CS412 Time and Global States

- Clocks and concepts of time
- Synchronization
- Global states
Global System Invariants and States

- Correctness of distributed systems frequently depends on the satisfaction of **global system invariants**
  - Examples
    - Absence of deadlocks
    - Write access to a distributed database never granted to more than one process
    - The sum of all account debits and ATM payments in an electronic cash system is zero
    - Objects are only subject to garbage collection when no further reference to them exists
  - Problem: how to observe global states in a distributed system
    - Asynchronous message passing
    - While processes maintain local clocks, impossible to synchronize these perfectly so as to use the local time stamps in order to determine global states.
    - Consequence: there is no process in a distributed system who at any given point in time has a view of the global system state
Clocks

♦ **Distributed Systems**
  ‣ Collection of n independent processes p₁,..,pₙ
    ‣ each process runs in a single thread of concurrency, strictly sequential
    ‣ interactions among pᵢ exclusively through message passing
    ‣ each eᵢ works as state transformer

  ![Diagram](image)

  where eₖ is either a
  • send
  • receive
  • internal action (e.g., variable assignment)

  – sequence of events e = e₁, .. of process pᵢ defines a total order, represented by the local happened before relation →ᵢ
    • e →ᵢ e' iff e happened before e' in pᵢ
  – history hᵢ of pᵢ: sequence of events that occur within pᵢ as ordered by →ᵢ
    hᵢ = <e₀, e₁, ..> for all eₖ in pᵢ
Clocks

**Physical clocks in computers**
- Realised as crystal oscillation counters at the hardware level
- Correspond to counter register \( H(t) \)
- Usually scaled to approximate physical time relative to some meaningful reference event, yielding software clock \( C(t) \)
  - \( C(t) = \alpha H(t) + \beta \)
  - e.g., 64 bit counter counting the number of nanoseconds since a convenience reference time.
  - note that progress of time between two clock queries will differ only if clock resolution is sufficiently smaller than processor cycle time
- Problems
  - **skew**: disagreement in the reading of two clocks
  - **drift**: difference in the rate at which two clocks count the time
    - due to physical differences in the used oscillation crystals, but also heat, humidity, voltage etc.
    - accumulated drift can lead to significant skew
  - clock **drift rate**: difference in precision between a perfect reference clock and a physical clock,
    - usually, \( 10^{-6} \) sec/sec, \( 10^{-7} \) to \( 10^{-8} \) for high precision clocks
Clocks

- **Coordinated Universal Time (UTC)**
  - Atomic Oscillator Clocks
    - Drift rate $10^{-13}$
    - Universally available via radio signal, telephone line or satellite (GPS)
Clocks

♦ Synchronization
  ‣ External synchronization
    – synchronize a process’s clock with an authoritative external reference clock $S(t)$ by limiting its skew to a delay bound $D > 0$
      $|S(t) - C_i(t)| < D$ for all $t$
    – e.g., synchronization with coordinated universal time source
  ‣ Internal synchronization
    – synchronize the local clocks within a distributed system to disagree on not more than a delay bound $D > 0$, without necessarily achieving external synchronization
      $|C_i(t) - C_j(t)| < D$ for all $i, j, t$
  ‣ Obviously, for a system with external synchronization bound of $D$, the internal synchronization is bounded by $2D$. 
Clocks

- **Internal synchronization in synchronous system**
  - synchronous system: known bounds for
    - clock drift rate
    - maximum message transmission delay (\(\text{max}\))
    - time to execute each step of a system
  - otherwise, asynchronous system
- synchronization
  - sender piggybacks its time \(t\) on message \(m\)
  - receiver sets own clock to \(t + T_{\text{trans}}\)
  - problem: how to estimate \(T_{\text{trans}}\)
  - possible to estimate \(\text{min}\): conservative assumptions if no cross traffic interferes
  - let \(u = \text{max} - \text{min}\)
    - if receiver sets its clock to \(t + (\text{max} + \text{min})/2\), then the skew is bounded by \(u/2\)
- useable in an internet setting?
Clocks

♦ Synchronization in systems using external synchronization (after F. Christian)

- Observations
  - round trip times between processes are often reasonably short in practice, yet theoretically unbounded
  - practical estimate possible if round-trip times are sufficiently short in comparison to required accuracy

- Principle
  - use UTC-synchronized time server S
  - process P sends requests to S and measures $T_{\text{round}}$
    - in LAN, $T_{\text{round}}$ should be around 1-10 millisecond
    - during this time, a clock with a $10^{-6}$ sec/sec drift rate varies by at most $10^{-8}$ sec
    - hence the estimate of $T_{\text{round}}$ is reasonably accurate
  - naive estimate: set clock to $t + T_{\text{round}}/2$
Clocks

♦ Synchronization in systems using external synchronization (after F. Christian)
  ‣ Accuracy of estimate?
    ‐ assumption:
      • requests and replies via same network
      • min is either known or can be estimated conservatively
    ‐ calculation
      • earliest time that S can have sent reply: t + min
      • latest time that S can have sent reply: t + T_{round} - min
      • width of range: T_{round} - 2 min
      • accuracy is +/- (T_{round}/2 - min)
  ‣ Discussion
    ‐ really only suitable for deterministic LAN environment or Intranet
    ‐ problem of failure of S
      • -> redundancy through group of servers, multicast requests
The Internet Network Time Protocol (NTP)

- all messages carry timing history information
  - local timestamps of send and receive of previous NTP message
  - local timestamp of send of this message

For each pair of messages \((m, m')\) exchanged between two servers, the following values are being computed (based on the three values carried with the message and the fourth value obtained through local timestamping):

- offset \(o_i\): estimate for the actual offset between two clocks
- delay \(d_i\): total transmission time for the pair of messages
Clocks

The Internet Network Time Protocol (NTP)

- Let $o$ to be the true offset of B’s clock relative to A’s clock, and let $t$ and $t’$ the actual transmission times of $m$ and $m’$
- delay
  \[ T_{i-2} = T_{i-3} + t + o \quad \text{and} \quad T_i = T_{i-1} + t’ - o \]
  which leads to
  \[ d_i = t + t’ = T_{i-2} - T_{i-3} + T_i - T_{i-1} \]
- offset
  \[ o = o_i + (t - t’)/2, \quad \text{where} \quad o_i = (T_{i-2} - T_{i-3} + T_{i-1} - T_i)/2 \]
- estimate and accuracy: using $t$, $t’ \geq 0$ it can be shown that
  \[ o_i - d_i/2 \leq o \leq o_i + d_i/2 \]
  hence, $o_i$ is an estimate for $o$, and $d_i$ is a measure for the estimate’s accuracy
Event Ordering

- Clock synchronization and ordering of events
  - Consider: with achievable internal synchronization of clocks (skew of about $10^{-3}$ sec), how many processor instructions can be executed during that skew?
  - Consequence: Clock synchronization (internal and external) cannot be sufficiently precise in order to use timestamping for the determination of total event orderings in different processes in a distributed system
Lamport’s *happened-before* relation

- Basic observations
  - if two events happen in the same process $p_i$, then they occurred in the order in which $p_i$ observed them (the local happened before relation $\rightarrow_i$)
  - for any message passing, the send event occurs before the receive event

- The HB (happened before) relation $\rightarrow$
  - HB1: for any pair of events $e$ and $e'$, if there is a process $p_i$ such that $e \rightarrow_i e'$, then $e \rightarrow e'$
  - HB2: for any pair of events $e$ and $e'$ and for any message $m$, if $e = \text{send}(m)$ and $e' = \text{receive}(m)$, then $e \rightarrow e'$
  - HB3: if $e$, $e'$ and $e''$ are events and if $e \rightarrow e'$ and $e' \rightarrow e''$, then $e \rightarrow e''$ (HB is identical to its transitive closure)

- HB defines a partial order
Event Ordering

♦ Lamport’s *happened-before* relation

Consequence

- for any tuple \( e \rightarrow e' \)
  - either \( e \) and \( e' \) are direct or indirect successors in the same process, or
  - there are events \( e'' \) and \( e''' \) and a message \( m \) such that \( e \rightarrow e'' \) and \( e''' \rightarrow e' \) and \( e'' = \text{send}(m) \) and \( e''' = \text{receive}(m) \)

- \( e \rightarrow e' \) does not necessarily express causality between \( e \) and \( e' \)

Concurrency

- for all \( e, e' \), \( e \) not \( \rightarrow e' \) and \( e' \) not \( \rightarrow e \), then we say that \( e \) and \( e' \) are concurrent (also written as \( e \parallel e' \))
Logical Clocks

- Lamport’s logical clocks
  - logical clocks: permit inference of the event ordering (HB ordering is captured numerically)
  - monotonically increasing counters that impose a total ordering of the observed events
  - every \( p_i \) maintains logical clock \( L_i \)
  - \( L(e) \) (or \( L_i(e) \)): timestamp of event \( e \) (at process \( i \))
  - messages piggyback the timestamp of the send event
  - rules to update logical clocks and message time stamps
    - LC1: \( L_i \) is incremented before each event at \( p_i \)
    - LC2: a process \( i \) piggybacks \( L_i \) on every message sent
    - LC3: on receiving \( (m, t) \),
      - a process \( p_j \) computes \( L_j := \max(L_j, t) \),
      - increments \( L_j \),
      - and then timestamps receive\( (m, t) \)
Lamport’s logical clocks

- Note
  \[ e \rightarrow e' \Rightarrow L(e) < L(e') \]
  but
  \[ L(e) < L(e') \text{ not } \Rightarrow e \rightarrow e' \]
- Possible to augment partial order of timestamps to total order, for instance by using the process number
  - can be used for instance to determine access of concurrent processes awaiting entry into critical section
Vector Clocks

- Vector Clocks
  - Overcome weakness of Lamport’s logical clocks:
    - array of N integers
    - each process i keeps its vector clock $V_i [1, ..N]$
    - piggybacking of timestamps as in Lamport’s protocol
    - clock update rules
      VC1: Initially, all clocks are 0 on all components
      VC2: i sets $V_i[i] := V_i[i] + 1$ just before timestamping an event
      VC3: i includes $t= V_i$ in every message (piggybacking)
      VC4: i receives a timestamp $t$, then
        $V_i[j] := \max(V_i[j], t[j]), \forall j=1,..,N$ (“merge”)

© Pearson Education 2001
Vector Clocks

- **Observation**
  - $V_{ij}$ is the number of events in $j$ that $i$ has potentially been affected by.
- **Vector timestamp comparison**:
  - $V = V'$ iff $V[j] = V'[j] \ \forall j=1,..,N$
  - $V \leq V'$ iff $V[j] \leq V'[j] \ \forall j=1,..,N$
  - $V < V'$ iff $V \leq V'$ and $V \neq V'$
- **In example**
  - $V(b) < V(d)$
  - $V(e)$ unordered to $V(d)$, i.e., $e \parallel d$
- **Theorem**
  - $e \rightarrow e' \iff V(e) < V(e')$
- **Critique**
  - storage and message overhead proportional to $N$
  - matrix clocks: reduced message overhead through partial vector transmission and local clock estimation
Global States

♦ Problems that would require the view on a global state
  ‣ Distributed deadlock detection: is there a cyclic wait-for-graph among processes and resources in the system?

  - problem: system state changes while we conduct observation, hence we may get an inaccurate observation result
Global States

- Problems that would require the view on a global state
  - Distributed garbage collection: is there any reference to an object left?

- Distributed termination detection: is there either an active process left or is any process activation message in transit?