CprE 450/550X
Distributed Systems and Middleware

Synchronization

Yong Guan
3216 Coover
Tel: (515) 294-8378
Email: guan@ee.iastate.edu

March 27, 2003

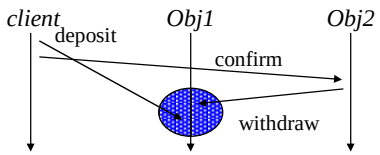
Readings for Today's Lecture

- References
  - Chapter 5 of "Distributed Systems: Principles and Paradigms"

3

Synchronization: Introduction

- A scary scenario:



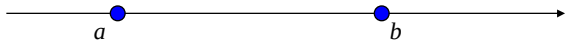
- **Synchronization**: temporal ordering of sets of events produced by concurrent processes in time.
  - Synchronization between senders and receivers of messages.
  - Control of joint activity.
  - Serialization of concurrent access to shared objects/resources.
- Why not Semaphores ?!
  - centralized systems: shared memory, central clock
  - distributed system: message passing, no global clock
- Events cannot be totally ordered!

4

A Partial Event Ordering for Distributed Systems (Lamport 1978)

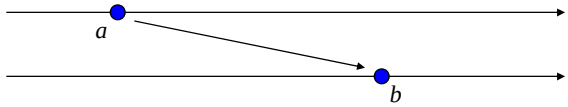
- Absence of central time means: no notion of *happened-when* (no total ordering of events)
- But can generate a *happened-before* notion (partial ordering of events)
- *Happened-Before* relation:

1. P_i



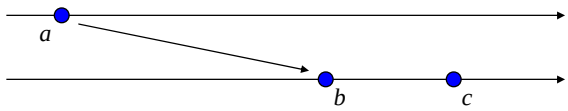
Event *a* happened-before Event *b*. (*a -> b)*

2. P_i / P_j



Event *a* happened-before Event *b*. (*a -> b)*

3. P_i / P_j

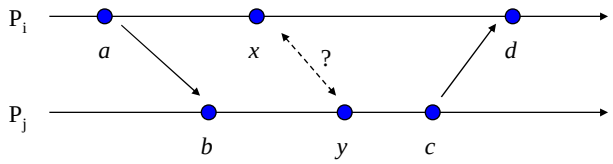


Event *a* happened-before Event *c*. (*a -> c) (transitivity)*

5

happened-before Relation

- What when no *happened-before* relation exists between two events?



Events *x* and *y* are concurrent.

- Problem:
  - only approximate knowledge of state of other processes
- Need global time:
  - common clock
  - synchronized clocks

6

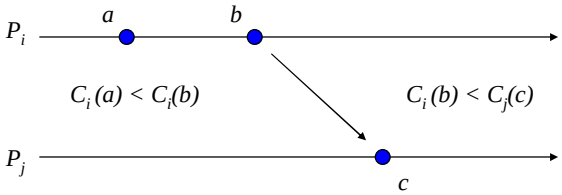
Logical Clocks

- Absolute time?
- Is chronological ordering necessary?
- Logical clock: assigns a number to each local event.

Clock Condition

$$\forall \text{ Events } a,b \;:\; \text{if } a \rightarrow b, \text{ then } C(a) < C(b)$$

- In Other Words:



7

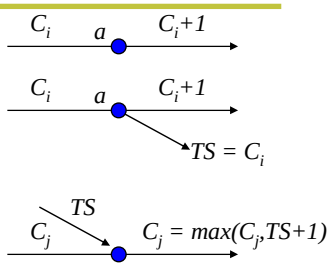
Total Ordering with Logical Clocks

- Rules:

  Rule 1: increment C_i after every local event.

  Rule 2: timestamp outgoing messages with current local clock

  Rule 3: Upon receiving message with timestamp TS, P_j updates local clock C_j to be
  $C_j = max\ (C_j, TS+1)$



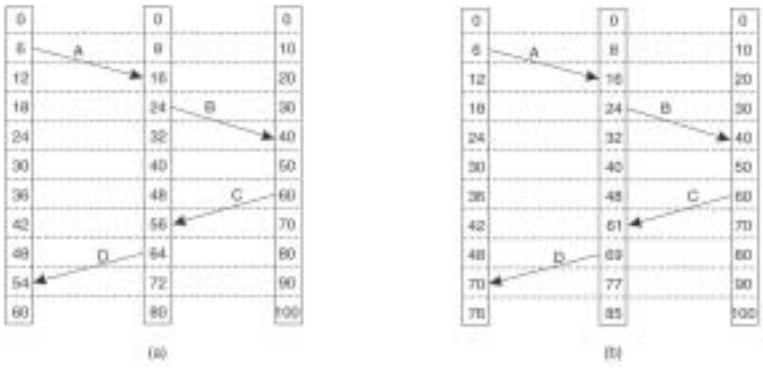
- Total ordering of events: assuming that clocks satisfy Clock Condition, define following relation:

$$a \Rightarrow b \quad \Leftrightarrow \quad \begin{array}{c} C_i(a) < C_j(b) \\ \text{or} \\ C_i(a) = C_j(b) \text{ and } i < j \end{array}$$

for events a on P_i and b on P_j.

8

Lamport Timestamps



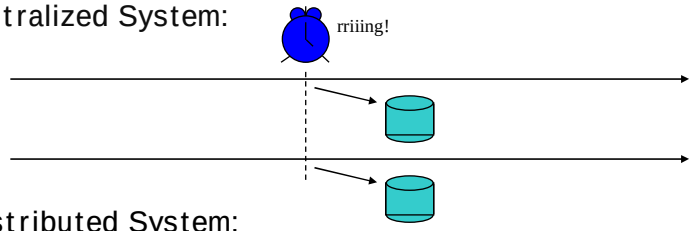
a) Three processes, each with its own clock. The clocks run at different rates.

b) Lamport's algorithm corrects the clocks.

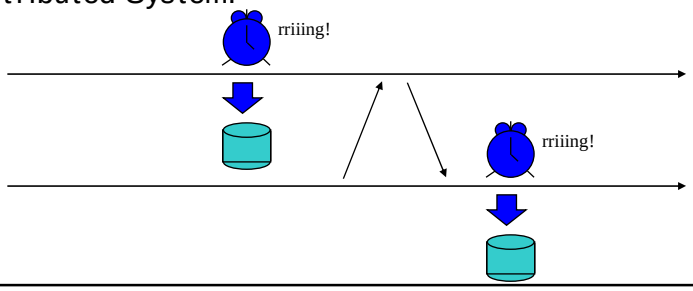
9

Example: Distributed Checkpointing

- "At 5pm everybody writes its state to stable storage!"
- Centralized System:



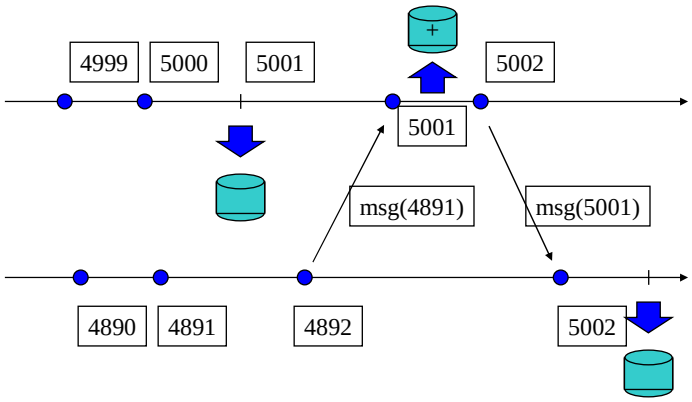
- Distributed System:



10

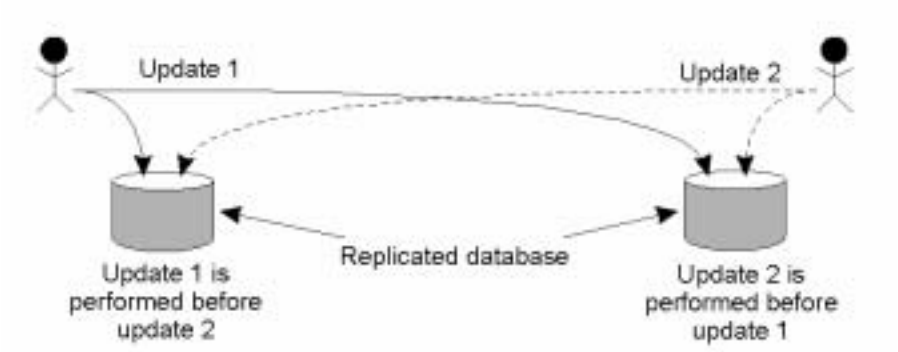
Distributed Checkpointing and Logical Clocks

"At logical-clock time 5000 write state to stable storage!"



11

Another Example: Totally-Ordered Multicasting



Updating a replicated database and leaving it in an inconsistent state.

12

Vector Timestamps

- Lamport timestamps: Can we say sth if C(a)<C(b)?
- Example: BBS message A and B, if totally ordered multicast is used, no way to say whether A is a reaction to B, or A and B are completely independent.

  The problem of Lamport timestamps does not capture causality.
- **Causality can be captured by Vector Timestamps**
  - VT(a): A vector timestamp assigned to event a.
  - If VT(a)<VT(b), then event a is known to causally precede event b.
  - Vector timestamp are contructed by letting each process Pi maintain a vector Vi with the following properties:
    1. $V_i[i]$ is the number of events happened so far at Pi
    2. If $V_i[j]=k$, then Pi knows that k events have occurred at Pj.

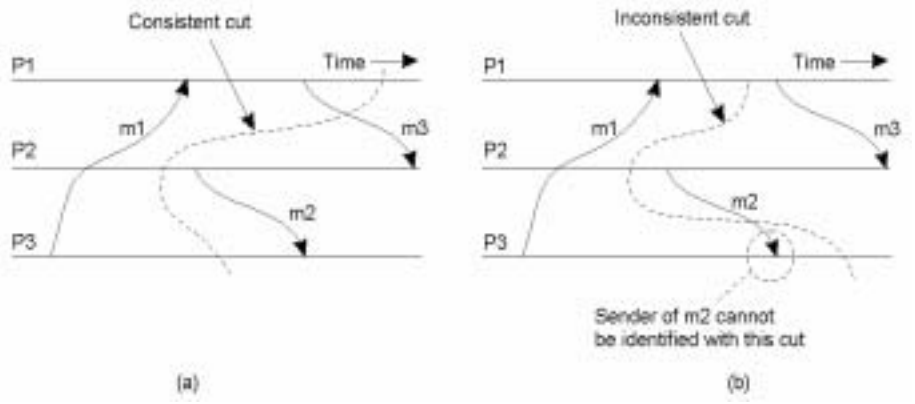
13

14

Global State (1)

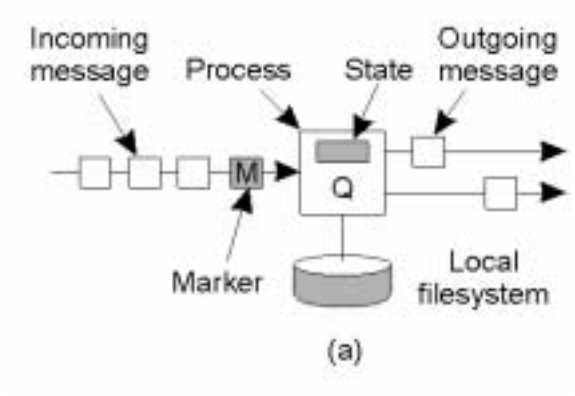
- Knowing the global state in distributed systems is useful on many occasions.
- The global state consists of the local state of each process, together with the messages-in-transit.
- Distributed Snapshot (Chandy and Lamport'85)

Global State (2)



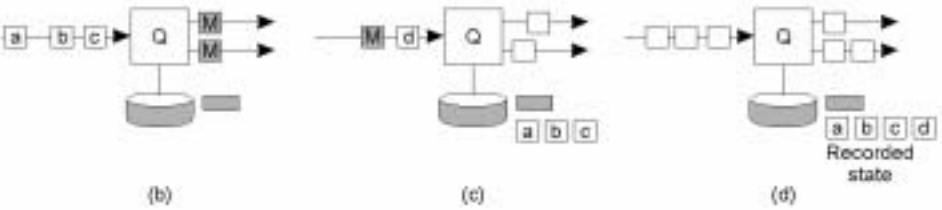
a) A consistent cut
b) An inconsistent cut

Global State (3)



a) Organization of a process and channels for a distributed snapshot

Global State (4)



b) Process Q receives a marker for the first time and records its local state

c) Q records all incoming message

d) *Q* receives a marker for its incoming channel and finishes recording the state of the incoming channel

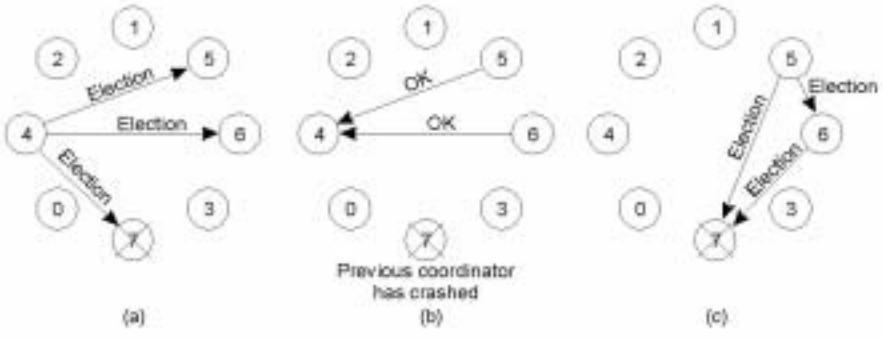
19

Election Algorithms

- Many distributed algorithms requires one process in the system acts as a leader (coordinator, initiator).
- It does not matter which process it is, but one of them has to do it.
- The goal of election algorithm is to ensure that when an election starts, it concludes with all processes agreeing on who the new coordinator is to be.

20

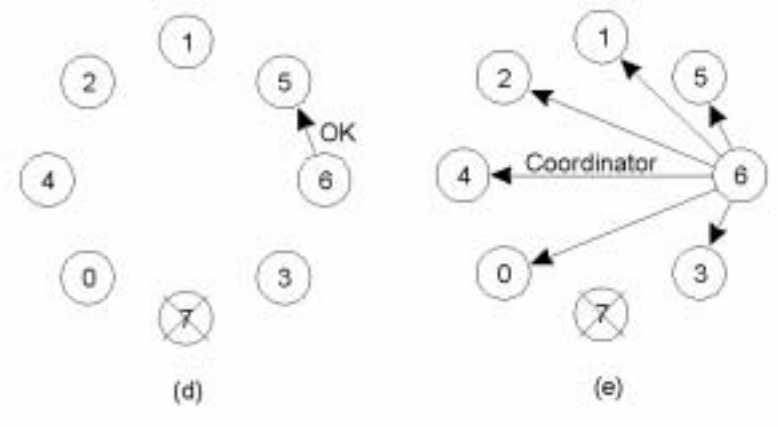
The Bully Algorithm (1)



The bully election algorithm

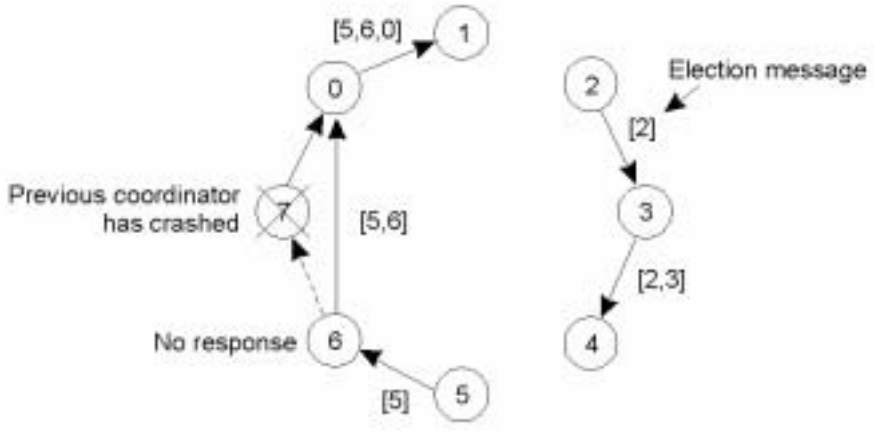
a) Process 4 holds an election
b) Process 5 and 6 respond, telling 4 to stop
c) Now 5 and 6 each hold an election

The Bully Algorithm (2)



d) Process 6 tells 5 to stop
e) Process 6 wins and tells everyone

A Ring Algorithm



Election algorithm using a ring.

Next class, we will discuss mutual exclusion and distributed transaction model.