15-440 Distributed Systems

Lecture 9 – Time Synchronization

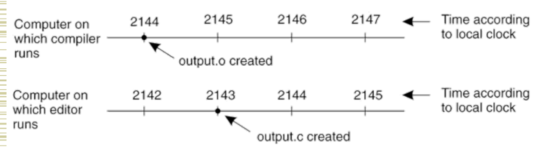
Today's Lecture

- Need for time synchronization
- Time synchronization techniques
- Lamport Clocks
- Vector Clocks

Why Global Timing?

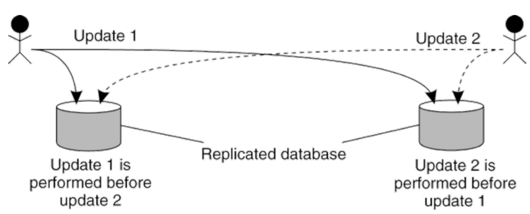
- Suppose there were a globally consistent time standard
- Would be handy
  - Who got last seat on airplane?
  - Who submitted final auction bid before deadline?
  - Did defense move before snap?

Impact of Clock Synchronization



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

Replicated Database Update

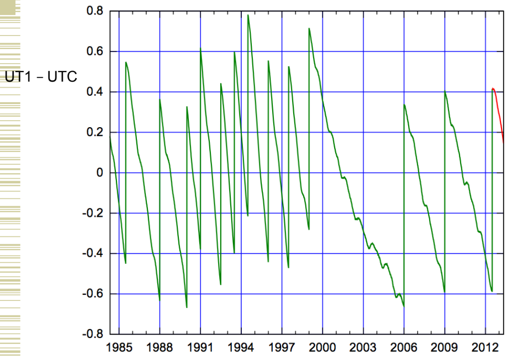


- Updating a replicated database and leaving it in an inconsistent state

Time Standards

- UT1
  - Based on astronomical observations
  - "Greenwich Mean Time"
- TAI
  - Started Jan 1, 1958
  - Each second is 9,192,631,770 cycles of radiation emitted by Cesium atom
  - Has diverged from UT1 due to slowing of earth's rotation
- UTC
  - TAI + leap seconds to be within 0.9s of UT1
  - Currently 35
  - Most recent: June 30, 2012

Comparing Time Standards

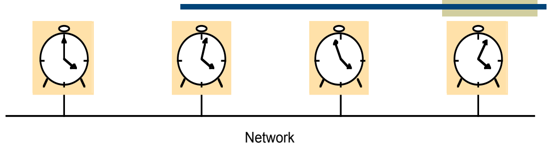


Coordinated Universal Time (UTC)

- Is broadcast from radio stations on land and satellite (e.g. GPS)
- Computers with receivers can synchronize their clocks with these timing signals
- Signals from land-based stations are accurate to about 0.1-10 millisecond
- Signals from GPS are accurate to about 1 microsecond
  - Why can't we put GPS receivers on all our computers?

8

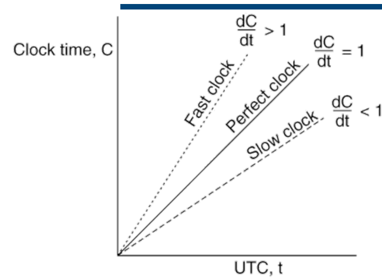
Clocks in a Distributed System



- Computer clocks are not generally in perfect agreement
  - **Skew**: the difference between the times on two clocks (at any instant)
- Computer clocks are subject to clock drift (they count time at different rates)
  - **Clock drift rate**: the difference per unit of time from some ideal reference clock
  - Ordinary quartz clocks drift by about 1 sec in 11-12 days. (10^{-6} secs/sec).
  - High precision quartz clocks drift rate is about 10^{-7} or 10^{-8} secs/sec

9

Clock Synchronization Algorithms



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

10

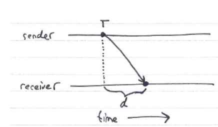
Today's Lecture

- Need for time synchronization
- Time synchronization techniques
- Lamport Clocks
- Vector Clocks

11

Perfect networks

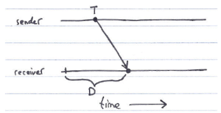
- Messages always arrive, with propagation delay exactly *d*



- Sender sends time *T* in a message
- Receiver sets clock to *T+d*
  - Synchronization is exact

Synchronous networks

- Messages always arrive, with propagation delay *at most D*



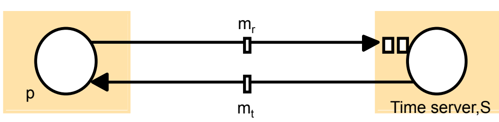
- Sender sends time T in a message
- Receiver sets clock to $T + D/2$
  - Synchronization error is at most $D/2$

Synchronization in the real world

- Real networks are asynchronous
  - Message delays are arbitrary
- Real networks are unreliable
  - Messages don't always arrive

Cristian's Time Sync

- A time server S receives signals from a UTC source
  - Process p requests time in m_r and receives t in m_t from S
  - p sets its clock to $t + T_{round}/2$
  - Accuracy ± ($T_{round}/2 - min$) :
    - because the earliest time S puts t in message m_t is min after p sent m_r.
    - the latest time was min before m_t arrived at p
    - the time by S's clock when m_t arrives is in the range [$t+min$, $t + T_{round} - min$]



T_{round} is the round trip time recorded by p
min is an estimated minimum round trip time

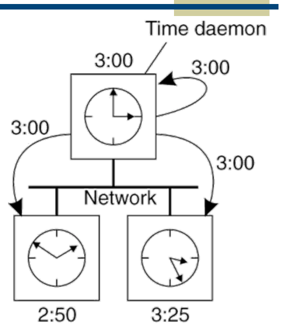
Berkeley algorithm

- Cristian's algorithm -
  - a single time server might fail, so they suggest the use of a group of synchronized servers
  - it does not deal with faulty servers
- Berkeley algorithm (also 1989)
  - An algorithm for internal synchronization of a group of computers
  - A *master* polls to collect clock values from the others (*slaves*)
  - The master uses round trip times to estimate the slaves' clock values
  - It takes an average (eliminating any above average round trip time or with faulty clocks)
  - It sends the required adjustment to the slaves (better than sending the time which depends on the round trip time)
  - Measurements
    - 15 computers, clock synchronization 20-25 millisecs drift rate < $2x10^{-5}$
    - If master fails, can elect a new master to take over (not in bounded time)

•

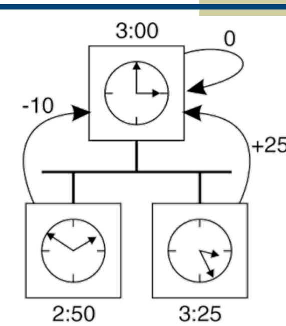
The Berkeley Algorithm (1)

- The time daemon asks all the other machines for their clock values.



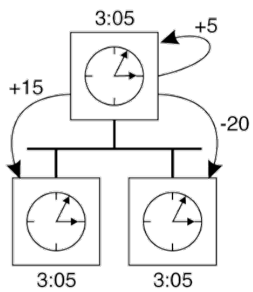
The Berkeley Algorithm (2)

- The machines answer.



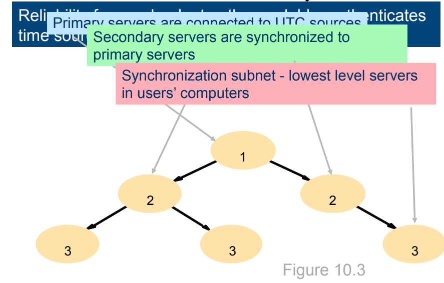
The Berkeley Algorithm (3)

- The time daemon tells everyone how to adjust their clock.



Network Time Protocol (NTP)

- A time service for the Internet - synchronizes clients to



The Network Time Protocol (NTP)

- Uses a hierarchy of time servers
  - Class 1 servers have highly-accurate clocks
    - connected directly to atomic clocks, etc.
  - Class 2 servers get time from only Class 1 and Class 2 servers
  - Class 3 servers get time from any server
- Synchronization similar to Cristian's alg.
  - Modified to use multiple one-way messages instead of immediate round-trip
- Accuracy: Local ~1ms, Global ~10ms

NTP Reference Clock Sources (1997 survey)

- In a survey of 36,479 peers, found 1,733 primary and backup external reference sources
- 231 radio/satellite/modem primary sources
  - 47 GPS satellite (worldwide), GOES satellite (western hemisphere)
  - 57 WWVB radio (US)
  - 17 WWV radio (US)
  - 63 DCF77 radio (Europe)
  - 6 MSF radio (UK)
  - 5 CHU radio (Canada)
  - 7 modem time service (NIST and USNO (US), PTB (Germany), NPL (UK))
  - 25 other (precision PPS sources, etc.)
- 1,502 local clock backup sources (used only if all other sources fail)
- For some reason or other, 88 of the 1,733 sources appeared down at the time of the survey

Udel Master Time Facility (MTF) (from January 2000)

Spectracom 8170 WWVB Receiver

Spectracom 8183 GPS Receiver

Spectracom 8170 WWVB Receiver

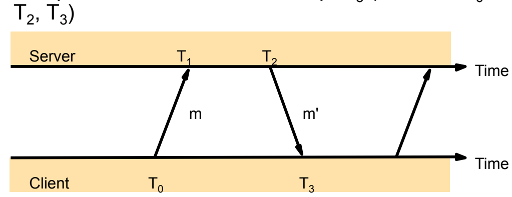
Spectracom 8183 GPS Receiver

Hewlett Packard 105A Quartz Frequency Standard

Hewlett Packard 5061A Cesium Beam Frequency Standard

NTP Protocol

- All modes use UDP
- Each message bears timestamps of recent events:
  - Local times of Send and Receive of previous message
  - Local times of Send of current message
- Recipient notes the time of receipt T_3 (we have T_0, T_1, T_2, T_3)



Accuracy of NTP

- Timestamps
  - t_0 is the client's timestamp of the request packet transmission,
  - t_1 is the server's timestamp of the request packet reception,
  - t_2 is the server's timestamp of the response packet transmission and
  - t_3 is the client's timestamp of the response packet reception.
- RTT = wait_time_client – server_proc_time = $(t_3-t_0) - (t_2-t_1)$
- Offset = $((t_1-t_0) + (t_3-t_2))/2$ = ((offset + delay) + (offset – delay))/2
- NTP servers filter pairs <*rtt_i*, *$offset_i$*>, estimating reliability from variation, allowing them to select peers
- Accuracy of 10s of millisecs over Internet paths (1 on LANs)

3/4/14 25

How To Change Time

- Can't just change time
  - Why not?
- Change the update rate for the clock
  - Changes time in a more gradual fashion
  - Prevents inconsistent local timestamps

26

Important Lessons

- Clocks on different systems will always behave differently
  - Skew and drift between clocks
- Time disagreement between machines can result in undesirable behavior
- Clock synchronization
  - Rely on a time-stamped network messages
  - Estimate delay for message transmission
  - Can synchronize to UTC or to local source
  - Clocks never exactly synchronized
- Often inadequate for distributed systems
  - might need totally-ordered events
  - might need millionth-of-a-second precision

27

Today's Lecture

- Need for time synchronization
- Time synchronization techniques
- Lamport Clocks
- Vector Clocks

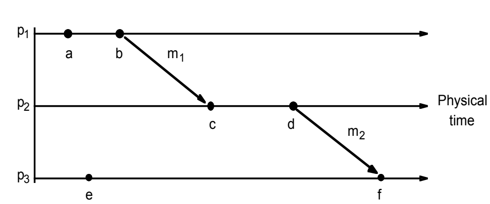
28

Logical time

- Capture just the "happens before" relationship between events
  - Discard the infinitesimal granularity of time
  - Corresponds roughly to causality

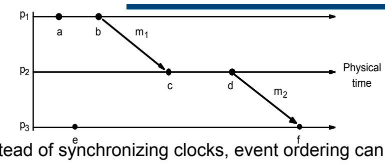
Logical time and logical clocks (Lamport 1978)

- Events at three processes



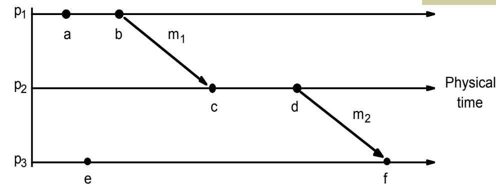
30

Logical time and logical clocks (Lamport 1978)



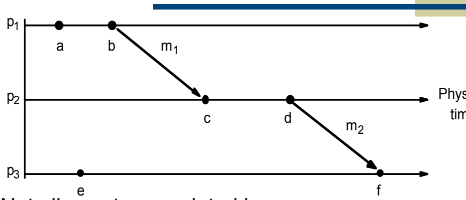
- Instead of synchronizing clocks, event ordering can be used
  1. If two events occurred at the same process p_i (i = 1, 2, … N) then they occurred in the order observed by p_i, that is the definition of: $\rightarrow_i$
  2. when a message, m is sent between two processes, send(m) happens before receive(m)
  3. The happened before relation is transitive
- The happened before relation is the relation of causal ordering

Logical time and logical clocks (Lamport 1978)



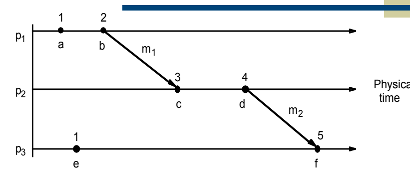
- $a \rightarrow b$ (at p_1) $c \rightarrow d$ (at p_2)
- $b \rightarrow c$ because of m_1
- also $d \rightarrow f$ because of m_2

Logical time and logical clocks (Lamport 1978)



- Not all events are related by $\rightarrow$
- Consider *a* and *e* (different processes and no chain of messages to relate them)
  - they are not related by $\rightarrow$; they are said to be concurrent
  - written as $a \parallel e$

Lamport Clock (1)

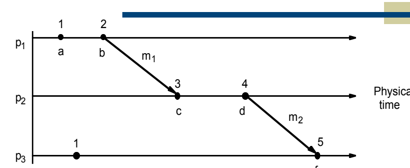


- A logical clock is a monotonically increasing software counter
  - It need not relate to a physical clock.
- Each process p_i has a logical clock, L_i which can be used to apply logical timestamps to events
  - Rule 1: L_i is incremented by 1 before each event at process p_i
  - Rule 2:
    - (a) when process p_i sends message m, it piggybacks $t = L_i$
    - (b) when p_j receives (m,t) it sets $L_j := max(L_j, t)$ and applies rule 1 before timestamping the event *receive (m)*

Lamport's algorithm

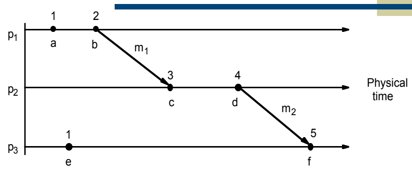
- Each process *i* keeps a local clock, L_i
- Three rules:
  1. At process *i*, increment L_i before each event
  2. To send a message *m* at process *i*, apply rule 1 and then include the current local time in the message: i.e., $send(m,L_i)$
  3. To receive a message (m,t) at process *j*, set $L_j = max(L_j,t)$ and then apply rule 1 before time-stamping the receive event
- The global time $L(e)$ of an event *e* is just its local time
  - For an event *e* at process *i*, $L(e) = L_i(e)$

Lamport Clock (1)



- each of p_1, p_2, p_3 has its logical clock initialised to zero,
- the clock values are those immediately after the event.
- e.g. 1 for a, 2 for b.
- for m_1, 2 is piggybacked and c gets max(0,2)+1 = 3

Lamport Clock (1)



- *e* →*e*' implies $L(e)<L(e')$
- The converse is not true, that is $L(e)<L(e')$ does not imply *e* →*e*'
  - e.g. $L(b) > L(e)$ but *b* || *e*

37

Lamport logical clocks

- Lamport clock *L* orders events consistent with logical "happens before" ordering
  - If e → e', then $L(e) < L(e')$
- But not the converse
  - $L(e) < L(e')$ does not imply *e* → *e'*
- Similar rules for concurrency
  - $L(e) = L(e')$ implies e‖e' (for distinct *e*,*e'*)
  - e‖e' does not imply $L(e) = L(e')$
  - i.e., Lamport clocks arbitrarily order some concurrent events

Total-order Lamport clocks

- Many systems require a total-ordering of events, not a partial-ordering
- Use Lamport's algorithm, but break ties using the process ID
  - $L(e) = M * L_i(e) + i$
    - *M* = maximum number of processes
    - i = process ID

Today's Lecture

- Need for time synchronization
- Time synchronization techniques
- Lamport Clocks
- Vector Clocks

40

Vector Clocks

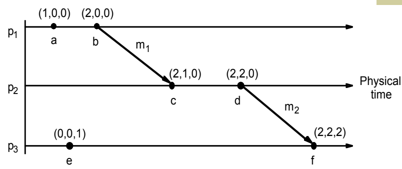
- Vector clocks overcome the shortcoming of Lamport logical clocks
  - $L(e) < L(e')$ does not imply *e* happened before *e*'
- Goal
  - Want ordering that matches causality
  - V(e) < V(e') if and only if e → e'
- Method
  - Label each event by vector V(e) $[c_1, c_2 \ldots, c_n]$
    - c_i = # events in process i that causally precede e

41

Vector Clock Algorithm

- Initially, all vectors [0,0,…,0]
- For event on process i, increment own c_i
- Label message sent with local vector
- When process j receives message with vector $[d_1, d_2, \ldots, d_n]$:
  - Set local each local entry k to $\max(c_k, d_k)$
  - Increment value of c_j

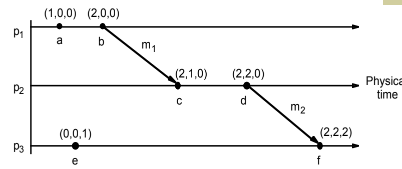
Vector Clocks



- At p_1
  - *a occurs at* (1,0,0); *b* occurs at (2,0,0)
  - piggyback (2,0,0) on m_1
- At p_2 on receipt of m_1 use *max* ((0,0,0), (2,0,0)) = (2, 0, 0) and add 1 to own element = (2,1,0)
- Meaning of =, <=, max etc for vector timestamps
  - compare elements pairwise

43

Vector Clocks



- Note that e →e' implies V(e)<V(e'). The converse is also true
- Can you see a pair of parallel events?
  - *c* || *e* (parallel) because neither *V*(c) <= *V*(*e*) nor *V*(*e*) <= *V*(*c*)

44

Important Points

- Physical Clocks
  - Can keep closely synchronized, but never perfect
- Logical Clocks
  - Encode causality relationship
  - Lamport clocks provide only one-way encoding
  - Vector clocks provide exact causality information

Last Lecture – Clock Sync Important Lessons

- Clocks on different systems will always behave differently
  - Skew and drift between clocks
- Time disagreement between machines can result in undesirable behavior
- Two paths to solution: synchronize clocks or ensure consistent clocks
- Clock synchronization
  - Rely on a time-stamped network messages
  - Estimate delay for message transmission
  - Can synchronize to UTC or to local source

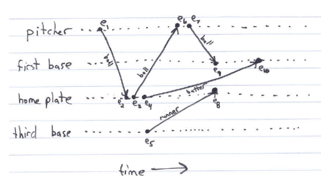
46

A baseball example

- Four locations: pitcher's mound, first base, home plate, and third base
- Ten events:
  - e_1: pitcher throws ball to home
  - e_2: ball arrives at home
  - e_3: batter hits ball to pitcher
  - e_4: batter runs to first base
  - e_5: runner runs to home
  - e_6: ball arrives at pitcher
  - e_7: pitcher throws ball to first base
  - e_8: runner arrives at home
  - e_9: ball arrives at first base
  - e_{10}: batter arrives at first base

A baseball example

- Pitcher knows e_1 happens before e_6, which happens before e_7
- Home plate umpire knows e_2 is before e_3, which is before e_4, which is before e_8, …
- Relationship between e_8 and e_9 is unclear



Ways to synchronize

- Send message from first base to home?
  - Or to a central timekeeper
  - How long does this message take to arrive?
- Synchronize clocks before the game?
  - Clocks drift
    - million to one => 1 second in 11 days
- Synchronize continuously during the game?
  - GPS, pulsars, etc

The baseball example revisited

- $e_1 \rightarrow e_2$
  - by the message rule
- $e_1 \rightarrow e_{10}$, because
  - $e_1 \rightarrow e_2$, by the message rule
  - $e_2 \rightarrow e_4$, by local ordering at home plate
  - $e_4 \rightarrow e_{10}$, by the message rule
  - Repeated transitivity of the above relations
- $e_8 \| e_9$, because
  - No application of the $\rightarrow$ rules yields either $e_8 \rightarrow e_9$ or $e_9 \rightarrow e_8$

Lamport on the baseball example

- Initializing each local clock to 0, we get

$L(e_1) = 1$	(pitcher throws ball to home)
$L(e_2) = 2$	(ball arrives at home)
$L(e_3) = 3$	(batter hits ball to pitcher)
$L(e_4) = 4$	(batter runs to first base)
$L(e_5) = 1$	(runner runs to home)
$L(e_6) = 4$	(ball arrives at pitcher)
$L(e_7) = 5$	(pitcher throws ball to first base)
$L(e_8) = 5$	(runner arrives at home)
$L(e_9) = 6$	(ball arrives at first base)
$L(e_{10}) = 7$	(batter arrives at first base)

- For our example, Lamport's algorithm says that the run scores!

Lamport on the baseball example

- Initializing each local clock to 0, we get

$L(e_1) = 1$	(pitcher throws ball to home)
$L(e_2) = 2$	(ball arrives at home)
$L(e_3) = 3$	(batter hits ball to pitcher)
$L(e_4) = 4$	(batter runs to first base)
$L(e_5) = 1$	(runner runs to home)
$L(e_6) = 4$	(ball arrives at pitcher)
$L(e_7) = 5$	(pitcher throws ball to first base)
$L(e_8) = 5$	(runner arrives at home)
$L(e_9) = 6$	(ball arrives at first base)
$L(e_{10}) = 7$	(batter arrives at first base)

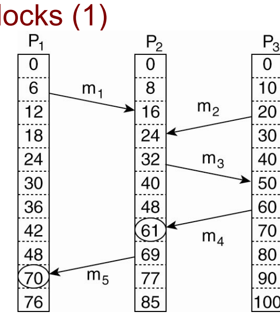
- For our example, Lamport's algorithm says that the run scores!

Vector clocks on the baseball example

Event	Vector	Action
e_1	[1,0,0,0]	pitcher throws ball to home
e_2	[1,0,1,0]	ball arrives at home
e_3	[1,0,2,0]	batter hits ball to pitcher
e_4	[1,0,3,0]	batter runs to first base)
e_5	[0,0,0,1]	runner runs to home
e_6	[2,0,2,0]	ball arrives at pitcher
e_7	[3,0,2,0]	pitcher throws ball to 1st base
e_8	[1,0,4,1]	runner arrives at home
e_9	[3,1,2,0]	ball arrives at first base
e_{10}	[3,2,3,0]	batter arrives at first base

- Vector: [p,f,h,t]

Vector Clocks (1)



- Concurrent message transmission using logical clocks.

54

Vector Clocks (2)

- Vector clocks are constructed by letting each process P_i maintain a vector VC_i with the following two properties:

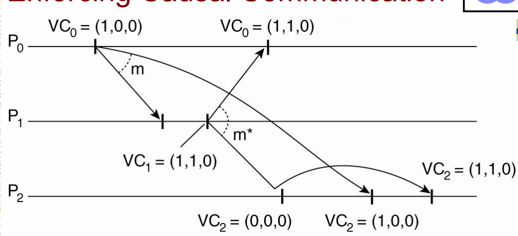
1. $VC_i[\,i\,]$ is the number of events that have occurred so far at P_i. In other words, $VC_i[\,i\,]$ is the local logical clock at process P_i.
2. If $VC_i[\,j\,] = k$ then P_i knows that k events have occurred at P_j. It is thus P_i's knowledge of the local time at P_j.

Vector Clocks (3)

- Steps carried out to accomplish property 2 of previous slide:

1. Before executing an event P_i executes $VC_i[\,i\,] \leftarrow VC_i[i] + 1$.
2. When process P_i sends a message m to P_j, it sets *m*'s (vector) timestamp *ts (m)* equal to VC_i after having executed the previous step.
3. Upon the receipt of a message m, process P_j adjusts its own vector by setting $VC_j[k] \leftarrow \max\{VC_j[k], ts(m)[k]\}$ for each *k*, after which it executes the first step and delivers the message to the application.

Enforcing Causal Communication



- Enforcing causal communication

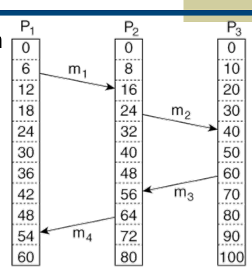
Lamport's Logical Clocks (1)

The "happens-before" relation → can be observed directly in two situations:

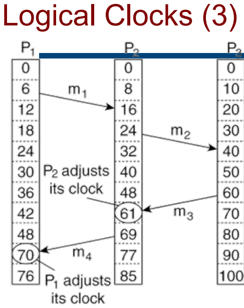
- If a and b are events in the same process, and a occurs before b, then a → b is true.
- If a is the event of a message being sent by one process, and b is the event of the message being received by another process, then a → b

Lamport's Logical Clocks (2)

- Three processes, each with its own clock. The clocks run at different rates.



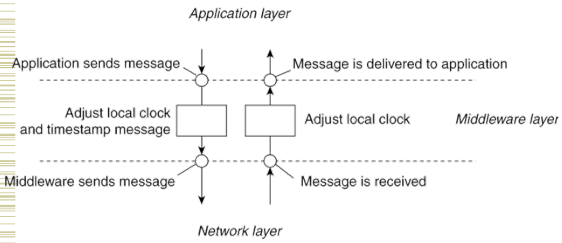
Lamport's Logical Clocks (3)



- Lamport's algorithm corrects the clocks.

Lamport's Logical Clocks (4)

- The positioning of Lamport's logical clocks in distributed systems.



61

Lamport's Logical Clocks (5)

- Updating counter C_i for process P_i

1. Before executing an event P_i executes $C_i \leftarrow C_i + 1$.
2. When process P_i sends a message m to P_j, it sets *m*'s timestamp *ts (m)* equal to C_i after having executed the previous step.
3. Upon the receipt of a message *m*, process P_j adjusts its own local counter as $C_j \leftarrow \max\{C_j, ts(m)\}$, after which it then executes the first step and delivers the message to the application.

62

Last Lecture – Clock Sync Important Lessons

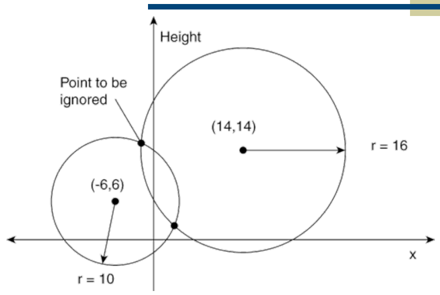
- Clocks on different systems will always behave differently
  - Skew and drift between clocks
- Time disagreement between machines can result in undesirable behavior
- Two paths to solution: synchronize clocks or ensure consistent clocks
- Clock synchronization
  - Rely on a time-stamped network messages
  - Estimate delay for message transmission
  - Can synchronize to UTC or to local source

63

Distributed time

- Premise
  - The notion of time is well-defined (and measurable) at each single location
  - But the relationship between time at different locations is unclear
    - Can minimize discrepancies, but never eliminate them
- Reality
  - Stationary GPS receivers can get global time with < 1μs error
  - Few systems designed to use this

Global Positioning System (1)



- Computing a position in a two-dimensional space.

65

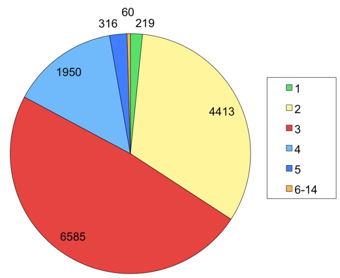
Global Positioning System (2)

Real world facts that complicate GPS

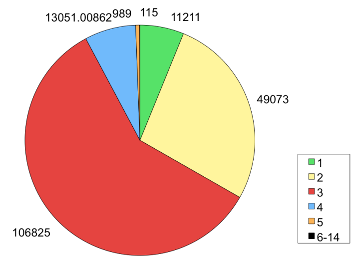
- It takes a while before data on a satellite's position reaches the receiver.
- The receiver's clock is generally not in synch with that of a satellite.

66

Server population by stratum (1997 survey)



Client population by stratum (1997 survey)



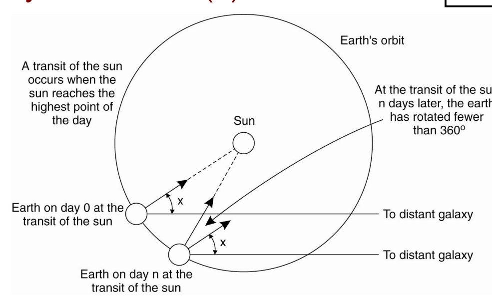
NTP - synchronisation of servers

- The synchronization subnet can reconfigure if failures occur, e.g.
  - a primary that loses its UTC source can become a secondary
  - a secondary that loses its primary can use another primary

Modes of synchronization:

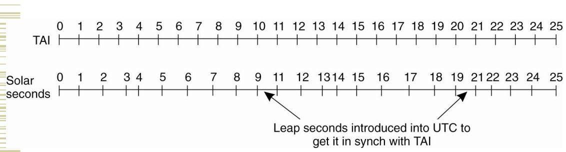
- Multicast
  - A server within a high speed LAN multicasts time to others which set clocks assuming some delay (not very accurate)
- Procedure call
  - A server accepts requests from other computers (like Cristiain's algorithm). Higher accuracy. Useful if no hardware multicast.
- Symmetric
  - Pairs of servers exchange messages containing time information
  - Used where very high accuracies are needed (e.g. for higher levels)

Physical Clocks (1)



- Figure 6-2. Computation of the mean solar day.

Physical Clocks (2)



- Figure 6-3. TAI seconds are of constant length, unlike solar seconds. Leap seconds are introduced when necessary to keep in phase with the sun.

Need for Precision Time

- Distributed database transaction journalling and logging
- Stock market buy and sell orders
- Secure document timestamps (with cryptographic certification)
- Aviation traffic control and position reporting
- Radio and TV programming launch and monitoring
- Intruder detection, location and reporting
- Multimedia synchronization for real-time teleconferencing
- Interactive simulation event synchronization and ordering
- Network monitoring, measurement and control
- Early detection of failing network infrastructure devices and air conditioning equipment
- Differentiated services traffic engineering
- Distributed network gaming and training

Vector Clocks

- $V_i[\ i\ ]$ is the number of events that p_i has timestamped
- $V_i[\ j\ ]$ ($j \neq i$) is the number of events at p_j that p_i has been affected by

Vector clock V_i at process p_i is an array of N integers

1. initially $V_i[j] = 0$ for $i, j = 1, 2, \ldots N$
2. before p_i timestamps an event it sets $V_i[i] := V_i[i] + 1$
3. p_i piggybacks $t = V_i$ on every message it sends
4. when p_i receives (m,t) it sets $V_i[j] := \max(V_i[j]\ ,\ t[j])$ $j = 1, 2, \ldots N$ *(then before next event adds 1 to own element using rule 2)*

73