Clock Synchronization with Bounded Global and Local Skew

Christoph Lenzen, ETH Zurich
Thomas Locher, ETH Zurich
Roger Wattenhofer, ETH Zurich

October 2008

Distributed **C**omputing **G**roup

49th Annual IEEE Symposium on
Foundations of Computer Science (FOCS)
Philadelphia, PA, USA

Motivation: No Global Clock

- Many tasks in distributed systems require a common notion of time
- Sometimes not all devices can be connected to a "global" clock

⇒ Equip each device with its own clock!

Problem 1: Different clocks have different clock rates

Even worse, these clock rates may vary over time!

⇒ Clock drifts!

Communication is required to synchronize the clocks!

Problem 2: What if the message delays vary?

Each message has a different delay…

How well can distributed clocks be synchronized?

Overview

Model: Clocks

- Each device has a hardware clock H ⇒ $H(t) := \int_0^t h(\tau)\, d\tau$.
- The hardware clock rate h is bounded ⇒ $\forall$ t: $h(t) \in [1-\epsilon, 1+\epsilon]$
- Each device computes a logical clock value L based on:

H(t)

L(t)

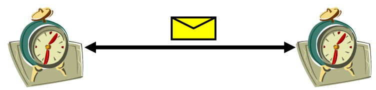
Its hardware clock H and its message history (the messages it received)

- Messages are required to correct clock skews!

Minimize clock skew of logical clocks!

- A **clock synchronization algorithm** specifies how the logical clock value L is adapted!

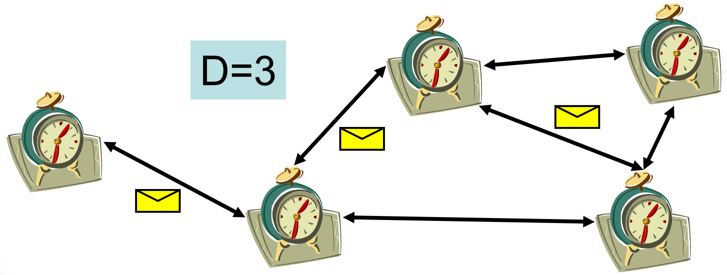
And triggers synch messages!



Model: Graph & Communication

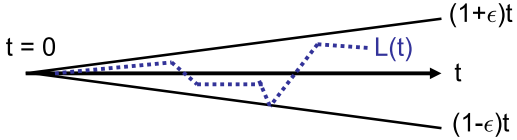
- Distributed system = Graph G of diameter D
  - ➢ Node = Computational device
  - ➢ Edge = Bidirectional communication link
- Nodes communicate via reliable, but delayed messages
  - ➢ Each message may be delayed by any value $\in [0,1]$

Simple normalization!



Model: Optimization Criteria

- Good real time approximation: $\forall$ v $\in$ V, $\forall$ t: $|L_v(t)-t| \leq \epsilon t$



- Minimum progress:

$$\forall v \in V, \forall t_2 > t_1: L_v(t_2)-L_v(t_1) \geq (1-\epsilon)(t_2-t_1)$$

L cannot go back in time!

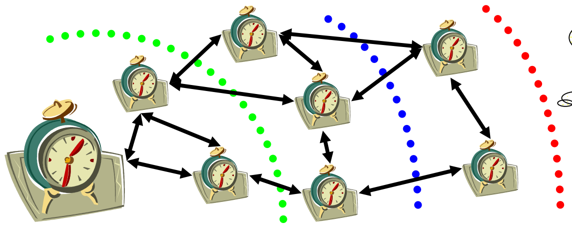
- Minimize the skew among all nodes:

$$\max_{v,w,t} |L_v(t)-L_w(t)|$$

Minimize the **global skew**!

Model: Optimization Criteria II

More importantly: We want a small clock skew between v and w, if the *distance* between v and w is short!



Allow more skew with increasing distance!

Minimize the skew among *neighboring nodes*:

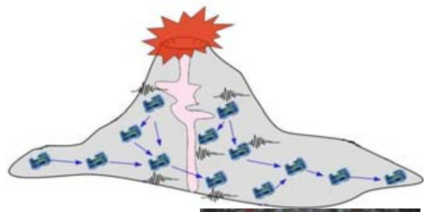
$$\max_{v,w \in N(v),t} |L_v(t)-L_w(t)|$$

Minimize the **local skew**!

N(v) = Neighbors of v

Model: Importance of Local Skew

For many applications, locally well synchronized clocks are more important!



- Monitoring applications

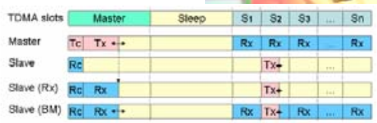
(record <event, timestamp>)

- Tracking applications

Use <event,time> recordings to determine movement/speed etc.

- More fundamental:

E.g., TDMA requires (locally) synchronized clocks!



Model: Old Results

A well-known result is that the skew between two nodes at distance d is **Ω(d)** in the worst case!
→ Ω(D) lower bound on global skew!

Guaranteeing a global skew of Θ(D) is easy…

„Always set L to largest clock value!“

Bounding the local skew is hard(er):

Many (reasonble) algorithms → O(D)

Best known bound → O(√D)

Lower bound → Ω(log D / log log D)

Diameter determines the local skew!!!

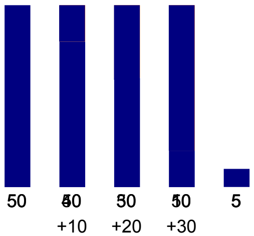
True bound probably Ω(log D)...

Overview

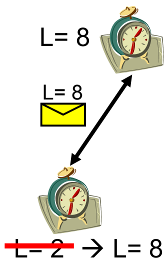
Algorithm: Simple Strategies

Strategy I: „Always set L to largest clock value!“

Problem:



O(D) local skew!



Strategy II: „Take the average clock value!“

Problem:

O(D²) global skew! (→ O(D) local skew...)

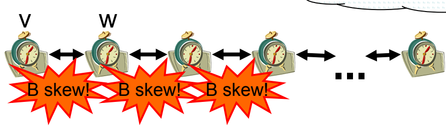
Algorithm: Better Strategies

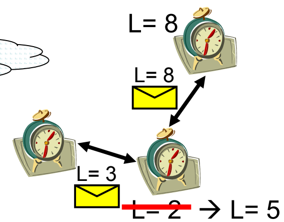
B = 2

Strategy III: „Always increase the clock value L UNLESS a neighbor is B behind.“

Problem:

Tolerate B skew!





Length of this chain → O(D/B)

v can built up skew to w at rate O(ϵ) for O(D/B) time → O(ϵ·D/B) = O(D) skew!!!

How can we fix this?!?

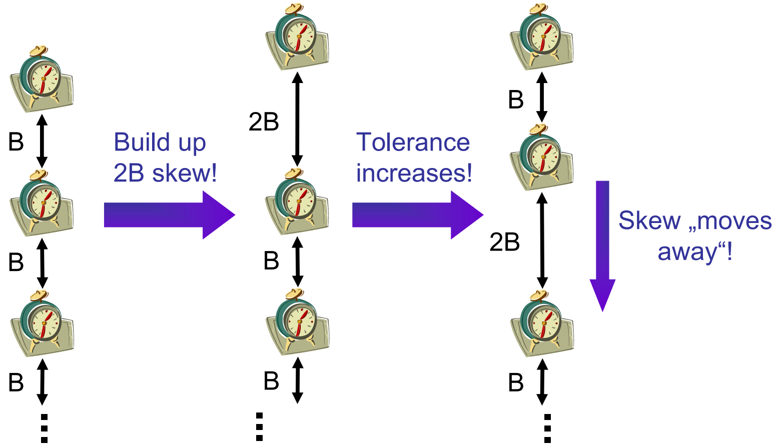
→ Choose B ∈ O(√D) → O(√D) local skew!!!

Ok, but can we do better?

Algorithm: Increase Tolerance

For any $i \in \{2,3,\ldots\}$

Strategy III+: „Tolerate B skew, but if v experiences a skew of i·B → Tolerate i·B skew!"



Algorithm: Intuition

If the adversary wants to build up 3B skew → A chain with 2B skew between neighbors is needed!

→The longer the better!

+B skew at rate $O(\epsilon)$ between l nodes!

→ Only O(D/B) time to build chain!

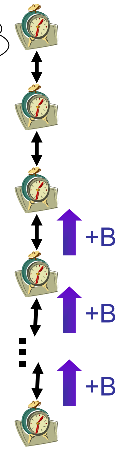
If l is the length of the chain → $\Omega(B \cdot l/\epsilon)$ time is needed

→ $\Omega(B \cdot l/\epsilon) \in O(D/B)$ → $l \in O(\epsilon \cdot D/B^2) \in O(D/B^2)$

Lose a factor of B!

Inductively:

A skew of (i+1)·B requires a chain with i·B skew between nodes → $l_i \in O(D/B^i)$



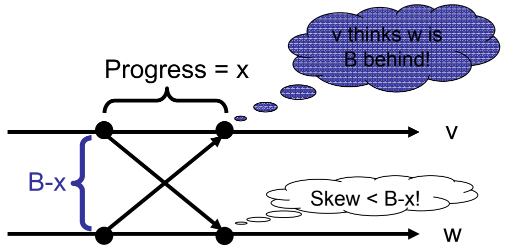
→ Local Skew $\in O(B \cdot \log_B D)$!

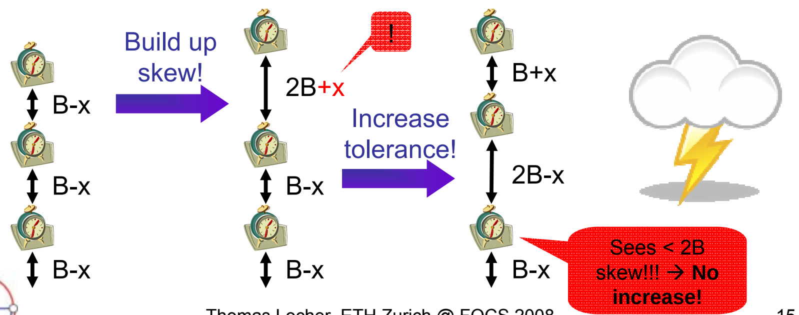
Algorithm: Why It Fails

That's it?

Unfortunately, no.

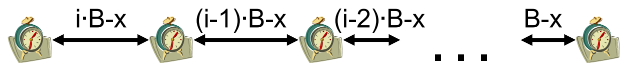
The message delays cause problems:





Algorithm: How bad is it? How can we fix it?

We get the following picture:



Local skew → $O(\sqrt{D})$

Since global skew $\in$ O(D)!

How can we fix this?!?

→ React earlier! If a neighbor w is > i·B-x behind, ask w to increase its clock value!!!

If iB-x+r behind, increase by r!

That's it?

Fortunately, yes.

Overview

I. Motivation

II. Model

III. Algorithms

IV. Conclusion

Conclusion: Results

- Local skew → O(log D)
  - $|L_v - L_w| \in O(d(v,w) \cdot \log(D/d(v,w)))$

Probably asymptotically optimal!

- Global skew → O(D)
  - $|L_v - L_w| \le (1+O(\epsilon))D$

In fact, only a factor ≈2 larger than the lower bound!

- Bit complexity → $O(\Delta \log^2 D)$

Δ = Maximum node degree

- Space complexity → $O(\Delta \log \log D + \log^2 D)$

Conclusion: Outlook

Open problems?

Clocks should „behave normally" even when correcting clock skew!

- Bound the logical clock rate!
  - Ideally: $l(t) \in [1-O(\epsilon), 1+O(\epsilon)]$

Clock skew is built up at a low rate!

- Reduce the bit complexity!
  - Send less bits per message
  - Reduce the message frequency
  - Enable piggybacking!
- Prove tight bounds for global/local skew!

Questions and Comments?

Thank you for your attention!

Thomas Locher
Distributed Computing Group
ETH Zurich, Switzerland
lochert@tik.ee.ethz.ch
http://dcg.ethz.ch/members/thomasl.html