Time

Michael George

November 10, 2005

The Problem

"All we do here is invent games to pass the time."
— John O'Donohue

The Problem

> *"All we do here is invent games to pass the time."*
> — *John O'Donohue*

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.

Why The Problem Is Interesting

Interesting for two reasons:

1. Good setting to examine general difficulties in distributed systems:
    - Fault tolerance
    - Consistent view of changing data
    - Trust
    - Interplay between strength of guarantees and practicality

Why The Problem Is Interesting

Interesting for two reasons:

1. Good setting to examine general difficulties in distributed systems:
    - Fault tolerance
    - Consistent view of changing data
    - Trust
    - Interplay between strength of guarantees and practicality
2. Useful primitive for distributed systems
    - Distributed checkpointing / stable property detection
    - Can be used to implement general state-machine algorithms reliably [Lamport 74]

Overview

We will discuss two papers that solve this problem:

1. Optimal Clock Synchronization [Srikanth and Toueg '87]
    - Assume reliable network
    - Provide logical clock with optimal agreement
    - Also optimal with respect to failures

Overview

We will discuss two papers that solve this problem:

1. Optimal Clock Synchronization [Srikanth and Toueg '87]
    - Assume reliable network
    - Provide logical clock with optimal agreement
    - Also optimal with respect to failures
2. Probabilistic Internal Clock Synchronization [Cristian and Fetzer '03]
    - Drop requirements on network
    - Provide very efficient logical clock
    - Only provide probabilistic guarantees

Some Assumptions

We assume. . .

- Clock drift is bounded:

$$\frac{1}{1+\rho}(t_2 - t_1) \leq R_i(t_2) - R_i(t_1) \leq (1+\rho)(t_2 - t_1)$$

Some Assumptions

We assume. . .

- Clock drift is bounded:

$$\frac{1}{1+\rho}(t_2 - t_1) \leq R_i(t_2) - R_i(t_1) \leq (1+\rho)(t_2 - t_1)$$

- Communication and processing are reliable:

$$t_{recv} - t_{send} \leq t_{del}$$

Some Assumptions

We assume. . .

- Clock drift is bounded:

$$\frac{1}{1+\rho}(t_2 - t_1) \leq R_i(t_2) - R_i(t_1) \leq (1+\rho)(t_2 - t_1)$$

- Communication and processing are reliable:

$$t_{recv} - t_{send} \leq t_{del}$$

- Authenticated messages (we will relax this later).

Our Goals

We want algorithms that satisfy the following:

- Agreement between clocks:

$$|C_i^k(t) - C_j^k(t)| \leq D_{max}$$

Our Goals

We want algorithms that satisfy the following:

- Agreement between clocks:

$$|C_i^k(t) - C_j^k(t)| \leq D_{max}$$

- Accuracy of clocks:

$$\frac{1}{1+\gamma}t + a \leq C_i^k(t) \leq (1+\gamma)t + b$$

Our Goals

We want algorithms that satisfy the following:

- Agreement between clocks:

$$|C_i^k(t) - C_j^k(t)| \leq D_{max}$$

- Accuracy of clocks:

$$\frac{1}{1+\gamma}t + a \leq C_i^k(t) \leq (1+\gamma)t + b$$

- Optimal accuracy (proved later):

$$\gamma = \rho$$

The Bad News. . .

Up to f processes can fail in the following ways:

- Clock too slow or fast

The Bad News. . .

Up to f processes can fail in the following ways:

- Clock too slow or fast
- Stuck clock bits

The Bad News. . .

Up to f processes can fail in the following ways:

- Clock too slow or fast
- Stuck clock bits
- Crash, lost connectivity, buggy code

The Bad News. . .

Up to f processes can fail in the following ways:

- Clock too slow or fast
- Stuck clock bits
- Crash, lost connectivity, buggy code
- Byzantine failure

The Bad News. . .

Up to f processes can fail in the following ways:

- Clock too slow or fast
- Stuck clock bits
- Crash, lost connectivity, buggy code
- Byzantine failure

Definitions: A *correct* process follows the protocol and has a working hardware clock. A non-correct process is *faulty*.

The Basic Algorithm

We proceed in rounds. On round k, process i will:

1. Wait for P units according to clock C_i^{k-1}

The Basic Algorithm

We proceed in rounds. On round k, process i will:

1. Wait for P units according to clock C_i^{k-1}
2. Broadcast “I’m ready to start round k”

The Basic Algorithm

We proceed in rounds. On round k, process i will:

1. Wait for P units according to clock C_i^{k-1}
2. Broadcast "I'm ready to start round k"
3. After receiving $f+1$ messages:
    - set C_i^k to $kP+\alpha$
    - rebroadcast the $f+1$ messages

The Basic Algorithm

We proceed in rounds. On round *k*, process *i* will:

1. Wait for *P* units according to clock C_i^{k-1}
2. Broadcast "I'm ready to start round *k*"
3. After receiving $f + 1$ messages:
    - set C_i^k to $kP + \alpha$
    - rebroadcast the $f + 1$ messages

Definitions:

- $ready^k$ is the real time of the first "I'm ready" message

The Basic Algorithm

We proceed in rounds. On round k, process i will:

1. Wait for P units according to clock C_i^{k-1}
2. Broadcast "I'm ready to start round k"
3. After receiving $f+1$ messages:
    - set C_i^k to $kP+\alpha$
    - rebroadcast the $f+1$ messages

Definitions:

- $ready^k$ is the real time of the first "I'm ready" message
- beg^k is the real time of first process to set clock C_i^k

The Basic Algorithm

We proceed in rounds. On round k, process i will:

1. Wait for P units according to clock C_i^{k-1}
2. Broadcast "I'm ready to start round k"
3. After receiving $f+1$ messages:
    - set C_i^k to $kP+\alpha$
    - rebroadcast the $f+1$ messages

Definitions:

- $ready^k$ is the real time of the first "I'm ready" message
- beg^k is the real time of first process to set clock C_i^k
- end^k is the last

The Basic Algorithm

We proceed in rounds. On round k, process i will:

1. Wait for P units according to clock C_i^{k-1}
2. Broadcast "I'm ready to start round k"
3. After receiving $f + 1$ messages:
    - set C_i^k to $kP + \alpha$
    - rebroadcast the $f + 1$ messages

Definitions:

- *ready*^{k} is the real time of the first "I'm ready" message
- *beg*^{k} is the real time of first process to set clock C_i^k
- *end*^{k} is the last
- The kth resynch period is the interval $[beg^k, end^k]$

Outline of Proof of Agreement

Sketch of Agreement:

- Proof is by induction on round number k.
- Show that if kth clocks agree then $(k+1)$st clocks also agree
- Uses bounds on sizes of intervals between rounds and within rounds.

Outline of Proof of Accuracy

We prove the two defining inequalities for accuracy separately:

- By considering the fastest possible clock and showing it forms an upper bound on any logical clock value, we can show

$$C_i^k(t) \leq \frac{P}{P-\alpha}(1+\rho)t + b$$

- Similarly, considering slowest possible clock yields

$$\frac{P}{P-\alpha+[t_{del}/(1+\rho)]}(1+\rho)^{-1}t + a \leq C_i^k(t)$$

- Putting these together we get Accuracy, which in turn gives correctness.

How close can we get?

What’s the best possible γ?

How close can we get?

What's the best possible γ?

- In run 1, let all clocks run as fast as possible:

$$C_i(t) \leq (1+\gamma)t + b$$

- In run 2, let all clocks run as slow as possible:

$$\frac{1}{1+\gamma}t + a \leq C_i(t)$$

How close can we get?

What's the best possible γ?

- In run 1, let all clocks run as fast as possible:

$$C_i(t) \leq (1+\gamma)t + b$$

- In run 2, let all clocks run as slow as possible:

$$\frac{1}{1+\gamma}t + a \leq C_i(t)$$

- Run 1 at time t looks the same as run 2 at time $(1+\rho)^2 t$, so

$$(1+\gamma)t + b \geq \frac{(1+\rho)^2}{1+\gamma}t + a$$

How close can we get?

What's the best possible γ?

- In run 1, let all clocks run as fast as possible:

$$C_i(t) \leq (1+\gamma)t + b$$

- In run 2, let all clocks run as slow as possible:

$$\frac{1}{1+\gamma}t + a \leq C_i(t)$$

- Run 1 at time t looks the same as run 2 at time $(1+\rho)^2 t$, so

$$(1+\gamma)t + b \geq \frac{(1+\rho)^2}{1+\gamma}t + a$$

- Taking $t \to \infty$ we see $\gamma \geq \rho$.

An Optimal Algorithm Drift-wise

Key insight:

- There's an interval of uncertainty in difference between arrival time:
    - it could be $P - \alpha$ if clock is fast
    - it could be $P - \alpha + t_{del}(1 + \rho)$ if clock is slow

An Optimal Algorithm Drift-wise

Key insight:

- There's an interval of uncertainty in difference between arrival time:
  - it could be $P - \alpha$ if clock is fast
  - it could be $P - \alpha + t_{del}(1 + \rho)$ if clock is slow
- Algorithm 1 chooses left endpoint of the interval

An Optimal Algorithm Drift-wise

Key insight:

- There's an interval of uncertainty in difference between arrival time:
    - it could be $P - \alpha$ if clock is fast
    - it could be $P - \alpha + t_{del}(1 + \rho)$ if clock is slow
- Algorithm 1 chooses left endpoint of the interval
- Let's choose midpoint instead

An Optimal Algorithm Drift-wise

Key insight:

- There's an interval of uncertainty in difference between arrival time:
  - it could be $P - \alpha$ if clock is fast
  - it could be $P - \alpha + t_{del}(1 + \rho)$ if clock is slow
- Algorithm 1 chooses left endpoint of the interval
- Let's choose midpoint instead

Proof of correctness goes through mostly unmodified, but drift rate is optimal.

Algorithm is Also Optimal Fail-wise

If an algorithm is correct, then $2f < n$.

- Easy proof - use the algorithm we have.

Algorithm is Also Optimal Fail-wise

If an algorithm is correct, then $2f < n$.

- Easy proof - use the algorithm we have.
- Authors give a different proof

Algorithm is Also Optimal Fail-wise

If an algorithm is correct, then $2f < n$.

- Easy proof - use the algorithm we have.
- Authors give a different proof

Thus this algorithm is optimal with respect to fault tolerance.

Extensions to the Basic Algorithm

We can remove some of the limitations from the basic algorithm:

- Strong authentication is too heavyweight. Only need:
    - Correctness
    - Unforgeability
    - Relay

  Can use a broadcast primitive from the literature.

Extensions to the Basic Algorithm

We can remove some of the limitations from the basic algorithm:

- Strong authentication is too heavyweight. Only need:
  - Correctness
  - Unforgeability
  - Relay

  Can use a broadcast primitive from the literature.
- Can slightly modify algorithm for related tasks
  - Initialization
  - Integration

Extensions to the Basic Algorithm

We can remove some of the limitations from the basic algorithm:

- Strong authentication is too heavyweight. Only need:
  - Correctness
  - Unforgeability
  - Relay

  Can use a broadcast primitive from the literature.
- Can slightly modify algorithm for related tasks
  - Initialization
  - Integration
- Can merge new clocks into a single continuous clock

Motivation for Probabilistic Synchronization

The Optimal scheme has some problems:

- Relies on guaranteed timely delivery (may not be an option)
- Performance depends on t_{del}, which can be large
- Bursty $O(n^2)$ messaging

Can we do without these limitations?

Probabilistic System Model

The system model for the second paper is similar...

- Correct clocks still have bounded drift
    - although assume $\rho^2 \ll \rho$

Probabilistic System Model

The system model for the second paper is similar...

- Correct clocks still have bounded drift
    - although assume $\rho^2 \ll \rho$
- No longer a maximum communication delay
    - delays given by probability distribution
    - this prevents us from stating results in terms of t_{max}.

Probabilistic System Model

The system model for the second paper is similar...

- Correct clocks still have bounded drift
  - although assume $\rho^2 \ll \rho$
- No longer a maximum communication delay
  - delays given by probability distribution
  - this prevents us from stating results in terms of t_{max}.
- There is a known minimum message delay t_{min}

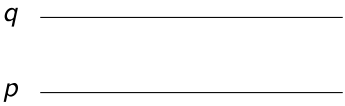
Failure Models

We distinguish between:

- Crash failure — process stops completely
- Performance failure — process runs too slow
- Read failure — process fails to read remote clock in time
- Arbitrary failure — anything else

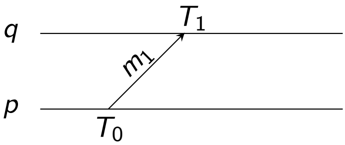
Probabilistic Remote Clock Reading

How does process p read process q's clock?



Probabilistic Remote Clock Reading

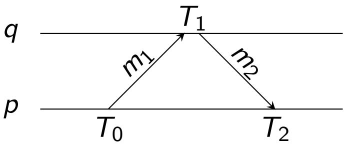
How does process p read process q's clock?



1. p sends a request m_1 with timestamp T_0 to q

Probabilistic Remote Clock Reading

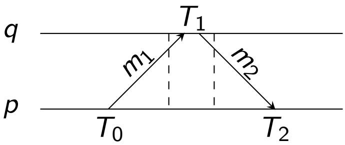
How does process p read process q's clock?



1. p sends a request m_1 with timestamp T_0 to q
2. q sends a response m_2 with timestamp T_1 to p

Probabilistic Remote Clock Reading

How does process p read process q's clock?



1. p sends a request m_1 with timestamp T_0 to q
2. q sends a response m_2 with timestamp T_1 to p
3. p can infer that T_1 is in a certain interval.

Properties

There are a number of properties that this protocol satisfies:

- Timeliness

Properties

There are a number of properties that this protocol satisfies:

- Timeliness
- Error Bound

Properties

There are a number of properties that this protocol satisfies:

- Timeliness
- Error Bound
- Crash Handling

Properties

There are a number of properties that this protocol satisfies:

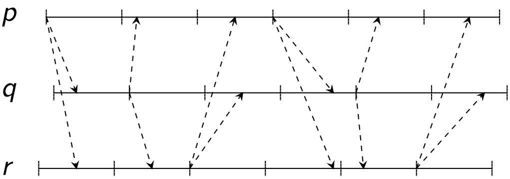
- Timeliness
- Error Bound
- Crash Handling
- Likely Success

Note that these are also satisfied by deterministic clock reading

The High Level Algorithm

The synchronization algorithm is organized as follows:

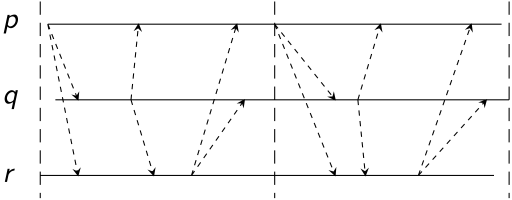
- A *slot* is a unit in which a single process gets to send



The High Level Algorithm

The synchronization algorithm is organized as follows:

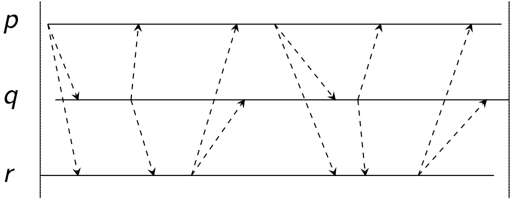
- A *slot* is a unit in which a single process gets to send
- A *cycle* is a unit in which all processes get a chance to send



The High Level Algorithm

The synchronization algorithm is organized as follows:

- A *slot* is a unit in which a single process gets to send
- A *cycle* is a unit in which all processes get a chance to send
- A *round* is a unit in which all processes must get estimates of other clocks



The Contents of Each Exchange

Each message from p to q in the above protocol contains:

- p's send timestamp
- p's best approximation of every clock
- The corresponding error bounds
- p's receive timestamp for each message from q

The Contents of Each Exchange

Each message from p to q in the above protocol contains:

- p's send timestamp
- p's best approximation of every clock
- The corresponding error bounds
- p's receive timestamp for each message from q

This data allows q to approximate p's clock as above, for up to k^2 message pairs.

The Contents of Each Exchange

Each message from p to q in the above protocol contains:

- p's send timestamp
- p's best approximation of every clock
- The corresponding error bounds
- p's receive timestamp for each message from q

This data allows q to approximate p's clock as above, for up to k^2 message pairs.

If q trusts p can also use it to approximate other clocks.

The Protocol

In each round, a process passes through the following *modes*:

1. It starts in *request* mode

The Protocol

In each round, a process passes through the following *modes*:

1. It starts in *request* mode
2. It moves to *reply mode* when it has all clocks

The Protocol

In each round, a process passes through the following *modes*:

1. It starts in *request* mode
2. It moves to *reply mode* when it has all clocks
3. Finally moves to *finish mode* when everyone has its clock

The Protocol

In each round, a process passes through the following *modes*:

1. It starts in *request* mode
2. It moves to *reply mode* when it has all clocks
3. Finally moves to *finish mode* when everyone has its clock

After *k*th cycle, it automatically returns to request mode for next round.

The Protocol

In each round, a process passes through the following *modes*:

1. It starts in *request* mode
2. It moves to *reply mode* when it has all clocks
3. Finally moves to *finish mode* when everyone has its clock

After kth cycle, it automatically returns to request mode for next round.

Total message complexity is kN in the worst case, $N+1$ in the best.

From Approximations to Shared Time

Thus far *p* has a separate approximation of everyone's clock, with error bounds.

We plug the data into a *midpoint convergence function*, which:

- Combines the estimates of the clocks to yield a single value
- Is responsible for detecting and correcting errors
- Is therefore fault-model specific

From Approximations to Shared Time

Thus far p has a separate approximation of everyone's clock, with error bounds.

We plug the data into a *midpoint convergence function*, which:

- Combines the estimates of the clocks to yield a single value
- Is responsible for detecting and correcting errors
- Is therefore fault-model specific

The authors provide four algorithms:

- Crash-fail (requires $n \geq f + 1$)
- Read-fail (requires $n \geq 2f + 1$)
- Arbitrary-fail (requires $n \geq 3f + 1$)
- Hybrid-fail (requires $n \geq 3f_A + 2f_R + f_C + 1$)

Discussion

Some thoughts for discussion:

- “Optimal” isn’t always optimal

Discussion

Some thoughts for discussion:

- “Optimal” isn’t always optimal
- Good demonstration of the end-to-end principle

Discussion

Some thoughts for discussion:

- "Optimal" isn't always optimal
- Good demonstration of the end-to-end principle
- It would be nice to see some data

Discussion

Some thoughts for discussion:

- “Optimal” isn’t always optimal
- Good demonstration of the end-to-end principle
- It would be nice to see some data