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

时间简史

- 人类生活有紧密联系的太阳的运动是比较均匀
  - 地球自转是一天，公转是一年
- 天文计量，自从17世纪机械钟表发明，GMT
  - 太阳到达天空中它出现的最高点时称为中天
  - 两次连续的太阳中天之间的时间称为一个太阳日
  - 每天有24小时，每小时有3600秒，1/86400个太阳日是太阳秒
- 物理计量，20世纪40年代
  - 1秒是铯133原子作9192631770次跃迁所用的时间
  - BIH将这些值平均起来产生国际原子时间，简称为TAI
  - TAI和太阳秒计时之间的差增加到800毫秒时使用一次闰秒，修正后的时间系统称作统一协调时间，简称UTC
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 standard, 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 every 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 $t$.
- $\frac{dC}{dt}$ ideally should be 1
- Clocks will drift.
  - If we can bound the drift, then we have the maximum drift rate.
  - $1 - \rho \leq \frac{dC}{dt} \leq 1 + \rho$

Clock Drift

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.

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
Network Time Protocol (RFC 1305)

- NTP is a protocol that allows all computers to synchronize their clocks to absolute time, such as UTC.
- NTP can use any of several methods to synchronize the clocks: atomic clock, radio clock, satellite system, or communication via the Internet.
- NTP uses a simple, efficient protocol that allows clocks to synchronize even if the network connections are lost for some time.
- NTP uses a hierarchy of time servers to ensure that clocks can be synchronized even when the primary time server is unavailable.
- NTP can handle packet loss and network latency to ensure that clocks remain synchronized even when network conditions are poor.
- NTP can handle large numbers of computers and can be used to synchronize clocks on a wide variety of computer systems.

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: Enable clients to resynchronize sufficiently frequently to offset the rates of drift found in most computers.
- Security: Protect against interference with the time service.
- Redundant server & redundant path between servers.

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).

Synchronization Measures of NTP

- Multicast mode
  - Intend for use on a high speed LAN.
  - Assuming a small delay.
  - Low accuracy but efficient.
- Procedure-call mode
  - Similar to Christian’s.
  - Higher accuracy than multicast.
- Symmetric mode
  - The highest accuracy.

NTP Architecture

- Arrows denote synchronization control, numbers denote strata.
- Reconfigure when servers become unreachable.
Let $t, t'$: actual transmission time of $m, m'$; $o$: actual B’s clock offset relative to A. We have $T_{i,2} = T_{i,3} + t + o$, $T_i = T_{i,1} + t' - o$.

**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:**
  \[ o = \frac{(T_{i,2} - T_{i,3} + T_i - T_{i,1})}{2} \]

where $o_i = \frac{(T_{i,2} - T_{i,3} + T_i - T_{i,1})}{2}$

we have $o = o_i + \frac{(t'-t)}{2}$

**Symmetric Mode Synchronization (3)**

we have $o = o_i + \frac{(t'-t)}{2}$

Accuracy analysis

Due $t, t' \geq 0$, then

$o_i - \frac{(t'+t)}{2} \leq o \leq o_i + \frac{(t'+t)}{2}$

Then

$-d_i/2 \leq o \leq o_i + d_i/2$

$o_i$ 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 $e, e', e''$ are events such that
  - $e \rightarrow e'$ and $e'' \rightarrow e''$, then $e \rightarrow e''$
- **Causal ordering**
Happen-before relation

- Example
  - a || e
- Shortcomings
  - Not suitable to processes collaboration that does not involve messages transmission
  - Capture potential causal ordering

Events occurring at three processes

![Diagram showing events occurring at three processes](image)

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 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.

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 \( queue_j \) 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, \( V[i][j] = 0 \) for \( i, j = 1, 2, \ldots, N \).
- VC2: Just before \( p_i \) timestamps an event, it sets \( V[i][i] := V[i][i] + 1 \).
- 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 \( V[j][j] := \max(V[j][j], t[j]) \) for \( j = 1, 2, \ldots, N \).

Vector Clocks - example

Vector Clocks - significance

- Compare vector timestamps
  - \( V = V' \) iff \( V[j][j] = V'[j][j] \) for \( j = 1, 2, \ldots, N \)
  - \( V \leq V' \) iff \( V[j][j] \leq V'[j][j] \) for \( j = 1, 2, \ldots, N \)
  - Otherwise \( V < V' \)
  - \( V(e) < V(e') \Leftrightarrow e \rightarrow e' \), \( V(e') \approx e/e' \)
Enforcing Causal Communication

- When a process \( P_j \) receives a message from \( P_i \), the middleware layer will delay until:
  - \( \alpha(m[j]) = VC[j][i] + 1 \)
  - \( \alpha(m[j]) = VC[j][k] \) 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

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 \([0,0,0,4]\)

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)

- Red cut is inconsistent
- Green cuts are consistent

Illustration of Consistent Cuts (2)

- A consistent cut
- An inconsistent cut

Chandy and Lamport Algorithm (1)

- Organization of a process and channels for a distributed snapshot
Chandy and Lamport Algorithm (2)

- Process Q receives a marker for the first time and records its local state.
- Q records all incoming messages.
- 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]
  - ....