Adaptive Internal Clock Synchronization

SRDS 2008

Zbigniew Jerzak, Robert Fach, and Christof Fetzer

October 6, 2008

Introduction

Build safer distributed systems using weaker assumptions [FC03]

In this paper/presentation:

- Relaxed specification for internal clock synchronization
- New clock synchronization algorithm
- New way to deal with crash failures
1. Assume a bounded deviation $\Delta_{\text{max}}$

2. However, in timed asynchronous applications:
   - messages get dropped or are arbitrarily delayed
   - different nominal and observed oscillator frequencies
   - clocks’ speeds fluctuate with time
   - processes crash

3. ...thus 1 cannot be guaranteed

Result:

An inherently inflexible and a failure-prone system
Current Approach

1. Assume a bounded deviation $\Delta_{\text{max}}$

2. However, in timed asynchronous applications:
   - messages get dropped or are arbitrarily delayed
   - different nominal and observed oscillator frequencies
   - clocks’ speeds fluctuate with time
   - processes crash

3. ...thus 1 cannot be guaranteed

Result:

An inherently inflexible and a failure-prone system
Current Approach

1. Assume a bounded deviation $\Delta_{\text{max}}$

2. However, in timed asynchronous applications:
   - messages get dropped or are arbitrarily delayed
   - different nominal and observed oscillator frequencies
   - clocks’ speeds fluctuate with time
   - processes crash

3. ...thus 1 cannot be guaranteed

Result:

An inherently inflexible and a failure-prone system
Current Approach

1. Assume a bounded deviation $\Delta_{\text{max}}$
2. However, in timed asynchronous applications:
   - messages get dropped or are arbitrarily delayed
   - different nominal and observed oscillator frequencies
   - clocks’ speeds fluctuate with time
   - processes crash
3. ...thus 1 cannot be guaranteed

Result:

An inherently inflexible and a failure-prone system
Our Approach

- No upper bound for the deviation ($\Delta_{\text{max}}$)
  - compute deviation between processes in real-time
  - use computed deviation to adapt system behavior
- No bound on the maximum number of crashed processes
  - mask crashes using clock readings extrapolation

Result:

An adaptive, safer system
Our Approach

- No upper bound for the deviation ($\Delta_{\text{max}}$)
  - compute deviation between processes in real-time
  - use computed deviation to adapt system behavior
- No bound on the maximum number of crashed processes
  - mask crashes using clock readings extrapolation

Result:

An adaptive, safer system
Our Approach

- No upper bound for the deviation ($\Delta_{\text{max}}$)
  - compute deviation between processes in real-time
  - use computed deviation to adapt system behavior
- No bound on the maximum number of crashed processes
  - mask crashes using clock readings extrapolation

Result:
An adaptive, safer system
Use Case

- $p$ is assigned a time slot $[S, T]$ for access to $R$
- $[S, T]$ defined w.r.t. some abstract clock
- $p$ is not synchronized with a priori known $\Delta_{\text{max}}$, instead...
- Use $E_p$ – maximum local clock synchronization error
  - an upper bound on the deviation from some abstract clock
  - calculated by every process
  - propagated to application layer for error handling
- $p$ uses $R$ only within $[S + E_p, T - E_p]$
  - $p$ knows how well it is synchronized
  - $p$ knows when it is allowed to access $R$
Use Case

- $p$ is assigned a time slot $[S, T]$ for access to $R$
- $[S, T]$ defined w.r.t. some abstract clock
- $p$ is not synchronized with a priori known $\Delta_{\text{max}}$, instead...
- **Use** $E_p$ – maximum local clock synchronization error
  - an upper bound on the deviation from some abstract clock
  - calculated by every process
  - propagated to application layer for error handling
- $p$ uses $R$ only within $[S + E_p, T - E_p]$
  - $p$ knows how well it is synchronized
  - $p$ knows when it is allowed to access $R$
Use Case

- $p$ is assigned a time slot $[S, T]$ for access to $R$
- $[S, T]$ defined w.r.t. some abstract clock
- $p$ is not synchronized with a priori known $\Delta_{\text{max}}$, instead...
- Use $E_p$ – maximum local clock synchronization error
  - an upper bound on the deviation from some abstract clock
  - calculated by every process
  - propagated to application layer for error handling
- $p$ uses $R$ only within $[S + E_p, T - E_p]$
  - $p$ knows how well it is synchronized
  - $p$ knows when it is allowed to access $R$
Motivation

Background

Adaptive Clock Sync.

Evaluation

Summary

6 of 25 slides

Processes and Hardware Clocks

based on TADSM [CF99]

- Set of $\mathbb{N} = \{p_0, p_1, \ldots, p_n\}$ processes
- Connected via communication bus
- Hardware clocks $- H_p(t)$:
  - bounded drift:
    $$\forall p \in \mathbb{N} \forall t : |\rho_p(t)| \leq \rho_{\text{max}}$$
  - within linear envelope of real time:
    $$(t - s)(1 - \rho_{\text{max}}) \leq H_p(t) - H_p(s) \leq (t - s)(1 + \rho_{\text{max}})$$
Generic Internal Clock Synchronization Algorithm

```plaintext
1 ClockVal Ap;  // current adjustment
2 ClockVal T;  // end of current round

4 void init() {
5   (Ap, T) = initialAdjustment();
6   // every P starting at T
7   schedule(synchronizer, P, T);
8 }

10 void synchronizer() {
11   // N – number of processes
12   ClockVal clk[N],
13   ClockVal err[N];
14   // remote clock reading
15   readClocks(clk, err);
16   // adjustment for the next round
17   Ap = adjust(Ap, T, clk, err);
18   // set T to next round
19   T = T + P;
20 }
```
Software Clocks and Remote Clock Reading

- We do not apply $A_p$ directly to the $H_p(t)$
- Instead we use software clocks:
  \[ S_p(t) = H_p(t) + a_p(t) \]

- $A_p$ is calculated based on the remote clock readings
  - probabilistic remote clock reading [FC99b]
  - provides values of the remote clocks $clk[]$
  - and accompanying remote clock reading errors $err[]$
Failure Model

- Messages are supposed to be delivered within $\delta$
  - might be delayed
  - delivered out of order
  - get dropped

- Arbitrary hardware clock failures: $|\rho_p(t)| > \rho_{\text{max}}$
  - converted to stop failures [FC99a]

- Processes’ value (byzantine) failures
  - handled by Software Encoded Processing [WF07]

Specifically:

No upper bound on the frequency of communication and process failures
Failure Model

- Messages are supposed to be delivered within $\delta$
  - might be delayed
  - delivered out of order
  - get dropped
- Arbitrary hardware clock failures: $|\rho_p(t)| > \rho_{\text{max}}$
  - converted to stop failures [FC99a]
- Processes’ value (byzantine) failures
  - handled by Software Encoded Processing [WF07]

Specifically:

No upper bound on the frequency of communication and process failures
Problem Statement

\(\forall p, q \in \mathbb{N} : \ S_p(t) = S_q(t)\) (1)

- However, achieving 1 is not possible
- Minimize & bound difference between software clocks:
  - \(\forall p, q \in \mathbb{N} : p, q \not\text{crashed}(t) \forall t : |S_p(t) - S_q(t)| \leq E_p(t) + E_q(t)\)
Problem Statement

\[ \forall p, q \in \mathbb{N} : \quad S_p(t) = S_q(t) \quad (1) \]

- However, achieving 1 is not possible
- Minimize & bound difference between software clocks:
  - \[ \forall p,q \in \mathbb{N} : p,q \not\in \text{crashed}(t) \forall t : \quad |S_p(t) - S_q(t)| \leq E_p(t) + E_q(t) \]
Problem Statement

\( \forall p, q \in \mathbb{N} : S_p(t) = S_q(t) \) (1)

- However, achieving 1 is not possible
- Minimize & bound difference between software clocks:
  - \( \forall p, q \in \mathbb{N} : p, q \not\text{crashed}(t) \forall t : |S_p(t) - S_q(t)| \leq E_p(t) + E_q(t) \)

\[ 
\begin{array}{c|c|c}
\text{q} & \text{Abstract Clock} & \text{p} \\
\hline
& E_q & \rightarrow \text{E} & \leftarrow E_p & \\
\end{array} 
\]
$E_p$ – Maximum Clock Synchronization Error

\[ E_p(T) = |C_p(T, p) - \text{cfn}| + \mathcal{E}_p(T) \]

- \( \text{cfn}() = \text{mid}(L^i_p, U^i_p) \)
  - determines the \( A_p \)

- \( \mathcal{E}_p(T) = \max \left\{ \text{cfn} - \hat{M}_p(T), \hat{M}_p(T) - \text{cfn} \right\} \)
  - determines the midpoint estimation error
$E_p$ – Maximum Clock Synchronization Error

$$E_p(T) = |C_p(T, p) - \text{cfn}()| + \mathcal{E}_p(T)$$

- $\text{cfn}() = \text{mid}([L_p^i, U_p^i])$
  - determines the $A_p$

- $\mathcal{E}_p(T) = \max \left\{ \text{cfn}() - \bar{\mathcal{M}}_p(T), \bar{\mathcal{M}}_p(T) - \text{cfn}() \right\}$
  - determines the midpoint estimation error
Convergence Functions: $\text{cfn}(()) = \text{mid}([L^i_p, U^i_p])$

$E_p(T) = |C_p(T, p) - \text{cfn}()| + \mathcal{E}_p(T)$

- A convergence function:
  - determines an abstract clock value which...
  - ...should be reached at the end of the next round

- Specifically:
  - $\text{cfn}_{V0}()$ — based on [WL88]
  - $\text{cfn}_{V1}()$ — based on [CF94]
Convergence Functions: \( \text{cfn}() = \text{mid}([L^i_p, U^i_p]) \)

\[ E_p(T) = |C_p(T, p) - \text{cfn}()| + \mathcal{E}_p(T) \]

- A convergence function:
  - determines an abstract clock value which...
  - ...should be reached at the end of the next round

- Specifically:
  - \( \text{cfn}_0() \) — based on [WL88]
  - \( \text{cfn}_1() \) — based on [CF94]
Convergence Functions: $cfn() = \text{mid}(\left[L_p^i, U_p^i\right])$

$E_p(T) = |C_p(T, p) - cfn()| + \epsilon_p(T)$

- **Perfect World** (no transmission delays, no drift):

  $p \quad C_q(T, p) \quad C_r(T, p)$

  $\text{cfn}()$
Convergence Functions: \( \text{cfn}(\cdot) = \text{mid}([L^i_p, U^i_p]) \)

\[ E_p(T) = |C_p(T, p) - \text{cfn}(\cdot)| + \varepsilon_p(T) \]

- Real Life (transmission delays, drift):
Midpoint Estimation Error: $\overrightarrow{M}_p(T), \overleftarrow{M}_p(T)$

$$E_p(T) = |C_p(T, p) - \text{cfn()}| + \epsilon_p(T)$$

- Possible values a midpoint can take:

$$\overrightarrow{M}_p(T) = \text{mid}\left(\overrightarrow{L}_p(T), \overrightarrow{U}_p(T)\right)$$

$$\overleftarrow{M}_p(T) = \text{mid}\left(\overleftarrow{L}_p(T), \overleftarrow{U}_p(T)\right)$$

\[\text{cfn()}\]
Extrapolation & Intersection

- Re-use clock readings from previous round
- Mask transient clock reading failures (extrapolation)
- Improve readings with large error (intersection)

Extrapolation:
- In round $i$ use the reading from round $i-1$
- Shift clock value by $P$, extend error by $(k+2)P\rho_{\text{max}}$

Intersection:
- Intersect readings from round $i$ with extrapolation from $i-1$
- Both correct $\rightarrow$ result also correct
Extrapolation & Intersection

- Re-use clock readings from previous round
- Mask transient clock reading failures (extrapolation)
- Improve readings with large error (intersection)
- Extrapolation:
  - in round \( i \) use the reading from round \( i - 1 \)
  - shift clock value by \( P \), extend error by \( (k + 2)P\rho_{\text{max}} \)
- Intersection:
  - intersect readings from round \( i \) with extrapolation from \( i - 1 \)
  - both correct \( \rightarrow \) result also correct
Extrapolation & Intersection

- Re-use clock readings from previous round
- Mask transient clock reading failures (extrapolation)
- Improve readings with large error (intersection)

**Extrapolation:**
- in round \(i\) use the reading from round \(i - 1\)
- shift clock value by \(P\), extend error by \((k + 2)P\rho_{\text{max}}\)

**Intersection:**
- intersect readings from round \(i\) with extrapolation from \(i - 1\)
- both correct \(\rightarrow\) result also correct
Evaluation Environment

- OMNeT++ discrete event based simulator
  - global observer view only possible in simulation
- TDMA
- 10 Mbit, half-duplex, shared Ethernet
- $2 \cdot 10^5$ km/s signal propagation speed
- 4 hosts:
Imprecision – $\text{cfn}_{V_0}()$ & $\text{cfn}_{V_1}()$
Intersection Gain

Adaptive Internal Clock Synchronization

Zbigniew Jerzak, Robert Fach, and Christof Fetzer
Intersection Gain

Adaptive Internal Clock Synchronization

Zbigniew Jerzak, Robert Fach, and Christof Fetzer
Extrapolation and Omissions

![Graph showing error vs. real-time]
Extrapolation and Omissions

The graph shows the error in nanoseconds (µs) over real-time in seconds. The y-axis represents error, and the x-axis represents real-time.

- **max. sync. error** is represented by red squares.
- **real sync. error** is represented by green circles.

The graph indicates that there is no infinity (extrapolation) in the data presented.

---

Adaptive Internal Clock Synchronization

Zbigniew Jerzak, Robert Fach, and Christof Fetzer
Asymmetric Permanent Omissions

The diagram shows a graph with the x-axis representing real-time in seconds (s) ranging from 4 to 10, and the y-axis representing the maximum sync. error in microseconds (µs) ranging from 0 to 120. The graph compares four hosts: A, B, C, and D. Host A has the highest max. sync. error, followed by host B, host C, and host D, which has the lowest. The error increases linearly with real-time for all hosts.
Asymmetric Permanent Omissions

Adaptive Internal Clock Synchronization

Zbigniew Jerzak, Robert Fach, and Christof Fetzer
Asymmetric Permanent Omissions

![Graph showing real/max sync error and real-time for host C and D](image)
Crash Failures

![Crash Failures Graph]

- host A - max. sync. error
- host B - max. sync. error
- host D - max. sync. error
- host A - real sync. error
- host B - real sync. error
- host D - real sync. error

Error [µs] vs. Real-Time [s]
Need to build critical systems, however:
- underlying components are not synchronous
- need to use COTS to cut cost
- need to use wireless to cut weight

We propose Adaptive Internal Clock Synchronization
- copes with message delays and process failures
- bounds deviation $E_p(T)$ from correct clocks in the system
- propagates the deviation $E_p(T)$ to upper level applications

Applications use the $E_p(T)$ to adapt their behavior
- no more synchronization layer crashes
- the system is safer
Summary

- Need to build critical systems, however:
  - underlying components are not synchronous
  - need to use COTS to cut cost
  - need to use wireless to cut weight
- We propose **Adaptive Internal Clock Synchronization**
  - copes with message delays and process failures
  - bounds deviation $E_p(T)$ from correct clocks in the system
  - propagates the deviation $E_p(T)$ to upper level applications
- Applications use the $E_p(T)$ to adapt their behavior
  - no more synchronization layer crashes
  - the system is safer
Summary

- Need to build critical systems, however:
  - underlying components are not synchronous
  - need to use COTS to cut cost
  - need to use wireless to cut weight
- We propose **Adaptive Internal Clock Synchronization**
  - copes with message delays and process failures
  - bounds deviation $E_p(T)$ from correct clocks in the system
  - propagates the deviation $E_p(T)$ to upper level applications
- Applications use the $E_p(T)$ to adapt their behavior
  - no more synchronization layer crashes
  - the system is safer
Thank You!

http://wwwse.inf.tu-dresden.de/
References

Flaviu Cristian and Christof Fetzer.
Probabilistic internal clock synchronization.

Flaviu Cristian and Christof Fetzer.
The timed asynchronous distributed system model.

Christof Fetzer and Flaviu Cristian.
Building fault-tolerant hardware clocks.

Christof Fetzer and Flaviu Cristian.
A fail-aware datagram service.

Christof Fetzer and Flaviu Cristian.
Fail-awareness: An approach to construct fail-safe systems.

Ute Wappler and Christof Fetzer.
Hardware failure virtualization via software encoded processing.

Jennifer Lundelius Welch and Nancy Lynch.
A new fault-tolerant algorithm for clock synchronization.