Cloud computing and distributed systems

Time and global services

Zeynep Yücel

Ca' Foscari University of Venice
zeynep.yucel@unive.it
yucelzeynep.github.io

Introduction

- Importance of time in
  - Determining event timing.
  - Data consistency, elimination of duplicates etc.
- Multiple frames of reference complicate time.
- Simultaneity differs for observers.
  - Events not simultaneous in all frames.

Introduction

- Special relativity disproves absolute time.
- I.e. event order varies, but with the exception of causality.
  - Cause-effect relation consistent for all observers.
- With no global time, how to determine event order?

Clocks, events and process states

- Consider events in single-process.
- Let N processes run on separate processors.
- Processes communicate by sending messages.
- Each process has individual state.
  - State includes variables, OS objects.
  - As the process executes, its state changes.
- Actions of a process include send/receive and operations on its state.
- Event is an individual action performed by the process.
- Ordering events e and e' within the process:

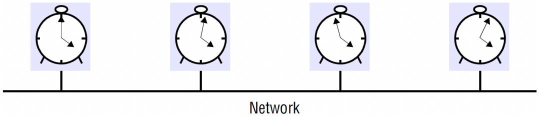
$$e \rightarrow_i e'$$

- History of process p_i: its ordered event series.

$$history(p_i) = \left\langle e_i^0, e_i^1, \ldots \right\rangle$$

Clocks, events and process states

Clocks



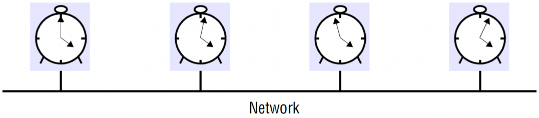
- Timestamping events is different than ordering them.
- Physical clocks count oscillations.
- For process p_i, hardware clock is $H_i(t)$
- Software clock $C_i(t)$ is derived from $H_i(t)$ by scaling translating:

$$C_i(t) = \alpha H_i(t) + \beta$$

- $C_i(t)$ not perfectly accurate, but can be used as timestamp if sufficiently good.
- If resolution adequate, different events have different time stamps.

Clocks, events and process states

Clock skew and clock drift



- Instantaneous difference: skew.
- Divergence: drift.
  - Small differences accumulate and cause large discrepancies.

Clocks, events and process states

Coordinated Universal Time

- Coordinated universal time: UTC standard.
- Sync with accurate sources, i.e. atomic clocks.
  - Astronomical time and atomic time diverge.
  - Leap seconds adjustment.
- UTC signals broadcast from radio stations and satellites.

Synchronizing physical clocks

- External synchronization: sync with an authoritative source S with accuracy D

$$|S(t) - C_i(t)| < D, \forall i$$

- Internal synchronization: sync with each other

$$|C_i(t) - C_j(t)| < D, \forall i, j$$

- External sync with accuracy D implies internal sync with accuracy $2D$.
- Internal sync does not imply anything about external sync.
- Correctness condition: drift less than a bound ρ

$$(1-\rho)(t'-t) \leq H(t') - H(t) \leq (1+\rho)(t'-t)$$

- Weaker correctness condition: monotonicity.

$$t' > t \Rightarrow C(t') > C(t)$$

Synchronizing physical clocks

- If a fast clock is corrected by a jump, some events "remain" in the future.
  - Jump update causes non-monotonicity.
- Adjust α and β to obtain monotonicity.

$$C_i(t) = \alpha H_i(t) + \beta$$

- Faulty clock: one which does not satisfy any correctness condition.
- Crash: clock that does not tick.
- Other failures: arbitrary (e.g. Y2K bug).

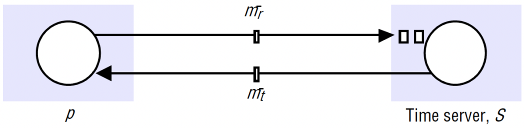
Synchronizing physical clocks

Synchronization in a synchronous system

- Internal synchronization between two processes.
  - Synchronous system: Known values of drift rate, maximum message transmission delay, and execution time.
- A process sends clock time t via message m.
- Another process adjusts its clock on receipt to $t + T_{trans}$, where T_{trans} is transmission time of m
  - T_{trans} unknown, varying.
  - Min/max for T_{trans}.
  - Transmission time uncertainty $u = max - min$
  - Clock skew can be large as u.
  - Midpoint minimizes skew.
- Real-world systems are asynchronous, i.e. no upper bound max.

Synchronizing physical clocks

Cristian's method for synchronizing clocks



- External synchronization.
- Process sends a request at t and server replies with its clock.
    - Round-trip time T_{round}.
- Clock setting formula: $t + (T_{round}/2)$.
- If minimum transmission time *min* is known, accuracy can be defined as
    - $\pm(T_{round}/2 - min)$.
- Multiple requests improve accuracy.

Synchronizing physical clocks

Discussion of Cristian's algorithm

- Christian relies on a single server: fault tolerance issues.
  - Multiple synchronized servers.
- Faulty or impostor servers
  - Issues addressed by Berkeley.
  - Authentication techniques for malicious servers.

Synchronizing physical clocks

The Berkeley algorithm

- Berkeley algorithm for internal sync.
- Coordinator (master) polls slaves.
- Master collects clocks of slaves.
- Master ignores reading, if round-trip time is too long.
- Master calculates average for smoothing.
- Instead of clock value, adjustment values sent.
- Master can be replaced in case of failure.

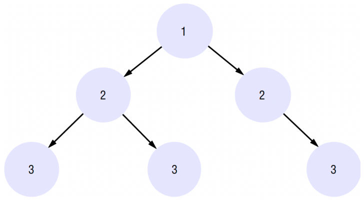
Synchronizing physical clocks

The Network Time Protocol (NTP)

- Christian, Berkley for Intranet; NTP for Internet.
- Goals of NTP:
  - Syncs to UTC.
  - Offers reliable service with redundant servers and paths.
  - Give frequent resynchronization opportunity against drift.
  - Offers protection against interference via authentication.

Synchronizing physical clocks

The Network Time Protocol



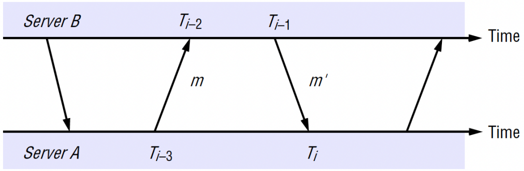
- NTP uses network of servers.
- Primary servers connect to time sources.
- Secondary servers sync with primary servers.
- Higher stratum numbers with less accuracy.
- NTP accounts for round-trip delays.
- Subnet reconfigures on failure.
  - Servers can switch roles (e.f. of time source fails, primary becomes a secondary in another subnet).

Synchronizing physical clocks
The Network Time Protocol

- NTP supports three sync modes.
- Multicast mode for LAN.
  - Used on high-speed LANs.
  - Servers multicast time.
  - Clients set clocks assuming with small delay.
- Procedure-call mode:
  - One server processes requests.
  - Replies with timestamp.
  - Higher accuracy than multicast.
- Symmetric mode:
  - Used by servers in LANs or high stratum time servers.
  - Servers exchange timing messages.
  - More communication improves accuracy.

Synchronizing physical clocks

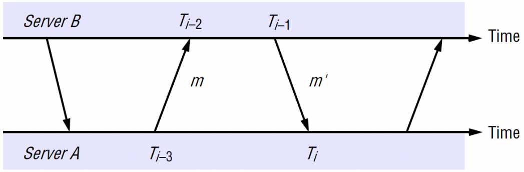
The Network Time Protocol



- In procedure call or symmetric mode, pairs of messages exchanged.
- UDP protocol can be unreliable.
- Recipient notes local time upon receiving.
- Offset o_i: estimate of offset between clocks.
- Delay d_i: total transmission time.

Synchronizing physical clocks

The Network Time Protocol



- Let actual transmission times of m, m' be t, t'.
- Let true clock offset of B relative to A be o.
- Then $T_{i-2} = T_{i-3} + t + o$ and $T_i = T_{i-1} + t' - o$
- d_i is $t + t'$, i.e.

$$d_i = T_{i-2} - T_{i-3} + T_i - T_{i-1}$$

- Account for asymmetric transmission delays.
- $o_i = (T_{i-2} - T_{i-3} + T_{i-1} - T_i)/2$.

Synchronizing physical clocks

The Network Time Protocol

- NTP servers apply filtering on o, d pairs.
  - Retain several recent pairs and choose the o_j for minimum d_j as estimate of o.
  - Filter dispersion indicates reliability.
- Communicates with several peers, rather than only one.
- Apply peer-selection process.
  - Prefers lower stratum peers.
  - Favors lower synchronization dispersion.

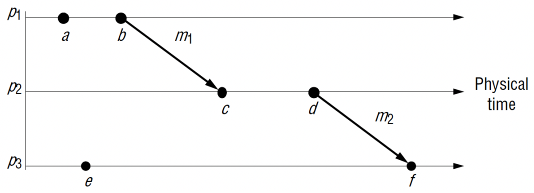
Synchronizing physical clocks

Logical time and logical clocks p1

- Perfect synchronization impossible.
- A scheme for ordering events.
  - Events in the same process ordered as they are observed by the process.
  - Send before receive.
- Happened-before relation (HB).
- HB defined by:
  - HB1: If $\exists\ p_i : e \rightarrow_i e'$, then $e \rightarrow e'$.
  - HB2: $\forall m$, $send(m) \rightarrow receive(m)$
  - HB3: If $e \rightarrow e'$ and $e' \rightarrow e''$, then $e \rightarrow e''$.

Synchronizing physical clocks

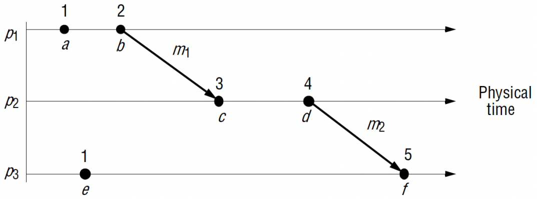
Logical time and logical clocks from it can also to end



- $a \rightarrow b$ (same p), $b \rightarrow c$ (send-receive), $c \rightarrow d$ (same p), $d \rightarrow f$ (send-receive).
- Not all events related by HB.
- Concurrent events, e.g. $a || e$.
- HB does not guarantee causation.

Synchronizing physical clocks

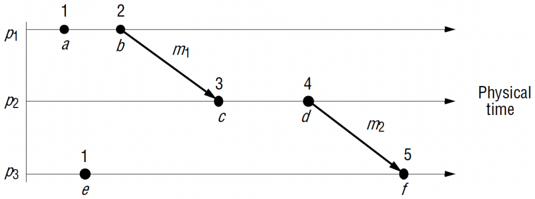
Logical clocks (Lamport timestamps)



- Each process has logical clock.
- LC1: Before any event, increment logical clock.
- LC2-a: On sender side, send message together with timestamp.
- LC2-b: On receiver side,
  - Update to max value of own clock and received clock.
  - Timestamp event *receive*(m) using LC1.

Synchronizing physical clocks

Logical clocks (Lamport timestamps)



- If $e \rightarrow e'$, then $L(e) < L(e')$.
- $L(e) < L(e')$ does not imply causality.
    - $L(e) < L(b)$ but $e||b$.

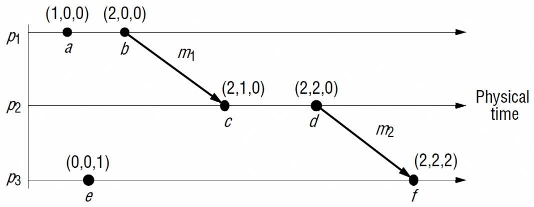
Synchronizing physical clocks

Totally ordered logical clocks

- Lamport timestamps can be identical for two events on two processes (L(a) = L(e) = 1).
- Total order can be achieved using process IDs.
    - Global timestamp (T_i, i).
- $(T_i, i) < (T_j, j)$, if $T_i < T_j$ or $T_i = T_j$ and $i < j$ (no actual significance).

Synchronizing physical clocks

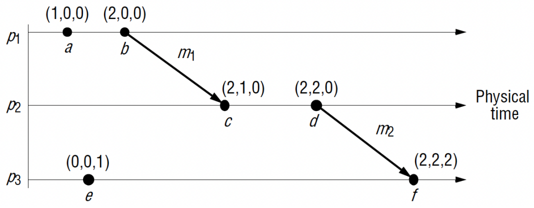
Vector clocks



- Vector clocks improve Lamport's timestamps.
- Vector clock is an array of N integers for a system with N processes.
- Each process maintains a vector.
    - Own event count: $V_i[i]$ is the number of events that p_i has timestamped.
    - Others' event counts: $V_i[j]$ $(j \neq i)$ is the number of events that have occurred at p_j.

Synchronizing physical clocks

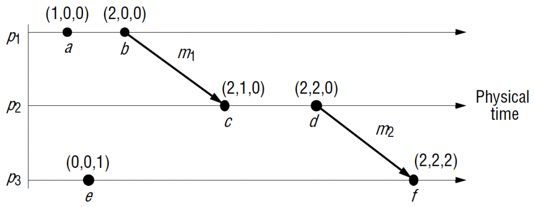
Vector clocks



- Timestamp local events.
  - VC1: Initialize $V_i[j] = 0, \forall i, j = 1, \ldots, N$.
  - VC2: Before p_i timestamps an event, it increments $V_i[i] = V_i[i] + 1$.
  - VC3: p_i includes its vector clock $t = V_i$ in the message it sends.
  - VC4: When p_i receives a message with timestamp t, it updates its vector clock $V_i[j] = max(V_i[j], t)$.

Synchronizing physical clocks

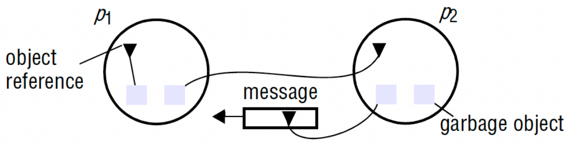
Vector clocks



- $V(x)$: vector timestamp applied by the process at which x occurs.
- Event order rules:
  - All entries of $V(e)$ smaller than $V(e')$: happened-before.
  - All entries of $V(e)$ larger than $V(e')$: happened-after.
  - Otherwise: concurrent events.
- Storage and payload issues.

Global states

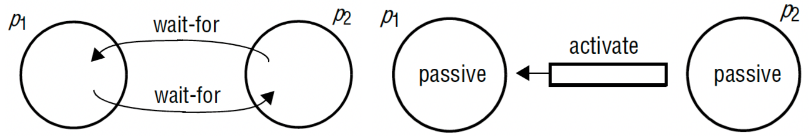
Distributed garbage collection



- Garbage objects lack references.
- Memory can be reclaimed.
- Example with two processes:
    - Process object references.
    - Channel state.

Global states

Distributed deadlock detection, Distributed Termination Detection, Distributed Debugging



- Deadlock: processes wait for each other.
- Detect termination.
- Halting check insufficient.
    - Process may become passive temporarily.
- Termination and deadlock detection similar.
- Debugging is complex in distributed systems.

Global states

Global states and consistent cuts

- Observing individual states possible.
  - Global state determination challenging.
- No global time.
- Creating global state from local:
  - Let each process p_i keep a history of events (e.g. internal actions)

$$history(p_i) = h_i = \langle e_i^0, e_i^1, e_i^2, \ldots \rangle$$

  - Let any finite prefix of the process's history be:

$$h_i^k = \langle e_i^0, e_i^1, \ldots, e_i^k \rangle$$

  - Let state of a p_i before the k-th event be s_i^k.

Global states

Global states and consistent cuts

- Track communication channel.
  - Record message states.
  - Infer message presence.
- The global history: combination of individual histories

$$H = h_1 \bigcup h_2 \bigcup \ldots \bigcup h_N$$

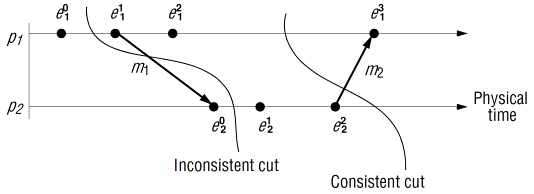
- Global state: a set of individual process states.

$$S = (s_1, s_2, \ldots, s_N)$$

- Global state as prefixes.

Global states

Global states and consistent cuts



- Cut as subset.

$$C = h_1^{c_1} \bigcup h_1^{c_2} \bigcup \dots h_N^{c_N}$$

  - Frontier of a cut: set of last events of processes.
    - Consistent cut: no message received without being sent.
  - Left cut with frontier $\langle e_1^0 e_2^0 \rangle$ inconsistent
  - Right cut with frontier $\langle e_1^2 e_2^2 \rangle$ consistent.

Global states

Global states and consistent cuts a cut to end

- Consistent cut condition: contains all events that happened any event that it contains.

$$f \rightarrow e \Rightarrow f \in C, \forall e \in C$$

- Consistent global state: consistent cut.
- Distributed system operation: transitions between global states.
- Single event at each transition.
- Run: total ordering of events aligning with histories.
- Linearization respects also happened-before relation.

Global states

The ‘snapshot’ algorithm of Chandy and Lamport

- Snapshot algorithm for determining global states.
- Record states of processes and channels.
- Assumptions:
  - No failures in channels or processes.
  - FIFO message delivery.
  - Strongly connected graph.
  - Any process can initiate snapshot
  - Processes remain active during snapshot.

Global states

The ‘snapshot’ algorithm of Chandy and Lamport

- Incoming channels of p_i.
- Outgoing channels of p_i.
- Record state and messages.
  - Account for unreceived messages.
  - Unreceived message state is "in channel".
- Special marker messages.
  - Prompt state saving.
  - Determine channel messages.

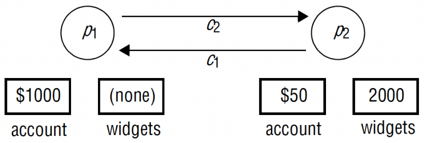
Global states

The ‘snapshot’ algorithm of Chandy and Lamport

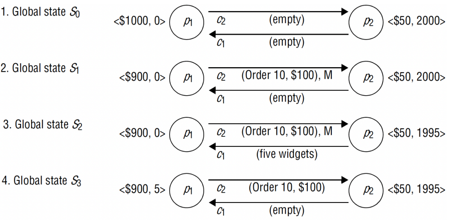
- Two main rules for recording states (i.e. taking snapshots):
    - Marker receiving rule:
        - First marker: Record state, track messages.
        - After snapshot: Record incoming channel state and messages.
    - Marker sending rule: Send marker after recording state.

Global states

The 'snapshot' algorithm of Chandy and Lamport



- Any process starts anytime.
- p_1 and p_2 connected by c_1, c_2.
- p_1 sends 10 widget orders, 10\$/widget and a marker to p_2.
- p_2 sends p_1's previous order of 5 widgets.



(M = marker message)

- p_1 records c_1's state (5 widgets).
- Recorded global state: $p_1 : \langle \$1000, 0 \rangle$; $p2 : \langle \$50, 1995 \rangle$; $c1 : \langle (5 widgets) \rangle$; $c2 : \langle \rangle$.

Discussion topic

A clock is reading 10:27:54.0 (hr:min:sec) when it is discovered to be 4 seconds fast.

Explain why it is undesirable to set it back to the right time at that point

Show (numerically) how it should be adjusted so as to be correct after 8 seconds has elapsed.

Discussion topic

A client attempts to synchronize with a time server. It records the round-trip times and timestamps returned by the server in the table below.

Round-trip (ms)	Time (hr:min:sec)
22	10:54:23.674
25	10:54:25.450
20	10:54:28.342

Which of these times should it use to set its clock? To what time should it set it? Estimate the accuracy of the setting with respect to the server's clock.

If it is known that the time between sending and receiving a message in the system concerned is at least 8 ms, do your answers change?

Discussion topic

An NTP server B receives server A's message at 16:34:23.480 bearing a timestamp 16:34:13.430 and replies to it. A receives the message at 16:34:15.725, bearing B's timestamp 16:34:25.7.

Estimate the offset between B and A and the accuracy of the estimate.