

LECTURE - XXIII

## DISTRIBUTED SYSTEMS - I

Tevfik Koşar

University at Buffalo  
November 22<sup>nd</sup>, 2011

## Motivation

- **Distributed system** is collection of loosely coupled processors that
  - do not share memory
  - interconnected by a communications network
- Reasons for distributed systems
  - Resource sharing
    - sharing and printing files at remote sites
    - processing information in a distributed database
    - using remote specialized hardware devices
  - Computation speedup - **load sharing**
  - Reliability - detect and recover from site failure, function transfer, reintegrate failed site
  - Communication - message passing

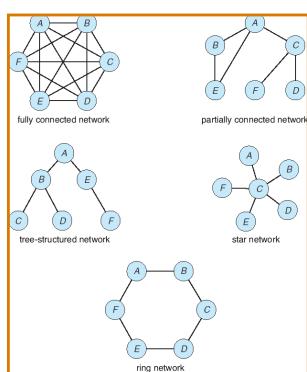
## Distributed-Operating Systems

- Users not aware of multiplicity of machines
  - Access to remote resources similar to access to local resources
- Data Migration - transfer data by transferring entire file, or transferring only those portions of the file necessary for the immediate task
- Computation Migration - transfer the computation, rather than the data, across the system

## Distributed-Operating Systems (Cont.)

- Process Migration - execute an entire process, or parts of it, at different sites
  - Load balancing - distribute processes across network to even the workload
  - Computation speedup - subprocesses can run concurrently on different sites
  - Hardware preference - process execution may require specialized processor
  - Software preference - required software may be available at only a particular site
  - Data access - run process remotely, rather than transfer all data locally

## Network Topology



## Robustness in Distributed Systems

- Failure detection
- Reconfiguration

## Failure Detection

- Detecting hardware failure is difficult
- To detect a link failure, a handshaking protocol can be used
- Assume Site A and Site B have established a link
  - At fixed intervals, each site will exchange an *I-am-up* message indicating that they are up and running
- If Site A does not receive a message within the fixed interval, it assumes either (a) the other site is not up or (b) the message was lost
- Site A can now send an *Are-you-up?* message to Site B
- If Site A does not receive a reply, it can repeat the message or try an alternate route to Site B

## Failure Detection (cont)

- If Site A does not ultimately receive a reply from Site B, it concludes some type of failure has occurred
- Types of failures:
  - Site B is down
  - The direct link between A and B is down
  - The alternate link from A to B is down
  - The message has been lost
- However, Site A cannot determine exactly **why** the failure has occurred

## Reconfiguration

- When Site A determines a failure has occurred, it must reconfigure the system:
  1. If the link from A to B has failed, this must be broadcast to every site in the system
  2. If a site has failed, every other site must also be notified indicating that the services offered by the failed site are no longer available
- When the link or the site becomes available again, this information must again be broadcast to all other sites

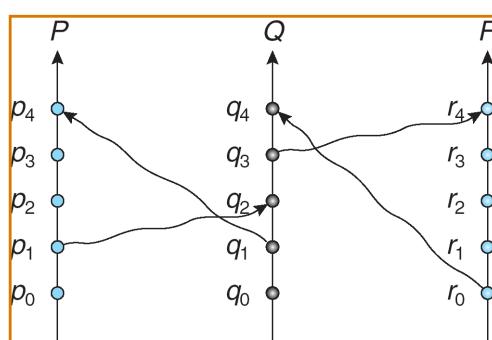
## Distributed Coordination

- Ordering events and achieving synchronization in centralized systems is easier.
  - We can use common clock and memory
- What about distributed systems?
  - No common clock or memory
  - *happened-before* relationship provides partial ordering
  - How to provide total ordering?

## Event Ordering

- **Happened-before** relation (denoted by  $\rightarrow$ )
  - If A and B are events in the same process (assuming sequential processes), and A was executed before B, then  $A \rightarrow B$
  - If A is the event of sending a message by one process and B is the event of receiving that message by another process, then  $A \rightarrow B$
  - If  $A \rightarrow B$  and  $B \rightarrow C$  then  $A \rightarrow C$
- If two events A and B are not related by the  $\rightarrow$  relation, then these events are executed **concurrently**.

## Relative Time for Three Concurrent Processes



Which events are concurrent and which ones are ordered?

## Exercise

Which of the following event orderings are true?

- (a)  $p_0 \rightarrow p_3$  :
- (b)  $p_1 \rightarrow q_3$  :
- (c)  $q_0 \rightarrow p_3$  :
- (d)  $r_0 \rightarrow p_4$  :
- (e)  $p_0 \rightarrow r_4$  :

Which of the following statements are true?

- (a)  $p_2$  and  $q_2$  are concurrent processes.
- (b)  $q_1$  and  $r_1$  are concurrent processes.
- (c)  $p_0$  and  $q_3$  are concurrent processes.
- (d)  $r_0$  and  $p_0$  are concurrent processes.
- (e)  $r_0$  and  $p_4$  are concurrent processes.

13

## Implementation of →

- Associate a timestamp with each system event
  - Require that for every pair of events A and B, if  $A \rightarrow B$ , then the timestamp of A is less than the timestamp of B
- Within each process  $P_i$ , define a **logical clock**
  - The logical clock can be implemented as a simple counter that is incremented between any two successive events executed within a process
    - Logical clock is **monotonically increasing**
- A process advances its logical clock when it receives a message whose timestamp is greater than the current value of its logical clock
  - Assume A sends a message to B,  $LC_1(A)=200, LC_2(B)=195 \rightarrow LC_2(B)=201$
- If the timestamps of two events A and B are the same, then the events are concurrent
  - We may use the process identity numbers to break ties and to create a total ordering

## Distributed Mutual Exclusion (DME)

- Assumptions
  - The system consists of  $n$  processes; each process  $P_i$  resides at a different processor
  - Each process has a critical section that requires mutual exclusion
- Requirement
  - If  $P_i$  is executing in its critical section, then no other process  $P_j$  is executing in its critical section
- We present two algorithms to ensure the mutual exclusion execution of processes in their critical sections

## DME: Centralized Approach

- One of the processes in the system is chosen to coordinate the entry to the critical section
- A process that wants to enter its critical section sends a request message to the coordinator
- The coordinator decides which process can enter the critical section next, and it sends that process a reply message
- When the process receives a reply message from the coordinator, it enters its critical section
- After exiting its critical section, the process sends a release message to the coordinator and proceeds with its execution
- This scheme requires three messages per critical-section entry:
  - request
  - reply
  - release

## DME: Fully Distributed Approach

- When process  $P_i$  wants to enter its critical section, it generates a new timestamp,  $TS$ , and sends the message *request* ( $P_i, TS$ ) to all processes in the system
- When process  $P_j$  receives a *request* message, it may reply immediately or it may defer sending a reply back
- When process  $P_i$  receives a *reply* message from all other processes in the system, it can enter its critical section
- After exiting its critical section, the process sends *reply* messages to all its deferred requests

## DME: Fully Distributed Approach (Cont.)

- The decision whether process  $P_j$  replies immediately to a *request* ( $P_i, TS$ ) message or defers its reply is based on three factors:
  - If  $P_j$  is in its critical section, then it defers its reply to  $P_i$
  - If  $P_j$  does not want to enter its critical section, then it sends a *reply* immediately to  $P_i$
  - If  $P_j$  wants to enter its critical section but has not yet entered it, then it compares its own request timestamp with the timestamp  $TS$ 
    - If its own request timestamp is greater than  $TS$ , then it sends a *reply* immediately to  $P_i$  ( $P_i$  asked first)
    - Otherwise, the reply is deferred
- **Example:** P1 sends a request to P2 and P3 (timestamp=10)  
P3 sends a request to P1 and P2 (timestamp=4)

## Undesirable Consequences

- The processes need to know the identity of all other processes in the system, which makes the dynamic addition and removal of processes more complex
- If one of the processes fails, then the entire scheme collapses
  - This can be dealt with by continuously monitoring the state of all the processes in the system, and notifying all processes if a process fails