Distributed Systems - Theoretical Foundations

CS239
Advanced Operating Systems
October 26, 2000
Dr. Richard Guy

Lecture Source Materials

Singhal, Ch. 5: **Theoretical Foundations**

Lamport: **Time, Clocks, and the Ordering of Events in a Distributed System,** CACM 21:7, July 1978, 558-565.

Chandy & Lamport: **Distributed Snapshots: Determining Global States of Distributed Systems**, ACM TOCS 3:1, Feb. `85,63-75.

Outline

- Motivation
- Model
- Inherent limitations of distributed system
- Virtual clocks
- Causal ordering of messages
- Global state

Motivation

- What are the final variable values after executing this ANSI C code fragment:

```
int i, j;
i = 5;
j = i * i;
```

- What is your reasoning?

Motivation, continued

- What are the final variable values after executing this ANSI C code fragment:

```
int i, a[2];
i = 0, a[0] = 2, a[1] = 3;
a[i] = i++;
```

- What is your reasoning?

Distributed System Model

- collection of spatially separated processors
- no common (shared) memory
- processes communicate via messages over communications channels; arbitrary delay

Inherent limitations in a DS

- No global clock
- No shared memory

No global clock

- therefore, **no global time**
- isn’t common clock adequate?
- how about synchronized clocks?

Common clocks

- Suppose all processes consult single clock (which exists at exactly one site)
- distinct remote processes observe identical clock value at *different* physical times, due to arbitrary message delays.
- may be incorrect to associate distinct physical times as same physical time.

Synchronized clocks

- Suppose one physical clock per processor
- Suppose very good synchronization method
- Clocks drift, differently
- Same problem as common clocks: distinct instants of physical time appear to be same

No shared memory

- Definitions
- Key question
- Example problem

Definitions

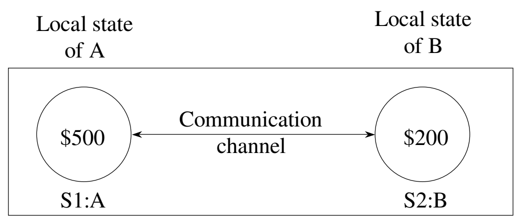
- ***coherent view*** is collection of observations of process state taken at same instant of physical time.
- ***incoherent view*** includes different time.
- ***complete view*** is state of all processes and messages in transit.
- complete view is same as ***global state***

Key question

- Is it possible to obtain a coherent global state of a distributed system, given the absence of a global clock, no shared memory, and the existence of arbitrary delay communication channels?

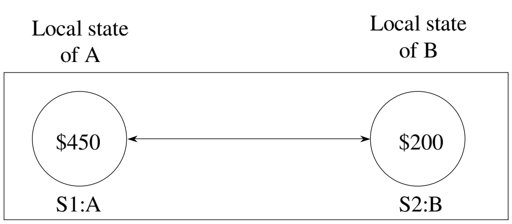
Example problem

Want to compute net balance; need global state?



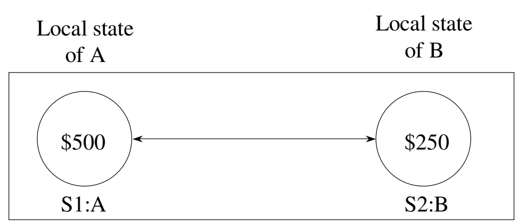
Incoherent View #1

Suppose S1 transfers $50 to from A to B



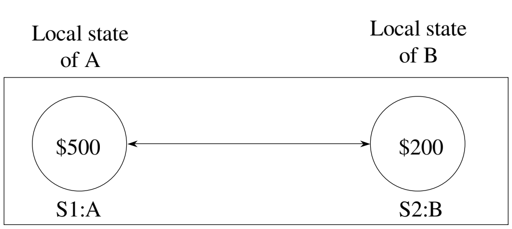
Incoherent View #2

Suppose S1 transfers $50 to from A to B



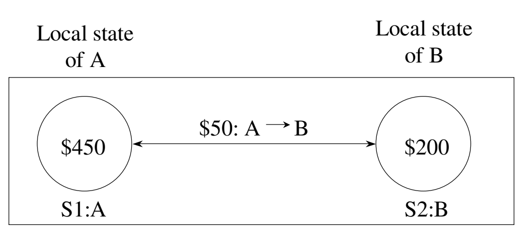
Coherent View #1

Suppose S1 transfers $50 to from A to B



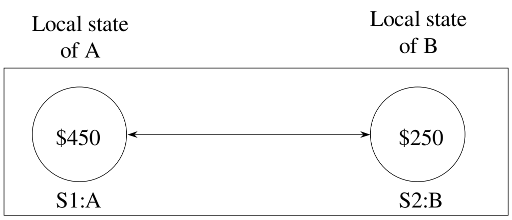
Coherent View #2

Suppose S1 transfers $50 to from A to B



Coherent View #3

Suppose S1 transfers $50 to from A to B



Observation

- must have some way of coordinating activity of processes on distinct processors, to overcome no global clock and no shared memory and arbitrary delay channels.
- solutions limited to same constraints
  - no global clock
  - no shared memory
  - arbitrary delay channels

"Virtual clock" solutions

- Lamport's logical clocks
  - limitations of Lamport's clocks
- Fidge/Mattern vector clocks

Logical clocks

- Model and definitions
- System of clocks

Model

- Process execution is sequence of "events"
- Event is arbitrary granularity
  - procedure call; single instruction
- Sending a message is single event
- Receiving a message is single event
- Event order not determinable by local time
- Order may be possible to determine by examining behavior of computation

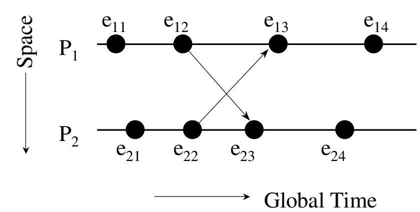
"Happened before" relation

- captures causal dependencies between events
- relation $\rightarrow$ is defined as follows:
  - $a \rightarrow b$ if a and b are events in the same process and a occurred before b
  - $a \rightarrow b$, if a is the event of sending a message m in a process and b is the event of receiving m in another process
  - if $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$ (transitive)

Happened before, continued

- Causally related events
  - Event *a* causally affects *b* if $a \rightarrow b$
- Concurrent events
  - Two distinct events *a* and *b* are concurrent (denoted $a \| b$) if $a \not\rightarrow b$ and $b \not\rightarrow a$. (concurrent events do not causally affect each other)
- For any two events *a* and *b* in a system, either $a \rightarrow b$, $b \rightarrow a$, or $a \| b$.

Space-time diagram



Practical realization of →

- System of logical clocks, C_i
  - one clock C_i at each process P_i in system
  - C_i assigns number $C_i(a)$ to any event *a* at P_i
    - timestamp of event *a*
  - values of C_i have no relation to physical time
  - values of C_i are monotonically increasing
  - implementable by simple counters

Practical realization, continued

- System of clocks satisfies:
  - if $a \rightarrow b$, then $C(a) < C(b)$
- Can use logical clocks to realize → if:
  - Correctness conditions, C1 and C2
  - Implementation rules, IR1 and IR2

Correctness conditions

- **[C1]** For any two events *a* and *b* in process P_i, if *a* occurs before *b*, then

  $C_i(a) < C_i(b)$
- **[C2]** If *a* is the event of sending a message *m* in process P_i and *b* is the event of receiving m in process P_j, then

  $C_i(a) < C_j(b)$

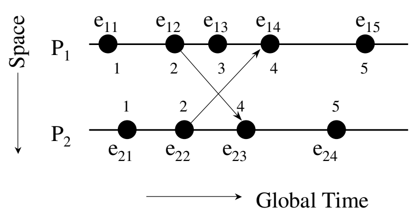
Implementation rules

- **[IR1]** Clock C_i is incremented between any two successive events in process P_i

  $C_i := C_i + d \qquad (d > 0)$
- **[IR2]** If *a* is the event of sending a message *m* by process P_i, then *m* is assigned a timestamp $t_m = C_i(a)$ (after applying IR1). When *m* is received by process P_j, then C_j is

  $C_j := \max(C_j, t_m) + d \quad (d > 0)$

Space-time diagram



Limitations of Lamport's clock

- If $a \rightarrow b$, then $C(a) < C(b)$
- Is the reverse true?
  - if $C(a) < C(b)$, is $a \rightarrow b$ always true? NO!
- Lamport's clock cannot distinguish between clock advancement from local vs remote events
- Hard to reason based on event timestamps

Limitations, continued

- How to overcome Lamport limitations?
- Attempt to capture "source" of clock ticks
- "Vector clocks" to the rescue!

Vector clocks

- Similar to logical clocks, but with vector rather than scalar values.
- Vector is length *n*, *n* is number of processes
  - $C_i[i]$, the ith entry of C_i, corresponds to P_i's logical time
  - $C_i[j]$, $j \neq i$ is P_i's best guess of P_j's logical time
    - $C_i[j]$ is time of last event at P_j which "happened before" the current time at P_i.

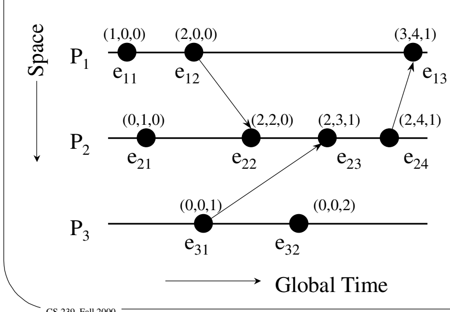
Implementation rules

- **[IR1]** Clock C_i is incremented between any two successive events in process P_i

  $C_i[i] := C_i[i] + d \qquad (d > 0)$
- **[IR2]** If *a* is the event of sending a message *m* by process P_i, then m is assigned a vector timestamp $t_m = C_i(a)$ (after applying IR1). When *m* is received by process P_j, then C_j is

  $\forall k \neq j, C_j[k] := \max(C_j[k], t_m[k]); C_j[j] += d$

Space-time diagram



Vector Timestamps

- are "pairwise" comparable to determine
  - (not) equal
  - (not) less than or equal to
  - (not) less than
  - concurrent
  - "less than or equal to" is a partial order
  - concurrent is not transitive

Vector Timestamps, continued

- Events *a* and *b* are causally related if
  - $t^a < t^b$, or $t^b < t^a$
  - a→b iff $t^a < t^b$
- Can look at timestamps to determine causal relationships, which was not possible with Lamport's logical clocks
- Scalable?

How to use vector clocks?

- Build "causal message" system
- Basis for global state detection