CSCI 5673
Distributed Systems

Lecture Set One

Event Ordering

Lecture Notes by
Shivakant Mishra
Computer Science, CU Boulder
Last update: January 17, 2017

Event Ordering

- A distributed program consists of two or more processes
- Execution of a process is characterized by a sequence of events

  read x, send y, add 1 to x, write x, …
- Ordering of events in a distributed system is only *partial*

- Partial order
  - Given two events, *a* and *b*, either *a* temporally precedes *b*, or *b* precedes *a*, or *temporal ordering between a and b cannot be determined*
  - Partial order among events can be illustrated via a tree

<Example tree>

Happened Before Relation

<u>Reference</u>: Lamport, L. "*Time, Clocks, and the Ordering of Events in a Distributed System*". CACM 21(7), 1977.

Happened before relation (→) captures the causal relation between events

a → *b* iff

1. *a* and *b* occur in the same process and *a* occurs before *b* based on the process's local clock, or
2. *a* is a send event and *b* is the corresponding receive event, or
3. there exists a third event *c*, such that *a* → *c* and *c* → *b*

- Causally related events: *a* causally affects *b*
  - e.g. write followed by read
- Happened before relation captures *potential* causality

  a → *b*: Event *a* has *potentially* caused or impacted the outcome of event *b*
- Concurrent events: Events *a* and *b* are said to be concurrent if

  a → *b* is false and

  b → *a* is false

Logical clock

- Based on happened before relation, a logical clock *C* can be constructed
  - For every event *a* in a distributed system, *C* assigns a timestamp *C*(*a*) such that

    For any two events *a* and *b*, if $a \rightarrow b$, then $C(a) < C(b)$

Logical Clock: Implementation

- C is comprised of several counters, C_i , one for each process i, such that

$C(a) = C_i(a)$, where event a occurred on process i

Logical Clock: Implementation

- Each process *i* maintains a counter C_i
- C_i is initialized to 0
- $C_i(a)$ = the value of C_i when *a* occurs on *i*
- C_i is incremented between any two successive events that occur on *i*
- Process *i* sends a message *m* to process *j*
  - *a* is the send event and *b* is the corresponding receive event
  - t_m : Timestamp of message m
  - $t_m = C_i(a)$
  - On receiving *m*, *j* updates C_j as follows:

$$C_j = \max(C_j, t_m + 1)$$

Example

Logical Clock

- How are logical clock and happened before relation related?
- If $a \rightarrow b$ then $C(a) < C(b)$
- If $C(a) < C(b)$ then ???

  $b \rightarrow a$ is false

- If a and b are concurrent, then $C(a)$? $C(b)$

  $C(a) < C(b)$ <u>or</u> $C(b) < C(a)$ <u>or</u> $C(a) == C(b)$

- Logical clock cannot be used to recognize concurrent events

Total Order

- Given any two events *a* and *b*, either *b* follows *a* or *a* follows *b*

  Why do we need total order?

- Logical clock can be used to implement a total order (=>)among all events in a distributed system
- This total order is consistent with the partial order among all events

  If $a \rightarrow b$ then $a \Rightarrow b$

- How ???

Recap …

- There is only a partial ordering among events in a distributed system
- Happened before relation: $a \rightarrow b$ iff
  1. *a* and *b* occur in the same process and *a* occurs before *b* based on the process's local clock, or
  2. *a* is a send event and *b* is the corresponding receive event, or
  3. there exists an event *c*: $a \rightarrow c$ and $c \rightarrow b$
- Logical clock
  - If $a \rightarrow b$ then $C(a) < C(b)$
  - If $C(a) < C(b)$ then b $\rightarrow$ a is false
  - Logical clock cannot recognize concurrent events
- Total order: Given any two events *a* and *b*, either *b* follows *a* or *a* follows *b*

Total Order

- Total order can be constructed using logical clock
- $a \Rightarrow b$ iff

  (1) $C(a) < C(b)$, or

  (2) If $C(a) == C(b)$, ???

  a occurred on i, b occurred on j, and $i < j$

<u>Important:</u> Total order constructed using logical clocks is artificial

Vector Clock

- Logical clock cannot recognize concurrent events
- Vector clock *V* can recognize concurrent events

  $a \rightarrow b$ iff $V(a) < V(b)$
- Implementation: Ideas???

Vector Clock

- References:

(1) Fidge, J. "*Timestamps in Message Passing Systems that Preserve the Partial Ordering*". In the *11th Australian Computer Science Conference*, 10(1), 1988.

(2) Mattern, F. "*Virtual Time and Global States of Distributed Systems*". *Parallel and Distributed Algorithms*, Elsevier Science, 1989.

(3) Strom, R.E. and Yemini, S. "*Optimistic Recovery in Distributed Systems*". ACM Transactions on Computer Systems, 3(3), 1985.

Vector Clock: Implementation

- Let n be the total number of processes in a distributed system
- Each process i maintains a vector C_i of size n
- <u>Notation</u>
  - $C_i[\ j\ ]$: j^{th} entry of C_i
- Initially, for all i, j: $C_i[\ j\ ] = 0$

Vector Clock: Implementation

- Comparison of vectors, *v1* and v2

  $v1 = v2$ iff for all i, $v1[\ i\ ] = v2[\ i\ ]$

  $v1 \leq v2$ iff for all i, $v1[\ i\ ] \leq v2[\ i\ ]$

  $v1 \neq v2$ iff there exists i, $v1[\ i\ ] \neq v2[\ i\ ]$

  $v1 < v2$ iff $v1 \leq v2$ and $v1 \neq v2$

Vector Clock: Implementation

- $V(a) = C_i(a)$: a occurred on i
- $C_i[\ i\ ]$ is incremented between any two successive events on i
- Process i sends a message m to j
  - t_m (timestamp of message m) = $V(a)$: a is the send event
  - On receiving m, j updates C_j as follows

    For all k, $C_j[\ k\ ] = \max(C_j[\ k\ ], t_m[\ k\ ])$

Vector Clock: Implementation

- Observations
  - $C_i[\ i\ ]$ is incremented before any event on i
    - Guarantees total order among all events that occur on the same process
  - On receiving m, j updates C_j as follows

    For all k, $C_j[\ k\ ] = \max(C_j[\ k\ ], t_m[\ k\ ])$
    - Guarantees $V(a) < V(b)$, where a is a send event and b is the corresponding receive event

$$\boxed{\text{If } a \rightarrow b \text{ then } V(a) < V(b)}$$

Vector Clock: Implementation

- What about concurrent events?
  - If events *a* and *b* are concurrent
    - Suppose *a* occurs on *i* and *b* occurs on *j*

      $V(b)[\ i\ ] < V(a)[\ i\ ]$ and $V(a)[\ j\ ] < V(b)[\ j\ ]$

If *a* and *b* are concurrent, then

$V(a) < V(b)$ is false

$V(b) < V(a)$ is false

Vector Clock

- $a \rightarrow b$ iff $V(a) < V(b)$

<Example>

Logical/Vector vs. physical clocks

- Logical and vector clocks keep track of event ordering
- Useful to have the system keep good *real* time
  - Useful in building distributed real-time services

Logical or vector clocks cannot be used to track real time