# Module -5

## Deadlock detection in distributed systems

- Deadlocks is a “fundamental problem in distributed systems.
- A process may request resources in any order, which may not be known a priori and a process can request resource while holding others.
- If the sequence of the allocations of resources to the processes is not controlled, deadlocks can occur.
- A deadlock is a state where a set of processes request resources that are held by other processes in the set”.

### System model - Models of deadlocks

- “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”

## Models of Deadlock

n distributed systems, deadlock refers to a situation where a set of processes are unable to make progress because each process is waiting for a resource that is held by another process in the set. Deadlock is particularly complex in distributed systems due to the distributed nature of resources and processes, making detection and resolution more challenging than in centralized systems.

**Knapp’s Classification of Distributed Deadlock Detection Algorithms**
**Knapp’s Classification** provides a systematic framework for understanding and categorizing various **deadlock detection** algorithms in **distributed systems**. The classification is based on a set of key characteristics that influence how deadlock detection is implemented, and it primarily addresses the complexity and nature of the algorithms used in distributed systems.
Knapp's classification considers the following factors:

1. **Detection Methodology** (Centralized vs. Decentralized)
2. **Detection Frequency** (Periodic vs. On-demand)
3. **State Representation** (Global vs. Local)
4. **Resolution Strategy** (Preventive vs. Reactive)
5. **Deadlock Detection Process** (Cycle detection vs. Resource allocation-based analysis)

By classifying deadlock detection algorithms according to these factors, Knapp offers a clear understanding of how different algorithms address the problem of deadlock in distributed systems.

**1. Detection Methodology: Centralized vs. Decentralized**
This dimension of classification refers to whether the deadlock detection is performed at a central location or distributed across multiple nodes.

- **Centralized Detection**:
    - A single process (or node) is responsible for detecting deadlocks by gathering information from all other processes in the system.
    - Example: A **central coordinator** that collects resource allocation and request information from all nodes and builds a **global wait-for graph** to detect deadlocks.
    - **Advantages**:
        - Simple to implement.
        - Easier to manage, as all detection logic is centralized.
    - **Disadvantages**:
        - Single point of failure (if the coordinator fails, detection fails).
        - Potential performance bottleneck due to frequent communication and data gathering.
- **Decentralized (Distributed) Detection**:
    - No single node is responsible for deadlock detection. Each node is responsible for detecting deadlock in its own local context and then communicates with other nodes to identify global deadlock conditions.
    - Example: **Chandy-Misra-Haas** algorithm, where processes communicate with each other to build a distributed wait-for graph.
    - **Advantages**:
        - No single point of failure.
        - More scalable than centralized detection.
    - **Disadvantages**:
        - More complex to implement.
        - Requires efficient communication between nodes.
        - May require additional coordination to detect global deadlocks.

**2. Detection Frequency: Periodic vs. On-demand**
This dimension refers to the frequency at which deadlock detection is performed.

- **Periodic Detection**:
    - Deadlock detection is performed at regular intervals, regardless of whether a deadlock is suspected or not.
    - Typically used when the system is highly dynamic and processes constantly interact with resources.
    - **Advantages**:
        - Simple to implement.
        - Deadlock detection happens periodically, ensuring that deadlocks are detected regularly.
    - **Disadvantages**:
        - Can introduce unnecessary overhead, especially if deadlocks are rare.
        - Might not detect deadlocks in a timely manner if the intervals are long.
- **On-demand Detection**:

    - Deadlock detection is only initiated when a deadlock is suspected or explicitly requested.
    - This approach is more efficient, as deadlock detection is not performed unnecessarily.
    - **Advantages**:
        - More efficient, as detection is triggered only when needed.
        - Reduces the overhead of periodic checks.
    - **Disadvantages**:
        - Deadlocks might not be detected promptly, leading to potential delays in the system.

**3. State Representation: Global vs. Local**

This refers to how the state of the system is represented during the detection process.

- **Global State Representation**:
    - The entire system's state is represented as a whole, typically involving a global resource allocation graph or wait-for graph that is shared and maintained by a central authority or all processes.
    - Example: A **global wait-for graph** that represents the waiting relationships of all processes in the system.
    - **Advantages**:
        - Clear, comprehensive view of the system's state.
        - Easier to detect deadlocks that span multiple processes or resources.
    - **Disadvantages**:
        - High communication overhead to maintain the global state.
        - A single point of failure in centralized approaches.
- **Local State Representation**:
    - Each process maintains only a **local state** and is responsible for detecting deadlocks within its own scope.
    - Example: A process that only keeps track of resources it holds and processes it waits for, without global knowledge of the system's state.
    - **Advantages**:
        - Scalable and more fault-tolerant.
        - Reduced communication overhead since processes only share local information.
    - **Disadvantages**:
        - Harder to detect global deadlocks.
        - Complexity increases as processes need to coordinate to detect global deadlocks.

**4. Resolution Strategy: Preventive vs. Reactive**

This category classifies algorithms based on their approach to handling detected deadlocks.

- **Preventive Algorithms**:
    - These algorithms aim to avoid deadlock before it occurs by ensuring that conditions like **circular wait**, **mutual exclusion**, and **hold-and-wait** are not violated.

    - Example: **Banker's Algorithm**, which checks the system's state for safety before granting resources to processes.
    - **Advantages**:
        - Deadlock is avoided proactively, and the system operates smoothly.
    - **Disadvantages**:
        - Overhead in managing resource allocation and preventing potential deadlocks.
        - Might restrict resource usage and reduce concurrency.
- **Reactive Algorithms**:
    - These algorithms detect deadlock after it occurs and take corrective actions such as process abortion or resource preemption to resolve the deadlock.
    - Example: Algorithms that **abort** one or more processes or **rollback** their actions to break the deadlock cycle.
    - **Advantages**:
        - Simpler than preventive algorithms since they react only when deadlock happens.
        - More flexible, as they allow for higher concurrency.
    - **Disadvantages**:
        - Deadlocks can persist for some time before they are detected and resolved.
        - May involve significant overhead in aborting processes or rolling back actions.

## Mitchell and Merritt's algorithm for the single resource model

Mitchell and Merritt's algorithm is a distributed deadlock detection algorithm designed for the single-resource model. In this model, there is only one type of resource in the system, and each resource can only be held by a single process at a time. The algorithm is designed to work in distributed systems where processes are spread across different nodes (computers) in a network, and there is a need to detect deadlock conditions efficiently in this environment.

## Main Steps of the Algorithm:

1. Wait-for Graph Representation:
    - Each process maintains a local wait-for graph that represents its current state of waiting for the resource.
    - A process P_i sends a request message to another process P_j if it is waiting for the resource that P_j holds.

2. Initiating the Detection:
    - When a process P_i requests the resource held by another process P_j, it sends a request message to P_j.
    - The request message is propagated along the chain of processes until either:
        - A cycle is detected, indicating a deadlock, or
        - The request reaches a process that does not have the resource (in which case the waiting chain is broken).
3. Cycle Detection:
    - Each process maintains a list of all the processes it has sent request messages to (i.e., processes it is waiting for).
    - When a process P_i sends a request message to P_j, it checks whether it has previously sent a request message to P_j or if P_j has sent a request message to P_i.
    - If such a message exists in the request history, a cycle is detected, meaning that deadlock exists.
4. Deadlock Detection:
    - If a process P_i detects a cycle in the wait-for graph (i.e., a circular chain of waiting processes), it knows that there is a deadlock among the processes involved in the cycle.
    - Once a deadlock is detected, the process can take corrective action such as aborting or preempting one of the processes involved in the deadlock to break the cycle.
5. Communication:
    - Each process communicates with other processes to propagate request messages and track dependencies.
    - The algorithm ensures that each process can monitor its interactions with other processes and identify when a process is involved in a cycle of resource request

## **Chandy–Misra–Haas algorithm for the AND model**

The **Chandy-Misra-Haas (CMH) algorithm** is one of the most widely studied distributed deadlock detection algorithms. It was originally designed to work in a **distributed system** where multiple processes share resources and communication happens through message passing.

Algorithm:

- if Pi is locally dependent on itself then declare a deadlock else for all Pj and Pk such that
- Pi is locally dependent upon Pj , and
- Pj is waiting on Pk , and
- Pj and Pk are on different sites,send a probe (i, j, k) to the home site of Pk

On the receipt of a probe (i, j, k), the site takes the following actions: if

- Pk is blocked, and
- dependentk (i) is false, and
- Pk has not replied to all requests Pj

  then
  begin
  dependentk(i) = true; if k=i
  then declare that Pi is deadlocked else for all Pm and Pn such that
  (a’) Pk is locally dependent upon Pm, and
  (b’) Pm is waiting on Pn, and
  (c’) Pm and Pn are on different sites, send a probe (i, m, n) to the home site of Pn
  end.

  - A probe message is continuously circulated along the edges of the global WFG graph and a deadlock is detected when a probe message returns to its initiating process.

**Algorithm**

The algorithm works as follows:
Initiate a diffusion computation for a blocked process Pi :
send query(i, i, j) to all processes Pj in the dependent set
DSi of Pi ;
numi (i):= |DSi |; waiti (i):= true;
When a blocked process Pk receives a query(i, j, k):
if this is the engaging query for process Pi
then send query(i, k, m) to all Pm in its dependent set DSk ;
numk (i): = |DSk |; waitk (i):= true
else if waitk (i) then send a reply (i, k, j) to Pj .
When a process Pk receives a reply(i, j, k):
if waitk (i)
then begin
numk (i):= numk (i) − 1;
if numk (i)= 0
then if i=k then declare a deadlock
else send reply(i, k, m) to the process Pm
which sent the engaging query.

## <u>Distributed Shared Memory</u>

- ➢ A distributed shared memory (DSM) system is made up of numerous computers, or nodes, connected by a network and each has its own local memory.
- ➢ All nodes' memory is managed by the DSM system. Every computer or node communicates and processes transparently with the others.
- ➢ Additionally, the DSM ensures that each node is independently accessing the virtual memory without any hindrance from other nodes.
- ➢ There is no physical memory in the DSM; instead, all nodes share a virtual space address, and data is transferred between them via the main memory.

**Types of Distributed shared memory:**

1. **On-Chip Memory:**
    1. All the data are stored in the CPU's chip.
    2. There is a direct connection between Memory and address lines.
    3. The On-Chip Memory DSM is very costly and complicated.
2. **Bus-Based Microprocessor**
    1. The connection between the memory and the CPU is established through a number of parallel wires that acts as a bus.
    2. All the computer follows some protocols to access the memory, and an algorithm is implemented to prevent memory access by the systems at the same time.
    3. The network traffic is reduced by the cache memory.
3. **Ring-based Microprocessor**
    1. In Ring-based DSM, there is no centralized memory present
    2. All the nodes are connected through some link/network, and accessing of memory is done by the token passing
    3. All the nodes/computers present in the shared area access a single address line in this system.

## Abstraction and Advantages

It is an abstraction provided to the programmer of a distributed system.
• It gives the impression of a single memory. Programmers access the data across the network using only read and write primitives.
• Programmers do not have to deal with send and receive communication primitives and the ensuing complexity of dealing explicitly with synchronization and consistency in the message passing model.
• A part of each computer's memory is earmarked for shared space, and the remainder is private memory.
• To provide programmers with the illusion of a single shared address space, a memory mapping management layer is required to manage the shared virtual memory space.

## <u>Memory consistency models</u>

The memory consistency model of a shared-memory system determines the order in which memory operations will appear to execute to the programmer.

Processor 1 writes to some memory location…
Processor 2 reads from that location…
Do I get the result I expect?
• Different models make different guarantees; the processor can reorder/overlap memory operations as long as the guarantees are upheld.

***Three models of memory consistency***

1. Sequential Consistency (SC): – Memory operations appear to execute one at a time, in some sequential order. – The operations of each individual processor appear to execute in program order.
2. Processor Consistency (PC): – Allows reads following a write to execute out of program order (if they're not reading/writing the same address!) – Writes may not be immediately visible to other processors, but become visible in program order.
3. Release Consistency (RC): – All reads and writes (to different addresses!) are allowed to operate out of program order