Chapter: 4
Agreement in Distributed System

2 Questions (15 marks)

(2 times)

1. Explain the necessity of leader process in distributed System.

OR

Why is coordinator needed in distributed System?

Ans: A distributed system is a collection of independent computers that work together to provide a unified computing service.

In such a system, there is a need for a leader process that helps in coordinating the activities of all the other processes in the system.

The following are some of the reasons why a leader process is necessary in a distributed system:

1. **Coordination**: In a distributed system, there are multiple processes that are executing simultaneously.
   These processes may need to coordinate their activities to achieve a common goal.
   A leader process can help in coordinating the activities of these processes by providing a centralized point of control.

2. **Resource allocation**: The leader could monitor the usage of network bandwidth, memory, and storage across all the nodes in the system, and allocate these resources fairly and efficiently.

For example, if one node is using a lot of network bandwidth, the leader could temporarily limit its usage to ensure that other nodes have sufficient bandwidth for their own tasks.

3. **Task scheduling:** The coordinator could help schedule tasks and manage the execution of distributed applications across multiple nodes.
   For example, it could prioritize tasks based on their importance, allocate resources to specific tasks, and coordinate communication between nodes to ensure that tasks are completed successfully.

4. **Fault tolerance**: If a node in the system fails or becomes unavailable, the coordinator could help manage the recovery and redistribution of resources to other nodes in the system.
   For example, it could redirect network traffic to other nodes, or reallocate processing power and memory to other tasks.

5. **Load balancing**: In a distributed system, there may be multiple nodes that are capable of executing a particular task.
   A leader process can help in load balancing by assigning tasks to the nodes that have the least workload.

6. **Security**: In a distributed system, security is a critical concern.
   A leader process can help in ensuring security by authenticating the nodes and controlling access to resources.

In summary, a leader process is essential in a distributed system because it helps in coordinating the activities of all the nodes, ensuring consistency, achieving fault tolerance, load balancing, and ensuring security.

2. What is election in Distributed System?

- In a distributed system, an election is a process where a group of nodes or processes selects a leader among themselves to coordinate their activities.
- A leader may fail.
- The election process is necessary to ensure that there is always a leader in the system that can provide a centralized point of control and coordination.
- The election process typically begins when a node or process identifies that the current leader is no longer functioning correctly or has failed.
- The node or process that detects the failure will initiate the election process by sending a message to all the other nodes in the system to inform them that a new leader needs to be selected.
- The election process can be challenging to implement correctly in a distributed system, and there are several algorithms available to help ensure that the election process is fair and efficient.
- Examples of election algorithms include Bully Algorithm, Ring Algorithm.

(4 times) Imp

3. Why do we need to perform election in a Distributed System?

OR

Explain the need of election in Distributed System.

Ans: We need to perform an election in a Distributed System to elect a leader.

In a distributed system, an election is a process where a group of nodes or processes selects a leader among themselves to coordinate their activities.

The election process ensures that there is always a leader in the system to coordinate the activities of the other nodes and to maintain the system's overall functionality.

Performing an election in a distributed system is important for several reasons, including:

1. **Leader Failure**:
   - ➢ In a distributed system, the node or process that acts as the leader is responsible for coordinating the activities of the other nodes.
   - ➢ If the leader fails or becomes unavailable, the system needs to elect a new leader to ensure that the system continues to function correctly.

2. **Load Balancing**:
   - ➢ In a distributed system, it is often desirable to distribute the workload among the nodes.
   - ➢ The election process can help to identify the node with the most resources, such as processing power and memory, to act

as the leader and coordinate the workload distribution among the nodes.

3. **Consistency**:
   - In a distributed system, there may be multiple nodes that hold copies of the same data.
   - To ensure consistency, the nodes need to coordinate their activities and agree on a common version of the data.
   - The leader can play a crucial role in this coordination, and the election process can ensure that a new leader is selected if the current leader fails.
4. **Resource allocation**:
   - The leader could monitor the usage of network bandwidth, memory, and storage across all the nodes in the system, and allocate these resources fairly and efficiently.
   - For example, if one node is using a lot of network bandwidth, the leader could temporarily limit its usage to ensure that other nodes have sufficient bandwidth for their own tasks.

5. **Fault Tolerance**:
   - A distributed system needs to be fault-tolerant and able to continue functioning even if one or more nodes fail.
   - The election process can help to identify a new leader if the current leader fails, ensuring that the system continues to function correctly.

6. **Security**:
   - The election process can also help to ensure the security of the distributed system by ensuring that only authorized nodes are allowed to participate in the election process and become the leader. This can help to prevent unauthorized nodes from taking control of the system.

4. Explain the importance of election algorithm with example of uses in distributed System.

Ans: An election algorithm is a set of rules and procedures used to elect a leader in a distributed system.

Election algorithms are important because they help ensure that a new leader is selected when the current leader fails, and the distributed system can continue to function correctly.

Assumptions in Election Algorithm:

- Each process has a unique number to distinguish them.
- Processes know each other's process number.

Here are two examples election algorithms in distributed systems.

1. **Bully Algorithm**:
    - The Bully Algorithm is an election algorithm used in distributed systems where nodes have a hierarchical structure.
    - In this algorithm, the node that identifies the current leader has crashed, initiates the election process, and each node sends a message to all other nodes with a higher priority.
    - The node with the highest priority becomes the new leader.
    - This algorithm is useful in systems where there is a clear hierarchy of nodes, such as in a local area network.

2. **Ring Algorithm**:
    - The Ring Algorithm is another election algorithm used in distributed systems where nodes are arranged in a circular manner.
    - In this algorithm, each node sends a message to its neighbor in a clockwise direction adding their id in the list.
    - The node with the highest ID becomes the new leader.
    - This algorithm is useful in systems where nodes are arranged in a circular manner, such as in a token ring network.

5. Explain Bully Election Algorithm.

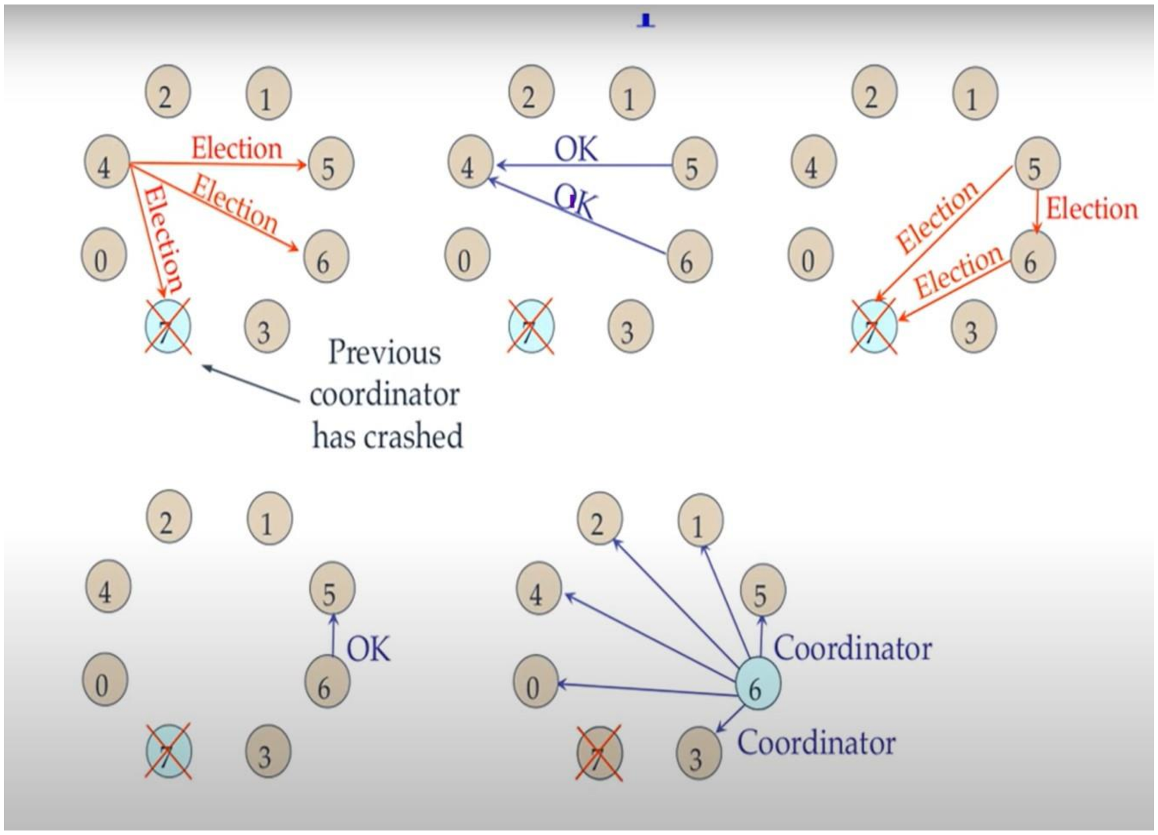
- The Bully Algorithm is an election algorithm used in distributed systems where nodes have a hierarchical structure.
- In this algorithm, the node that identifies the current leader has crashed, initiates the election process, and each node sends a message to all other nodes with a higher priority.
- The node with the highest priority becomes the new leader.
- This algorithm is useful in systems where there is a clear hierarchy of nodes, such as in a local area network.

Algorithm

1. When a Process P, notices that the coordinator is no longer responding to requests, it initiates an election.
    a. P sends an **election** message to all processes with higher numbers.
    b. If no one responds, P wins the election and becomes a coordinator.
    c. If one of the higher process answers, higher process takes over. P's job is done.
2. When a process gets an **election** message from one of its lower-numbered colleagues:
    a. Receiver sends an **Ok** message back to the sender to indicate that he is alive and will take over.
    b. Eventually, all processes give up and one will be remaining, and that one is the new coordinator.
    c. The new coordinator announces its victory by sending all processes a CO-ORDINATOR message telling them it is the new coordinator.
3. If a process that was previously down comes back:
    a. It holds an election.
    b. If it happens to be the highest process currently running, it will win the election and take over the coordinator's job.

"Biggest guy" always wins and hence the name "bully" algorithm.

Example:



- ➢ First, Process 4 sends requests to its coordinator and doesn't get a respond within a specified time. It assumes the coordinator is dead.
- ➢ Now it will send election message to its higher processes 5, 6 and 7.
- ➢ Process 5 and 6 sends OK message to 4 and now the job of 4 is done here.
- ➢ Process 5 sends election message to 6 and 7, and process 6 sends election message to 7.
- ➢ Process 6 sends ok message to 5, and the job of process 5 is done here.
- ➢ Process 6 doesn't receive any message from its higher node, so it is now the coordinator.
- ➢ It announces its victory to all the processes in the system.

6. Can multiple election appear in bully algorithm?

- In the Bully algorithm, multiple elections can occur if multiple nodes detect that the coordinator is dead or not responding simultaneously.
- When a node detects that the coordinator is dead, it sends an election message to all other nodes with a higher priority number.

- If multiple nodes detect that the coordinator is dead at the same time, they may all initiate an election simultaneously, resulting in multiple elections.
- However, the Bully algorithm is designed to handle this scenario by ensuring that only one election is successful.

- The node with the highest priority number will win the election and become the new coordinator.
- Other nodes that initiated elections will recognize that they are not the highest priority node and withdraw from the election.
- The winning node will then send a coordinator message to all other nodes to inform them of its new role.

- Therefore, while multiple elections can occur simultaneously in the Bully algorithm, only one will be successful, and the node with the highest priority number will become the new coordinator.

7. Explain the Chang and Roberts election algorithm. (Ring Algorithm)

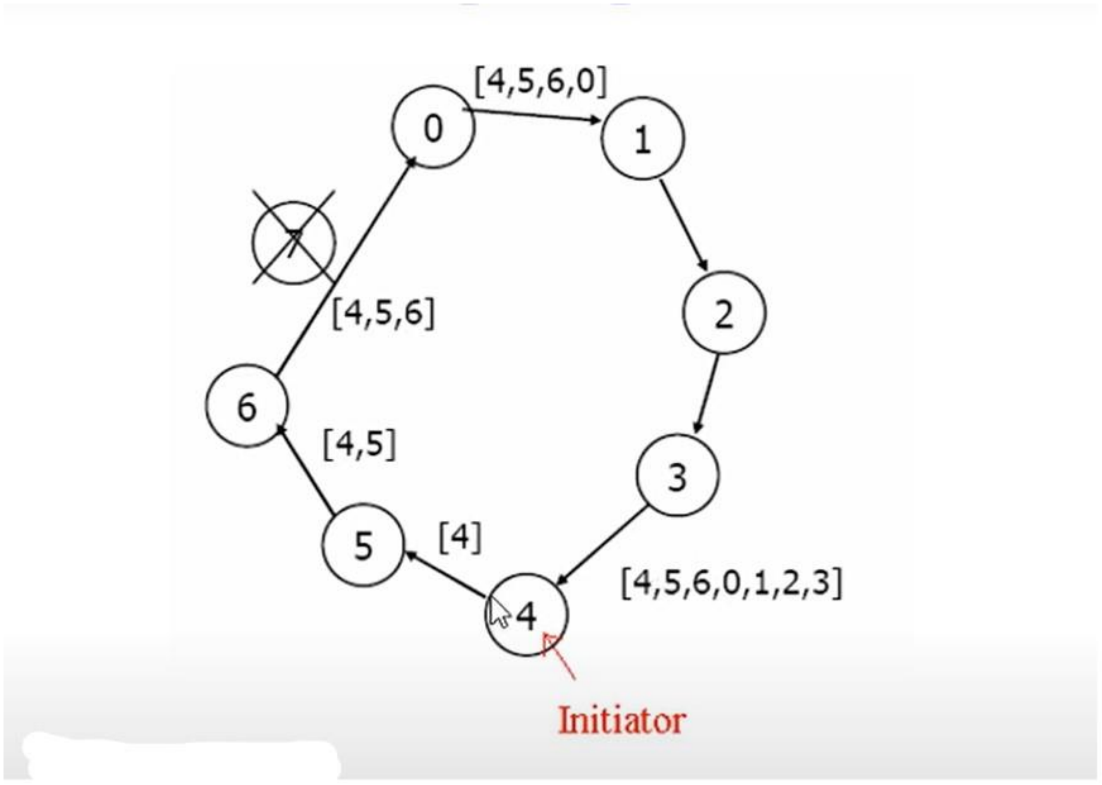
- The Ring Algorithm is another election algorithm used in distributed systems where nodes are arranged in a circular manner.
- Simply, this algorithm used to elect a leader in a ring network topology.
- In a ring network, each node is connected to its immediate neighbors, forming a circular topology.

- In this algorithm, each node sends a message to its neighbor in a clockwise direction adding their id in the list.
- The node with the highest ID becomes the new leader.

Algorithm:

1. When a Process P, notices that the coordinator is no longer responding to requests, it initiates an election.
    a. P builds an election message (containing its own process number)
    b. Sends the message to its neighbor in clockwise direction (if the neighbor is down, sender skips over it and goes to the next running process along the ring).
    c. At each step, sender adds its own process number to the list in the message.
2. When the message gets back to the process that initiated the election.
    a. Message comes back to the initiator process P.
    b. In the queue the process with maximum Id number wins.
    c. Initiator announces the winner by sending another message around the ring.

Example:



- First, Process 4 sends requests to its coordinator and doesn’t get a respond within a specified time. It assumes the coordinator is dead.
- It builds an election messages and sends the message in clockwise direction to process 5.
- Process 5 adds its id in the list.
- Similarly, all the process adds their ids in the list, and it comes back to the initiator process i.e., 4.
- The maximum id in the list is 6.
- Hence, 6 is the new coordinator.
- Initiator process sends leader announcement message to all the process in the ring. The message contains the id of the new leader i.e., 6.

(3 times)

8. How multiple ongoing election is handled using ring-based election algorithm.

OR

How can we resolve the multiple active elections at the same time in ring election algorithm?

- In the Ring Election algorithm, multiple ongoing elections can be handled using a priority scheme.
- Each node in the ring is assigned a priority level based on its ID.
- The priority level is used to determine which election should be given priority in case of multiple ongoing elections.

- If a node detects that the coordinator is dead or not responding, it starts a new election by sending a message to its neighbor in one direction.
- The message contains the ID of the node that initiated the election and the priority level of the ongoing election.

- If a node receives another message for an ongoing election with a higher priority level, it stops its ongoing election and participates in the higher priority election.
- The node sends a message to its neighbor in the opposite direction to inform them that it is no longer participating in the ongoing election and forwards the message for the higher priority election to its neighbor in the same direction.

- Once the new leader is identified, the node that initiated the election with the highest priority level sends a leader announcement message to all nodes in the ring to inform them of the new leader.

- The priority scheme ensures that only one election is ongoing at any given time, and the election with the highest priority level is given priority over other ongoing elections.
- Therefore, multiple ongoing elections can be handled using the Ring Election algorithm by using a priority scheme to give priority to the election with the highest priority level.

Note 1:

Multiple ongoing elections will choose same leader. What is the issue?

- One issue is the increased network traffic and communication overhead caused by multiple ongoing elections.
- Each election involves a series of messages being sent between nodes in the network, and if multiple elections are happening simultaneously, the network can become congested, leading to delays and reduced performance.

Note 2:

- A process ID is a unique identifier assigned to each process in the system.
- On the other hand, a priority number is used to prioritize the election messages in distributed systems.
- In some distributed systems, the priority number may be based on the process ID, but this is not always the case.
  The priority number can be assigned in any way that makes sense for the system, such as based on the node's computing power, network bandwidth, or other relevant factors.

(2 times)

9. Compare and Contrast Ring algorithm with Bully Algorithm.

Ans: Explain both (Que no 5 and

(6 times) (VVIP)

10. What is distributed mutual exclusion?

OR

What is mutual exclusion in Distributed system?

- Mutual exclusion is a mechanism used in distributed systems to ensure that only one process or thread can access a shared resource or critical section at any given time.
- In a distributed system, multiple processes may be running simultaneously on different nodes, and they may need to access shared resources such as databases, files, or network devices.
- However, allowing multiple processes to access the same resource simultaneously can lead to conflicts, data inconsistencies, and other problems.

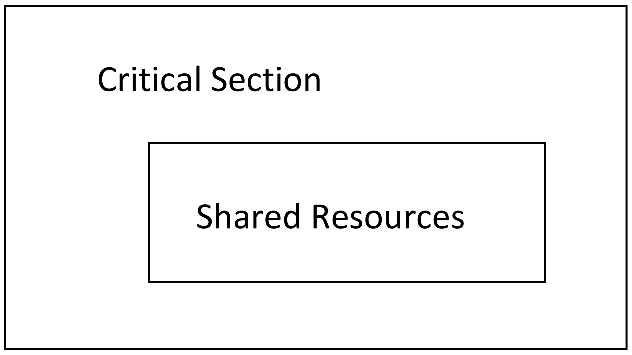
- Distributed mutual exclusion provides a way for processes to coordinate access to shared resources in a distributed system.
- It works by ensuring that only one process is granted access to a resource at a time, while other processes are prevented from accessing the resource until the current process releases it.
- It is an important mechanism in distributed systems to ensure correct operation and data consistency.

- It is used in various applications such as distributed databases, distributed file systems, and distributed resource management systems.

Critical Section: It is a section which is accessible only to one process at a time.

Shared Resource: it is a resource that can be accessed and used by multiple processes. Example: Printer, Files, Database, etc.

Mutual Exclusion: It makes sure process access shared resources or data in serialized way.



P1, P2, … Pn are waiting in a queue to access shared resources.

A critical section is a part of a program or code that accesses shared resources, and only one process or thread can enter the critical section at a time. The shared resources are typically kept inside the critical section to ensure that they are accessed in a mutually exclusive manner.

11. Why do we need a mutual exclusion?

Ans: We need mutual exclusion in distributed systems for several reasons:

a. **Data consistency:**
When multiple processes are accessing and modifying the same shared resource simultaneously, there is a risk of data inconsistency.
Mutual exclusion ensures that only one process can access the shared resource at any given time, preventing conflicts and inconsistencies.

b. **Resource allocation:**
In a distributed system, resources such as network devices, processors, and memory are shared among processes.
Mutual exclusion ensures that resources are allocated fairly and efficiently among processes.

c. **Deadlocks:**
Without mutual exclusion, multiple processes may get stuck in a deadlock situation, where each process is waiting for another process to release a resource.
Mutual exclusion helps prevent deadlocks by allowing processes to request and release resources in an orderly manner.

d. **Security:**
Mutual exclusion can be used to ensure that only authorized processes can access sensitive resources in a distributed system.

12. What are the basic requirements for the distributed mutual exclusion?

Ans: The basic requirements for distributed mutual exclusion include:

a. **Correctness**: The distributed mutual exclusion algorithm must ensure that only one process can access the shared resource at any given time, while other processes are blocked from accessing the resource.

b. **Deadlock avoidance:** The algorithm should avoid the possibility of deadlocks, where processes are waiting indefinitely for each other to release resources.

c. **Scalability**: The algorithm should be able to handle a large number of processes and shared resources in a distributed system.

d. **Fault tolerance:** The algorithm should be able to handle process and network failures and should continue to function correctly even in the presence of failures.

e. **Efficiency**: The algorithm should minimize the amount of time and network resources required for process coordination.

f. **Fairness**: The algorithm should ensure that processes are granted access to the shared resource in a fair and equitable manner.

Note:

Distributed Mutual Exclusion is typically achieved through the use of a centralized server or a distributed algorithm.

In a **centralized server approach**, a dedicated server is responsible for managing access to shared resources.
Processes request access to the server, which grants access to one process at a time while blocking access to other processes.
This approach is simple and easy to implement but can be a single point of failure and can cause performance bottlenecks.

In a **distributed algorithm approach**, each process in the system participates in a protocol that allows them to coordinate access to shared resources.
The algorithm ensures that only one process can access the resource at any given time, while others are prevented from accessing the resource until the current process releases it.
Examples: Ricart-Agrawala algorithm, Maekawa's algorithm, Lamport’s mutual exclusion algorithm.

13. What are the major advantages and disadvantages of centralized controlled system?

Ans: Centralized control systems have some advantages and disadvantages, which are:

Advantages:

1. **Simplicity**: Centralized control systems are relatively simple to implement because there is only one control point, the coordinator. This makes it easier to manage and modify the system.
2. **Coordination**: Centralized control systems provide a single point of control that can coordinate the activities of all the processes in the system. This makes it easier to ensure that all processes are working towards the same goal and that there are no conflicts or duplications.
3. **Security**: Centralized control systems can be more secure than decentralized systems because access to the central control point can be tightly controlled. This can prevent unauthorized access to critical resources and improve the overall security of the system.

Disadvantages:

1. **Single point of failure**: Centralized control systems have a single point of failure, the coordinator. If the coordinator fails, the entire system may fail or become unusable.
2. **Scalability**: Centralized control systems may not be easily scalable because the coordinator may become overwhelmed with requests as the number of processes in the system grows. This can lead to delays and decreased system performance.
3. **Cost**: Centralized control systems may be more expensive to implement because they require a central control point and may need additional hardware or software to ensure that the system is reliable and can handle a large number of requests.

14. Central Coordinator Algorithm in Mutual Exclusion. (Not asked yet)

Ans: The Central Coordinator Algorithm is a mutual exclusion algorithm that uses a centralized coordinator to manage access to shared resources.

The central coordinator algorithm ensures mutual exclusion by only granting access to the shared resource to one process at a time.

In this algorithm, all processes communicate with a central coordinator process to request access to the shared resource.

The coordinator grants access to the processes in a way that ensures mutual exclusion.

Algorithm:

a. Each process sends a request message to the coordinator when it wants to enter the critical section.
b. The coordinator receives the request messages and determines which process should be granted access to the critical section based on some criteria. For example, it may use a timestamp or priority scheme to decide which process gets access first.
c. The coordinator sends a reply message to the selected process, granting it access to the critical section.
d. The selected process enters the critical section, performs its task, and then releases the resource by sending a release message to the coordinator.
e. The coordinator updates its state to reflect that the resource is now available and processes the next request message.

15. Explain token and non-token based mutual exclusion algorithm.

Ans: In mutual exclusion algorithms, token-based and non-token based algorithms are two different approaches to ensure that only one process can access a shared resource at a time.

In a token-based algorithm

- A logical token is passed among the processes.
- A token represents access right to the shared resources.
- The process which holds the token is granted access to critical section.
- The token moves around in a circular or linear fashion, and only the process that holds the token can enter the critical section.
- Once the process finishes its critical section, it passes the token to the next process in the sequence, which then gains access to the shared resource.
- The advantage of a token-based algorithm is that it guarantees fairness, as each process gets a turn to access the resource.
- It includes:
    a. Ricart-Agrawala second Algorithm
    b. Token Ring Algorithm

In a non-token-based algorithm,

- The processes do not pass any token to each other.
- Instead, they use some form of signaling mechanism to determine which process can access the shared resource.
- It includes:
    a. Central Coordinator Algorithm
    b. Ricart-Agrawala Algorithm
    c. Lamport’s mutual exclusion Algorithm

Note: Ricart-Agrawala second Algorithm and Ricart-Agrawala Algorithm are different. Don’t get confused.

(2 times)

16. Explain Ricart Agrawala token-based algorithm for distributed mutual exclusion.

Ans: **Ricart Agrawala token-based** algorithm **means Ricart-Agrawala second** Algorithm.

- In the Ricart-Agrawala second algorithm, there is only one token in the entire system that is used to coordinate access to the critical section.
- The token is initially held by one of the processes in the system, and only that process can enter the critical section.
- When a process that does not hold the token wants to enter the critical section, it must first request the token from the process that currently holds it.

- A process is allowed to enter the critical section when it got the token.
- In order to get the token, it sends a request to all other processes competing for the same resource.
- The request message consists of the requesting process' timestamp (logical clock) and its identifier.

- Initially the token is assigned arbitrarily to one of the processes.
- When a process Pi leaves a critical section, it passes the token to one of the processes which are waiting for it; this will be the first process Pj.
- If no process is waiting, Pi retains the token (and is allowed to enter the CS if it needs); it will pass over the token as result of an incoming request.

RELEASED: The RELEASED token state indicates that the process does not currently hold the token and is not waiting for it.

WANTED: The WANTED token state indicates that the process wants to enter the critical section and is waiting for the token.

Algorithm:

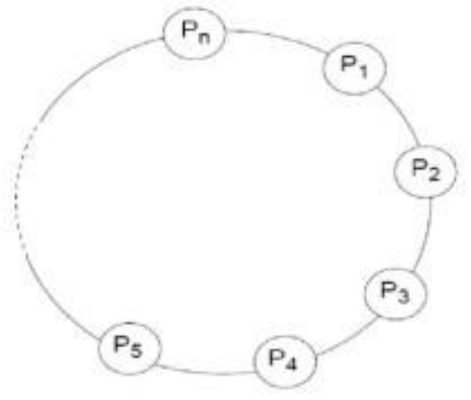
1. Initialization
   a. Each process Pi initializes its local clock and token state.
2. Requesting access:
   a. When a process Pi wants to enter the critical section, it sets its token state to WANTED and sends a REQUEST message to all other processes.
   b. If a process Pj receives a REQUEST message from process Pi:
      i. If process Pj is currently in its critical section or its token state is WANTED, it adds Pi's ID to its queue of outstanding requests.
      ii. Otherwise, if Pj's token state is RELEASED, it sends a REPLY message to process Pi.
3. Entering the critical section:
   a. A process Pi can enter the critical section when it has received a REPLY message from all other processes, and it has the token.
4. Releasing access:
   a. When process Pi exits the critical section, it releases the token and sends RELEASE messages to all other processes.
   b. If there are outstanding requests in Pi's queue, Pi forwards the token to the process with the highest request timestamp in its queue, removes that process from the queue, and sets its token state to RELEASED.

Advantages

1. It requires only n-1 requests and one reply.

17. Explain Token Ring Algorithm. (Not Asked)

The n processes are arranged in a logical ring as shown in given figure:



Algorithm:

1. Token is initially given to one process.

2. When a process requires to enter CS, it waits until it gets token from its left neighbor and retains it. After it left CS, it passes token to its neighbor in clockwise direction.

3. If a process gets token but does not require to enter CS, it immediately passes token along the ring.

Problem:

1. It adds load to the network as token should be passed even the process do not need it.

2. If one process fails, no progress is possible until the faulty process is extracted from the ring.

3. Election process should be done if the process holding the token fails.

18. Explain the token-based algorithm to obtain mutual exclusion.

Explain either Ricart-Agrawala second algorithm or Ring Token Algorithm.

Que no 16 or Que no 17

Non-Token Based Algorithms

1. Lamport's mutual exclusion Algorithm.
2. Ricart-Agrawala Algorithm.

19. Explain Lamport's distributed mutual exclusion algorithm.

Ans: Lamport's distributed mutual exclusion algorithm is a non-token-based algorithm for achieving mutual exclusion in a distributed system.

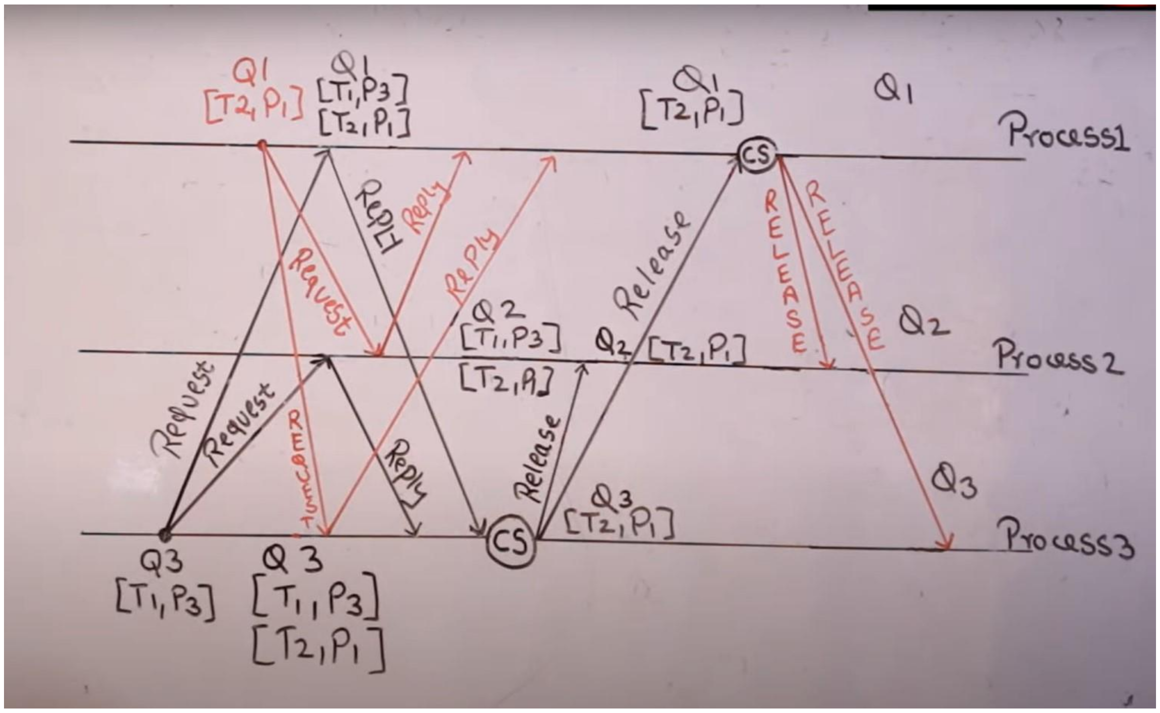
Three types of messages (REQUEST, REPLY and RELEASE) are used and communication channels are assumed to follow FIFO order.

Algorithm:

1. Each process maintains a logical clock (Timestamp) that is used to order events across different processes.
2. When a process wants to enter the critical section, it sends a REQUEST message to all other processes in the system, along with its Timestamp.
3. Upon receiving a request message, a process compares the timestamp of the incoming message with its own timestamp to determine which process has the right to access the shared resource first.
4. If the process that received the request has the higher timestamp, it adds the request to its own queue and sends a reply message back to the requesting process, along with its own current timestamp.

5. If the process that received the request has the lower timestamp, it enters its critical section.
6. The requesting process can only enter the critical section if:
- It is in first in the queue.
- It received REPLY from all the processes.
7. It should send RELEASE message to all the processes after exiting the critical section.

Example:



In the example above,

- There are three Processes P1, P2 and P3.
- Each process has its own queue. P1, P2, P3 has Q1, Q2 and Q3 as their queue respectively.
- CS is a critical section.

- Process P3 wants to enter the critical section. So, it needs to send REQUEST to all the processes i.e., P1 and P2 with its own timestamp.

- It adds its timestamp and Process id in queue. Q3 [T1, T3]
- Since it is the first one to send the request the timestamp will be T1.

- P2 doesn't want to enter critical section so it sends REPLY to P3 as soon as it receives REQUEST.
- But P2 adds P3 in its queue. Q2 [T1, P3]

- After some time, P1 also wants to enter the critical section and it adds itself in its queue. Q1 [T2, P1].
- It also sends its request to all the processes.

- P1 receives a REQUEST from P1 so it compares the timestamp of P3 and itself. Since the timestamp of P3 is lower, P1 adds P3 in its queue in ascending order of timestamp.
- Updated Queue: Q1 [T1, P3], [T2, P1]
- And now it sends REPLY to P3.

- On the other hand, P3 also RECEIVES request from P1, so it compares the timestamp of P1 and itself. It adds P1 in queue in ascending order.
- Updated Queue: Q1 [T1, P3], [T2, P1]

- P3 receives REPLY from P1 of its earlier REQUEST, P3 received REPLY from all the processes.
- Checks the two conditions:
  -It is in first in the queue.
  -It received REPLY from all the processes.
- Both satisfies, it can now enter the critical section.

- Once, P3 completes its processing and leaves the Critical Section, it needs to send RELEASE message to all the processes.
- All the processes remove P3 from their queue after receiving the RELEASE message.

- P1 received the RELEASE message from P3, it now checks the condition.
- It satisfies the condition to enter critical section.
- It enters the critical section, completes its processing, and sends RELEASE message to all the processes.

- Now all the process removes P1 from their queue.

Note:

Higher Timestamp: Occurs earlier. Time 3:00

Lower Timestamp: Occurs later. Time 2:00

If timestamps are same, they compares the process id.

(Asked 4 times)

20. Explain Ricart-Agrawala algorithm for distributed mutual exclusion.

Ans: Ricart–Agrawala algorithm is an algorithm to for mutual exclusion in a distributed system proposed by Glenn Ricart and Ashok Agrawala.

This algorithm is an extension and optimization of Lamport's Distributed Mutual Exclusion Algorithm. Like Lamport's

Algorithm, it also follows permission-based approach to ensure mutual exclusion.

In this algorithm, two type of messages (REQUEST and REPLY) are used and communication channels are assumed to follow FIFO order.

A process sends a REQUEST message to all other processes to get their permission to enter critical section.

A process sends a REPLY message to other process to give its permission to enter the critical section.

The request message contains the timestamp of the requesting process.

Timestamp is used to determine priority of critical section requests. Smaller timestamp gets high priority over larger timestamp. The execution of critical section request is always in the order of their timestamp.

Algorithm

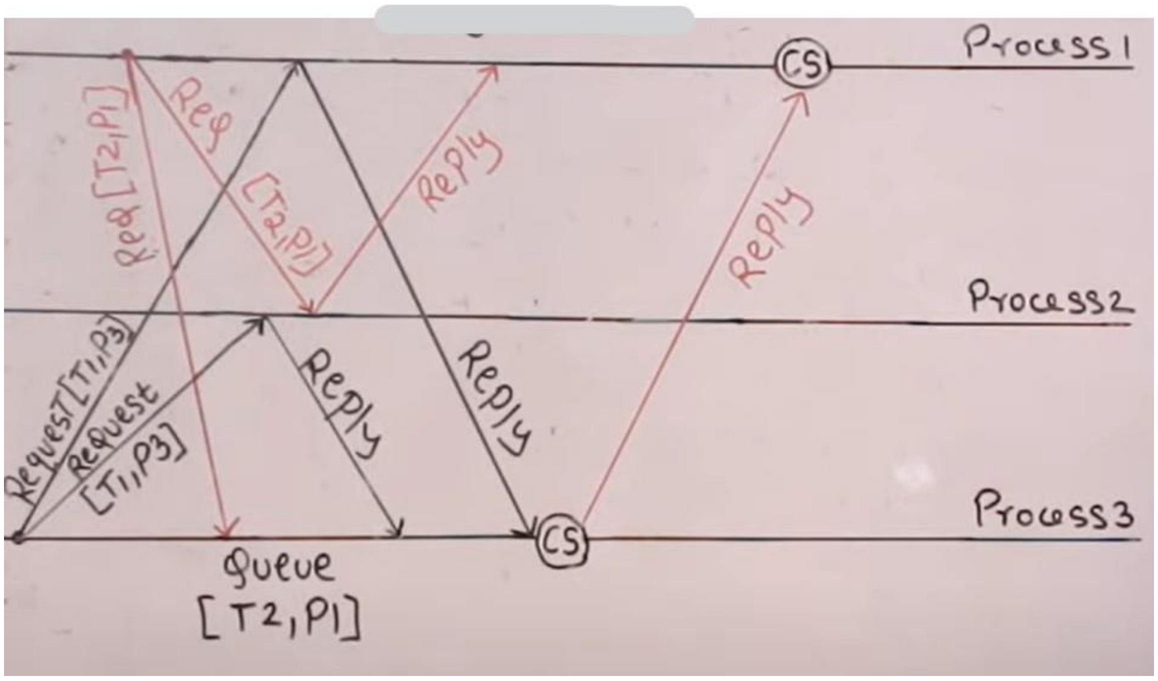
1. Each process maintains a logical clock (Timestamp) that is used to order events across different processes.
2. When a process wants to enter the critical section, it sends a REQUEST message to all other processes in the system, along with its Timestamp.
3. Upon receiving a request message, a process compares the timestamp of the incoming message with its own timestamp to

determine which process has the right to access the shared resource first.

4. If the process that received the request has the higher timestamp, it simply sends a reply message back to the requesting process, along with its own current timestamp.
   Otherwise, if the process that received the request has lower timestamp, it adds the requesting process to its queue of pending request, enters the Critical Section and only sends it reply after exiting the critical section.
5. The requesting process can only enter the critical section if:
   1. It received REPLY from all the processes.
6. On exiting the Critical Section, the process sends REPLY message to only the process which is in its queue.

Suppose process P3 wants to enter Critical Section. And it receives a REQUEST from process P1 that it also wants to enter Critical Section. Now process P3 will compare its timestamp with process P1. If the timestamp of P1 is higher than P3, then P3 will not respond to it.

Example:



In the example above,

- There are three Processes P1, P2 and P3.
- Each process has its own queue. P1, P2, P3 has Q1, Q2 and Q3 as their queue respectively.
- CS is a critical section.

- Process P3 wants to enter the critical section. So, it sends REQUEST to all the processes i.e., P1 and P2 with its own timestamp.
- P1 also wants to enter the critical section so it also sends REQUEST to all the processes.
- P1 receives request from P3 so it compares it timestamp with P3, since the timestamp of P3 is smaller than P1. P1 sends a REPLY message to P1.
- P3 also receives a REQUEST from P1, and it compares its timestamp with P1, since its timestamp is lower than P1, it doesn't sends a REPLY message.

- It adds P1 in its queue so that it will sends REPLY after it completes it processing in Critical Section.
- Queue: Q3[T2, P1].

- P2 doesn't needs to enter Critical Section so it sends REPLY message to both P1 and P3.

- P3 received reply message from all the processes so now it can enter the critical section.
- P3 exits critical section. It sends a REPLY message to P1 after checking its queue.

- Now P1 can also enter critical section after receiving REPLY from P3.

(Asked 2 times)

25. Difference Between Lamport's and Ricart Agrawala Mutual Exclusion Algorithm.

In case of Lamport's Algorithm

1. There were two condition to enter Critical Section:
    a. Process that wants to enter critical section has to be in the top of its queue.
    b. It should receive REPLY message from all the processes.
2. There were three types of messages: REQUEST, REPLY and RELEASE
3. Additional to Ricart Agrawal, it should send RELEASE message to all the processes after coming out of Critical Section.

In case of Ricart Agrawala Algorithm

1. There is only one condition to enter Critical Section:
    a. Process that wants to enter critical section should receive REPLY message from all the processes.
2. There were two types of messages: REQUEST and REPLY
3. It doesn't need to send RELEASE message like in Lamport's algorithm.

Chapter: 5
Agreement in Distributed System

2 or 3 Questions (15-23 marks)

(Asked 2 times)

1. What is error, fault, failure, and fault tolerant?

Ans: In the context of distributed systems, the terms "error," "fault," "failure," and "fault-tolerant" are used to describe different aspects of system behavior and reliability.

Error:

- An "error" refers to a human mistake or incorrect use of the system, such as providing incorrect input data or using a feature in an unintended way.
- Example: A user enters an incorrect password while trying to log in to a system.

Fault:

- A "fault" refers to a defect in the system's software or hardware that causes it to behave incorrectly or produce incorrect results.
- A fault may be caused by a software bug, hardware malfunction, or other issues.

Failure:

- A "failure" occurs when the system is unable to perform its intended function.
- This can be caused by one or more faults in the system.
- A system failure can result in data loss, downtime, or other negative consequences.
- Example: Failure: A network outage causes a distributed system to be unavailable to users.

Fault tolerance:

- "Fault tolerance" refers to a system's ability to continue operating in the presence of faults or failures.
- A fault-tolerant system is designed to detect and recover from errors and faults, either by masking them or by continuing to operate in a degraded state.
- For example, a distributed database might have multiple replicas of data across different servers, so that if one server fails, the data can still be accessed from another replica.

In summary, errors are human mistakes, faults are defects in the system, failures are when the system is unable to perform its intended function, and fault tolerance is the ability of the system to continue operating in the presence of faults and failures.

Q. Differentiate errors and faults.

(Asked 3 times)

2. What is Replication? Why do we need Replication?

OR

What is the reason for replication in distributed system?

- Replication in distributed systems refers to the process of creating and maintaining multiple copies of data, services, or components across different nodes in the system.
- The goal of replication is to increase system availability, performance, and reliability by providing redundancy and fault tolerance.
- Simply, data Replication is the process of generating numerous copies of data. We then store these copies also called replicas in various locations while keeping the data updated and synced with the source.
- The data replicates can be stored on on-site and off-site servers, as well as cloud-based hosts, or all within the same system.

Replication is important because:

- Availability of the data is an important factor often accomplished by data replication.
- If a node stops working, the distributed network will still work fine due to its replicas which will be there. Thus, it increases the fault tolerance of the system.
- It also helps in load sharing where loads on a server are shared among different replicas.
- It enhances the availability of the data. If the replicas are created and data is stored near to the consumers, it would be easier and faster to fetch data.

Replication can be used for various purposes, including: (Importance)

1. **Improving system performance:**
   By replicating frequently accessed data or services across multiple nodes, the system can reduce response times and improve overall performance.

2. **Increasing system availability:**
   By replicating data or services across multiple nodes, the system can continue to function even if one or more nodes fail or become unavailable.
   Clients can still access the replicated data or services from other nodes in the system, improving overall availability.

3. **Reduced latency:**
   By replicating data or services closer to the clients that need them, the system can reduce the time it takes to access them.
   This can improve performance and reduce network traffic.

4. **Increased fault tolerance:**
   If a node fails, clients can still access the replicated data or services from other nodes.
   This can help ensure that the system remains available and responsive even in the face of failures.

5. **Enhancing system reliability:**
   Replication can help prevent data loss or corruption by ensuring that multiple copies of data are available in case of failures or errors.

6. **Facilitating load balancing:**
   Replication can be used to distribute load across multiple nodes, allowing the system to handle more requests and users.

3. What do you mean by object replication?

Object replication is a technique used in distributed systems to improve system availability and fault-tolerance by creating multiple copies of an object and storing them in different locations.

In object replication, the original object is referred to as the primary copy, and the copies are referred to as replicas.

When a client request is made for an object, the request can be handled by any of the replicas.

The replicas work together to provide a consistent view of the object, which means that updates made to one replica are propagated to all other replicas.

Object replication can provide several benefits, including:

1. Improving system performance
2. Increasing system availability
3. Reduced latency.
4. Increased fault tolerance
5. Enhancing system reliability
6. Facilitating load balancing

(Asked 5 times) Imp.

4. Explain Active and Passive Replication Technique:

There are two types of replications, and they are:

1. **Active Replication**
   - ➢ In active replication each client request is processed by all the servers/replicas.
   - ➢ In this, all replicas actively participate in the execution of requests.
   - ➢ Each replica independently executes the request and compares its result with other replicas to ensure consistency.
   - ➢ If a replica fails or produces an inconsistent result, the other replicas can correct it or remove it from the system.
   - ➢ Active replication is often used in systems where availability is critical, such as web services or distributed applications.
   - ➢ It ensures that the system can continue to function even if one or more replicas fail or become unavailable.

   Advantage
   - ➢ Even if a node fails, it will be easily handled by replicas of that node.

   Disadvantage
   - ➢ It increases resource consumption. The greater the number of replicas, the greater the memory needed.
   - ➢ It increases the time complexity. If some change is done on one replica it should also be done in all others.

2. Passive Replication

- In passive replication there is only one server (called primary replica) that processes client requests.
- After processing a request, the primary replica updates the state on the other (backup) replicas and sends back the response to the client.
- If the primary replica fails, one of the backup replicas takes its place.
- Passive replication is often used in systems where consistency is critical, such as financial systems or databases.
- It ensures that all replicas are consistent with each other by having the primary replica act as the source of truth.

Advantage

- The resource consumption is less as backup servers only come into play when the primary server fails.
- The time complexity of this is also less as there's no need for updating in all the nodes replicas, unlike active replication.

Disadvantage

- The disadvantage of passive replication compared to active is that in case of failure the response is delayed.

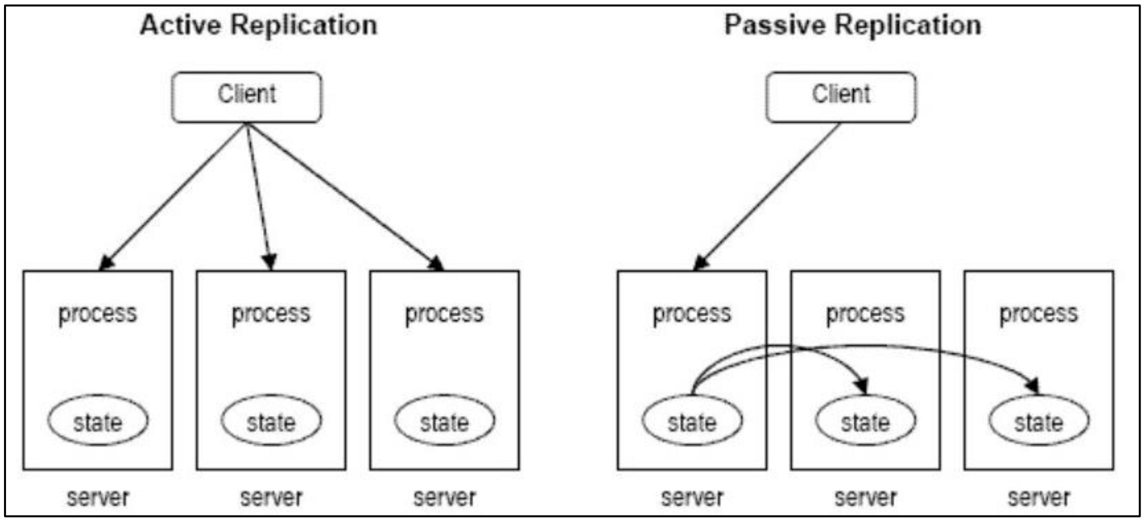


Fig: Types of Replications

(Asked 5 times)

5. What is meaning of fault tolerant system?

OR

What is fault tolerant service?

- ➢ A fault-tolerant system(service) is a type of system that is designed to continue functioning even if there is a failure in one or more components of the system.
- ➢ The primary goal of a fault-tolerant system is to provide high availability and reliability by preventing or minimizing the impact of failures on the system.
- ➢ To achieve fault tolerance, fault-tolerant systems typically employ redundancy, which means having backup components or systems that can take over in the event of a failure.
- ➢ This redundancy can be implemented at different levels of the system, from the hardware level to the software level, and can involve various techniques, such as error detection and correction, replication, and failover.
- ➢ Fault-tolerant systems are commonly used in critical applications where system failure can have serious consequences, such as in aviation, healthcare, finance, and telecommunications.
- ➢ By ensuring high availability and reliability, fault-tolerant services can help to minimize downtime, improve customer satisfaction, and prevent financial losses.

(Asked 2 times)

6. Explain the different approaches of fault tolerance.

There are several approaches to achieving fault tolerance, and the choice of approach depends on the specific needs and requirements of the system. Some of the commonly used approaches are:

1. **Redundancy**:
   This approach involves creating backup components or systems that can take over in the event of a failure.
   For example, a system may have redundant servers, so that if one fails, the other can take over.
   Redundancy can be implemented at different levels, from the hardware level to the software level.

2. **Replication**:
   This approach involves creating multiple copies of the same component or system and running them simultaneously.
   Each copy is identical, and if one copy fails, the others can take over.
   This approach is commonly used in distributed systems, where data or services are replicated across multiple nodes.

3. **Failover**:
   This approach involves automatically switching to a backup component or system in the event of a failure.
   Failover can be used in conjunction with other fault tolerance approaches, such as replication or redundancy.
   There are three types of failovers: Cold, Warm and Hot failover.

4. **Checkpoints**:
   This approach involves periodically saving the state of the system so that if a failure occurs, the system can be restored to the last known good state.
   Checkpoints can be taken at regular intervals and can be used in conjunction with other fault tolerance approaches.

5. **Error detection and correction**: This approach involves detecting errors in the system and correcting them automatically or with human intervention.
   Error detection and correction can be implemented at different levels, from the hardware level to the software level.

Overall, the different approaches to fault tolerance are used to ensure that critical systems remain available and reliable, even in the event of hardware or software failures.

Redundancy Vs Replication

Redundancy allows us to duplicate components of our system.

Replication is what makes those duplicated values actually useful to us.

Replication is the process of creating a redundant node and ensuring that it is identical to all the copies.

(Asked 2 times)

7. Explain fault tolerant mechanism using active and passive replication.

Ans: Active and passive replication are two common approaches to achieving fault tolerance in distributed systems.

(Explain Que no 4)

8. What is failover. Explain cold failover, warm failover, and hot failover.

Failover is a process in which a backup component or system takes over when the primary component or system fails.

his is done to ensure that the system remains available and operational in the event of a failure.

There are three main types of failovers: cold failover, warm failover, and hot failover.

1. **Cold Failover**:
   In a cold failover scenario, the backup component or system is not running, but it is configured and ready to take over if the primary component or system fails.
   The backup component needs to be manually started up, which can take some time.
   Cold failover is typically used in systems with longer recovery times, where it is acceptable to have some downtime.

2. **Warm Failover**: In a warm failover scenario, the backup component or system is running, but it is not actively processing requests. Instead, it is waiting in a standby mode, ready to take over if the primary component or system fails.
   This type of failover typically involves some delay as the backup system takes over, but it is faster than cold failover.
   Warm failover is typically used in systems with shorter recovery times, where downtime needs to be minimized.

3. **Hot Failover**:
   In a hot failover scenario, the backup component or system is actively processing requests, and it is kept in sync with the primary component or system in real-time.
   This means that if the primary component or system fails, the backup system can take over almost instantly, without any loss of data or interruption in service.
   Hot failover is typically used in systems that require high availability and minimal downtime.

(Asked 2 times)

9. What are different types of failure models?

Ans: There are several types of failure models in distributed systems. Here are some of the most common ones:

1. **Crash failures**:
   This type of failure occurs when a node in the distributed system stops functioning abruptly, without any indication or warning.
   In other words, a process in the system fails silently and does not respond to any messages.

2. **Omission failures**:
   In this type of failure, a node in the distributed system fails to send or receive a message, leading to communication failures.
   But the node continues to execute other actions.

3. **Timing failures**:
   This type of failure occurs when the timing of a message or operation in the distributed system exceeds the expected time limit.
   Timing failures can result in message delays, timeouts, and inconsistencies in the system.

4. **Byzantine failures**:
   Byzantine failures are the most severe type of failure, where a node in the distributed system can behave arbitrarily, intentionally or unintentionally, leading to incorrect behavior of the entire system.

   Byzantine failures are the most difficult to detect and recover from because the system cannot distinguish between faulty and malicious behavior.

5. **Network failures:**
   This type of failure occurs when the network connecting the nodes in the distributed system fails or becomes unstable, leading to communication failures.
6. **Process failure**
   Process failure refers to the situation where a process in the distributed system stops functioning or crashes.
   This can happen due to various reasons, such as hardware or software errors, network connectivity issues, or other factors.
   When a process fails, it can no longer send or receive messages, and it may also leave the system in an inconsistent state.
7. **Channel failure**
   This refers to the situation where the communication channel between two processes in the distributed system becomes unreliable or unavailable.
   This can happen due to network congestion, packet loss, or other factors.
   When a channel fails, messages may be lost or delayed..

Each of these failure models requires different techniques and strategies for ensuring fault tolerance and maintaining the consistency of the distributed system.

It is important for developers and engineers to understand the different failure models and design the system accordingly to handle them.

(Asked 2 times)

10. Explain Triple Modular Redundancy (TDR).

- Triple Modular Redundancy (TMR) is a fault-tolerant technique used in distributed systems to improve the reliability of critical components.

- In TMR, three identical copies of a component are created, and each copy is run on a separate processor or computer.
- The three copies work in parallel, and their outputs are compared.
- If one of the copies produces a different output from the other two, it is considered faulty and is replaced by a spare copy.
- This ensures that the system continues to function correctly even if one of the copies fails.

- TMR is commonly used in safety-critical systems such as aviation, nuclear power plants, and medical devices, where failures can have catastrophic consequences.
- By providing three redundant copies of critical components, TMR increases the system's fault tolerance and reduces the likelihood of a failure.
- However, TMR comes at a cost, as it requires three times as much hardware and processing power as a non-redundant system.

11. What is Distributed Commit? (Not Asked yet)

- Distributed commit refers to the process of ensuring that transactions are committed or rolled back consistently across a distributed system, where different components or nodes may be responsible for different parts of the transaction.
- In a distributed system, transactions may involve multiple nodes that are geographically distributed, and each node may have its own copy of the data that is being updated.
- The goal of distributed commit is to ensure that all nodes involved in a transaction either commit or abort the transaction together, so the distributed system is left in a consistent state.
- This is important because if one node commits the transaction while another node aborts it, inconsistencies can arise in the data that is being updated, which can lead to errors and data corruption.
- To ensure distributed commit, various distributed commit protocols have been developed, including **two-phase commit (2PC), three-phase commit (3PC)**, and others.
- These protocols ensure that all nodes involved in a transaction agree on whether to commit or abort the transaction before any changes are made to the data.
- This helps to ensure that the distributed system remains consistent and free from errors.

(Asked 2 times)

12. What is atomic commit protocol?

- Atomic commit protocol is a **type of distributed commit protocol** that ensures that a transaction is either committed or aborted atomically, meaning that either all changes are made to the distributed system or none at all.
- The term "atomic" in this context refers to the indivisibility of the transaction, where either all changes are made or none are made, and there are no intermediate states.
- The atomic commit protocol is used in distributed systems to ensure that all nodes involved in a transaction agree on whether to commit or abort the transaction before any changes are made to the data.
- This helps to ensure that the distributed system remains consistent and free from errors.
- There are different types of atomic commit protocols, including **two-phase commit (2PC), three-phase commit (3PC),** and others.
- These protocols ensure that all nodes involved in a transaction agree on whether to commit or abort the transaction before any changes are made to the data.
- This helps to ensure that the distributed system remains consistent and free from errors.
- The atomic commit protocol is an important mechanism for ensuring that distributed transactions are completed correctly and consistently, despite the challenges presented by the distributed nature of the system.

(Asked 5 times)

13. Explain two phase and three phase distributed commit protocol.

- Distributed commit protocols are used to ensure that transactions are committed or rolled back consistently across a distributed system, where different components or nodes may be responsible for different parts of the transaction.
- There are several types of distributed commit protocols, including two-phase commit (2PC) and three-phase commit (3PC).

Two phase commit (2PC)

In the two-phase commit (2PC) protocol, the commit process takes place in two phases:

Two modules included are: **Co-Ordinator and Participant**

1. **Prepare phase:**

- The coordinator (typically the node responsible for initiating the transaction) sends a "prepare" message to all participants (other nodes involved in the transaction) asking if they are ready to commit the transaction.
- Each participant then responds with either a "vote to commit" or a "vote to abort" message.

2. **Commit phase:**

- If all participants vote to commit, the coordinator sends a "commit" message to all participants, indicating that the transaction should be committed.
- If any participant votes to abort, the coordinator sends an "abort" message to all participants, indicating that the transaction should be rolled back.

The two-phase commit protocol ensures that all participants either commit or abort the transaction together, so the distributed system is left in a consistent state.

Three Phase Commit

Three-phase commit (3PC) protocol is an extension of the two-phase commit (2PC) protocol, which ensures that a transaction is either committed or rolled back consistently across a distributed system.

3PC is designed to address some of the limitations of 2PC, such as blocking and the inability to handle certain types of failures.

Here, the commit process takes place in three phases:

1. **Can-commit phase:**
   - The coordinator sends a "can-commit" message to all participants, asking if they are able to commit the transaction. Each participant responds with either a "yes" or a "no" message.
   - It is similar to prepare phase in Two Phase Commit

2. **Pre-commit phase:**
   - If all participants respond with a "yes" message in the can-commit phase, the coordinator sends a "pre-commit" message to all participants, indicating that the transaction will be committed.
   - The participants then prepare to commit the transaction, but do not actually commit it yet.

3. **Commit phase:**
   - If all participants are able to commit the transaction successfully, the coordinator sends a "commit" message to all participants, indicating that the transaction should be committed.
   - If any participant is unable to commit the transaction, the coordinator sends an "abort" message to all participants, indicating that the transaction should be rolled back

The three-phase commit protocol adds an additional phase to ensure that the transaction can be committed before actually committing it.

This reduces the likelihood of a participant aborting the transaction after the pre-commit phase, which could cause inconsistencies in the distributed system.

Overall, both two-phase and three-phase commit protocols are designed to ensure that transactions are consistently committed or rolled back across a distributed system, but three-phase commit provides an additional layer of safety to prevent inconsistencies in the system.

14. Why two-phase commit protocol is widely used than three phase commit protocol?

For several reasons:

1. **Simplicity**:

- 2PC is simpler to implement and understand compared to 3PC.
- It involves only two phases - prepare and commit - which makes it easier to reason about and debug.

2. **Compatibility**:
    - Many existing distributed systems and databases already support 2PC, which makes it easier to integrate new components into these systems without requiring major changes.

3. **Performance**:
    - 2PC is generally faster than 3PC, since it involves only two message exchanges between the coordinator and participants, whereas 3PC requires three message exchanges.
    - This can be especially important in high-transaction-rate environments, where the overhead of extra message exchanges can become significant.

(Asked 4 times)

15. Forward and Backward Recovery

- Recovery in a distributed system refers to the process of restoring the system to a consistent and functional state after a failure or disruption.
- In a distributed system, failures can occur at various levels, such as hardware, software, network, or human errors, and can result in various types of data inconsistencies or system failures.
- Recovery mechanisms typically involve detecting the failure, identifying the scope and impact of the failure, and taking appropriate actions to restore the system to a consistent state.
- Forward and backward recovery are two different approaches to recovering from failures in a distributed system.

Forward Recovery

- In forward recovery, the system detects a failure and continues to run, attempting to execute the remaining transactions in the system.
- Forward recovery attempts to recover from a failure while keeping the system running.
- Forward recovery is often used in systems where failures are relatively common, and it is more important to keep the system running than to roll back transactions.
- Forward recovery is generally faster than backward recovery, which involves rolling back transactions to a consistent state prior to the failure.
- Forward recovery can be less expensive than backward recovery, as it does not require the system to store as much transaction data.
- Examples of forward recovery techniques include redoing transaction, compensating transactions, and retrying failed transactions.

- It is used in application such as Online shopping applications, banking applications, Online Gaming, etc.
  When a customer places an order, the system processes the transaction and updates the inventory. If a failure occurs before the update is complete, the system can use forward recovery to redo the operation and update the inventory to maintain consistency.

Backward Recovery

- Backward recovery is a technique used to recover a distributed system to a previous consistent state in case of failure.
- It involves restoring the system to a state that existed before a particular transaction or set of transactions were executed.
- The technique is called "backward" because it moves the system state back in time to a previous state, rather than moving forward with new updates.
- Backward recovery is typically used in systems where transactions are atomic, meaning they are either fully executed or not executed at all.
- To achieve backward recovery, the system must keep track of all transactions and their effects on the system state. This information is typically stored in a log.
- If a failure occurs, the system can use the log to roll back transactions and restore the system to a previous state.
- The system may use mechanisms such as checkpoints, transaction logging, and transaction undo/redo to recover from the failure and ensure that the system is in a consistent state.
- Backward recovery is often used in systems where failures are relatively rare and it is critical to maintain strong consistency across the distributed system.
- Examples of backward recovery techniques include two-phase commit protocols, transaction undo/redo, and database snapshots.

In summary, forward recovery attempts to recover from a failure while keeping the system running.

While backward recovery rolls back transactions to maintain strong consistency across the distributed system.

(Asked 2 times)

16. Explain Naïve Snapshot algorithm for recovery mechanism.

The Naïve Snapshot algorithm is a recovery mechanism used in distributed systems to provide a consistent global snapshot of the system's state.

This snapshot can be used to recover the system in case of a failure or inconsistency.

It is a simple and straightforward algorithm that involves the following steps:

Write Snapshot algorithm from Chapter 3 Que No 18 and continue..

In case of a failure, the global snapshot can be used to restore the system to a consistent state.

Overall, the Naïve Snapshot algorithm provides a simple and straightforward mechanism for ensuring consistency and recoverability in distributed systems.

(Asked 2 times)

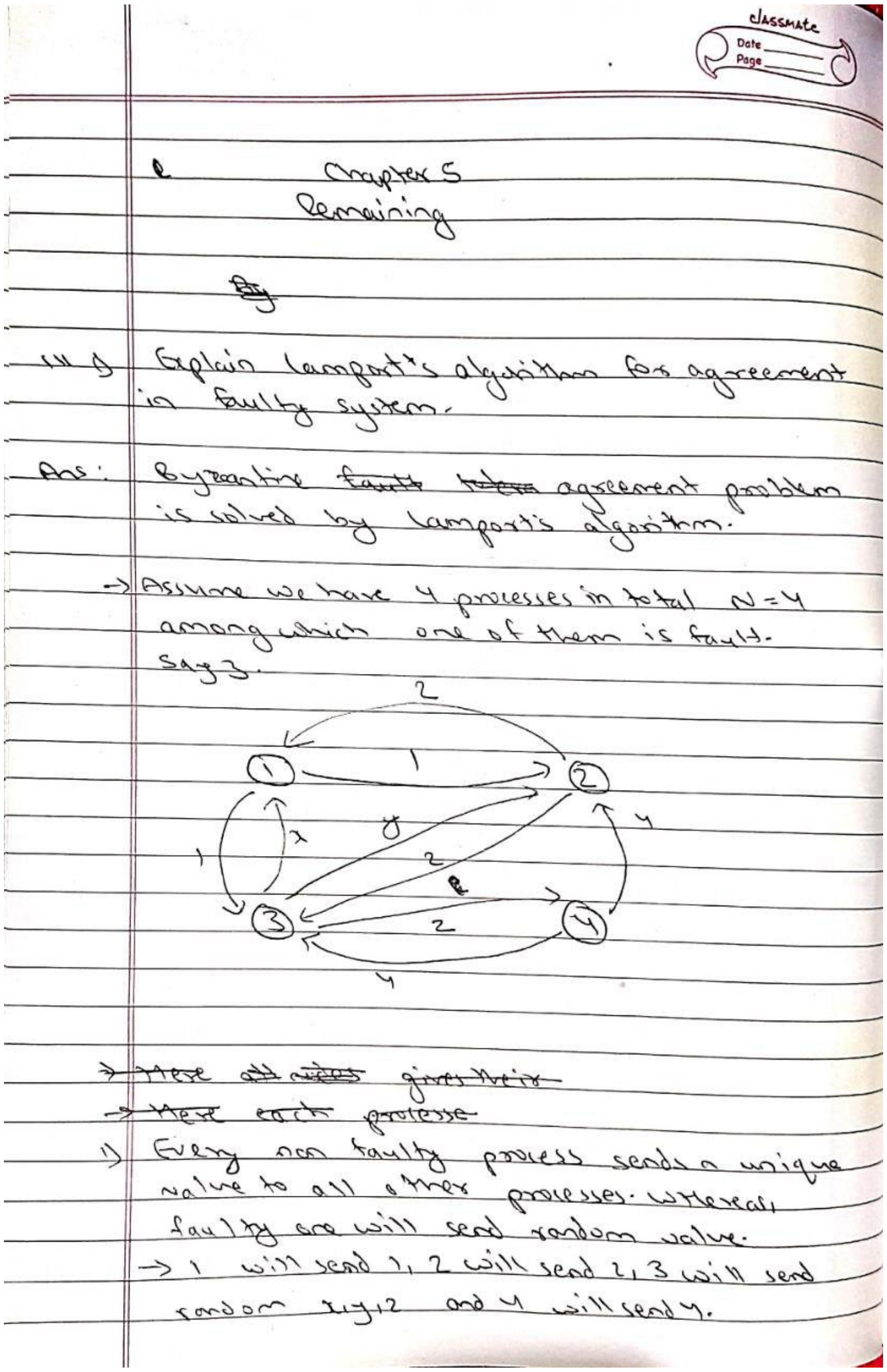
17. Byzantine agreement problem and solution algorithm

- The Byzantine agreement problem is a problem where a group of nodes needs to agree on a common decision in the presence of faulty nodes.
- The faulty nodes behave arbitrarily and send conflicting information to other nodes.
- One solution to this is by using Lamport's algorithm for agreement in faulty system.

(Explain Que no 18)

(Asked 4 times)

18. Explain Lamport's algorithm for agreement in faulty system.

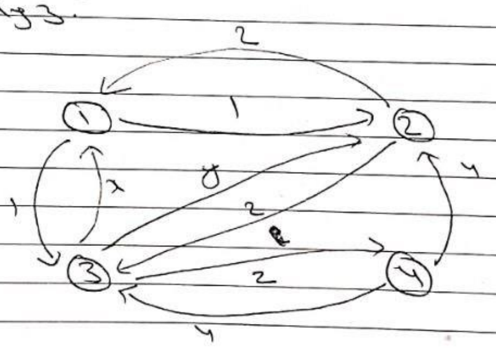


Chapter 5
Remaining

14) Explain lamport's algorithm for agreement in faulty system.

Ans: Byzantine ~~faulty~~ ~~[illegible]~~ agreement problem is solved by lamport's algorithm.

→ Assume we have 4 processes in total N = 4 among which one of them is faulty. say 3.



→ ~~Here all votes gives their~~
→ ~~Here each processe~~
1) Every non faulty process sends a unique value to all other processes. whereas, faulty one will send random value.
→ 1 will send 1, 2 will send 2, 3 will send random x, y, z and 4 will send 4.

1 : Got (1,2,x,4)
2 : Got (1,2,y,4)
3 : Got (1,2,3,4)
4 : Got (1,2,z,4)

2) All values are collected and stored in the form of vector.

3) Now each process will send ~~all~~ vector list to each other.

1 Got	2 Got	4 Got
(1,2,y,4)	(1,2,x,4)	(1,2,x,4)
(a,b,c,d)	(e,f,g,h)	(1,2,y,4)
(1,2,z,4)	(1,2,z,4)	(i,j,k,l)

Again faulty process sent random list to 1,2, and 4 i.e ~~(1,1,3,4)~~ (a,b,c,d), (e,f,g,h) and (i,j,k,l) respectively.

4) Finally each process examines the element of each of the newly received vector.
-> If the list has values that are same many times i.e (1,2,4) then they are selected.
-> Whereas, 3 is not selected.

-> ~~Hence~~, ~~it~~ This is Byzantine faulty problem.

-> Agreement can ~~be~~ ~~st~~ be reached by Lamport's Algorithm.

~~[illegible]~~ Here, n → total process/site
m ⇒ faulty process

If $m \le \left[\frac{n-1}{3}\right]$ then agreement
else no agreement

In our example
$n = 4$, $m = 1$

$\therefore 1 \le \left[\frac{4-1}{3}\right]$

$1 \le 1$ (true)

∴ Hence, agreement is reached.

In this way lamport's algorithm is used for agreement in faulty system.

Chapter: 6
Agreement in Distributed System

2 Questions (15 marks)

1. What is transaction?

- Transaction is a set of operations used to perform logical unit of work.
- A transaction generally represent change in database.

2. What is distributed transaction?

- A distributed transaction is a set of operations on data that is performed across two or more databases.
- It is a transaction that involves multiple systems or databases that need to work together to complete the transaction.
- They all either commit or roll back the transaction as a whole.
- For example, Transferring money from one bank account to another back account. They will be in different locations and will have different balances. To complete this transaction, the customer's account in one region must be debited, and the other account in a different region must be credited. To achieve this, the transaction must be distributed across the two databases in different regions. This is an example of a distributed transaction.
- Coordinating the transaction across multiple systems can be challenging, but it's done to improve performance, scalability, and availability.
- Special protocols like Two-Phase Commit and Three-Phase Commit are used to manage distributed transactions.

(Asked 2 times)

3. Explain the different properties of transaction.

- Transactions have several properties that ensure their correctness and consistency.
- Understanding the properties of transactions is essential for designing database systems that are reliable, efficient, and maintain data consistency.
- The different properties of transactions are also known as ACID properties. They are described below:

1. **Atomicity:**
    - This property ensures that a transaction is treated as a single, indivisible unit of work.
    - If any part of the transaction fails, the entire transaction is rolled back, and any changes made to the data are undone.
    - This ensures that the database remains consistent and that transactions are not left in an incomplete state.

2. **Consistency:**
    - This property ensures that a transaction brings the database from one valid state to another.
    - It enforces rules, that the database has to ensure that the data remains valid and consistent.
    - If a transaction violates any constraints or rules, it is rolled back, and the data is restored to its previous state.
3. **Isolation:**
    - This property ensures that a transaction operates independently of other transactions.
    - Each transaction should see the database as if it were the only transaction accessing it, even though there may be other transactions running concurrently.
    - This is achieved by locking the data that a transaction is accessing, to prevent other transactions from modifying it.

4. **Durability:**
    - This property ensures that once a transaction is committed, its changes are permanently stored in the database, even in the event of a system failure.
    - This is typically achieved by writing the changes to a durable storage device, such as a hard disk or solid-state drive.

4. Explain the different types of transaction.

(2 times)

1. **Flat transaction**

- A flat transaction is a simple transaction that involves a single operation that modifies the database.
- It is the most basic type of transaction that refers to a single, standalone transaction that is not nested within any other transaction.
- It is a simple transaction that performs a specific task or set of tasks and is committed or rolled back as a single unit of work.
- Flat transactions are commonly used in database applications to perform simple operations, such as updating a single record or inserting a new record.

(2 times)

2. **Distributed transaction**

- A distributed transaction is a set of operations on data that is performed across two or more databases.
- It is a transaction that involves multiple systems or databases that need to work together to complete the transaction.
- They all either commit or roll back the transaction as a whole.
- For example, Transferring money from one bank account to another back account.

(3 times)

3. **Nested Transaction**

- A nested transaction is a transaction that is executed within the context of another transaction.
- This means that a transaction can have one or more sub-transactions that are executed within its scope.
- The outer transaction is called the parent transaction, and the inner transaction is called the child/sub transaction.
- Each child transaction is treated as a separate transaction and can have its own set of operations that are executed within the scope of the parent transaction.
- The primary advantage of nested transactions is that they provide a way to break down complex transactions into smaller, more manageable parts.
- By dividing a transaction into multiple sub-transactions, each with a specific purpose or set of operations, it is easier to debug, test, and maintain the transaction.
- Nested transactions also provide a way to improve performance by allowing child transactions to run in parallel with the parent transaction.
- This can reduce the amount of time needed to complete the entire transaction and can improve concurrency and throughput.
- The outcome of the child transactions affects the outcome of the parent transaction, and if any child transaction rolls back, it can cause the parent transaction to roll back as well.

4. Sub transaction

- In a database management system that supports nested transactions, a sub-transaction refers to a transaction that is executed within the scope of another transaction, which is called the parent transaction.
- Like any other transaction, a sub-transaction can be committed or rolled back, but its outcome depends on the outcome of the parent transaction.
- Sub-transactions can be used to break down complex transactions into smaller, more manageable parts, each with its own set of operations.
- This makes it easier to design, test, and maintain the transaction.
- The parent transaction can have one or more sub-transactions, each representing a child transaction that is executed within the scope of the parent transaction.

Q. Two remote objects are related and should be updated in one operation. What kind of service do you need for this if you assume that both objects can be on different systems and that multiple operations can happen concurrently?

Ans: To ensure that two remote objects are updated in one operation, you need a **distributed transaction service**.

Explain distributed transaction.

(Asked 2 times)

5. Describe and differentiate flat transaction and nested transaction with their advantages and disadvantages.

Flat Transactions:

- A flat transaction is a standalone transaction that is not nested within any other transaction.
- It is a simple transaction that performs a specific task or set of tasks and is committed or rolled back as a single unit of work.
- Flat transactions are easy to use and understand, and their simplicity makes them well-suited for simple operations like updating a single record or inserting a new record.
- The primary disadvantage of flat transactions is that they can become difficult to manage when dealing with complex operations or transactions that involve multiple steps.

Nested Transactions:

- A nested transaction is a transaction that includes other transactions within it.
- A nested transaction can contain one or more sub-transactions that are executed within the scope of the parent transaction.
- Nested transactions are more flexible and powerful than flat transactions, allowing for complex transactions to be broken down into smaller, more manageable parts.
- The primary advantage of nested transactions is that they provide greater modularity and flexibility, making it easier to manage complex transactions and improve performance.
- The main disadvantage of nested transactions is that they can be more difficult to understand and use, particularly for users who are not familiar with the nesting structure or who may have limited access to the system.

V-Imp (Asked 5 times)

6. Explain different concurrency control mechanism for distributed transaction.

OR

Explain different concurrency control mechanism.

OR

Discuss about the various technique of concurrency control in a distributed system.

- Concurrency refers to the ability of a database management system to allow multiple users or applications to access and manipulate the same data at the same time.
- It is important because it allows multiple users to access and use the database simultaneously, improving the overall efficiency and productivity of the system.
- However, it can also create potential problems, such as data inconsistency or conflicts, if multiple users try to modify the same data at the same time.

- **Concurrency control mechanism** is a technique used in database management systems to manage concurrent access to data by multiple users or transactions.
- The main purpose of concurrency control is to ensure that transactions are executed in a correct and consistent manner, even when they are executed simultaneously.

Different concurrency control mechanisms are **Locks**, **Optimistic concurrency control** and **Timestamp ordering**.

They are explained below:

1. **Locks**:

- Lock-based concurrency control involves locking the data that a transaction is accessing, to prevent other transactions from modifying it at the same time.
- It is used to prevent multiple transactions from modifying the same data at the same time, or to ensure that one transaction completes its work before another transaction starts accessing the same data.

There are two types of locks - **shared** and **exclusive**.

a. **Shared Lock**: A shared lock is a type of lock that allows multiple transactions to read the same data simultaneously but prevents any transaction from modifying that data until the shared lock is released.
Other transactions can read data but cannot modify data.

b. **Exclusive locks**: Exclusive locks are used to ensure that only one transaction can access and modify the data at a time, and no other transaction can read or modify the data until the lock is released.
Other transactions cannot read nor modify data.

2. **Optimistic concurrency control:**

- Optimistic concurrency control assumes that conflicts between transactions are rare and allows multiple transactions to access the same data concurrently, without locking it.
- When a transaction tries to commit, the system checks if any other transactions have modified the same data.
- If so, the transaction is rolled back, and the data is reprocessed.

- For example, imagine two people try to buy the last item in a store online. Optimistic concurrency control would allow both people to add the item to their cart, but when they try to check out, the system would detect the conflict and roll back the customer's transaction which added to the cart later.

3. **Timestamp ordering:**

- Timestamp-based concurrency control uses timestamps to order transactions and ensures that only transactions that have a valid timestamp can access the data.
- In timestamp ordering, each transaction is assigned a unique timestamp that indicates when the transaction started.
- It is useful for ensuring that transactions are executed in a consistent order, which can help avoid conflicts and inconsistencies in the database.
- **Example**: Imagine two people try to book the same hotel room at the same time. The system would assign a timestamp to each transaction and process them in timestamp order, ensuring that only one person can book the room i.e., the one which have earlier timestamp.

Q) Which one is best? Provide reasons for your answer.

- The choice of the best concurrency control mechanism depends on the specific requirements of the system.
- Each mechanism has its own strengths and weaknesses, and the most appropriate mechanism will depend on factors such as the workload of the system, the type of transactions being processed, and the level of concurrency required.
- Explain any one of them with example.

Q) Why do we need a locking service in distributed system? What kind of problems does a locking service prevent? Explain in detail.

Ans: Explain about locks and how it helps to solve concurrency problem.

7. What are the problems of concurrency in transaction? List and describe with example.

Concurrency control is the process of managing access to shared resources in a way that ensures correct results while allowing multiple users to access the resource simultaneously. While concurrency is essential for improving system performance, it can also introduce several problems. Here are some of the common problems of concurrency in transactions:

1. **Lost Updates:**
   Lost updates occur when two or more transactions attempt to update the same data at the same time.
   A lost update happens when one transaction writes data to a record, and then another transaction overwrites that data before the first transaction is finished, causing the first transaction's changes to be lost.
   The result is that the database ends up with the data from the second transaction only, and the changes made by the first transaction are lost

2. **Dirty Reads:**
   Dirty reads occur when a transaction reads data that has been modified by another transaction that has not yet been committed. This can result in incorrect data being read and used by the transaction.
   The problem with dirty reads is that the uncommitted data may be changed or rolled back by the original transaction that wrote it, which can lead to inconsistencies in the database.
   For example, suppose a bank customer requests a balance inquiry while another transaction is updating the balance. If the balance inquiry reads the uncommitted data, it may return an incorrect balance.

3. **Non-Repeatable Reads:**

- Non-repeatable reads occur when a transaction reads the same data multiple times, but gets different results each time.
- This can happen when another transaction modifies the data between the two reads.
- For example, suppose a salesperson checks the inventory of a product and sees that there are 10 units available. However, when they attempt to sell the product, another transaction has already sold 5 units, and the inventory is now only 5 units.

4. **Phantom Reads:**

- Phantom reads occur when a transaction reads a set of data, but when it tries to read the same data again, the result set has changed due to another transaction modifying the data.
- For example, suppose a user searches for all orders placed on a specific day. However, before the search is complete, another transaction adds a new order for that day. When the search is re-executed, the result set includes the newly added order.
- The problem with phantom reads is that they can lead to inconsistencies in the database because the results of the query are not consistent between the two reads.

These are just a few examples of the problems that can arise in concurrent transactions.

To avoid these issues, database systems use concurrency control mechanisms such as locks, optimistic concurrency control, and timestamp ordering to ensure that transactions are processed in a correct and consistent manner.

(Asked 2 times)

8. What is distributed deadlock? What are the causes and solution of deadlock?

- Deadlock is a situation where 2 processes sharing the same resources are effectively preventing each other from accessing the resources.
- Distributed deadlock occurs when multiple processes running on different computers in a distributed system are blocked because they are each waiting for resources that are currently held by other processes in the system.
- For example, consider a distributed system with multiple nodes, each holding a set of resources. If two transactions, each running on different nodes, try to acquire a set of resources that are held by the other transaction, they can become deadlocked, as each is waiting for the other to release the resources they need.
- The distributed system is unable to resolve the deadlock by itself and may require human intervention or a distributed deadlock detection algorithm to detect and resolve the deadlock.

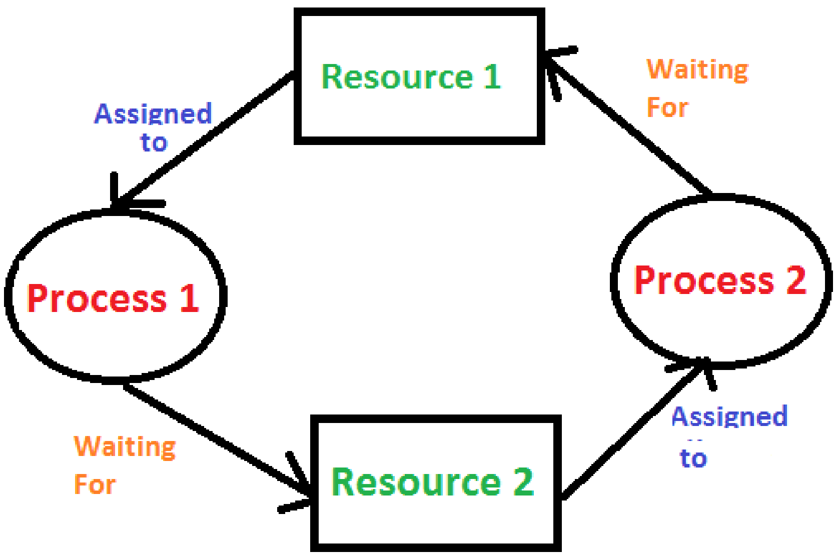


Fig: Deadlock

Causes of distributed deadlock:

1. **Resource contention:**
   When multiple transactions in a distributed system compete for the same resources, such as database tables, files, or network connections, it can lead to deadlock.
2. **Communication delays:**
   When there are delays in message transmission between different nodes in a distributed system, it can cause transactions to wait for response of messages that are never delivered, leading to deadlock.
3. **Synchronization issues:**

- In a distributed system, synchronization issues can arise when transactions use different mechanisms to coordinate access to shared resources, such as locks or semaphores.
- If two transactions try to acquire conflicting locks on the same resource, it can lead to a situation where both transactions are waiting for each other to release their locks, resulting in a deadlock.
- For example, if transaction A has acquired a lock on resource X and is waiting to acquire a lock on resource Y, while transaction B has acquired a lock on resource Y and is waiting to acquire a lock on resource X, both transactions will be waiting for each other to release their locks, resulting in a deadlock.

Solutions to distributed deadlock:

1. **Prevention**:

- Prevention involves designing the system in a way that avoids the possibility of deadlock altogether.
- This can be achieved by ensuring that transactions always request resources in a certain order, such as always requesting resources in alphabetical order.
- Another approach is to use timeout mechanisms to prevent transactions from waiting indefinitely.

2. **Detection and Recovery:**

- Detection involves periodically checking the system for deadlocks.
- This can be done by analyzing the resource allocation graph and looking for cycles.
- Once a deadlock is detected, the system can take steps to break the cycle, such as aborting one or more transactions involved in the deadlock.

3. **Avoidance**:

- Avoidance involves dynamically managing resource allocation to prevent deadlocks from occurring.
- This is done by using algorithms that predict which resources will be needed by each transaction and allocating resources accordingly.
- By carefully managing resource allocation, the system can avoid the possibility of deadlock altogether.

9. Explain wait-die and wound wait schemas for deadlock prevention?

Wait-die Scheme:

- In the wait-die scheme, if a transaction requests a resource that is currently held by another transaction with a higher timestamp, the requesting transaction is forced to wait until the other transaction releases the resource.
- If a younger transaction attempts to request a resource held by an older transaction.
- If the younger transaction has a lower timestamp than the older transaction, it is not allowed to proceed and is instead rolled back to its previous checkpoint, effectively "dying" and being restarted from an earlier state.

Wound-die Scheme:

- It operates in a similar way to wait-die, but with different rules.
- In the wound-wait scheme, if a transaction requests a resource that is currently held by another transaction with a lower timestamp, the requesting transaction is allowed to force the other transaction to release the resource.
- Simply, A transaction with a higher priority can "wound" a transaction with a lower priority by forcing it to abort and restart.

In wait-die, the newer transactions are killed when the newer transaction makes a request for a lock being held by an older transactions.

In wound-wait, the newer transactions are killed when an older transaction makes a request for a lock being held by the newer transactions.

10. What is starvation?

- Starvation occurs when a process is blocked from accessing a resource or entering a critical section because other processes are holding onto it for an extended period.
- Deadlock starvation occurs when a deadlock resolution algorithm always chooses the same transaction or resource to be sacrificed in order to break a deadlock, resulting in that transaction or resource being consistently blocked or delayed.
- For example, consider a distributed system with multiple transactions and resources, where deadlock detection and resolution is performed using a timeout mechanism.
- If the timeout value is set too high, some transactions or resources may be starved because they are consistently chosen as the ones to be sacrificed to break deadlocks.

11. Communication Deadlock and Resource deadlock.

Communication Deadlock:

- Deadlock in message communication, also known as communication deadlock, occurs in a distributed system when two or more processes or nodes are waiting for each other to send messages, causing all of them to be blocked or deadlocked.
- For example, consider two nodes in a distributed system, A and B, where each node is waiting for a message from the other to proceed. Node A sends a message to B and waits for a response, while node B is also waiting for a message from A. However, due to a network failure or a bug in the application, the message from A to B is lost, causing both nodes to wait indefinitely for each other's messages, resulting in a communication deadlock.

Resource Deadlock:

- Resource deadlock is a type of deadlock that occurs when two or more processes are unable to proceed because each is waiting for the other to release a resource.
- In other words, each process is holding a resource that another process needs to continue, resulting in a deadlock situation.
- For example, if process A holds a lock on resource X and waits for resource Y, while process B holds a lock on resource Y and waits for resource X, a resource deadlock occurs.

classmate
Date
Page

Chap6 Remaining Question

Imp

//// 1) Explain different distributed deadlock detection technique/algorithm.
or
How deadlocks are handled in distributed system?

Ans: There are two deadlock detection algorithm.
-> They are:-

a) Obermarck's Path Pushing Algorithm
b) Chandy-Mishra-Haas Edge Chasing Algorithm.

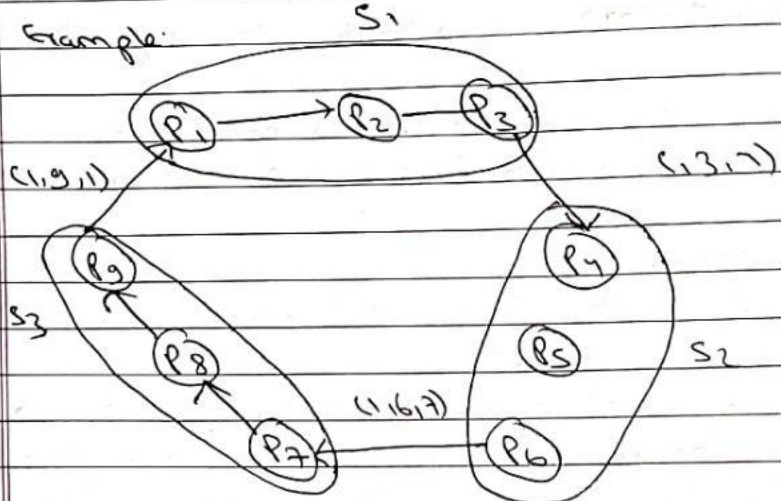
a) Obermarck's Path Pushing Algorithm {Path Pushing}

Ans: In path pushing algorithm, distributed deadlocks are detected by maintaing an explicit WFG (Wait for Graph).
-> If there is cycle then deadlock is present.

Algorithm
1. The first step of algorithm is to build a global WFG for each site of ~~the~~ the distributed system.
2. At each site whenever a deadlock computation is performed, it sends its local WFG to all the neighbouring sites.

There are 3 sites S1,S2 and S3

Example:



The ~~site~~ process that started the transaction is P1.
i=1, it will same for all now.

site 1 is sending Probe to site 2.
Here,
(i=1, j=3, k=4).
i is the one who started transaction, it sent the probe message P3 so j=3 and it's sending probe message to P4 so k=4.
∴ (1,3,4)

Similarly when sending probe message from S2 to S3. Probe message is:
∴ (1,6,7)

Similarly when sending probe message from S3 to S1 Probe message is
∴ (1,9,1)
∴ Since i=j (1=1). Deadlock is detected.

{Simply Edge chasing}

b) Chandy-Mishra-Haas Edge Chasing Algorithm.

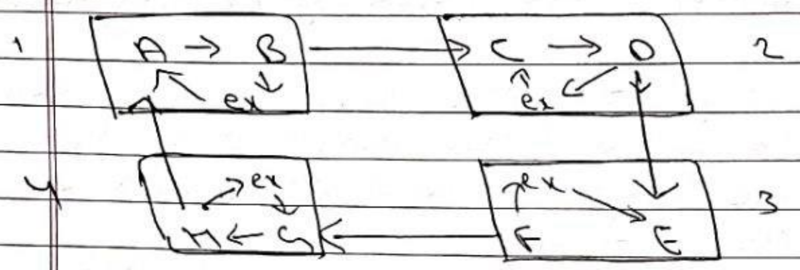
→ Edge chasing algorithm uses probe message to detect ~~the~~ deadlock in a distributed database.

Algorithm

1. Each site in the distributed database is assigned a unique identifier.
2. Whenever a transaction starts at a site, it sends out a probe message to all its neighbours. The probe message contains three pieces of information: the identifier of the site that started the transaction (i), the identifier of the probe (j), and the identifier of the site that received the probe (k). {i,j,k}
3. When a site receives a probe message, it checks if the transaction associated with the probe message has completed. If the transaction has not completed, it forwards the probe message to its neighbours.
4. If a site receives a probe message where ~~(i=k)~~ ~~[illegible]~~ i = k a deadlock is detected.
5. Once the deadlock is detected, necessary steps will be taken to resolve the deadlock.

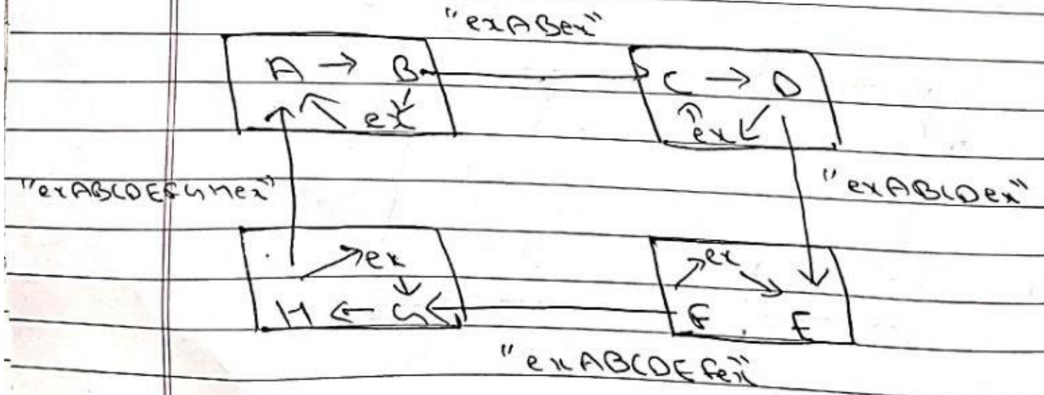
Exame:

Step1: Create a local wait for Graph.



Here, B is waiting for external site to complete its processing, whereas external site is waiting for A to complete it's processing. This is local WFG of site 1.
Similarly create WFG of site 2,3 & 4.

Step2: Each site will send a string to other site about current situation.
Then the string is concatenated and sent to all (~~site 4 will recognize they are in deadlock & announces it to everyone~~)

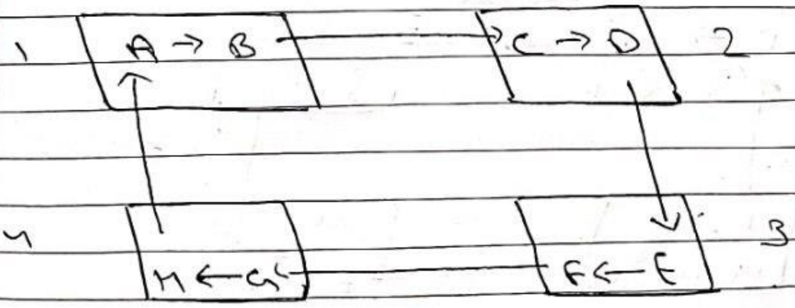


Step3: Site 4 will ~~recognize~~ ~~to~~ detect the deadlock & announce it to everyone. Necessary steps will be taken to resolve the deadlock.

3. After that the WFG of each site is updated, this updated WFG is then passed along the other sites.
4. Repeat 2 and 3 until some site has a sufficiently complete picture of the global state to announce deadlock or to establish that no deadlocks are present.
5. Once the deadlock cycle has been detected, the system can either terminate the processes in the cycle or release resources to break the cycle.
6. If the deadlock has been resolved, the algorithm can be restarted to check ~~the~~ for any additional deadlocks.

This feature of sending around the paths of WFG has led to the term path-pushing algorithms.

Example
→ Let site 1,2,3,4 are having transactions A & B, C&D, E&F and G&H respectively.



Here, A is waiting for B to complete its processing, B is waiting for external node to finish its processing and so on.

More Questions

1. What is race condition and semaphore?

- **A race condition** is a situation that can occur in computer systems when two or more processes or threads access shared resources in an unpredictable order. In a race condition, the result of the computation depends on the order in which the processes or threads execute, and the outcome may be incorrect or unpredictable.
- A common example of a race condition is when two processes or threads attempt to modify the same variable simultaneously. Depending on the order of execution, the final value of the variable may be different from what either process intended.

- A **semaphore** is a synchronization primitive used in computer systems to control access to shared resources and prevent race conditions.
- A semaphore is essentially a counter that is used to manage access to a shared resource such as a file, a printer, or a database.

- In a binary semaphore, the counter can have only two values: 0 and 1. If the value of the counter is 0, it means that the resource is currently unavailable and any process or thread that attempts to access it will be blocked until the resource becomes available.
- If the value of the counter is 1, it means that the resource is currently available and any process or thread that attempts to access it will be granted access.

2. What is group communication?

- Group communication in distributed systems refers to the exchange of messages between a group of processes or nodes in a coordinated and reliable manner.
- The group of processes can be located on different machines and can communicate with each other using a network.
- Group communication is used to achieve fault tolerance, scalability, and reliability in distributed systems.
- Group communication protocols ensure that messages are delivered reliably to all members of the group, even if some members fail or are temporarily unavailable.
- Broadcast, multicast, and consensus algorithms are common mechanisms used in group communication protocols.
- Group communication is used in many applications, including distributed databases, messaging systems, and distributed computing frameworks.

(Asked 3 times)

3. What is reliable multicasting?

- Reliable multicasting is a communication technique where a message is sent from one sender to multiple recipients over a network, ensuring that all recipients receive the message in the correct order and without errors.
- Special protocols are used to ensure that all recipients receive the message and acknowledge its receipt, detecting and recovering from errors or failures that occur during transmission.
- Reliable multicasting is commonly used in applications such as video conferencing, online gaming, and distributed computing.

- It ensures that all users receive the same information at the same time, regardless of their location or network conditions.

4. What is virtual synchrony?

- Virtual synchrony is a communication paradigm used in distributed systems to ensure consistent and reliable views of the system's state.
- It ensures that all processes in the system have the same view of the state of the system at all times, even if some processes fail or are temporarily unavailable.
- Virtual synchrony divides the processes in the system into groups and ensures that all processes in a group receive the same messages in the same order.
- This guarantees that all processes in the group have the same view of the state of the system, even if some processes are temporarily unavailable or fail.
- Virtual synchrony is commonly used in distributed systems that require high availability, fault tolerance, and consistency, such as distributed databases, messaging systems, and distributed computing frameworks.
- By using virtual synchrony, these systems can ensure that all processes have a consistent view of the state of the system, even in the face of failures or delays.

5. Explain the principle of virtual synchronous multicast.

Virtual synchronous multicast (VSM) is a way for computers in a group to communicate with each other reliably and consistently.

The principle of VSM is based on the concept of virtual synchrony, which ensures that all computers in the group have the same view of the system, even if some of them have problems or there are delays in the network.

The principle of VSM has four main steps:

Creation of a multicast group, Group membership and view management, Message ordering and delivery, and Failure detection and recovery.

1. In the **creation step**, a group of processes is created to communicate with each other using VSM.
2. In **group membership and view management step**, the processes exchange messages to maintain membership and view information.
3. In **the message ordering and delivery step**, messages are sent to the multicast group using the VSM protocol, which ensures that all the processes receive the message in the same order.
4. In the **failure detection and recovery step**, VSM provides mechanisms to detect and recover from failures of processes or communication links.