock Reading

- Can reduce the number of messages per round to $N + 1$

\[ C_r(T,q) = C_r(T,p) + T - C_q(T,p) \]
Transitive Remote Clock Reading

- Can reduce the number of messages per round to $N + 1$

- Cannot be used when arbitrary failures can occur!
Round Message Exchange Protocol
Round Message Exchange Protocol

**Request Mode**

Clock times:

<table>
<thead>
<tr>
<th></th>
<th>p</th>
<th>q</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>err</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
Round Message Exchange Protocol

**Request Mode**

Clock times:

<table>
<thead>
<tr>
<th></th>
<th>p</th>
<th>q</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>err</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

*request messages*

**Reply Mode**

Clock times:

<table>
<thead>
<tr>
<th></th>
<th>p</th>
<th>q</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td>10</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>err</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

*reply messages*
Round Message Exchange Protocol

**Request Mode**
- **Clock times:**
  - $p$: 
  - $q$: 
  - $r$: 
  - $t$: ?
  - $err$: ?
- **Request messages**

**Reply Mode**
- **Clock times:**
  - $p$: 10
  - $q$: 11
  - $r$: 10
  - $t$: ?
  - $err$: ?
- **Reply messages**

**Finish Mode**
- **Clock times:**
  - $p$: 10
  - $q$: 11
  - $r$: 10
  - $t$: 1
  - $err$: 1

**Finish messages**
Outline of Algorithms

Round clock $C_p^k$ of process $p$ for round $k$:

$$C_p^k(t) = H_p(t) + A_p^k$$

```c
Void synchronizer() {
    ReadClocks(..)
    $A = A + cf(n(rank()), Clocks, Errors)$
    $T = T + P$
}
```
Convergence Functions

Let $I(t) = [L, R]$ be the interval spanned by at $t$ by correct clocks. If all processes would set their virtual clocks at the same time $t$ to the midpoint of $I(t)$, then all correct clocks would be exactly synchronized at that point in time.

Unfortunately, this is not a perfect world!
Convergence Functions

- Each correct process makes an approximation $I_p$ which is guaranteed to be included in a bounded extension of the interval of correct clocks $I$:

$$I_{\Lambda}^k(t) = [\min\{C_s^k(t) - \Lambda\}, \max\{C_s^k(t) + \Lambda\}]$$

Deviation of clocks is bounded by $\delta$, so length of $I_{\Lambda}^k(t)$ is bounded by $\delta + 2\Lambda$. 

## Failure classes

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Tolerated Failures</th>
<th>Required Processes</th>
<th>Tolerated types of failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSA Crash</td>
<td>F</td>
<td>F + 1</td>
<td>Crash</td>
</tr>
<tr>
<td>CSA Read</td>
<td>F</td>
<td>2F + 1</td>
<td>Crash, Reading</td>
</tr>
<tr>
<td>CSA Arbitrary</td>
<td>F</td>
<td>3F + 1</td>
<td>Arbitrary, Reading</td>
</tr>
<tr>
<td>CSA Hybrid</td>
<td>Fc, Fr, Fa</td>
<td>3Fa + 2Fr + Fc + 1</td>
<td>Crash, Read., Arb.</td>
</tr>
</tbody>
</table>
Conclusions – Which one is better?

First Paper (deterministic algorithm)

- Simple algorithm
- Unified solution for different types of failures
- Achieves optimal accuracy
- Assumes bounded communication
- $O(n^2)$ messages
- Bursty communication
Conclusions – Which one is better?

- **Second Paper (probabilistic algorithm)**
  - Takes advantage of the current working conditions, by invoking successive round-trip exchanges, to reach a tight precision)
  - Precision is not guaranteed
  - Achieves *optimal accuracy*
  - $O(n)$ messages
Conclusions – Which one is better?

- Second Paper (probabilistic algorithm)
  - Takes advantage of the current working conditions, by invoking successive round-trip exchanges, to reach a tight precision)
  - Precision is not guaranteed
  - Achieves *optimal accuracy*
  - $O(n)$ messages

If both algorithms achieve optimal accuracy,

Then why is there still work being done?