Synchronization


Hongfei Yan
School of EECS, Peking University
3/10/2008

Outline

- Introduction
- Physical Clocks
  - Time
  - Synchronization Algorithms
    - Cristian's method
    - The Berkeley Algorithm
    - The Network Time Protocol
- Logical Clocks
  - Happened-before relation
  - Synchronization Algorithms
    - Lamport timestamps
    - Vector timestamps
- Global State
  - Chandy and Lamport Algorithm

Why is it important?

- Need to measure accurately
  - E.g., auditing in e-commerce
- Algorithms depending on
  - E.g., consistency,
    - make
- The satisfaction of global system invariants.
  - E.g., Absence of deadlocks
  - Write access to a distributed database never granted
to more than one process
  - Objects are only subject to garbage collection when no
further reference to them exists

E.g., UNIX make program

When each machine has its own clock, an event
that occurred after another event may
nevertheless be assigned an earlier time.

Outline

- Introduction
- Physical Clocks
  - Time
  - Synchronization Algorithms
    - Cristian's method
    - The Berkeley Algorithm
    - The Network Time Protocol
- Logical Clocks
- Global State
Physical Clocks

- A high-frequency oscillator, counter, and holding register
  - Counter decrements on each oscillation, generates a "tick" and is reset from the holding register when it goes to 0.
- On a network, clocks will differ: "skew".
- Two problems:
  - How do we synchronize multiple clocks to the "real" time?
  - How do we synchronize with each other?

Measuring Time

- How long is a second?
- How long is a day?
- One solar day is 86,400 solar seconds.
  - Some variation in the rotational speed of the earth: mean solar second.

A Solar Day

- Computation of the mean solar day.

Changes

- Big problem: the earth's rotation is slowing.
- Seconds are getting longer.
- Is this acceptable for science and engineering?
  - Use physical clocks not tied to the Earth: TAI.
- Hm...but then there are more seconds in a day!
  - What are leap years for?
  - Leap seconds are periodically added.

Leap Seconds

- TAI seconds are of constant length, unlike solar seconds. Leap seconds are introduced when necessary to keep in phase with the sun.
- UTC is TAI with leap seconds.
  - Suppose we want to know how much time elapsed between two events. Which do we use?

Time Standards

- International Atomic Time
  - TAI is a physical time standard that defines the second as the duration of 9,192,631,770 periods of the radiation of a specified transition of the cesium atom 133. TAI is a chronoscopic timescale, i.e., a timescale without any discontinuities. It defines the epoch, the origin of time measurement, as January 1, 1958 at 00:00:00 hours, and continuously increases the counter as time progresses

- Universal Time Coordinated
  - UTC is an astronomical time standard that is the basis for the time on the "wall clock". In 1972 it was internationally agreed that the duration of the second should conform to the TAI stand, but that the number of seconds in an hour will have to be occasionally modified by inserting a leap second into UTC to maintain synchrony between the wall clock time and the astronomical phenomena, like day and night.
Notation and Terms

Assume a timer that causes an interrupt \( H \) times a second.
- When it fires, adds 1 to a software clock, \( C \).
- Let \( C_p(t) \) be the value of the clock on machine \( p \) when the UTC time is \( t \).
- \( dC/dt \) ideally should be 1
- Clocks will drift.
  - If we can bound the drift, then we have the maximum drift rate.
  - \( 1 - \rho \leq dC/dt \leq 1 + \rho \)

Clock Drift

The relation between clock time and UTC when clocks tick at different rates.

Outline

- Introduction
- Physical Clocks
  - Physical Clocks
  - Synchronization Algorithms
    - Cristian’s method
    - The Berkeley Algorithm
    - The Network Time Protocol
- Logical Clocks
- Global State

Clock Synchronization Algorithms

- Cristian’s Algorithm
  - The time server is passive
- The Berkeley Algorithm
  - The time server is active
- The Network Time Protocol

Cristian’s Algorithm (1)

Both \( T_0 \) and \( T_1 \) are measured with the same clock.

Both \( T_0 \) and \( T_1 \) are measured with the same clock.

- Getting the current time from a time server.

Cristian’s Algorithm (2)

- Two problems
  - One major, time must never run backward
    - When slowing down, the interrupt routine adds less msec to the time until the correction has been made
    - Be advanced gradually by adding more msec at each interrupt
  - One minor, it takes a nonzero amount of time for the time server’s replay to get back to the sender
    - \( (T_1 - T_0) / 2 \)
    - \( (T_1 - T_0 - 1) / 2 \)
    - A series of measurement to be averaged
    - The message that came back fastest
The Berkeley Algorithm

- The time daemon asks all the other machines for their clock values
- The machines answer
- The time daemon tells everyone how to adjust their clock

Clock Synchronization

- **Internal:**
  - processors run clock sync protocol, e.g.: broadcasting their clock readings
  - each processor receives set of values from others (may differ)
  - algorithm would pick a synchronized value from the set

- **External:**
  - satellite system launched by military in early 1990's, became public and inexpensive
  - can think of satellites broadcasting the time
  - small radio receiver picks up signals from three satellites and triangulates to determine position
  - same computation also yields extremely accurate clock [milliseconds]

Network Time Protocol (RFC 1305)

- NTP is a way to synchronize computer time periodically.
- It can synchronize with other time servers (but not all can synchronize), which is why it's not considered a protocol.
- It can provide very accurate time synchronization, with an accuracy of milliseconds.
- NTP uses a client-server architecture.

Design Aims of NTP

- **External synchronization**
  - Enable clients across the Internet to be synchronized accurately to UTC

- **Reliability**
  - Can survive lengthy losses of connectivity

- **Scalability**
  - Can handle many clients

Synchronization Measures of NTP

- **Multicast mode**
  - Intend for use on a high speed LAN

- **Symmetric mode**
  - The highest accuracy
Let \( t, t' \): actual transmission time of \( m, m' \); \( o \): actual B’s clock offset relative to A. We have:

\[
T_{i-2} = T_{i-3} + t + o, \quad T_{i} = T_{i-1} + t' - o
\]

Symmetric Mode Synchronization (2)

\[
T_{i-2} = T_{i-3} + t + o, \quad T_{i} = T_{i-1} + t' - o
\]

Then:

addition:

\[
d_i = t + t' = T_{i-2} - T_{i-3} + T_{i} - T_{i-1}
\]

subtraction:

\[
o = (T_{i-2} - T_{i-3} + T_{i-1} - T_{i} + t' - t)/2
\]

where \( o_i = (T_{i-2} - T_{i-3} + T_{i-1} - T_{i})/2 \)

we have \( o = o_i + (t' - t)/2 \)

Symmetric Mode Synchronization (3)

we have \( o = o_i + (t' - t)/2 \)

Accuracy analysis

Due \( t, t' \geq 0 \), then

\[
o_i - (t' + t)/2 \leq o \leq o_i + (t' + t)/2
\]

Then

\[
o_i - d_i/2 \leq o \leq o_i + d_i/2
\]

\( o_i \) is the measured offset

\( d_i \) is the measured delay

Symmetric Mode Sync. Implementation

- NTP servers retain 8 most recent pairs \(<o, d>\)
- The value \( o \) of that corresponds to the minimum value \( d_i \) is chosen to estimate \( o \)
- A NTP server exchanges with several peers in addition to with parent
  - Peers with lower stratum numbers are favored
  - Peers with the lowest synchronization dispersion are favored

Outline

- Introduction
- Physical Clocks
- Logical Clocks
  - Happened-before relation
  - Synchronization Algorithms
  - Lamport timestamps
  - Vector timestamps
- Global State

Happen-Before (HB) Relation

\( \rightarrow \) denotes HB relation

HB1: If \( \exists \) process \( p_i: e \rightarrow e' \), then \( e \rightarrow e' \)

HB2: For any message \( m \),

\[
send(m) \rightarrow receive(m)
\]

HB3: If \( e, e' \) and \( e'' \) are events such that

\( e \rightarrow e' \) and \( e' \rightarrow e'' \), then \( e \rightarrow e'' \)

Causal ordering
Happen-before relation

- **Example**
  - \( a \parallel e \)
- **Shortcomings**
  - Not suitable to processes collaboration that does not involve messages transmission
  - Capture potential causal ordering

Events occurring at three processes

Lamport timestamps algorithm

- **LC1**
  - \( L_i \) is incremented before each event is issued at process \( P_i \)
  - \( L_i := L_i + 1 \)
- **LC2**
  - (a) When a process \( P_i \) sends a message \( m \), it piggybacks on \( m \) the value \( t = L_i \)
  - (b) On receiving \((m,t)\), a process \( P_j \) computes \( L_j := \max(L_j, t) \) and then applies LC1 before timestamping the event \( \text{receive}(m) \)

E.g., Lamport’s Logical Clocks

- (a) Three processes, each with its own clock. The clocks run at different rates.
- (b) Lamport’s algorithm corrects the clocks.

Adjustments take place in the middleware layer

Totally ordered logical clocks

**Assumption**

- \( T_i \): local timestamp of \( e \) that is an event occurring at \( P_i \)
- \( T_j \): local timestamp of \( e' \) that is an event occurring at \( P_j \)

Define the timestamps of \( e, e' \) are \((T_i, i), (T_j, j)\)

Define:

\( (T_i, i) < (T_j, j) \) if \( T_i < T_j \), or \( T_i = T_j \) and \( i < j \)
Example: Totally Ordered Multicasting (1)

- Updating a replicated database and leaving it in an inconsistent state.

Example: Totally Ordered Multicasting (2)

- Solution:
  - Process $P_i$ sends timestamped message $msg_i$ to all others.
The message itself is put in a local queue $queue_i$.
- Any incoming message $P_j$ is queued in $queue_j$, according to its timestamp, and acknowledged to every other process.

$P_i$ passes a message $msg_i$ to its application if:
1. $msg_i$ is at the head of $queue_i$.
2. For each process $P_k$, there is a message $msg_k$ in $queue_i$ with a larger timestamp.

Note: We are assuming that communication is reliable and FIFO ordered.

Lamport timestamps algorithm

- $e \rightarrow e' \Rightarrow L(e) < L(e')$
- $L(e) < L(e') \Rightarrow e \rightarrow e'$ or $e || e'$

Vector Clocks - algorithm

- Each process $p_i$ keeps a vector clock $V_i$.
- VC1: Initially, $V_{ij} = 0$, for $i, j = 1, 2, \ldots, N$.
- VC2: Just before $p_i$ timestamps an event, it sets $V_{ij} := V_{ij} + 1$.
- VC3: $p_i$ includes the value $t = V_i$ in every message it sends.
- VC4: When $p_i$ receives a timestamp $t$ in a message, it sets $V_{ij} := \max(V_{ij}, t[j])$, for $j = 1, 2, \ldots, N$.

Vector Clocks - example

- Compare vector timestamps
  - $V = V' \iff V_{ij} = V'_{ij}$ for $j = 1, 2, \ldots, N$.
  - $V \leq V' \iff V_{ij} \leq V'_{ij}$ for $j = 1, 2, \ldots, N$.
  - Otherwise $V > V'$.
  - $V(e) < V(e') \iff e \rightarrow e'$, $V(e) \leq V(e') \iff e || e'$.
When a process $P_j$ receives a message from $P_i$, the middleware layer will delay till:

1. $\alpha(m)[i] = VC_j[i] + 1$
2. $\alpha(m)[k] = VC_j[k]$ for all $k \neq i$

First condition means that this message is the next expected (no skipping).
Second condition means that we wait till we have seen all messages that $P_i$ saw when it sent the message.

Illustration of vector timestamps

<table>
<thead>
<tr>
<th>$P_0$</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1,0,0,0]</td>
<td>[2,0,0,0]</td>
<td>[2,1,1,0]</td>
<td>[2,2,1,0]</td>
</tr>
<tr>
<td>[0,0,1,0]</td>
<td>[0,0,0,1]</td>
<td>[0,0,0,0]</td>
<td>[0,0,0,1]</td>
</tr>
</tbody>
</table>

Vector timestamps accurately represent the happens-before relationship!

Define $VT(e) < VT(e')$ if,
- for all $i$, $VT(e)[i] \leq VT(e')[i]$, and
- for some $j$, $VT(e)[j] < VT(e')[j]$

Example: if $VT(e) = [2,1,1,0]$ and $VT(e') = [2,3,1,0]$ then $VT(e) < VT(e')$

Notice that not all VT's are "comparable" under this rule: consider $[4,0,0,0]$ and $[0,0,0,4]$

Now can show that $VT(e) < VT(e')$ if and only if $e \rightarrow e'$:
- If $e \rightarrow e'$, there exists a chain $e_0 \rightarrow e_1 \ldots \rightarrow e_n$ on which vector timestamps increase "hop by hop"
- If $VT(e) < VT(e')$ suffices to look at $VT(e')[proc(e)]$, where $proc(e)$ is the place that $e$ occurred. By definition, we know that $VT(e')[proc(e)]$ is at least as large as $VT(e)[proc(e)]$, and by construction, this implies a chain of events from $e$ to $e'$

Examples of VT's and happens-before

Example: suppose that $VT(e) = [2,1,0,1]$ and $VT(e') = [2,3,0,1]$, so $VT(e) < VT(e')$

How did $e'$ "learn" about the 3 and the 1?
- Either these events occurred at the same place as $e'$, or
- Some chain of send/receive events carried the values!

If VT's are not comparable, the corresponding events are concurrent!
Outline

- Introduction
- Physical Clocks
- Logical Clocks
- Global State
  - Chandy and Lamport Algorithm

Interpretations of Temporal Terms

- Understand now that “a happens before b” means that information can flow from a to b
- Understand that “a is concurrent with b” means that there is no information flow between a and b
- What about the notion of an “instant in time”, over a set of processes?

Chandy and Lamport: Consistent cuts

- Draw a line across a set of processes
- Line cuts each execution
- Consistent cut has property that the set of included events is closed under happens-before relation:
  - If the cut “includes” event b, and event a happens before b, then the cut also includes event a
  - In practice, this means that every “delivered” message was sent within the cut

Illustration of Consistent Cuts (1)

- Green cuts are consistent
- Red cut is inconsistent

Illustration of Consistent Cuts (2)

(a) A consistent cut
(b) An inconsistent cut

Chandy and Lamport Algorithm (1)

(a) Organization of a process and channels for a distributed snapshot
b) Process Q receives a marker for the first time and records its local state

- Q records all incoming message
- Q receives a marker for its incoming channel and finishes recording the state of the incoming channel

Summary

- Introduction
- Physical Clocks
  - Physical Clocks
  - Synchronization Algorithm
    - Cristian's method
    - The Berkeley Algorithm
    - The Network Time Protocol
- Logical Clocks
  - Happened-before relation
  - Synchronization Algorithm
  - Lamport timestamps
  - Vector timestamps
- Global State
  - Chandy and Lamport Algorithm

References

- Chapter 5 of [Tanenbaum, 2002]
- Chapter 11 of [Coulouri, 2005]

Textbooks

  - book1, [Coulouri, 2005]
  - book2, [Birman, 2005]
  - book3, [Tanenbaum, 2006]
  - book4, [Tanenbaum, 2002]
  - ....