Rashmi Khadka Sameer Shrestha
Shiwani Shah Sujan Lamichhane

DISTRIBUTED SYSTEM

2074 (Regular)

Question 1: What is True Distributed System (TDS)? How distributed system can be organized as middleware? Explain.

Ans:

A distributed system refers to a network of interconnected computers that work together to achieve a common goal. The computers, often referred to as nodes, communicate and coordinate their activities in order to provide a unified service to users. These systems can range from small clusters of machines to massive networks of computers spanning multiple geographical locations.

Distributed system with all the challenges like transparency, scalability, dependability , performance, and flexibility being solved to its full extent is called a True Distributed system(TDS). However because of the different problems (like a need of new component (network), security and software complexity face during the development of a distributed system, a TDS is virtually possible.

Middleware, in the context of distributed systems, refers to software layers or frameworks that facilitate communication and interaction between various components in a distributed environment. It acts as an intermediary layer between the application and the underlying operating system and network infrastructure. Middleware provides a set of services and APIs (Application Programming Interfaces) that abstract the complexities of distributed communication, synchronization, and data management.

Here's how a distributed system can be organized using middleware:

- Communication Abstraction: Middleware provides a communication abstraction that hides the low-level details of network communication from applications. It offers APIs for

sending and receiving messages, allowing different nodes to communicate regardless of their underlying hardware or network protocols.

- Remote Procedure Calls (RPC): Middleware often supports RPC, allowing applications to invoke procedures or methods on remote nodes as if they were local. This simplifies the development of distributed applications by abstracting the network interaction.
- Message Queues: Middleware can offer message queuing services, enabling asynchronous communication between nodes. This is useful for decoupling components and handling high loads of messages.
- Data Replication and Consistency: Middleware can provide mechanisms for data replication and consistency across distributed nodes. This ensures that data remains coherent and up-to-date in the face of failures or concurrent updates.
- Security and Authentication: Middleware often includes security features to protect communication between nodes, ensuring that sensitive data remains confidential and that nodes are authenticated.
- Transaction Management: Middleware can facilitate distributed transactions, allowing multiple operations across different nodes to be treated as a single transaction with atomicity and consistency guarantees.

Question 2: Define Remote Procedure Call (RPC)? Describe various RPC communication semantics of client-server communication in a distributed system.

Ans:

Remote Procedure Call (RPC) is a communication protocol that allows a program or process to invoke a procedure (function) on a remote server as if it were a local procedure call. RPC abstracts the complexities of network communication, providing a transparent way to develop distributed applications.

RPC Communication Semantics:

1. At Most Once (AMO):

In AMO semantics, the client sends an RPC request to the server and waits for a response. If the client does not receive a response within a specified time, its re-issues the request. This ensures that the request is executed at most once, preventing duplicate executions. However, it might lead to duplicate requests in case of network delays.

2. At Least Once (ALO):

ALO semantics guarantee that the RPC is executed at least once. The client sends the request, waits for the response, and re-issues the request if no response is received within a specified time. This ensures the request's execution, but it can result in duplicate executions and responses due to re-issuing.

3. Exactly Once (EO):

EO semantics aims for a single, correct execution of the RPC. Achieving EO semantics involves coordination. The client sends the request, the server executes it, sends a response acknowledgment, and the client re-issues the request only if no acknowledgment is received. This approach ensures the request's correctness and uniqueness.

4. Perhaps or Possibly Call:

It is the weakest one, here, the caller is waiting until a predetermined timeout period and then continues with its execution. It is used in services where periodic updates are required.

5. Last-one Call:

Based on timeout, retransmission of call message is made. After the elapsing of the timeout period, the obtained result from the last execution is used by the caller. It sends out orphan calls. It finds its application in designing simple RPC.

6. Last-of-Many Call:

It is like Last-one Call semantics but the difference here is that it neglects orphan calls through call identifiers. A new call-id is assigned to the call whenever it is repeated. It is accepted by the caller only if the caller id matches the most recent repeated call.

Question 3: Mention the role of stub and skeleton in distributed system. Explain the architectural details of the Network File System (NFS).

Ans:

A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.

Skeleton is the object which resides on the server side. stub communicates with this skeleton to pass request to the remote object.

Stubs and skeletons provide a high-level programming abstraction for remote communication in distributed systems. Stubs allow clients to invoke remote methods as if they were local, while skeletons handle the mechanics of unpacking method calls, dispatching them to the server's implementation, and packaging the results for transmission back to the client. This separation of concerns simplifies distributed application development and enables transparent communication across distributed components.

The Network File System (NFS) is a widely used protocol for sharing files and directories across a network of computers. It allows remote systems to access and manipulate files on a server as if they were local to the client system.

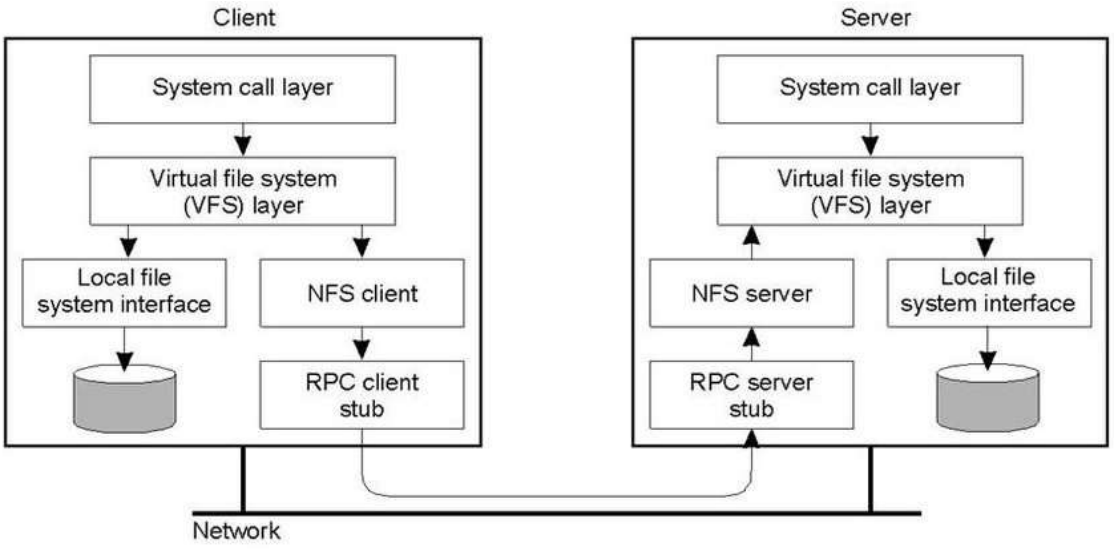


Fig:- Network File System

The NFS architecture comprises three main layers that work together to provide seamless remote file access and sharing. These layers are the System Call Layer, Virtual File System (VFS) Layer, and the NFS Service Layer.

System Call Layer

The System Call Layer is the lowest level of the NFS architecture and serves as the interface between user applications and the operating system kernel. When a user application makes file-related requests like opening, reading, or closing a file, the requests are handled by the system call layer.

In the context of NFS, when a user application initiates an NFS operation, such as accessing a remote file, the request is translated into a system call. The system call layer processes this request and forwards it to the Virtual File System layer for further handling.

Virtual File System Layer

The Virtual File System Layer sits above the system call layer and acts as an abstraction layer for managing various file systems in the operating system. It maintains a table with one entry, known

as a v-node (virtual node), for each open file.The v-node entries indicate whether a file is local or remote (on an NFS server). If the file is remote, the v-node contains enough information to access it over the network. For local files, the VFS layer records information about the local file system and the i-node (index node) associated with the file

NFS Service Layer

The NFS Service Layer is the top layer of the NFS architecture and is responsible for implementing the actual NFS protocol. It handles the communication between the NFS client and server components.

When the VFS layer identifies an NFS request, it passes it to the NFS Service Layer. This layer then takes care of sending the NFS request over the network to the appropriate NFS server using a transport protocol (such as TCP or UDP).

Question 4: What do you mean by Context Switching in a distributed system? How is a distributed OS different from a network OS?

Ans:

Context Switching: Context switching in a distributed system refers to the process of saving the current state of a running process (including its program counter, registers, and stack pointer) and restoring the saved state of another process. This allows the system to switch between executing processes, enabling multitasking and sharing computing resources efficiently.

Distributed OS vs. Network OS:

S.N.	Network Operating System	Distributed Operating System

1	Network operating systems are server-based operating systems that provide networking-related functionality.	The distributed OS manages a set of independent, networked, communicating computers and makes them look like an ordinary centralized operating system.
2	Its main objective is to provide the local services to remote client.	The main objective is to manage the hardware resources.
3	Communication takes place on the basis of files.	Communication takes place on the basis of messages and shared memory.
4	Its fault tolerance is less.	Its fault tolerance is high.
5	Resources are managed at every node.	Global central or distributed management is used to manage resources.
6	Easy to implement.	Comparatively hard to implement.

Question 5: Explain CORBA Invocation methods with its services.

Ans:

CORBA (Common Object Request Broker Architecture) is a middleware technology that enables communication and interaction between distributed objects in a heterogeneous environment. It allows objects to communicate across different programming languages, operating systems, and network protocols. CORBA provides a standardized way to invoke methods on remote objects through its invocation mechanisms and associated services.

There are two primary methods of invoking methods in CORBA: synchronous method invocation and asynchronous method invocation. Each of these methods is supported by various services to manage the complexities of distributed communication and error handling.

Synchronous Method Invocation:

In synchronous method invocation, the client sends a request to the server to execute a specific method on a remote object. The client then waits for a response from the server, which contains the result of the method call or an exception if an error occurs. This method ensures that the client waits for the method to complete before continuing with its execution.

Associated services for synchronous method invocation include:

Request and Reply Service: This service manages the communication between the client and the server for synchronous method invocations. It ensures that requests are properly sent to the server, and the corresponding replies are received by the client.

Asynchronous Method Invocation:

Asynchronous method invocation allows the client to invoke a remote method and then continue its execution without waiting for the method to complete. The client specifies a callback mechanism to handle the server's response or the completion of the remote method. This approach is suitable for scenarios where the client doesn't need to block while waiting for the method to finish.

Associated services for asynchronous method invocation include:

Event Service: This service allows objects to generate and handle asynchronous events. Clients can register interest in certain events, and servers can notify clients when relevant events occurs.

The CORBA services includes the following:

- Interface Repository Service: This service provides a centralized repository of interface definitions for objects in the system. It helps clients and servers dynamically discover and understand each other's interfaces, even if they were developed in different languages or on different platforms.
- Naming Service: This service provides a way to name objects and locate them across the distributed system. It enables clients to find objects using human-readable names instead of complex network addresses.
- Transaction Service: This service supports distributed transactions across multiple objects and resources. It ensures that a sequence of operations is executed as an atomic unit, with commit or rollback guarantees.
- Security Service: The Security Service provides authentication, authorization, and encryption mechanisms to secure communication and data in the distributed environment.

Question 6: Define clock synchronization. What is the need for clock synchronization? Explain Castain's algorithm for physical clock synchronization along with a necessary diagram.

Ans:

Clock Synchronization:

Clock synchronization in a distributed system refers to the process of aligning the physical clocks of multiple nodes to a common reference time. This is essential to ensure consistency in distributed computations, coordination, and ordering of events.

Clock synchronization is necessary to address issues like event ordering, distributed transaction coordination, and synchronization of distributed algorithms. Without synchronized clocks, accurate event sequencing becomes challenging, leading to confusion and incorrect outcomes.

Cristian's Algorithm for Physical Clock Synchronization:

Rashmi Khadka Sameer Shrestha
Shiwani Shah Sujan Lamichhane

Cristian's Algorithm is a clock synchronization algorithm used to synchronize time with a time server by client processes. This algorithm works well with low-latency networks where Round-Trip Time is short as compared to accuracy.

Round Trip Time refers to the time duration between the start of a Request and the end of the corresponding Response.

1) The process on the client machine sends the request for fetching clock time (time at the server) to the Clock Server at time.

2) The Clock Server listens to the request made by the client process and returns the response in form of clock server time.

3) The client process fetches the response from the Clock Server at time and calculates the synchronized client clock time using the formula given below.

$$T_{Client} = T_{Server} + (T_1 - T_0)/2$$

where,

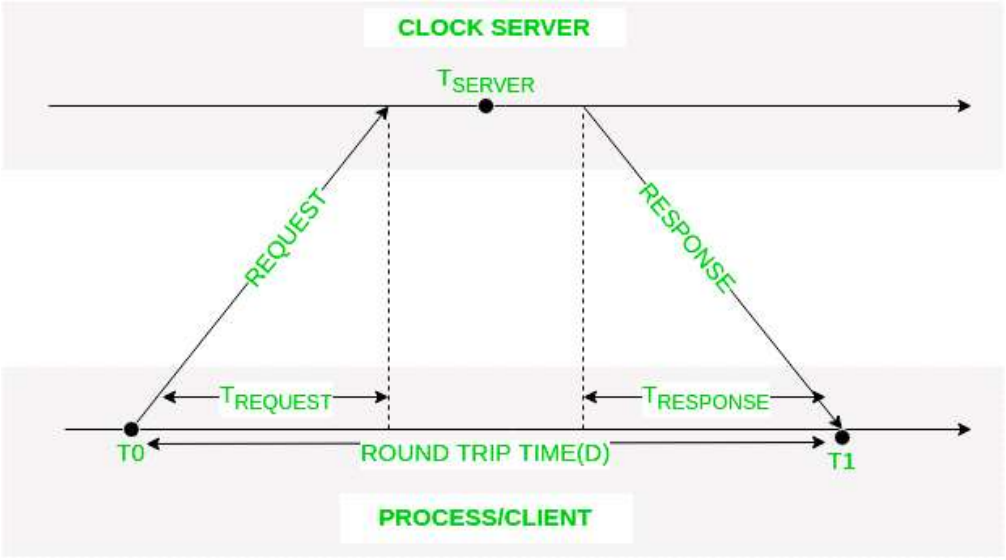
T_{Client}, refers to the synchronized clock time,

T_{Server}, refers to the clock time returned by the server,

T_0, refers to the time at which request was sent by the client process,

T_1, refers to the time at which response was received by the client process.

Diagram:



Question 7: Why Consensus is needed in DS? Explain BULLY algorithm with suitable example.

Ans:

Consensus in distributed systems refers to the process of reaching an agreement among a group of nodes (computers) on a certain value or decision. Achieving consensus is crucial for ensuring that all nodes in the system agree on a common state or value, even in the presence of failures and communication delays. Consensus is required for various distributed algorithms and applications, such as distributed databases, distributed transactions, leader election, and more.

The BULLY algorithm is a leader election algorithm used in distributed systems to elect a coordinator or leader from a group of nodes. The leader is responsible for making critical decisions and coordinating activities within the distributed system. The BULLY algorithm ensures that the highest-priority node (with the highest identifier) becomes the leader.

Here's how the BULLY algorithm works, along with a suitable example:

Assumptions and Scenario:

Rashmi Khadka Sameer Shrestha
Shiwani Shah Sujan Lamichhane

Let's consider a distributed system consisting of several nodes, each with a unique identifier. Nodes communicate with each other, and a leader must be elected. The node with the highest identifier is considered to have the highest priority.

Algorithm Steps:

- When a node wants to initiate the leader election process, it sends an election message to all nodes with higher identifiers. This indicates its intention to become the leader.
- If a higher-priority node receives an election message, it responds with an OK message to the initiating node. This indicates that it is still active and intends to participate in the election.
- If a node doesn't receive any OK message after sending an election message, it assumes that it has the highest priority among the active nodes and becomes the leader.
- If a node receives an OK message, it backs off and waits for a victory message. This indicates that a higher-priority node has been elected as the leader
- Once the initiating node receives OK messages from all nodes with higher priorities, it broadcasts a victory message to inform all other nodes that it has become the leader.
- Example:
- Let's illustrate the BULLY algorithm with an example involving five nodes with unique identifiers: Node 1, Node 2, Node 3, Node 4, and Node 5.
- Node 5 wants to initiate the leader election process because it noticed that the current leader (if any) has failed.
- Node 5 sends election messages to nodes with higher identifiers: Node 4 and Node 3.
- Nodes 4 and 3 respond with OK messages to Node 5, acknowledging that they are still active.
- Node 5, having not received any OK message from nodes with higher priorities (there are none), assumes that it has the highest priority and becomes the leader.
- Node 5 broadcasts a victory message to all nodes, informing them that it is the new leader.

In this scenario, Node 5 becomes the leader because it successfully communicated with all nodes with higher identifiers and received no OK messages. The BULLY algorithm ensures that the node

with the highest priority becomes the leader and handles coordination tasks within the distributed system.

Question 8: What is Object Replication? Explain the Process Resilience approach.

Ans:

Object Replication: Object replication involves creating and maintaining multiple copies of an object across distributed nodes. This approach aims to enhance availability, fault tolerance, and load balancing in a distributed system.

Process Resilience Approach: The Process Resilience approach refers to the strategies and techniques implemented within a system to ensure its continued operation and performance even in the face of unexpected failures, errors, or disruptions. This approach is particularly important in distributed systems, where components may be geographically dispersed and interconnected through networks. The goal is to maintain the desired functionality and availability of the system despite adverse conditions.

Key concepts and strategies within the Process Resilience approach include:

1. Primary-Backup Model: Each object has a primary copy and one or more backup copies. The primary copy handles client requests, while the backups maintain consistent copies of the object.

2. Replication Protocol: When a client request modifies an object, the request is first sent to the primary copy. The primary applies the update and then propagates the change to the backup copies using a replication protocol.

3. Quorum-Based Techniques: Replication protocols often involve quorum-based techniques to ensure consistency among replicas. Quorum defines the minimum number of nodes that must agree on an update for it to be considered successful.

4. Handling Failures: If the primary fails, one of the backups is promoted to the new primary using an election process. This ensures that the system remains operational even in the presence of failures.

Question 9: What are the benefits and drawbacks of using lock in Transaction Processing? Explain the lost-update problems with suitable example.

Ans:

Locks are commonly used in transaction processing systems to manage concurrent access to shared resources, such as databases or files, in order to maintain data consistency and integrity. Locks help prevent multiple transactions from simultaneously modifying the same resource, which could lead to data corruption or inconsistency. However, using locks also comes with both benefits and drawbacks.

Benefits of Using Locks:

- Data Consistency: Locks ensure that only one transaction at a time can access and modify a shared resource, preventing conflicting changes and maintaining data consistency.
- Isolation: Locking provides a level of isolation between transactions, ensuring that one transaction's changes are not visible to other transactions until the first transaction is completed.
- Controlled Access: Locks allow for controlled access to resources, preventing potential issues like race conditions and ensuring proper sequencing of operations.
- Deadlock Avoidance: Lock management systems often include mechanisms to detect and resolve deadlocks, which occur when multiple transactions are waiting for resources held by each other, causing a circular dependency.

Drawbacks of Using Locks:

- Performance Overhead: Locking introduces overhead due to the need to acquire and release locks, which can affect system performance, especially in high-concurrency scenarios.

- Blocking: If one transaction holds a lock on a resource, other transactions requiring the same lock must wait, potentially causing contention and decreased throughput.
- Deadlocks: While deadlocks can be avoided or detected, resolving them can be complex and resource-intensive.
- Lock Granularity: Deciding on the appropriate level of lock granularity (e.g., whether to lock an entire table or just a single row) can be challenging and impact both performance and data integrity.
- Lost Updates and Starvation: The use of locks can lead to situations where certain transactions are delayed (starvation) or where updates made by one transaction are overwritten by another, resulting in lost updates.

The lost-update problem occurs when multiple transactions concurrently modify the same data, and the updates from one transaction are overwritten by the updates from another transaction. This can lead to inconsistent or incorrect data. As we know, the data of transaction if first read by the system and operation is performed and then write operation comes in the action which permanently saves the change in the actual database. Similarly, two or more transaction read the data from the database and performs their own operations but during the write operation sometimes the updated data from one transaction is overwritten by another transaction due to which the new data from the first transaction got lost and this situation is called lost update problem.

Example :

T1	T2
Read(X)	
	Read(X)
Update X=X+1000	
	Update X=X-500
Write(X)	
	Write(X)

Fig: Lost-Update Problem

Question 10: What are independent checkpointing and coordinated checkpointing? Explain the snapshot algorithm used for backward recovery in a distributed system.

Ans:

Independent Checkpointing: In independent checkpointing, each process in a distributed system takes checkpoints of its local state independently without coordination with other processes. These checkpoints are taken periodically or when certain conditions are met. Independent checkpointing is simpler to implement but can lead to inefficient resource utilization and difficulties in ensuring global consistency during recovery.

Coordinated Checkpointing: Coordinated checkpointing involves coordinating the checkpoint process among multiple processes in the system. This ensures that a consistent global state is captured across all processes. The challenge is to determine a global checkpoint time that does not violate the causality relationship between events.

Snapshot Algorithm for Backward Recovery:

The Chandy-Lamport Snapshot Algorithm is a popular approach for taking coordinated snapshots of distributed processes.

It allows for backward recovery in case of failures. This algorithm captures a consistent global state of a distributed system for the purpose of checkpointing and recovery.

1. Initiation: An initiator process triggers the snapshot algorithm. The initiator marks its own state and sends "marker" messages to all its outgoing channels.

2. Marker Propagation: When a process receives a marker message for the first time on a channel, it marks its state, records the channel state, and forwards the marker to other processes through its outgoing channels.

3. State Recording: When a process receives a marker on an incoming channel, it records its state, including the messages in transit on that channel now of marker reception.

4. Local State Recording: A process can record its local state at any time while processing the marker messages.

5. Completion Detection: When a process has recorded its state and received marker messages on all incoming channels, it is considered to have completed the snapshot.

6. Recovery: If a process fails and needs to recover, it uses the recorded snapshot to restore its state to a consistent global state.

Question 11: Write short notes on: (any two)

i) JINI Distributed Event Specification

ii) IDL

iii) Distributed cut

Rashmi Khadka Sameer Shrestha
Shiwani Shah Sujan Lamichhane

Ans:

i) JINI Distributed Event Specification

Jini, now known as Apache River, is a Java-based technology that facilitates the development of distributed systems, including services and applications that can dynamically discover, interact, and collaborate within a network. One of the important aspects of Jini is its support for distributed events through its Event Specification. The Jini Distributed Event Specification allows services to communicate and collaborate in response to events without requiring centralized coordination.

The Jini Distributed Event Specification includes the following key components and concepts:

- Event Registration: Services can register interest in specific types of events or notifications that might occur within the network. These events could be related to changes in the state of a service, updates in data, or any other relevant occurrence.
- Event Notification: When an event of interest occurs, the service generating the event broadcasts a notification to the network. This notification includes information about the event itself.
- Event Consumers: Services that have registered interest in a specific type of event are referred to as event consumers. These services can be notified of the event occurrence and take appropriate actions in response.
- Event Sources: Services that generate events and notify the network about them are referred to as event sources. These services produce events based on certain conditions or changes in their state.
- Distributed Event Model: The Jini Distributed Event Specification operates based on a distributed event model. Events are generated, communicated, and consumed in a decentralized manner, allowing for scalability and dynamic collaboration.
- Decentralized Event Processing: Events can be processed by event consumers distributed across the network. This allows for event-driven collaboration and coordination without relying on a centralized component.

- Service Discovery: Jini's service discovery capabilities play a crucial role in the event specification. Services can discover each other dynamically through multicast or unicast communication, enabling them to interact without prior knowledge.

ii) IDL

IDL stands for Interface Definition Language. It is a language-agnostic specification used to define the interfaces of software components in a distributed system. IDL acts as a bridge between different programming languages, allowing components written in different languages to communicate with each other seamlessly. IDL defines the methods, data structures, and types that a component exposes to the external world, enabling interoperability in distributed systems.

Key characteristics and concepts of IDL include:

- Language Neutrality: IDL is designed to be independent of any specific programming language. It provides a way to define interfaces and data structures in a standardized format that can be understood by various programming languages.
- Interface Specification: In IDL, interfaces are defined by listing the methods that a component exposes. Each method includes its name, input parameters (if any), output parameters (if any), and exceptions that it may throw.
- Data Types: IDL supports various primitive and complex data types, including integers, floating-point numbers, strings, arrays, structures, and enums. These data types are defined in a language-independent way.
- Remote Procedure Calls (RPC): IDL is often used in the context of remote procedure calls (RPC), where a client in one language invokes methods on a remote component implemented in another language. IDL defines the methods and their signatures in a way that both client and server can understand.
- Stubs and Skeletons: When using IDL, code generation tools typically generate stubs (client-side proxies) and skeletons (server-side handlers) that handle the marshaling and unmarshaling of data for remote communication.

- Marshaling and Unmarshaling: IDL specifies how data should be marshaled (serialized) before being transmitted over the network and unmarshaled (deserialized) upon receipt. This ensures that data can be properly exchanged between different languages and platforms.

Rashmi Khadka Sameer Shrestha
Shiwani Shah Sujan Lamichhane

2074 (Back)

Qn 1: Why there are challenges in achieving some requirements of a distributed system? Explain the challenges associated with different requirements of distributed system.

Ans Challenges in achieving requirements of distributed systems often stem from issues such as network communication, data consistency, fault tolerance, and scalability. These challenges arise due to the inherent complexity of coordinating multiple interconnected components across different nodes, leading to issues like latency, synchronization problems, and ensuring reliable operation in the face of failures.

The challenges associated with different requirements of distributed system are:-

Scalability:

Scalability refers to a system's ability to handle an increasing load of work without sacrificing performance. Achieving scalability in a distributed system is challenging due to the following reasons:

1. Load Balancing: Distributing the workload evenly across multiple nodes is complex, as varying workloads and node capabilities can lead to imbalances, causing some nodes to become overwhelmed while others remain underutilized.

2. State Management: Maintaining consistency and synchronization of shared state across distributed nodes becomes complex, as adding more nodes can increase the chances of conflicts and contention.

3. Communication Overhead: As the number of nodes increases, the amount of communication between them also grows. This can lead to increased network overhead and latency, potentially affecting overall system performance.

4. Data Partitioning: Deciding how to partition data across nodes to achieve optimal performance and distribution is challenging, especially for dynamic workloads.

Fault Tolerance:

Fault tolerance refers to a system's ability to continue functioning properly in the presence of hardware or software failures. Achieving fault tolerance in a distributed system is challenging due to:

1. Node Failures: Nodes can fail due to hardware issues, software bugs, or network problems. Detecting and recovering from these failures while maintaining system consistency and availability is complex.

2. Network Partitions: Network failures can lead to split-brain scenarios, where nodes are partitioned into separate groups. Ensuring data consistency and preventing divergent behaviors in such scenarios is a challenge.

3. Replication Management: Replicating data across nodes to ensure availability introduces challenges related to consistency, synchronization, and conflict resolution in case of updates to replicated data.

Consistency and Availability:

Achieving the right balance between data consistency and system availability is a fundamental challenge in distributed systems:

1. CAP Theorem: The CAP theorem states that a distributed system can achieve only two out of three properties: Consistency, Availability, and Partition Tolerance. Balancing these aspects to meet application requirements is a challenge.

2. Eventual Consistency: Ensuring that distributed data eventually becomes consistent after updates while maintaining high availability can be complex, as it involves managing conflicts and reconciliation.

3. Latency vs. Consistency: Maintaining strict consistency may require synchronous operations, which can increase latency. Achieving low-latency responses while ensuring consistency is a trade-off.

Q.no 2: Define distributed object and IDL. Compare RPC and RMI

Ans A distributed object is an entity within a distributed system that encapsulates both data and the methods (functions) that can operate on that data. It represents a real-world object or an abstract concept that can be shared and accessed across multiple nodes in a network. Distributed objects facilitate communication and interaction between different components of a distributed system, allowing them to collaborate seamlessly even if they reside on different machines

The Interface Definition Language (IDL) is a specification language used in distributed computing to define the interface of distributed objects or components. IDL provides a language-neutral way to describe the methods, data structures, and interactions that can be performed on a distributed object.

Remote Procedure Call (RPC)	Remote Method Invocation (RMI)
1.RPC enables bidirectional communication between a client and a remote server. Clients can call procedures or functions on the server as if they were local calls.	1. RMI also facilitates bidirectional communication, but it's tailored for Java-specific remote object-oriented interactions.

2. In RPC, communication between different programming languages and platforms requires explicit marshaling and unmarshaling of data, which can be complex and error-prone.	2. RMI is Java-specific and doesn't naturally extend to other programming languages and platforms.
3. Data serialization involves manually converting data structures into a format that can be transmitted over the network and then converting it back on the receiving end.	3. RMI provides automatic serialization and deserialization of objects, making data exchange for Java objects straightforward
4. RPC doesn't typically have built-in support for remote object-oriented interactions, focusing more on procedure calls.	4. RMI is designed specifically for remote object-oriented interactions. It treats objects as remote entities that can be manipulated remotely.
5. : Parameters are explicitly passed and received as arguments and return values during procedure calls.	5. RMI abstracts away the lower-level parameter passing details, and objects can be passed and returned as method arguments and return values
6. RPC often lacks dynamic discovery and invocation mechanisms for remote procedures. The client needs to be aware of the server's interface.	6. RMI includes mechanisms for dynamic discovery of remote objects and their methods, providing more flexibility in interacting with distributed components.

Rashmi Khadka Sameer Shrestha
Shiwani Shah Sujan Lamichhane

Q.no3: What is stateful and stateless service in file system? Explain the DNS working mechanism with suitable practical example.

Ans A stateful service in a file system refers to a service or component that maintains and relies on specific state information throughout interactions or operations. In the context of a file system, a stateful service maintains information about the current state of files, directories, and their attributes. This state might include metadata such as file sizes, access permissions, timestamps, and data block locations. Stateful services typically require a persistent storage mechanism to store and retrieve this state information, ensuring its availability across sessions or restarts.

A stateless service in a file system refers to a service or component that does not rely on maintaining specific state information between interactions. In a stateless service, every operation is independent and self-contained. In the context of a file system, a stateless service might receive all the necessary information for an operation (e.g., file read or write) as input and produce a result without relying on previously stored state.

The Domain Name System (DNS) is a hierarchical and distributed naming system that translates human-readable domain names into IP addresses, which are used to locate resources on the internet. DNS serves as a critical component of the internet infrastructure, enabling users to access websites, send emails, and perform various online activities using easily memorable domain names.

Explanation of how DNS works using a practical example:

Step 1: User Types a URL : Imagine a user wants to access a website by typing its URL in a web browser. For instance, the user types "www.example.com" and presses Enter.

Step 2: Querying the Local DNS Resolver

The user's computer doesn't directly know the IP address of "www.example.com." It first sends a query to a local DNS resolver (usually provided by the Internet Service Provider or ISP).

Step 3: Local DNS Resolver Cache Check

The local DNS resolver checks its cache to see if it has the IP address for "www.example.com" from previous queries. If the IP address is found and is still valid, the resolver returns the IP address to the user's computer.

Step 4: Recursive Query

If the local DNS resolver doesn't have the IP address in its cache, it becomes responsible for finding the IP address. It sends a recursive query to the root DNS servers, asking for the IP address of "www.example.com."

Step 5: Root DNS Server Response

The root DNS servers respond to the local DNS resolver, indicating the authoritative DNS server for the top-level domain (TLD) ".com."

Step 6: TLD DNS Server Query

The local DNS resolver sends another query, this time to the TLD DNS server responsible for ".com," asking for the authoritative DNS server for "example.com."

Step 7: Authoritative DNS Server Query

The TLD DNS server responds with the IP address of the authoritative DNS server for "example.com."

Step 8: Authoritative DNS Server Response

The local DNS resolver sends a query to the authoritative DNS server for "example.com," asking for the IP address of www.example.com.

Step 9: IP Address Response

The authoritative DNS server for "example.com" responds with the IP address of "www.example.com."

Step 10: Local DNS Resolver Cache Update

The local DNS resolver caches the IP address for "www.example.com" to expedite future queries.

Step 11: User's Computer Accesses the Website

Armed with the IP address, the user's computer can now establish a connection to the web server hosting "www.example.com" and retrieve the web content.

Qno4: What are the characteristics of distributed operation system? Explain ORB and its interfaces.

Ans Distributed operating systems possess several distinct characteristics that differentiate them from traditional single-machine operating systems. Some key characteristics include:

1. *Distribution*: Distributed operating systems manage resources and tasks across multiple interconnected machines, enabling collaboration and efficient resource utilization.
2. Transparency: They aim to provide transparency to users and applications, hiding the complexities of distributed infrastructure and making it appear as a single, cohesive system.
3. Concurrency: Distributed systems handle concurrent execution of processes across different nodes, often requiring synchronization mechanisms to maintain consistency.
4. Heterogeneity: These systems can involve diverse hardware, software, and network technologies, leading to challenges in interoperability and communication.
5. Scalability: Distributed operating systems can scale by adding more nodes to the network, accommodating increasing workloads and users.
6. Fault Tolerance: They incorporate mechanisms to manage failures, ensuring system availability even when individual nodes or components fail.

7. Security: Security mechanisms are vital due to the distributed nature and potential vulnerabilities introduced by network communications.
8. Resource Sharing: Distributed operating systems facilitate sharing of resources like files, printers, and computational power among nodes.
9. Location Independence: Users and applications can access resources irrespective of their physical location in the network.

An Object Request Broker (ORB) is a middleware component that facilitates communication between distributed objects in a distributed system. It acts as an intermediary, managing requests and responses between clients and server objects, allowing them to interact seamlessly even if they reside on different machines. ORBs provide various interfaces to enable developers to define object interactions and communication protocols:

Interface Definition Language (IDL): IDL is a specification language used to define the interfaces and methods of distributed objects. It abstracts away implementation details, allowing objects written in different programming languages to communicate transparently. IDL interfaces define the methods a remote object supports, their parameters, and return values.

Client Stub: The client-side stub is generated based on the IDL interface. When a client wants to invoke a method on a remote object, it calls the method on the stub, which takes care of marshaling the method arguments, sending the request over the network, and waiting for the response.

Server Skeleton: The server-side skeleton is generated based on the IDL interface as well. When the ORB receives a request from the client, it dispatches the request to the appropriate server object's skeleton, which unmarshals the arguments, invokes the method, and marshals the response back to the client.

Object Adapter: The Object Adapter is responsible for managing the lifecycle of objects, activating and deactivating them as needed. It provides a bridge between the ORB and the application objects.

Naming Service: Some ORBs also include a naming service that allows clients to locate objects by name instead of needing to know their network locations.

Rashmi Khadka Sameer Shrestha
Shiwani Shah Sujan Lamichhane

Q.no:5 Why is clock synchronization is necessary ? Explain clock synchronization algorithm using vector clock along with example.

Ans Clock synchronization is essential in distributed systems to ensure that various nodes have a consistent and accurate notion of time. Accurate time synchronization is crucial for various reasons:

- ⇨ Event Ordering: Many distributed algorithms rely on the correct ordering of events. Inconsistent clocks can lead to incorrect sequencing of events and cause anomalies in system behavior.
- ⇨ Concurrency Control: Accurate time synchronization aids in handling concurrent operations, ensuring proper synchronization of processes and preventing conflicts.
- ⇨ Distributed Coordination: Many coordination and consensus algorithms require synchronized clocks to make correct decisions.
- ⇨ Logging and Debugging: Synchronized clocks help in correlating log entries and events across different nodes, making debugging and troubleshooting easier.
- ⇨ Security: Timestamps are often used in security protocols to prevent replay attacks and ensure the freshness of data.

Vector Clock is an algorithm that generates partial ordering of events and detects causality violations in a distributed system. These clocks expand on Scalar time to facilitate a causally consistent view of the distributed system, they detect whether a contributed event has caused another event in the distributed system. It essentially captures all the causal relationships. This algorithm helps us label every process with a vector(a list of integers) with an integer for each local clock of every process within the system. So for N given processes, there will be vector/ array of size N.

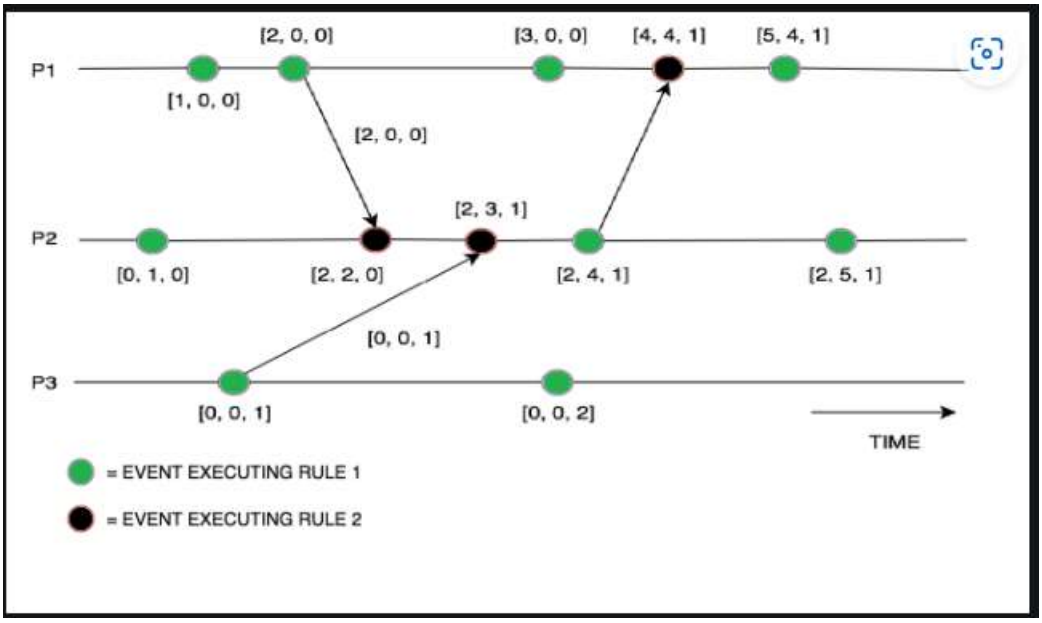
How does the vector clock algorithm work :

- ⇨ Initially, all the clocks are set to zero.
- ⇨ Every time, an Internal event occurs in a process, the value of the processes's logical clock in the vector is incremented by 1

- ⇨ Also, every time a process sends a message, the value of the processes's logical clock in the vector is incremented by 1.

Every time, a process receives a message, the value of the processes's logical clock in the vector is incremented by 1, and moreover, each element is updated by taking the maximum of the value in its own vector clock and the value in the vector in the received message (for every element).

Example : Consider a process (P) with a vector size N for each process: the above set of rules mentioned are to be executed by the vector clock:



The above example depicts the vector clocks mechanism in which the vector clocks are updated after execution of internal events, the arrows indicate how the values of vectors are sent in between the processes (P1, P2, P3).

Rashmi Khadka Sameer Shrestha
Shiwani Shah Sujan Lamichhane

Q.no:6 Describe non-token based centralized and Ricart Agrawala algorithm with example and compare them.

Ans Non-token based centralized algorithm is a mutual exclusion algorithm in which a central coordinator is responsible for granting permission to processes to enter the critical section. The central coordinator maintains a list of all processes that are currently in the critical section, and any process that wants to enter the critical section must first send a request to the central coordinator. The central coordinator will then grant permission to the process with the highest priority, and the process will be able to enter the critical section.

Ricart-Agrawala algorithm is a non-token based mutual exclusion algorithm that does not require a central coordinator. In this algorithm, each process maintains a list of all the other processes in the system, and each process also maintains a timestamp. When a process wants to enter the critical section, it sends a REQUEST message to all the other processes in the system. The REQUEST message includes the process's timestamp. The other processes then reply to the REQUEST message with a REPLY message. The REPLY message includes the process's timestamp and a flag that indicates whether or not the process is currently in the critical section.

The comparison between Non-token based centralized algorithm and Ricart-Agrawala algorithm is:-

- ⇨ The non-token based centralized algorithm requires a central coordinator, while the Ricart-Agrawala algorithm does not.

- ⇨ Ricart-Agrawala algorithm more scalable, as it does not require a dedicated server to act as the central coordinator. Whereas the other is less scalable.

- ⇨ Ricart-Agrawala algorithm is more complex than the non-token based centralized algorithm, and it may require more messages to be exchanged.

- ⇨ Non-token based centralized algorithm requires only 3 messages per access to critical situation but Ricart-Agrawala algorithm needs more.

Rashmi Khadka Sameer Shrestha
Shiwani Shah Sujan Lamichhane

Q.no:7 Differentiate between active and passive replication. Explain working mechanism of active replication.

Ans The differences between active and passive replication are:

Active Replication	Passive Replication
1. Active replication is a replication strategy in which all replicas receive and process the same client request.	1. Passive replication is a replication strategy in which only one replica, called primary, receives and processes client requests.
2. All replicas are always in sync.	2. Backup replicas may be out of sync with the primary
3. All replicas are available to clients.	3. Only the primary replica is available to clients.

The working mechanism of active replication is as follows:

i. A client sends a request to any replica. The client does not need to know which replica is the primary or secondary. This is because all replicas are equally important in active replication.

ii. The replica receives the request and processes it. The replica executes the request and updates its states accordingly.

iii. The replica sends the response to the client. The replica sends the response to the client, along with a copy of the uploaded state.

iv. The other replicas receive the response and update their state to match the state of the replica that processed the request. The other replicas receive the response from the first

replica and update their state to match the state of the first replica. This ensures that all replicas are always in sync with each other.

Q.no:8 How cascading aborts occurs and solved? Explain three phase commit protocol with state diagram.

Ans Cascading aborts occur in distributed systems when a transaction is aborted, and this causes other transactions that have already committed to be aborted as well. This can happen if the aborted transaction has updated data that was also read by other transactions. If these other transactions are not aware that the data has been updated, they may continue to use the old data, which can lead to inconsistencies.

Cascading aborts can be solved by using a protocol called three-phase commit (3PC). 3PC is a distributed protocol that ensures that all participants in a transaction either commit or abort the transaction together. This prevents cascading aborts by ensuring that no transaction can update data that is shared with other transactions until all participants have agreed to commit the transaction.

Three-Phase Commit (3PC) Protocol is an extension of the Two-Phase Commit (2PC) Protocol that avoids blocking problem under certain assumptions. In particular, it is assumed that no network partition occurs, and not more than k sites fail, where we assume 'k' is predetermined number. With the mentioned assumptions, protocol avoids blocking by introducing an extra third phase where multiple sites are involved in the decision to commit.

Instead of directly noting the commit decision in its persistent storage, the coordinator first ensures that at least 'k' other sites know that it intended to commit transaction.

In a situation where coordinator fails, remaining sites are bound to first select new coordinator. This new coordinator checks status of the protocol from the remaining sites. If the coordinator had decided to commit, at least one of other 'k' sites that it informed will be up and will ensure that commit decision is respected. The new coordinator restarts third phase of protocol if any of rest

sites knew that old coordinator intended to commit transaction. Otherwise, new coordinator aborts the transaction.

The states of the coordinator and each participant satisfy the following two conditions:

i. There is no single state from which it is possible to make a transition directly to either a COMMIT or an ABORT state.

ii. There is no state in which it is not possible to make a final decision, and from which a transition to a COMMIT state can be made.

Co-ordinator, sends vote-request message to all the participants, after which it waits for incoming response.

If any participant votes to abort the transactions the final decision will be to-abort as well, so the coordinator sends global abort.

However the transaction can be committed, a prepare-commit message is sent. Only after each participants has acknowledge, it is now prepared to commit, will the coordinator send the final commit message by which the transaction is actually committed.

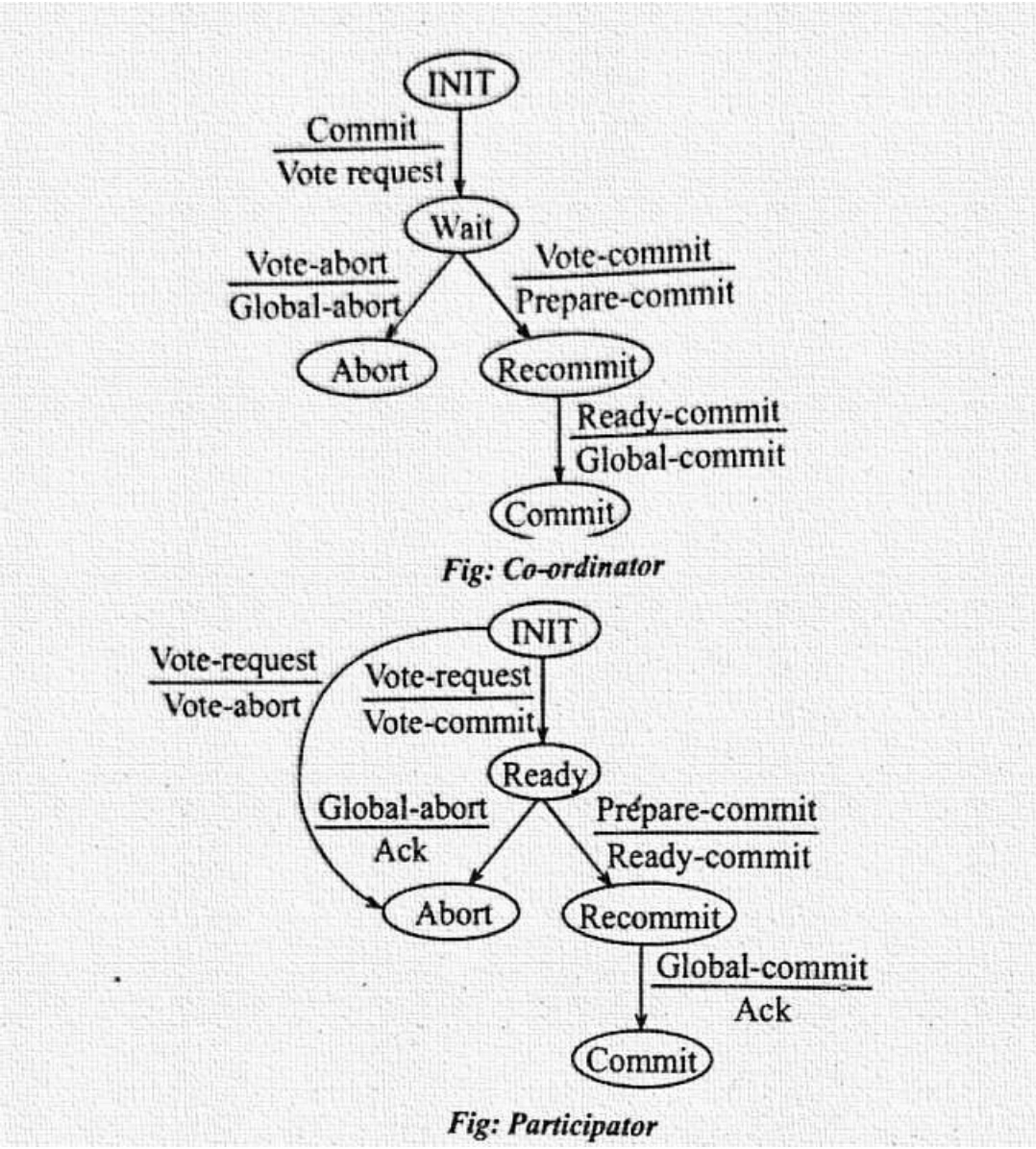


Fig: Co-ordinator

Fig: Participator

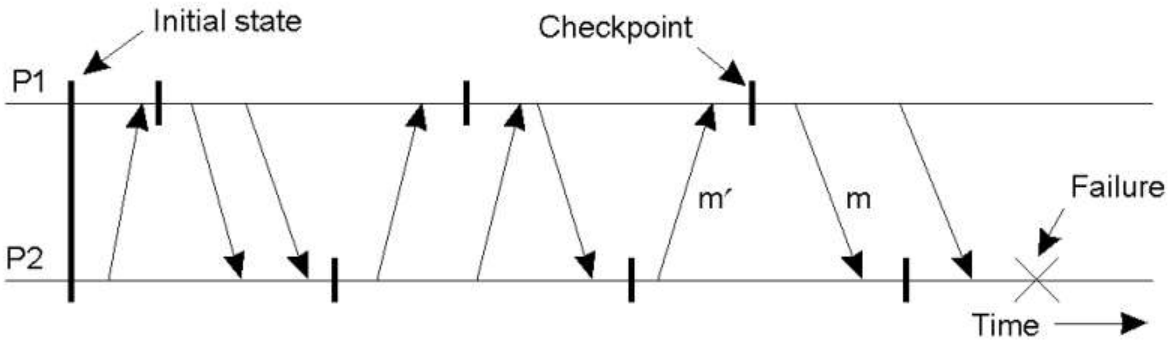
Q.no:9 What is K-fault tolerant system? Explain fault recovery techniques.

Ans A k-fault tolerant system is a system that can continue to operate even if k of its components fail. This is achieved by having redundant components in the system, so that if one component fails, another component can take over its function.

The fault recovery techniques are:-

a. Independent checkpoint

- Each process periodically checkpoints independent of other processes.
- Upon failures, locate a consistent cut background.
- Needs to rollback until consistent cut is found.



b. Coordinated checkpoint

A checkpointing protocol in distributed systems can be coordinated, independent or quasi-synchronous. Coordinated Checkpointing is an attractive checkpointing strategy as it is domino-free and requires a minimum number of checkpoints (maximum two checkpoints) to be stored per process on stable storage. In the coordinated checkpointing, all processes take checkpoints .cooperatively and in a synchronized way. It is, therefore, also known as "Synchronous checkpointing". In this scheme the synchronization of processes is required for checkpointing and so, extra check pointing messages are required to be exchanged between processes. Also, the underlying computation may have to be frozen.

Once a synchronous checkpoint of the distributed computation has been recorded on stable storage, it can be used to recover from any future fault. If a fault occurs, all processes roll back and restart from the global consistent state given by this checkpoint. A global state of a distributed system contains one checkpoint of each process and is consistent if it does not contain any orphan messages. An orphan message is a message, whose receiving has been recorded by the destination process in its checkpoint but sending is not recorded by the sender in its checkpoint.

c. Message logging

Message logging is a common technique used to build systems that can tolerate process crash failures. These protocols require that each process periodically records its local state and log the messages received since recording that state. When a process crashes, a new process is created in its place: the new process is given the appropriate recorded local state, and then it replays the logged messages in the order they were originally received. All message-logging protocols require that the state of a recovered process be consistent with the states of the other processes. This consistency requirement is usually expressed in terms of orphan processes, which are surviving processes whose state is inconsistent with the recovered state of a crashed process. Therefore, the big question is how our implementation of message logging, will guarantee that after recovery no process is orphan.

Q.no:10 Write short notes on:

a) Distributed deadlock and recovery:

A distributed deadlock is a situation in which two or more processes in a distributed system are blocked, waiting for each other to release resources. This can happen when processes need to access resources that are held by other processes.

Deadlocks can be prevented by using deadlock avoidance or deadlock detection and recovery.

- Deadlock avoidance involves preventing situations where deadlocks can occur. This can be done by using a variety of techniques, such as resource allocation graphs and timestamps.

- Deadlock detection and recovery involves detecting deadlocks when they occur and then taking steps to recover from them. This can be done by using a variety of techniques, such as the wait-for graph algorithm and the banker's algorithm.

When a deadlock is detected, there are a few different recovery strategies that can be used:

- Rollback: This involves aborting one or more of the deadlocked processes and then restarting them.

- Timeout: This involves waiting for a period of time and then aborting one or more of the deadlocked processes.

- Victim selection: This involves selecting one of the deadlocked processes to abort. This is a more complex strategy, as it requires choosing a process that will cause the least amount of damage.

The choice of recovery strategy depends on the specific requirements of the system. For example, a system that requires high availability may use rollback. A system that is more concerned with cost may use timeout.

b. MACH

Mach is a powerful and flexible operating system that is well-suited for distributed systems. Its modular design and message passing support make it a good choice for systems that need to be scalable and extensible.

The key features of Mach are:

- Microkernel design: The microkernel design makes Mach very flexible and extensible, as it allows new services to be added without having to modify the kernel itself.

- Message passing: Mach uses message passing as its primary mechanism for communication between processes. This makes it well-suited for distributed systems, as it allows processes to communicate with each other regardless of their physical location.

- Portability: Mach is designed to be portable to a variety of hardware platforms. This makes it a good choice for distributed systems that need to be deployed on a variety of different machines.

- Security: Mach provides a variety of security features, such as process isolation and access control lists. This makes it a good choice for distributed systems that need to be secure.