Oblivious Gradient Clock Synchronization

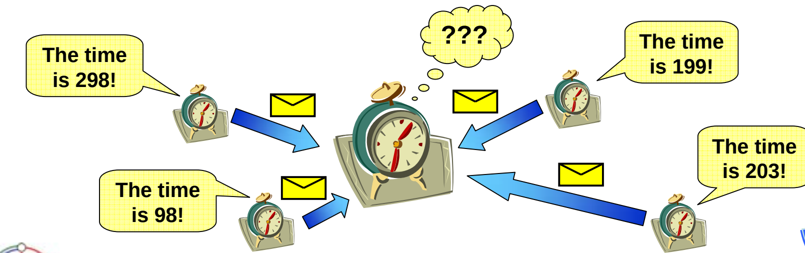
Thomas Locher, ETH Zurich
Roger Wattenhofer, ETH Zurich

Distributed Computing Group

20th IEEE Int. Conference on Distributed Computing (DISC)
Stockholm, Sweden, September 2006

Motivation: Clock Synchronization

- Clock synchronization is a classic, important problem!
  - Many results have been published about different subtopics (skew minimization, communication cost, fault-tolerance…).
- More and more distributed applications are appearing!
  - Distributed systems become even more popular (Internet, wireless networks…).
  - These applications often require *synchronized clocks*!



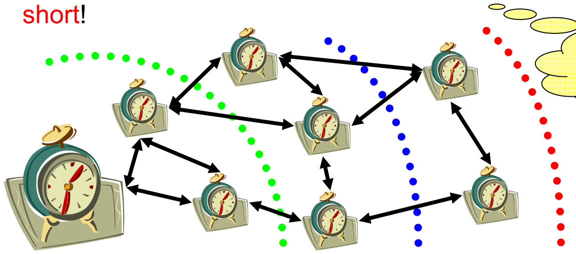
Motivation: Gradient Property

- We focus on the minimization of the *clock skew*!
- We would like the clock skew to be small between *any two nodes*, even if they are *not directly connected*!

This is called the global property!

- More importantly, we want the skew between two nodes to be small, if the *length of the shortest paths* between those nodes is short!

This is called the gradient property!



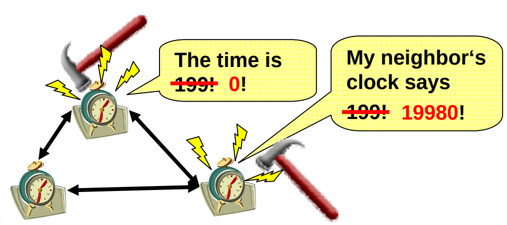
Motivation: Obliviousness

- How can the *local clock value* be computed?
  - Store message arrival times and their time stamps!
  - Use old messages to *estimate* the current clock value of the neighboring nodes and/or the message delays…

How much and what has to be stored?
What if nodes do not have much memory?

Really? How?

- *Oblivious model*: Each node can store only one clock value for each neighboring node!



Advantage: Oblivious algorithms can easily be transformed into *self-stabilizing* algorithms!

Motivation: Goal

- We study the effect of obliviousness on clock synchronization!
- The goal is to get insights into the difficulty of gradient clock synchronization!
  - What level of synchronization can be achieved without storing a larger history?
- Find good gradient clock synchronization algorithms in this restricted model!
  - Such an algorithm might facilitate the development of a general gradient clock synchronization algorithms guaranteeing even better bounds on the skew!

Outline

I. Motivation

II. Model / Results

III. Synchronization Algorithms

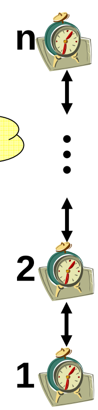
IV. Conclusion / Outlook

Model

- The graph used in all examples is the linear list!
- Messages have variable delays in the range [0,1].
- Distance d(i,j) is defined as the length of the shortest path between node i and j.

In the list: d(i,j) = |j-i|+1

- Each node i has a hardware clock $H_i(\cdot)$.
- The hardware clock rate of node i at time t is $h_i(t) \in [\mathcal{L},\mathcal{U}]$, where $0 < \mathcal{L} < \mathcal{U}$ and $\mathcal{U} \geq 1$.
- The time of node i at time t is $H_i(t) = \int_0^t h_i(\tau)\, d\tau$!
- Each node i further has a logical clock $L_i(\cdot)$!



$H_i(t)$

$L_i(t)$

Goal: Minimize the skew between the logical clocks!

Model

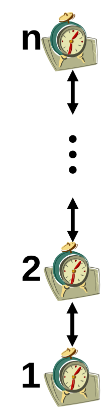
- $L_i(\cdot)$ increases at the rate of $H_i(\cdot)$!
- A message received with a fresh clock value from a neighboring node → Update $L_i(\cdot)$ according to the algorithm in use!
- If $L_i(\cdot)$ changed → Inform all neighboring nodes about the new clock value!

$L_i(\cdot)$ cannot run backwards!

- An algorithm $\mathcal{A}$: L x ψ → L specifies how node i adapts its logical clock at time t, given its current value $L_i(t)$ and the stored message history ψ!

Oblivious! The most accurate clock value from each neighbor!

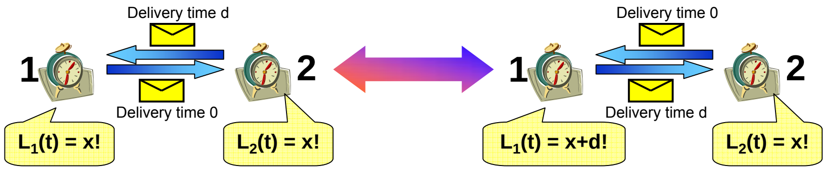
- An execution is $\mathcal{E} = (\mathcal{M},\mathcal{R})$, where $\mathcal{M}(t,i,j) \in [0,1]$ specifies how long it takes for a message from node i sent at time t to arrive at node j, and $h_i(t) = \mathcal{R}(t,i)$!



Results

A well-known result is that the skew between two nodes at distance d is at least **Ω(d)**!

Proof Sketch: The following scenarios cannot be distinguished!



There is a clock synchronization algorithm with a worst-case skew of **Θ(d)** between any two nodes at distance d!

The only result on *gradient clock synchronization* [Fan, Lynch @ PODC 2004] is that nodes at distance 1 cannot be synchronized better than **Ω(log D / log log D)** where D denotes the *diameter* of G!

Results [Fan, Lynch @ PODC 2004]

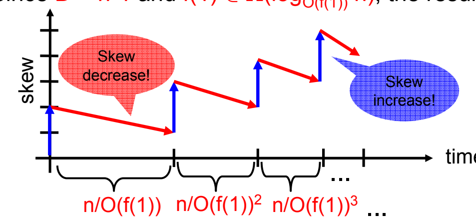
Proof Sketch for the **Ω(log D / log log D)** lower bound:

The skew between all neighbors among k nodes can be increased by O(1) in O(k) time, but the skew can only be decreased by O(f(1)) in O(1) time!

f(1) = Worst-case skew allowed between neighbors!

Recursive skew induction: Induce a skew of c_1 in O(n) time. Let the algorithm run again for n/O(f(1)) time → Skew decreases by $c_2 < c_1$! Increase the skew between O(n/f(1)) nodes during this time again by c_1! Repeat this for $\log_{O(f(1))} n$ steps!

→ Since D = n-1 and $f(1) \in \Omega(\log_{O(f(1))} n)$, the result follows!



No algorithm with f(1) = o(D) published!

Results

Our results:

- We show that for several intuitive algorithms the worst-case skew between two neighboring nodes is Θ(D)!

Not easy to find a good gradient clock synchronization algorithm!

- We present an algorithm with a worst-case skew of $O(d + \sqrt{D})$ between any two nodes at distance d in any graph!

First algorithm with a worst-case bound of o(D) between nodes at distance 1!

Outline

I. Motivation

II. Model / Results

III. Synchronization Algorithms

IV. Conclusion / Outlook

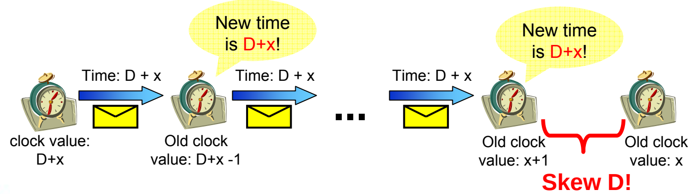
Synchronization Algorithms: $\mathcal{A}^{max}$

A simple algorithm: Always set the clock to the maximum clock value received from any neighbor (if > own clock value)!

Intuition: Nodes have to keep up with the fastest node anyway! Synchronize to its clock as closely as possible!

This is a poor gradient clock synchronization algorithm!!!

A skew of 1 between all n (= D-1) neighbors cannot be avoided
→ Fast propagation of the largest clock value incurs a large skew between two neighboring nodes!



Synchronization Algorithms: $(\mathcal{A}^{max})'$

The problem of $\mathcal{A}^{max}$ is that the clock is always increased to the maximum value!

→ Idea: Allow a slack between the maximum clock value and the own value!
The algorithm $(\mathcal{A}^{max})'$ sets the clock value to

$$L_i(t) := \max(L_i(t), \max_{j \in \mathcal{N}_i} L_j(t) - \gamma)$$

The constant slack!

The worst-case clock skew between two neighboring nodes is still $\Theta(D)$ independent of the choice of γ!!!

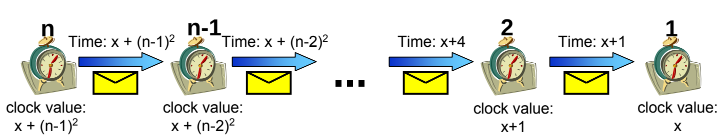
Synchronization Algorithms: $\mathcal{A}^{avg}$

Only considering the largest clock value is a bad idea!
→ Idea: Take the clocks of all neighboring nodes into account and choose the *average clock value*!

Surprisingly, this algorithm $\mathcal{A}^{avg}$ is even worse!!!

- Assume that the message delay is always 1.
- Assume that the clock rate of node n is always 1 and the clock rates of all other nodes are arbitrary values less than 1.

"After a while", the skew between node n and n-1 is $2n-3 \in \Theta(n)$!

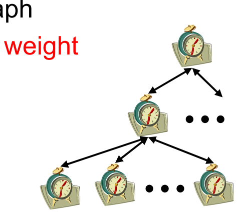


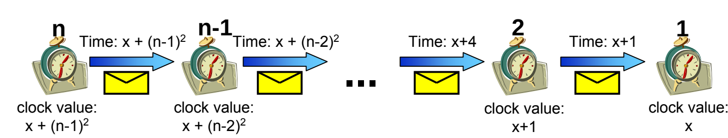
Synchronization Algorithms: $\mathcal{A}^{avg}$

Bad global property: The skew between node n and 1 is $(n-1)^2 \in \Theta(n^2) \in \Theta(D^2)$!

$\mathcal{A}^{max}$ guarantees a bound of $\Theta(D)$ between any two nodes!

Note: Nodes have at most 2 neighbors in this graph
→ The maximum clock value must have a larger weight than ½!
→ This also holds for other graphs: In a k-ary tree, if the k children have a weight larger or equal to ½, the worst-case skew is also $\Theta(D^2)$ between the root and the leaf nodes!





Synchronization Algorithms: $\mathcal{A}^{\text{bound}}$

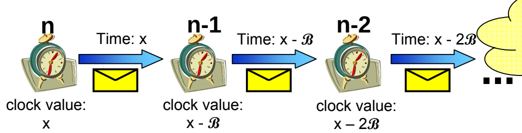
Minimizing the skew to the fastest neighbor or all neighbors does not work…

→ Idea: Minimize the skew to the slowest node! Give the slowest node time to „catch up!"

All nodes wait for each other!

Algorithm $\mathcal{A}^{\text{bound}}$ does not increase its logical clock due to a message if any neighboring node's clock is $\mathcal{B}$ behind!

Problem with approach: Chain of dependency!



Node n-1 has to wait for node n-2, node n-2 has to wait for node n-3...

Chain of length $\Theta(n) = \Theta(D)$ results in $\Theta(D)$ waiting time

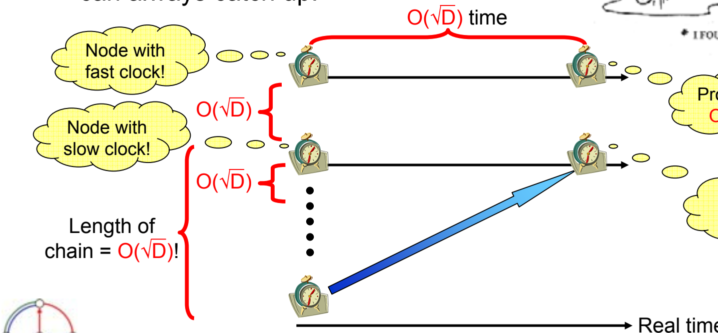
→ $\Theta(D)$ skew!

Synchronization Algorithms: Idea!

Waiting for slower nodes is not such a bad idea…

Do it smarter: Set $\mathcal{B} = O(\sqrt{D})$ → Skew is allowed to be $O(\sqrt{D})$! But the waiting time is at most $O(D/\mathcal{B}) = O(\sqrt{D})$ as well!

Set the constants right and slow nodes can always catch up!



Synchronization Algorithms: $\mathcal{A}^{\text{root}}$

Algorithm $\mathcal{A}^{\text{root}}$ works as follows:

When a message is received, execute the following steps:

max := Maximum clock value of all neighboring nodes

min := Minimum clock value of all neighboring nodes

if(max > own clock and $min + \mathcal{U}\sqrt{D}+1$ > own clock →

own clock := min(max, $min + \mathcal{U}\sqrt{D}+1$)

inform all neighboring nodes about new clock value!

end if

Reminder: $\mathcal{U}$ is the maximum hardware clock rate!

Synchronization Algorithms: $\mathcal{A}^{\text{root}}$

Properties of $\mathcal{A}^{\text{root}}$

- Global Property:

The logical clock skew between *any* two nodes is at most $\mathcal{U}D + 1$.

$\Theta(D)$ is asymptotically *optimal*!!! This fact is required to prove the **gradient property** of $\mathcal{A}^{\text{root}}$!

- Gradient Property:

The logical clock skew between any two neighboring nodes is at most $2\mathcal{U}\sqrt{D}+1$.

$O(\sqrt{D})$ is the best known bound so far!

Outline

I. Motivation

II. Model / Results

III. Synchronization Algorithms

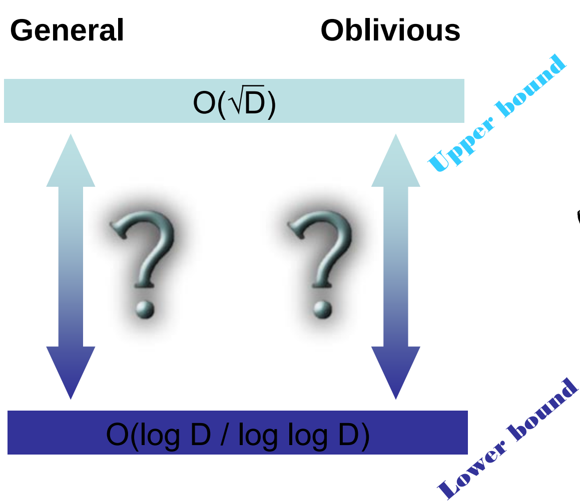
IV. Conclusion / Outlook

Conclusion

- General results:
  - Dilemma: Focusing on the maximum clock value does not work. However, this value must have a large weight!
  - Considering all clocks to be equally important does not work!

- Algorithmic result:
  - Algorithm with a worst-case skew of $O(d+\sqrt{D})$!

Outlook



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