LTAT.06.007 Distributed Systems
Lecture 6 – Coordination I (Clock primitives)

Huber Flores, PhD
ASSOCIATE PROFESSOR

Tartu, Estonia 23/03/2020
Recap

• Studied the different layers that enable communications in distributed systems
• Analyzed different types of communications, and their primitives
• Learned how applications used different communications
Agenda

• **Goal:** To study the purpose of clocks in distributed systems

• **Content:**
  - The importance of clocks and synchronization
  - Different types of clocks
  - Time protocols
  - Logical clocks
    - Lamport’s logical clocks
    - Vector clocks

After this lecture, you should be able to:

• Understand the synchronization and logical ordering of events in distributed systems
Example: Two Generals’ Problem

<table>
<thead>
<tr>
<th>Two generals need to coordinate an attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Must agree on time to attack</td>
</tr>
<tr>
<td>• They’ll win only if they attack simultaneously</td>
</tr>
<tr>
<td>• Communicate through messengers</td>
</tr>
<tr>
<td>• Messengers may be killed on their way</td>
</tr>
</tbody>
</table>
Example: Two Generals’ Problem
Example: Two Generals’ Problem

attack!
Example: Two Generals’ Problem
Example: Two Generals’ Problem
Example: Failing coordination

Diagram: 1 is planning to march during the night, but 2 is expecting to march during the day, resulting in ambush!
Physical clocks

Universal Coordinated Time (UTC)

- Based on the number of transitions per second of the cesium 133 atom (pretty accurate)
- At present, the real time is taken as the average of some 50 cesium clocks around the world. TAI (International Atomic Time)
- Introduces a leap second from time to time to compensate that days are getting longer.

Note

UTC is broadcast through short-wave radio and satellite. Satellites can give an accuracy of about ±0.5 ms
## Example: Global Positioning System (GPS)

### Characteristics

- 31 satellites now
- Each satellite has up to four atomic clocks, regularly calibrated
- Satellite broadcasts its position and timestamps each message with its local time
  - receiver can determine its location and actual time
  - Requires antenna and signal from at least 4 satellites

### GPS time

- Zero at 0h 6-Jan-1980
- Accuracy ±10 ns
- No leap seconds → 18 seconds ahead of UTC now (always 19 seconds behind TAI)
Synchronization

Essence

• In distributed system multiple processes need to
  ▪ Cooperate on accessing shared resources
  ▪ Agree on the ordering of events

• Synchronization in distributed systems is often much more difficult than in uniprocessor or multiprocessor systems

• Synchronization is critical to avoid blocked resources, ensure data consistency, and schedule tasks.
Synchronization mechanisms

Mechanisms
- Synchronization based on actual time
- Synchronization based on logical time
- Mutual exclusion
- Election
Clock synchronization (Actual time)

Precision
The goal is to keep the deviation between two clocks on any two machines within a specified bound, known as the precision $\pi$:

$$\forall t, \forall p, q : |C_p(t) - C_q(t)| \leq \pi$$

with $C_p(t)$ the computed clock time of machine $p$ at UTC time $t$.

Accuracy
In the case of accuracy, we aim to keep the clock bound to a value $\alpha$

$$\forall t, \forall p : |C_p(t) - t| \leq \alpha$$

Synchronization
- **Internal synchronization**: keep clocks precise
- **External synchronization**: keep clocks accurate
Clock synchronization

Observations

- In a centralized system time is unambiguous
- There is no global clock for a distributed system!

- Computer clocks drift!
  - A cheap quartz has a temperature dependence of 1 ppm/°C or more
  - 1 ppm corresponds to 1 μs per second (1 μs/s results in 1s/12 days)
- Protocol stack and execution environment cause both constant and varying delay components
Clock drift

Clock specifications

- A clock comes specified with its maximum clock drift rate $\rho$.
- $F(t)$ denotes oscillator frequency of the hardware clock at time $t$.
- $F$ is the clock’s ideal (constant) frequency $\rightarrow$ living up to specifications:

$$\forall t : (1 - \rho) \leq \frac{F(t)}{F} \leq (1 + \rho)$$

Observation

By using hardware interrupts we couple a software clock to the hardware clock, and thus also its clock drift rate:

$$C_p(t) = \frac{1}{F} \int_0^t F(t)dt \Rightarrow \frac{dC_p(t)}{dt} = \frac{F(t)}{F}$$

$$\Rightarrow \forall t : 1 - \rho \leq \frac{dC_p(t)}{dt} \leq 1 + \rho$$

Fast, perfect, slow clocks
Network time protocol

Essence (NTP, RFC 1305)

- Based on delay estimates between two servers

- Assuming T2-T1≈T4-T3, then
  
  offset $\theta = T3 - \frac{((T2-T1)+(T4-T3))}{2} = \frac{(T2-T1)+(T3-T4)}{2}$
  
  delay $\delta = \frac{(T2-T1)+(T4-T3)}{2}$

- Eight pairs of ($\theta$, $\delta$) are buffered and minimum of the $\delta$ values is chosen as the estimate of the delay, with the associated $\theta$ as the estimate of the offset
Network time protocol

**Essence**

NTP servers are divided into strata

- When A contacts B, A will only adjust its time if its own stratum level is higher than that of B

- Up to 15 stratum levels

- NTP uses UDP as transport protocol (port 123) both in multicast and unicast modes
Network time protocol

Primitives

- NTP timestamps
  - 64 bits with an epoch of January 1, 1900
    - 32 bits for seconds: 232 seconds (136 years) time scale
    - 32 bits for fractional seconds: 2-32 (0.233 ns) theoretical resolution
  - Future versions will extend representation to 128 bits

- NTPv4 accuracy
  - ~10 ms in public Internet
  - ~200 μs in LAN in ideal conditions

- SNTP (Simple NTP version 4, RFC 2030)
  - Recommended to be used at extremities of synchronization subnet
### Observations (PTP, IEEE 1588)

- Precision clock synchronization protocol for networked measurement and control systems (typically in Ethernet LAN).

- Periodic 4-phase synchronization and timestamp messages allow slave determine precisely delay/offset, and compensate for drift/offset of its own clock accordingly.

- Influence of protocol stack and execution environment can be eliminated by taking timestamps as close to physical layer as possible with hardware assistance.

- Provides us accuracy with HW-assisted timestamps.
Precision time protocol

Observations (PTP, IEEE 1588)

$$O = \text{Offset} = \text{Clock}_{\text{Slave}} - \text{Clock}_{\text{Master}}$$

Measured values $t_1, t_2, t_3, t_4$

$$A = t_1 - t_0 = D + O$$

$$B = t_3 - t_2 = D - O$$

$$\text{Delay } D = \frac{A + B}{2}$$

$$\text{Offset } O = \frac{A - B}{2}$$
The Berkeley algorithm

Definition

- For systems with no access to external accurate timing signal
- Basic operation
  (a) Time daemon asks all other machines for their clock values
  (b) Machines answer
  (c) Time daemon tells everyone how to adjust their clock
Logical clock synchronization (Logical time)

Lamport’s (1990) classical notion of logical time

• For many purposes, it is sufficient that all machines agree on the same time
  ▪ It is not essential that this time also agrees with the real time
  ▪ Emphasis on internal consistency of the logical clocks

• If two processes do not interact, lack of synchronization will not be observable and thus will not cause problems

• **Ordering of events** is needed to avoid ambiguities
## Lamport’s logical clocks

### The Happened-before relationship

- **Happens-before** relation $a \rightarrow b$, “$a$ happens before $b$”
- All processes agree on that first event $a$ occurs, then afterward, event $b$ occurs
- Can be observed in three situations
  1. If $a$ and $b$ are events in the same process, and $a$ occurs before $b$, then $a \rightarrow b$ is true
  2. 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 $a \rightarrow b$ is also true
  3. Transitive: if $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$
Lamport’s logical clocks

**Problem**
How do we maintain a global view on the system’s behavior that is consistent with the happened-before relation?

**All processes agree on time** $C(e)$ **assigned for every event** $e$

- C must always go forward (increase), never backward (decrease)

- If $a \rightarrow b$ ($a$ happens before $b$ in the same process), then $C(a) < C(b)$

- If $a$ and $b$ represent the sending and receiving of a message, respectively, then $C(a) < C(b)$

- For all distinctive events $a$ and $b$, $C(a) \neq C(b)$
Lamport’s logical clocks

Implementation

• Each process \( P_i \) maintains local counter \( C_i \)
  1. \( P_i \) executes \( C_i \leftarrow C_i + 1 \) before executing an event (e.g. sending a message over the network, delivering a message to application)
  2. When sending message \( m \) to \( P_j \), \( P_i \) sets m’s timestamp \( \text{ts}(m) = C_i \) (after having executed step 1)
  3. Upon receiving message \( m \), \( P_j \) adjusts its own local counter as \( C_j \leftarrow \max\{C_j, \text{ts}(m)\} \), after which it executes step 1 and delivers the message to application

• Total order by taking into account process ID
  \( (C_i, i) < (C_j, j) \) if \((C_i < C_j \text{ or } (C_i = C_j \text{ and } i < j)) \)
Lamport’s logical clocks

Observations
(a) Three processes each with clocks running at different rates
(b) Lamport’s algorithm corrects the clocks
Lamport’s logical clocks

where is implemented

Application layer

Application sends message

Adjust local clock and timestamp message

Middleware sends message

Middleware layer

Message is delivered to application

Adjust local clock

Message is received

Network layer
Example: Totally-ordered multicasting

Problem

Updating a replicated database and leaving it in an inconsistent state
- P1 adds $100 to an account (initial value: $1000)
- P2 increments account by 1%
- There are two replicas

In absence of proper synchronization:
replica #1 ← $1111, while replica #2 ← $1110
Solution: totally ordered-multicast

• Each message is multicast, with timestamp=current (logical) time
  ▪ Recipient ACKs each message, via multicast
• Each process puts received messages in its local queue, sorted according to the timestamp
  ▪ All processes will eventually have the same copy of the local queue
  ▪ Timestamps reflect a consistent global ordering of events!
• A process delivers a message only when it is at the head and it has been ACKed by all processes
  ▪ All messages are delivered in the same order everywhere
### Analogy with total-ordered multicast

- With total-ordered multicast, all processes build identical queues, delivering messages in the same order.
- Mutual exclusion is about agreeing in which order processes are allowed to enter a critical section.
Casual dependency

Precedence vs. dependency

- We say that a causally precedes b.
- b may causally depend on a, as there may be information from a that is propagated into b.
Lamport’s logical clocks

**Essence**

Lamport clocks do not guarantee **causality** of events

- FIFO delivery ≠ causal delivery
- $f > e$, although $a < b$
Example

Message

Instruction or step

34
Example

- $a \rightarrow b$
- $b \rightarrow f$
- $a \rightarrow f$
Example

\[ h \rightarrow g \]
\[ f \rightarrow j \]
\[ h \rightarrow j \]
\[ c \rightarrow j \]
Example

How to assign logical (Lamport) timestamps to each event?
Example
Example

Message send event

Message carries ts= 1
Example

\[
\begin{align*}
\text{ts} &= \max(\text{local}, \text{msg}) + 1 \\
&= \max(0, 1) + 1 \\
&= 2
\end{align*}
\]

Message carries \( \text{ts} = 1 \)
Example

Message carries $ts = 2$

$\text{Max}(2, 2) + 1 = 3$
Example

Max(3, 4) + 1 = 5
Example

Max(2, 6) + 1 = 7
Example
Example

✓ a → b :: 1 < 2
✓ b → f :: 2 < 3
✓ a → f :: 1 < 3
Example

✓ h → g :: 1 < 4
✓ f → j :: 3 < 7
✓ h → j :: 1 < 7
✓ c → j :: 3 < 7
Example

- $c (?) \rightarrow f (?) :: 3 = 3$
- $h (?) \rightarrow c (?) :: 1 < 3$

$(c, f)$ and $(h, c)$ are pairs of concurrent events
Vector clocks

**Essence**

- Capture causality: if $\text{VC}(a) < \text{VC}(b)$, then event $a$ is known to causally precede event $b$

- Each process $\text{Pi}$ maintains a vector clock $\text{VCi}$ (N numbers, one per process)
  - $\text{VCi}[i]$ is the number of events that have occurred at $\text{Pi}$
  - If $\text{VCi}[j] = k$ ($j \neq i$) then $\text{Pi}$ knows that $k$ events have occurred at $\text{Pj}$

- Maintenance
  1. $\text{Pi}$ executes $\text{VCi}[i] \leftarrow \text{VCi}[i] + 1$ before executing an event
  2. When $\text{Pi}$ sends message $m$ to $\text{Pj}$, it sets $m$’s timestamp $\text{ts}(m)$ equal to $\text{VCi}$ (after having executed step 1)
  3. When $\text{Pj}$ receives $m$, it executes $\text{VCj}[k] \leftarrow \max\{\text{VCj}[k], \text{ts}(m)[k]\}$ for each $k$, after which it executes step 1 and delivers the message to the application
Vector clocks

Enforcing causal communication with causally ordered multicasting

• Clocks are only adjusted when sending and receiving messages
  ▪ Upon sending a message, process Pi will only increment VC[i][i] by 1
  ▪ When receiving message m with timestamp ts(m), Pi only adjusts VC[i][k] to max{VC[j][k], ts(m)[k]} for each k

• When Pj receives message m from Pi with timestamp ts(m), the delivery of the message to the application layer is delayed until:
  ▪ ts(m)[i] = VC[j][i]+1
    m is the next message Pj was expecting from Pi
  ▪ ts(m)[k] ≤ VC[j][k] for all k≠i
    Pj has seen all messages that have been seen by Pi when it sent m
Vector clocks

Essence

\[ VC_0 = (1,0,0) \quad VC_0 = (1,1,0) \]

\[ VC_1 = (1,1,0) \]

\[ VC_2 = (0,0,0) \quad VC_2 = (1,0,0) \]
Example
Example
Example
Example
Example

Message (2,0,0)

(0,0,0) (1,0,0) (2,0,0)

(0,0,1) (0,1,1) (2,2,1)
Example

Diagram showing points a, b, c, d, e, f, g, h, i, j in 3D space with time progression.
Example

- $a \rightarrow b :: (1,0,0) < (2,0,0)$
- $b \rightarrow f :: (2,0,0) < (2,2,1)$
- $a \rightarrow f :: (1,0,0) < (2,2,1)$
- $h \rightarrow g :: (0,0,1) < (2,3,1)$
- $f \rightarrow j :: (2,2,1) < (5,3,3)$
- $h \rightarrow j :: (0,0,1) < (5,3,3)$
- $c \rightarrow j :: (3,0,0) < (5,3,3)$

Diagram:

- Points:
  - $(0,0,0)$
  - $(1,0,0)$
  - $(2,0,0)$
  - $(3,0,0)$
  - $(4,3,1)$
  - $(5,3,1)$
  - $(0,0,1)$
  - $(2,2,1)$
  - $(2,3,1)$
  - $(5,3,3)$

- Arrows:
  - $a \rightarrow b$
  - $b \rightarrow f$
  - $a \rightarrow f$
  - $h \rightarrow g$
  - $f \rightarrow j$
  - $h \rightarrow j$
  - $c \rightarrow j$
Example

- $c(?) \& f(?) :: (3,0,0) \lll (2,2,1)$
- $h(?) \& c(?) :: (0,0,1) \lll (3,0,0)$

Ill Concurrent
Issues on ordered message delivery

Observations

• Only potential causality is captured
  ▪ Middleware treats all messages from a sender causally related even though they may not be, which can lead to performance problems

• External communication (hidden channels)
  ▪ Data can flow in ways other than message passing!
  ▪ Example

  ▪ Event a: pipe rapture, detected by sensor #1
  ▪ Event b: pressure drop, detected by sensor #2
  ▪ Event c: controller (p3) increases heat (to increase pressure), then receives notification of pipe rapture (event d)
  ▪ The pipe acts as a hidden communication channel
Summary

What we learned?

• The importance of clocks in distributed systems
• The goal of synchronization and how to achieve it
• Internal and external clocks
• Approaches to synchronize (distributed) computer systems
References

- Indranil Gupta, Cloud computing Concepts, University of Illinois at Urbana-Champaign
Next lecture

Coordination II
Questions?

E-mail: huber.flores@ut.ee