Distributed Information Processing

2nd Lecture

Eom, Hyeonsang (엄현상)
Department of Computer Science & Engineering
Seoul National University

©Copyrights 2012 Eom, Hyeonsang All Rights Reserved
Outline

- Clock
  - Logical Clocks & Ordering of Events
  - Vector Clocks
- Q&A
Partial Ordering of Events

Assumptions

- System Consists of N Processes:
  \[ p_i \quad (i = 1, \ldots, N) \]
- System Is Composed of a Collection of Events

  - Examples of events
    - Execution of a subprogram
    - Execution of a single machine instruction
    - Sending or receiving a message

\[
\text{history}(p_i) = h_i = \left\langle e_i^k \right\rangle \quad (k = 1, \ldots) \\ (i = 1, \ldots, N)
\]
Partial Ordering of Events
[Lamport78]

“Happened Before” Relation (→)

1. $k < l \implies (if) \quad e^k_i \rightarrow e^l_i \quad (i = 1, \ldots, N)$
2. $e_i = \text{send}(m), e_j = \text{receive}(m)$ for Message $m$
   \implies \quad e_i \rightarrow e_j$
3. $e \rightarrow e' \land e' \rightarrow e'' \implies e' \rightarrow e''$
   * $e \rightarrow e$

Concurrent Relation (||)

\[ e \rightarrow e^* \land e^* \rightarrow e \implies e \parallel e^* \]
Logical Clocks [Lamport78]

- Ways of Assigning Numbers to Events
  
  \( LC_i(e) \), where \( e \) is an event in process \( p_i \)

- Clock (Correctness) Condition

  \[ e \to e^* \implies LC(e) < LC(e^*) \]

- Satisfied if the following two conditions hold:

  1. \( k < l \implies LC(e_i^k) < LC(e_i^l) \) \( (i = 1, ..., N) \)
  2. \( e_i = \text{send}(m), e_j = \text{receive}(m) \) for Message \( m \)

  \[ \implies LC(e_i) < LC(e_j) \]
Logical Clocks (Cont’d) [Lamport 78]

Implementation Rules

(1) $LC(e_i^{k+1}) = LC(e_i^k) + 1$ for Successive Events $e_i^k, e_i^{k+1}$ ($i = 1, ..., N$)

(2) $e_i = \text{send}(m), e_j^l = \text{receive}(m)$ for Message $m$

\[ \Rightarrow \quad t = LC(e_i) \in m, \quad LC(e_j^l) = \max\{LC(e_j^{l-1}), t\} + 1 \]
Ordering the Events Totally
[Lamport78]

**Total Ordering** (⇒_{t.o.})

\[ e \Rightarrow_{t.o.} e^* \iff LC_i(e) < LC_j(e^*) \lor \{LC_i(e) = LC_j(e^*) \land i < j\} \]

\[ e \rightarrow e^* \Rightarrow e \Rightarrow_{t.o.} e^* \]

- **Use Case: Mutual Exclusion Problem**
  - Processes sharing a single resource
    1. A process using it must release it before the following use
    2. The processes must use it in their request order
    3. If it is eventually released, every request is eventually granted
Ordering the Events Totally (Cont’d) [Lamport78]

Nontrivial Scheduling Problem

- The Requests Cannot Be Granted in the Order They Are Received

Scheduler

This Earlier Request Is Received Later
Ordering the Events Totally (Cont’d)

- Nontrivial

- Assumptions for an Algorithm
  - In-order message delivery
  - Each process’ request queue
    - Initially containing a request with T0:P0

- Algorithm
  - Pi sends a Tm:Pi Req to all others and puts it in the queue
    - Each receiver puts it in the queue and sends a timestamped Ack to the requestor
  - When releasing the resource, Pi removes the Req and sends a timestamped Rel to all others
    - Each receiver removes Pi’s Req from its queue
  - The Tm:Pi Req is granted if it is the first in ⇒, and Pi has received a message from all others timestamped later than Tm

(1) A process using it must release it before the following use
(2) The processes must use it in their request order
(3) If it is eventually released, every request is eventually granted

Smallest Time Value: Scheduler ID

Send Time
Vector Clocks

Shortcomings of Lamport’s Logical Clock

\[ LC_i(e) < LC_j(e^*) \implies e \rightarrow e^* \]
Vector Clocks (Cont’d)

**Implementation Rules**

1. \( VC_i[j] = 0 \) for \( i, j = 1, \ldots, N \)

2. \( VC_i(e_i^{k+1})[i] = VC_i(e_i^k)[i] + 1 \) for Successive Events \( e_i^k, e_i^{k+1} \) for \( i = 1, \ldots, N \)

3. \( e_i = \text{send}(m), \ e_j^l = \text{receive}(m) \) for Message \( m \)

\( \Rightarrow \ t = VC_i(e_i) \in m, \ VC_j(e_j^l) = \max\{ VC_j(e_j^{l-1}), t \} + j \ 1 \)
Vector Clocks (Cont’d)

- Vector-Timestamp Comparisons

Vector Clocks ($VC$) are compared using the following relations:

- $VC = VC^*$ if and only if $VC[j] = VC^*[j]$ ($j = 1, ..., N$)
- $VC \leq VC^*$ if and only if $VC[j] \leq VC^*[j]$ ($j = 1, ..., N$)
- $VC < VC^*$ if and only if $VC \leq VC^* \land VC \neq VC^*$

- Properties

For any event $e$, the following holds:

$$e \rightarrow e^* \text{ iff } VC(e) < VC(e^*)$$

- Hints for the Proof

- $e \parallel e^* \Rightarrow \neg\{VC(e) \leq VC(e^*) \lor VC(e) \geq VC(e^*)\}$

(Note That $VC_i[j] \leq VC_j[j]$)

- Strong Clock Condition
Vector Clocks

Properties

\[ e_i \rightarrow e_j \text{ iff } VC_i[i] \leq VC_j[i], \text{ where } i \neq j \]

Hint for the Proof

(Note That \( VC_i[j] \leq VC_j[j] \))

\[ \exists k(\neq j) \text{ s.t. } VC_i[k] < VC_j[k] \]

\[ \Rightarrow \exists e_k \text{ s.t. } (e_k \leftrightarrow e_i) \land (e_k \rightarrow e_j) \]
Vector Clocks (Cont’d)

Properties

\[ \exists k (\neq j) \text{ s.t. } VC_i(e_i)[k] < VC_j(e_j)[k] \]
\[ \Rightarrow \exists e_k (\neq e_i) \text{ s.t. } (e_k \rightarrow e_i) \land (e_k \rightarrow e_j) \]

Suppose That \( i = k \),

\[ \exists k (\neq j) \text{ s.t. } VC_k(e_k^*)[k] < VC_j(e_j)[k] \]
\[ \Rightarrow \exists e_k (\neq e_k^*) \text{ s.t. } (e_k \rightarrow e_k^*) \land (e_k \rightarrow e_j) \]

iff \[ \exists e_k (\neq e_k^*) \text{ s.t. } e_k^* \rightarrow e_k \rightarrow e_j \]

\[ \exists e_k (\neq e_k^* \land k \neq j) \text{ s.t. } e_k^* \rightarrow e_k \rightarrow e_j \]
\[ \Rightarrow VC_k(e_k^*)[k] < VC_j(e_j)[k] \]
Vector Clocks (Cont’d)

**Properties**

\[ \forall (k \neq j), \ VC_k(e_k^*)[k] \geq VC_j(e_j)[k] \]

\[ \Rightarrow \neg \{ \exists e_k(\neq e_k^*) \} \ s.t. \ e_k^* \rightarrow e_k \rightarrow e_j \]

- **If the Monitoring Process Keeps the Value of the k-th Elt of VC from Pk, It Can Decide Whether There Is No Event That Happened before a Given Event**