Recap

• Code Migration using java
  • Java Reflection Utility
  • Sockets
  • Code Offloading
Agenda

• **Goal:** Calculate timestamps in Vector Clock
• **Content:**
  - Happens before relation
  - Need for Vector Clocks
  - Causality Relation
  - Rules for calculating timestamps in vector clocks
  - Comparison of vector clock timestamps

• Quiz

After this lecture, you should be able to:
• Understand how to calculate vector clock timestamps
Description

• The use of Vector clocks is an attempt to move away from the use of a physical clock for measuring ordering of events in a Distributed System.

Observation

Instructions to complete this practical session can be found in the course website: https://https://courses.cs.ut.ee/2021/ds/spring/Main/Instructions2
Vector Clocks

Definition:

- The vector clocks provide a sort of a Logical Clock that assigns a unique id to events in a Distributed System, such that it captures the casual order among events in a Distributed System.
- The Distributed System can be viewed as a system where series of processes are executed. These Processes communicate with each other via messages.
The Happens before Relation (<)

- If $a$, $b$ are events in the same process, if $a$ occurs before $b$, then $a < b$
- If $a$ denotes sending of message in process-$a$ and $b$ denotes receipt of message in process-$b$ then $a < b$
- The relation is transitive, $a < b$ and $b < c => a < c$
Need for Vector Clocks

Eliminate limitations set by Lamport's Logical Clock:

- Looking at the Lamport's timestamps, we can not detect Casual Relationship in Lamport's logical clock and we can not conclude which events are casually related.

- Lamport's Logical Clock is a Partial Order of events.
Need for Vector Clocks

• Consider the example:
  - if A -> B, then it is true that TS(A) < TS(B)
  - But if TS(A) < TS(B), then we can not conclude that A -> B (Not true for Concurrent Events)
  - Even though C(e11) < C(e32), we can not say whether e11 happened before e32 or not. In other words, we cannot establish the casual relationship between these two events.
  - The solution to this problem is to make use of Vector Clock
Casuality Relation:

• It states: *If an event e1 possibly influences the generation of event e2, then e1 < e2*

• The better way to establish such a relation is to rely on **message-passing** among the processes.

• Message passing establishes a **happens before order** on events based on the communication pattern.
Rules for Vector Clocks (Algorithm)

- Initially, all clocks are set to 0
- $C_i[i] = C_i[i] + 1$, for two successive internal events at $P_i$
- At Process $P_i$, $C_i[j] = \text{Max}(C_i[j], \text{tm}[j])$, for all processes $P_j$, where $\text{tm}$ is the vector timestamp borne by process $m$ received by process $P_i$ from some other process.
Vector Clocks (Example)
Comparison of Vector clock timestamps:

- **Equal**: $T_a = T_b$, iff for all processes $P_i$, $T_a[i] = T_b[i]$
  - For instance, $(2,2,2) = (2,2,2)$

- **Not Equal**: $T_a \neq T_b$, iff $T_a[i] \neq T_b[i]$ for at least one $P_i$
  - For instance, $(2,1,2) \neq (2,2,2)$

- **Less than or Equal**: $T_a \leq T_b$, iff $T_a[i] \leq T_b[i]$ for all $P_i$
  - For instance, $(2,1,2) \leq (2,2,2)$

- **Less than**: $T_a < T_b$, iff
  - for all $P_i$, $T_a[i] \leq T_b[i]$ and
  - there exists at least one $P_i$, for which $T_a < T_b$
  - For instance, $(2,1,2) < (2,2,2)$

- **Concurrent**: $T_a \parallel T_b$, iff $T_a !< T_b$ and $T_b !< T_a$
Session Instructions at Course Page
Quiz

Content

• Lecture 6 (Clock synchronization for distributed processes)
• Two attempts
  ▪ One in Seminar Session
  ▪ Next available until Monday 23:50 (Deadline)
• Open Quiz in Moodle
• Total Quiz Points = 100

Observation

Quiz review is available after the quiz is closed
Questions?

E-mail: farooq.ayoub.dar@ut.ee