DISTRIBUTED SYSTEMS (DS)

Unit I

1. What is DS? Explain the characteristics of DS.
2. What is distributed mutual exclusion? How it can be classified?
3. Describe the system model in detail.
4. Write short notes on :-
   - (a) Logical clock.
   - (b) Casual ordering of msgs.
   - (c) Termination detection
   - (d) Token & non-token based algo.
5. What are the requirements of distributed mutual exclusion theorem? Also discuss the performance metrics of it.

Unit II

1. What is distributed deadlock detection? What are the issues in deadlock detection?
2. ~~Write~~ Explain the deadlock prevention techniques with eg.
3. Write short notes on :-
   - (a) Path pushing algo.
   - (b) Edge chasing algo.
4. What do u understand by agreement protocol? Also discuss its applica.
5. How can we classify agreement protocol? Explain in detail.

DS:- a s/w system that consists of a collec^n of autonomous comps., connected through a n/w & distribu^n middleware which enables comps. to coordinate their activities & to share the resources of the system, so that users perceive the system as a single integrated computing facility.

Centralised System	Distributed System
→ one component with non-autonomous parts	→ Multiple autonomous components
→ Component shared by users all the time.	→ components are not shared by all the users.
→ All resources accessible.	→ Resources may not be accessible.
→ S/w runs in a single process.	→ S/w runs in concurrent processes on diff. processors.
→ Single point of control	→ Multiple pts of control.
→ Single point of failure	→ Multiple pts of failure.

Applica^{n}s of DS:-

- telecommunica^n n/ws.
- telephone & cellular n/ws.
- comp. n/ws like Internet
- routing algos.
- wireless sensor networks (WSN)
- multiplayer online games.
- VR communities
- www & peer to peer n/ws.
- distributed dbms / filesystems.
- aircraft control systems.

Characteristics of DS:-

→ **Resource Sharing** - Ability to use any hw, sw or data anywhere in the system.

→ **Openness** - concerned with extensions & improvements of ds.

→ **Concurrency** - Components in DS are executed in concurrent processes

→ **Scalability** - Adapta^n of DS to accomodate more users & respond faster

- **Faults Tolerance** - achieved by recovery & redundancy in cases of n/w, s/w & n/w failure.

- **Transparency** - DS should be perceived by users & programmers as a whole rather than as a collec^n of cooperating components

→ It has various dimensions: ◎ Scalability Transparency ◎ Replica^n T.
◎ Performance " ◎ Concurrency T.
◎ failure " ◎ Access T.
◎ Migra^n " ◎ Loca^n T.

Mutual Exclusion in DS

Mutual Exclusion:- Its a concurrency contrd prop. ~~that~~ to prevent race condⁿ.
- A process can't enter its critical secⁿ while another concurrent process is currently executing in its critical secⁿ i.e. only one process is allowed to execute the critical secⁿ at any given instance of time.

ME in single computer system v/s distributed system:

→ In single systems, the status of shared resources & the status of users is easily available in the shared memory, so with the help of shared var. (eg. semaphores) mutual exclusion prob. can be easily solved.

→ In distributed systems, we neither have shared memory nor a common physical clock & ∴ we can't solve ME prob. using shared vars., so we use message passing based approach to eliminate ME prob. in DSs.

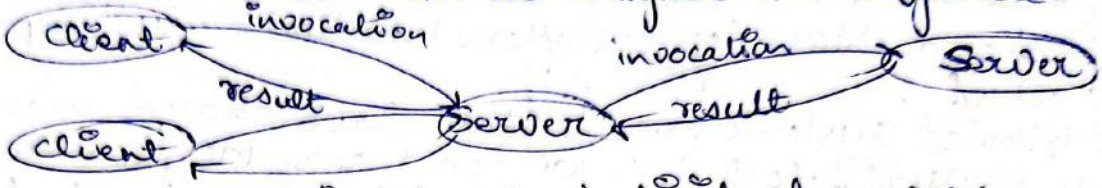
Classification and solution of ME in DS:-

Message passing is a way to implement ME which can be classified into 3 approaches:-

1. **Token Based Algo.:-**
   - A unique token is shared among all the sites.
   - If a site possesses the unique token, its allowed to enter its critical section.
   - This approach uses sequence no. to order the requests for the critical secⁿ.
   - This approach ensures Mutual Exclusion as the token is unique.
   - Eg. Suzuki-Kasami's Broadcast Algo.
2. **Non-Token Based Approach:-**
   - A site communicates with other sites in order to determine which sites should execute critical secⁿ next. This requires exchange of 2 or more successive round of msgs among sites.
   - This approach uses timestamps instead of seq. nos. to order the requests for the critical secⁿ.
   - An algo. which follows non-token based approach maintains a logical clock which gets updated acc'ing to Lamport's scheme.
   - Eg. Lamport's Algo., Ricart-Agrawala algo.
3. **Quorum based approach:-**
   - Instead of requesting permission to execute the critical secⁿ from all other site's, each site requests only a subset of sites, which is called a quorum.
   - Any 2 subsets of sites or Quorum contains a common site.
   - This common site is responsible to ensure ME.
   - Eg. Maekawa's Algo.

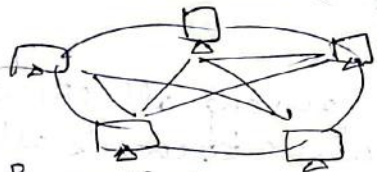
System Models:- Systems that are intended for use in real-world envs. should be designed to function correctly in the widest possible range of circumstances & in the face of many possible difficulties & threats.

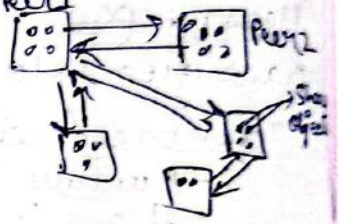
1. **Physical Models:-** They capture the hw composition of a system in terms of the comps. (& devices) & their interconnecting n/ws.
2. **Architectural Models:-** These describe a system in terms of the computational & comm^n tasks performed by its computational elements.
   - **Client-Server Model:-** Most imp. & most widely used DS arch.
     - Client-Server roles are assigned & changeable.



Client invoke individual servers

   - **Peer to Peer Model:-** Unlike client-server, P2P-model does not distinguish b/w client/server, instead each node can act as both be a client or server depending on whether the node is requesting or providing the services.
     - Each node is considered as a peer.





     - Each obj. is replicated in several comps. to further distribute the load & to provide resilience in the event of discon^n of individual comps.
3. **Fundamental Models:-** These take an abstract perspective inorder to examine individual aspects of a distributed system.
   - These examine 3 imp. aspects of DS:-
   - **Interaction Models:-** It considers the str. & sequencing of the comm^n b/w the elements of the system.
     - Synchronous:- with time bounds } processes execute
     - Asynchronous:- without time bounds }
   - **Failure Models:-** It considers the ways in which a system may fail to operate correctly.
     - Omission failure:- cases when a process fails to perform actions that it is supposed to do
     - Arbitrary failures:- worst possible failure in which any type of error may occur.
     - Timing failure:- processes fail to provide response in time.
   - **Security Models:-** It considers how the system is protected against attempts to interfere with its correct operation or to steal its data.
     - Defeating security threats → Cryptography & shared secrets.
       - Authentication
       - Secure Channels

(a) Logical Clock:- It refers to implementing a protocol on all machines within your DS, so that the machines are able to maintain consistent ordering of events within some virtual timespan.
- DSs may have no physically synchronous global clock, so a logical clock allows global ordering on events from diff. processes in such sys.
- Eg. We have 10 PCs in a DS, then how we make them work together, There comes a sol^n to this ie Logical clock.

(b) Casual Ordering of Msgs:-
- Its one of the 4 semantics of multicast communica^n namely unordered, totally ordered, casual, & sync-ordered comm^n.
- It describes the casual relation ship b/w a msg. sent event & a msg. received event.
- Eg. if send(M1) → send(M2), then every recepient of both the msgs. M1 & M2 must receive the msg M1 before receiving the msg. M2.
- In DS, the casual ordering of msgs. is not automatically guaranteed
- Reasons for Viola^n of COOM:- due to transmission delay.
  congestion in the n/w.
  failure of a system.

(c) Termination Detection:-
- In DS, inferring if a distributed computa^n has ended is essential so that the results produced by the computa^n can be used.
- Also in some cases, the execution of a subprob. can't begin until the execu^n of the previous subprob. is complete.
- So when we are interested in inferring when the underlying computa^n has ended, a termina^n detection algo. is used for this purpose.
- A distributed computa^n is considered to be globally terminated if every process is locally terminated & there is no msg. in transit b/w any processes.
- The detec^n of the termina^n of distributed computa^n is non-trivial since no process has complete knowledge of the global state, & global time does not exist.
- Eg. Huang's Termination Detection Algo.

(d) Token & Non token based Algo.
- In token based algo., a unique token is shared among all the sites, & the site possessing a token is only allowed to enter its critical section. It uses seq. no. to order the requests.
- In non-token based algo., a site communicates with other sites in order to determine which sites should execute critical sec^n's next. It uses timestamps to order the requests for critical sec^n's

Requirements of Distributed Mutual Exclusion Theorem :-

→ **No deadlock** :- 2 or more site should not endlessly wait for any msg. that will never arrive.

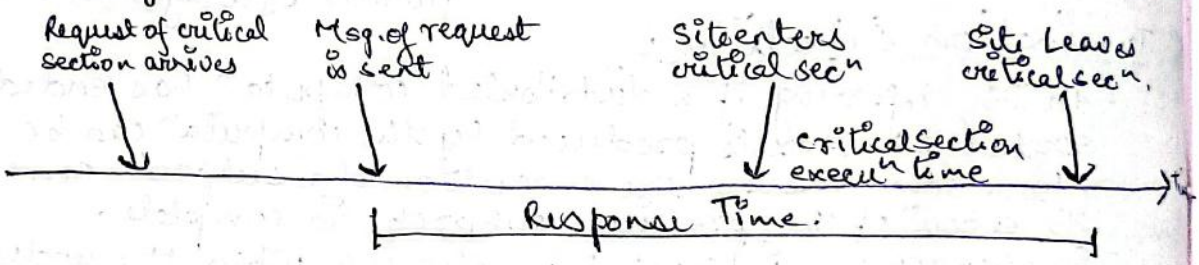
→ **No starvation** :- Any site should not wait indefinitely to execute critical section while other site are repeatedly executing critical sec^{n}.

→ **Fairness** :- Each site should get a fair chance to execute critical sec^{n}. Any request to execute critical sec^{n} must be executed in the order of their arrival in the system.

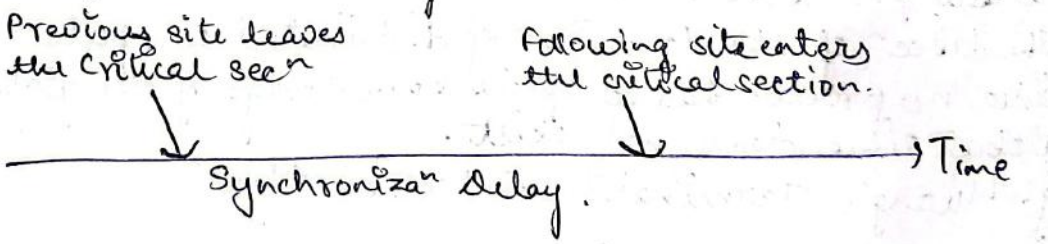
→ **Fault Tolerance** :- In case of failure, it should be able to recognize it by itself in order to continue functioning without any disruption.

Performance Metrics of Mutual Exclusion :-

1. **Response Time** :- The interval of time when a request waits for the end of its critical sec^{n} execution after its solicita^{n} msgs. have been conveyed.



2. **Synchronization Delay** :- The time reqd. for the next process to enter the critical section after a process leaves the critical sec^{n} is k/a Synchroniza^{n} delay.



3. **Message Complexity** :- The no. of msgs. needed to execute each critical section by the process.

4. **Throughput** :- It is the amount at which the system executes requests for the critical sec^{n}.

$$\text{Throughput} = \frac{1}{\text{Synchroniza}^{n}\text{ Delay} + \text{Avg. critical sec}^{n}\text{ execu}^{n}\text{ time}}$$

5. **Low & High Load Performance** :- The amt. of request that arrives for critical sec^{n} execu^{n} denotes the load. A site is only occasionally in this inactive State in heavy load condi^{n}s.

- For some ME algos., the performance metrics can be registered effectively under low & heavy loads through simple mathematical reasoning.

Deadlock detection in Distributed Systems :-

- In a distributed sys., deadlock can neither be prevented, nor avoided as the system is so vast ∴ Only deadlock detec^n can be implemented. The techniques in deadlock detec^n require following :-

Progress :- The method should be able to detect all deadlocks in the system.

Safety :- The method should not detect false or phantom deadlocks.

3 approaches to deadlock detection :-

1. Centralised approach :- Only one node responsible to detect deadlock.
   - Simple & easy to implement.
   - Excessive workload at one node.
   - Single pt-failure which makes the system less reliable.
2. Distributed approach :- Different nodes work together to detect deadlocks.
   - No single pt. failure as the workload is equally divided among all nodes.
   - Speed of deadlock detec^n also ↑s.
3. Hierarchical Approach :- Most advantageous.
   - Combination of centralised & distributed approaches.
   - In this approach, some selected nodes or clusters of nodes are responsible for deadlock detection & these selected nodes are controlled by a single node.

Issues of deadlock detection :-

1. It requires addressing 2 fundamental issues : first, detecting existing deadlocks, & second resolving detected deadlocks.
2. It entails tackling 2 issues : WFG (Wait for Graph) maintenance & searching the WFG for the presence of cycles.
3. In a DS, a cycle may include multiple sites. The search for cycles is highly dependant on the system's WFG as represented across the system.

Resolution of Deadlock Detec^n :-

- It includes breaking existing wait-for dependencies in the system WFG.
- It includes rolling multiple deadlocked processes & giving their resources to the blocked processes in the deadlock so that they may resume execution.

Deadlock detection Algorithms in D.S

1. Path Pushing Algos.
2. Edge Chasing Algos.
3. Diffusing Computations Based Algos.
4. Global State Detection Based Algos.

Deadlock:- Its a situation where a set of processes are blocked because each process is holding a resource & waiting for a resource that is held by some other process.

4 necessary conditions for deadlock:

1. **Mutual Exclusion**:- There is atleast one resource that is non-shareable & can be used by only one process at a time.
2. **Hold & wait**:- A process is holding atleast on resource & waiting for another.
3. **No Preemption**:- A process can't be taken from a process until it releases the resource.
4. **Circular wait**:- At least 2 processes should form a circular chain by holding a resource & waiting for a resource that is held by the next process in the chain.

Handling deadlock
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection & Recovery

Deadlock Prevention:- So if any of the above 4 necessary conditions are prevented, we can prevent a deadlock to occur.
- So there are 3 methods for deadlock prevention:-

1. **Collective requests**:- It prevents hold & wait condition of deadlock.
   - (a) Allocate all reqd. resources to the process before the start of its execution.
     - It will lead to low device utilization.
     - Eg. if a process requires printer at a later time, but resource allocated it before the start of its execution, printer will remain blocked till it has completed its execution.
   - (b) The process will make a new request for resources after releasing the current set of resources.
     - It may lead to starvation.
2. **Ordered Request**:- It prevents circular wait condition.
   - Ordering is imposed on the resources, & thus process request for resources in rising order.
   - An ordering strictly indicates that a process never asks for a low resource while holding a high one.
   - Its better to give priority to the old processes bcoz of their long existence & might be holding more resources.
   - It also eliminates starvation issues as the younger transactions will eventually be out of the system.
3. **Preemption**:-
   - (a) **Wait-die**:- If an older process requires a resource held by a younger process, the younger will have to wait. A young process will be destroyed if it requests a resource controlled by an older process.
   - (b) **Wound-Wait**:- If an old process seeks a resource held by a young process, the young process will be preempted, wound, & killed, & the old process will resume & wait. If a young process needs a resource held by an older process, it will have to wait.

Deadlock Avoidance :- can be done by Banker's Algo.

Banker's Algo.- Its a resource allocation & deadlock avoidance algo. which test all the request made by processes for resources, it checks for the safe state at every step, if after granting request system remains in the safe state it allows the request & if there is no safe state it doesn't allow the request made by the proces

Deadlock Detection Algos :- (4)

1. Path Pushing Algos.- The detec" is carried out by maintaining an explicit global WFG for each distributed system site.
   - When a site performs a deadlock computa", it sends its local WFG to all neighboring sites.
2. Edge chasing Algos:- It verifies a cycle in a distributed graph str. by sending special msgs. called probes along the graph's edges.
   - These probing msgs. are distinct from request & response msgs.
   - If a site receives the matching probe that it previously transmitted it can cancel the forma" of cycle.
3. Diffusing Computa"s Based Algos.:-
   - In this, deadlock detec" computa" is diffused over the system's wo
   - The underlying distributed computa" is superimposed on this computa".
   - If this computa" fails, the initiator reports a deadlock global state detec".
4. Global State Detection based Algos:- The detec" of distributed deadlocks can be made by taking a snapshot of the system & then inspecting it for signs of a deadlock.

Aggreement Protocol :-

- In distributed system, where several sites are communicating via msgs, some type of trust b/w them is reqd. for this, sites make some sort of agreements for mutual working.

System Model for agreement protocol :-

- There are 'n' processors in the system, & at most 'm' of the processors can be faulty.
- The processors can directly communicate with other processors by message passing.
- A receiver processor always knows the identity of the sender processor of the message.
- The communica" medium is reliable & only processors are prone to failure.

Application of Agreement Algos :-

- Fault-tolerant clock Synchronization.
- In distributed db systems (DDBS), data managers at sites, must agree on whether to commit or to abort a transaction.

Classification of Agreement Protocols (Problems) :-

→ Byzantine agreement protocol - A single value, which is to be agreed on, is initialized by an arbitrary processor & all non-faulty processors have to agree on that value.

→ Consensus Problem - Every processor has its own initial value & all non-faulty processors must agree on a single common value.

→ Interactive consistency prob :- Every processor has its own initial value & all non-faulty processors must agree on a set of common values.

1. Byzantine Agreement Prob :-

- Source processor is an arbitrarily chosen processor.
- Source processor broadcasts its initial value to all other processors.
- The soln should meet the following condi"s :-

Agreement - All non-faulty processors agree on same value.

Validity - If the source processor is non-faulty, then the common agreed upon value by all non-faulty processors should be the initial value of the source.

- Note :- If source processor is faulty, then all non-faulty processors can agree on any common value.
- Its irrelevant what value faulty processors agree on or whether they agree on a value at all.

2. Consensus Prob :-

- Every processor broadcasts its initial value to all other processors.
- Initial value of the processors may be diff.
- Condi"s :

Agreement - All non-faulty processors agree on the same single value.

Validity - If the initial value of every non-faulty processor is 'v', then the agreed upon common value by all non-faulty processors must be 'v'.

- Note :- If the initial values of non-faulty processors are diff, then all non-faulty processors can agree on any common value.
- We don't care what value faulty processors agree on.

3. Interactive Consistency Prob :-

- Every processor broadcasts its initial value to all other processors.
- The initial values may be diff.
- Condi"s :

Agreement - All non-faulty processors agree on the same vector $(v_1, v_2, \ldots)$

Validity - If the 'i'th processor is non-faulty & its initial value is v_i, then the 'i'th value to be agreed on by all non-faulty processors must be 'v_i'.

- Note :- If the 'j'th processor is faulty, then all non-faulty processors can agree on any common value for 'j'.