DEADLOCK DETECTION IN DISTRIBUTED SYSTEMS

Deadlock can neither be prevented nor avoided in distributed system as the system is so vast that it is impossible to do so. Therefore, only deadlock detection can be implemented. The techniques of deadlock detection in the distributed system require the following:

- **Progress:**The method should be able to detect all the deadlocks in the system.
- **Safety:** The method should not detect false of phantom deadlocks.

There are three approaches to detect deadlocks in distributed systems.

Centralized approach:

- Here there is only one responsible resource to detect deadlock.
- The advantage of this approach is that it is simple and easy to implement, while the drawbacks include excessive workload at one node, single point failure which in turns makes the system less reliable.

Distributed approach:

- In the distributed approach different nodes work together to detect deadlocks. No single point failure as workload is equally divided among all nodes.
- The speed of deadlock detection also increases.

Hierarchical approach:

- This approach is the most advantageous approach.
- It is the combination of both centralized and distributed approaches of deadlock detection in a distributed system.
- In this approach, some selected nodes or cluster of nodes are responsible for deadlock detection and these selected nodes are controlled by a single node.

System Model

- A distributed program is composed of a set of n asynchronous processes p1, p2, . . . , pi , . . . , pn that communicates by message passing over the communication network.
- Without loss of generality we assume that each process is running on a different processor.
- The processors do not share a common global memory and communicate solely by passing messages over the communication network.

- There is no physical global clock in the system to which processes have instantaneous access.
- The communication medium may deliver messages out of order, messages may be lost garbled or duplicated due to timeout and retransmission, processors may fail and communication links may go down.

We make the following assumptions:

- The systems have only reusable resources.
- Processes are allowed to make only exclusive access to resources.
- There is only one copy of each resource.
- A process can be in two states: running or blocked.
- In the running state (also called active state), a process has all the needed resources and is either executing or is ready for execution.
- In the blocked state, a process is waiting to acquire some resource.

Wait for graph

This is used for deadlock deduction. A graph is drawn based on the request and acquirement of the resource. If the graph created has a closed loop or a cycle, then there is a deadlock.

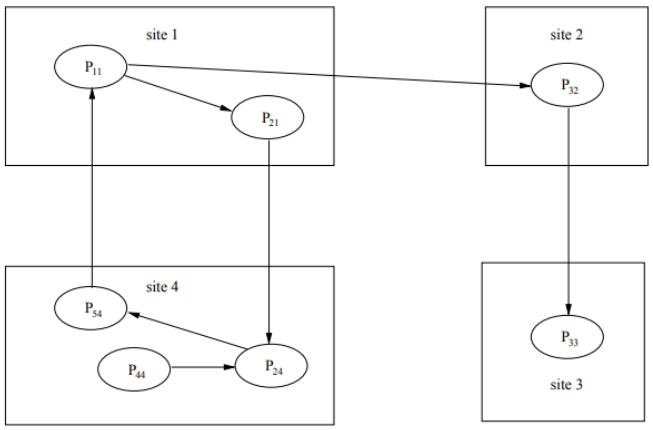

Fig : Wait for graph

Preliminaries

Deadlock Handling Strategies

Handling of deadlock becomes highly complicated in distributed systems because no site has accurate knowledge of the current state of the system and because every inter-site communication involves a finite and unpredictable delay. There are three strategies for handling deadlocks:

- **Deadlock prevention:**
  - This is achieved either by having a process acquire all the needed resources simultaneously before it begins executing or by preempting a process which holds the needed resource.
  - This approach is highly inefficient and impractical in distributed systems.
- **Deadlock avoidance:**
  - A resource is granted to a process if the resulting global system state is safe. This is impractical in distributed systems.
- **Deadlock detection:**
  - This requires examination of the status of process-resource interactions for presence of cyclic wait.
  - Deadlock detection in distributed systems seems to be the best approach to handle deadlocks in distributed systems.

Issues in deadlock Detection

Deadlock handling faces two major issues

1. Detection of existing deadlocks
2. Resolutionof detected deadlocks

Deadlock Detection

- Detection of deadlocks involves addressing two issues namely maintenance of the WFG and searching of the WFG for the presence of cycles or **knots**.
- In distributed systems, a cycle or knot may involve several sites, the search for cycles greatly depends upon how the WFG of the system is represented across the system.
- Depending upon the way WFG information is maintained and the search for cycles is carried out, there are centralized, distributed, and hierarchical algorithms for deadlock detection in distributed systems.

Correctness criteria

A deadlock detection algorithm must satisfy the following two conditions:

1. Progress-No undetected deadlocks:

The algorithm must detect all existing deadlocks in finite time. In other words, after all wait-for dependencies for a deadlock have formed, the algorithm should not wait for any more events to occur to detect the deadlock.

2. Safety -No false deadlocks:

The algorithm should not report deadlocks which do not exist. This is also called as called **phantom or false deadlocks.**

Resolution of a Detected Deadlock

- Deadlock resolution involves breaking existing wait-for dependencies between the processes to resolve the deadlock.
- It involves rolling back one or more deadlocked processes and assigning their resources to blocked processes so that they can resume execution.
- The deadlock detection algorithms propagate information regarding wait-for dependencies along the edges of the wait-for graph.
- When a wait-for dependency is broken, the corresponding information should be immediately cleaned from the system.
- If this information is not cleaned in a timely manner, it may result in detection of phantom deadlocks.