Recovery

In the context of distributed databases, recovery refers to the process of restoring the database to a consistent state after a failure or disruption. Distributed databases are spread across multiple locations or nodes, and recovery mechanisms ensure that data remains consistent and available even in the event of failures.

Importance of Recovery in Distributed Databases -

➔ **Consistency**: Ensures that all nodes have a consistent view of the data, even after failures.

➔ **Availability**: Keeps the database available and operational despite node or network failures.

➔ **Durability**: Ensures that once a transaction is committed, it remains permanent, even in the event of failures.

➔ **Fault Tolerance**: Enhances the system's ability to withstand and recover from failures without data loss.

NOTE: *Recovery in distributed databases is crucial for maintaining data integrity, availability, and reliability across multiple nodes and locations. It involves a combination of transaction management, failure handling, data replication, checkpointing, and log-based recovery techniques.*

Transaction: A **transaction** is a sequence of one or more operations (such as read, write, update, or delete) performed as a single logical unit of work, that ensures data integrity and consistency across multiple distributed nodes or sites.

Transactions in distributed databases are governed by specific principles to ensure that they are processed reliably and that the database remains in a consistent state, even in the presence of failures.

Types of Transactions in Distributed Databases -

(a) **Local Transactions**:

- **Definition**: Transactions that involve operations on data located at a single site or node.
- **Example**: Updating a user's profile information stored on a single server.

(b) **Global Transactions**:

- **Definition**: Transactions that involve operations on data distributed across multiple sites or nodes.
- **Example**: Processing an online order that involves inventory checks at different warehouses located in various geographical regions.

Managing Distributed Transactions -

(a) **Two-Phase Commit Protocol (2PC):**

- **Definition:** A widely-used protocol to ensure that all nodes involved in a global transaction either commit or roll back the transaction in a coordinated manner.

- **Phases:**
  - ➔ **Prepare Phase:** The coordinator node asks all participating nodes to prepare to commit the transaction. Each node responds with a "prepared" or "abort" message.
  - ➔ **Commit Phase:** If all nodes are prepared, the coordinator sends a commit message. If any node aborts, the coordinator sends a rollback message.

(b) Concurrency Control:

- **Definition:** Mechanisms to ensure that multiple transactions can execute concurrently without causing data inconsistency.
- **Example:** Locking mechanisms and timestamp ordering to manage access to shared data.

(c) Fault Tolerance and Recovery:

- **Definition:** Ensuring that the system can recover from failures without losing transaction data or compromising consistency.
- **Example:** Using transaction logs and checkpoints to restore the database to a consistent state after a failure.

Challenges in Distributed Transactions -

(a) **Network Latency:** The delay in communication between distributed nodes can impact the performance of distributed transactions.

(b) **Data Consistency:** Ensuring data consistency across multiple nodes, especially in the presence of concurrent transactions and failures, is challenging.

(c) **Coordination Overhead:** Coordinating global transactions requires additional overhead for communication and synchronization between nodes.

NOTE: *Transactions in distributed databases are critical for maintaining data integrity and consistency across multiple nodes or sites. They are governed by the ACID properties and managed through protocols like the Two-Phase Commit (2PC) to ensure reliable execution. Despite challenges such as network latency and coordination overhead, distributed transactions enable robust and fault-tolerant database systems.*

Transaction Recovery: Transaction Recovery in distributed databases refers to the process of restoring the database to a consistent state after a transaction failure. The goal is to ensure that all committed transactions are preserved and that incomplete transactions are properly rolled back, maintaining the integrity and consistency of the database. Transaction Recovery in distributed databases refers to the process of restoring the database to a consistent state after a transaction failure. The goal is to ensure that all committed transactions are preserved and that incomplete transactions are properly rolled back, maintaining the integrity and consistency of the database.

System Recovery: System Recovery refers to the process of restoring a computer or digital system to its normal operating state after experiencing a failure, crash, or other disruptive event. The goal of system recovery is to minimize downtime, prevent data loss, and ensure that the system operates correctly again.

Media Recovery: Media Recovery refers to the process of restoring data from storage media that has become damaged, corrupted, or inaccessible. This process is crucial for retrieving lost files, ensuring data integrity, and maintaining the functionality of digital storage devices. Media recovery can be applied to a variety of storage media, including hard drives, solid-state drives (SSDs), USB drives, CDs, DVDs, and memory cards.

Two-Phase Commit (2PC):

The **Two-Phase Commit (2PC)** protocol is a distributed algorithm used to ensure all participants in a distributed transaction either commit the transaction or abort it in a coordinated manner. This protocol is essential for maintaining atomicity and consistency in distributed databases. The 2PC protocol is particularly useful in scenarios where a single transaction involves multiple nodes or databases that must all agree on the transaction outcome.

Phases of the Two-Phase Commit Protocol

1. **Prepare Phase (Voting Phase)**:
    - **Coordinator**: The coordinating node (transaction manager) sends a prepare message to all participating nodes (participants) involved in the transaction.
    - **Participants**: Each participant node performs the necessary checks and prepares to commit the transaction. If a participant is ready to commit, it sends a vote-commit message back to the coordinator. If a participant encounters any issues, it sends a vote-abort message.
    - **Outcome**: The coordinator collects votes from all participants.
2. **Commit Phase (Completion Phase)**:
    - **Coordinator**: Based on the votes received, the coordinator decides the outcome of the transaction:
        - If all participants vote to commit, the coordinator sends a commit message to all participants, instructing them to commit the transaction.
        - If any participant votes to abort, the coordinator sends an abort message to all participants, instructing them to abort the transaction.
    - **Participants**: Upon receiving the final decision from the coordinator, participants either commit or abort the transaction as instructed.

Advantages of Two-Phase Commit

- **Atomicity**: Ensures that the entire distributed transaction is committed or aborted as a single unit, maintaining atomicity.
- **Consistency**: Helps maintain data consistency across multiple nodes, ensuring that all nodes reflect the same state after the transaction.
- **Coordination**: Provides a structured and coordinated approach to handle distributed transactions involving multiple participants.

Disadvantages of Two-Phase Commit

- **Blocking**: If the coordinator fails during the commit phase, participants may be left in an uncertain state, leading to a potential blocking scenario.
- **Performance Overhead**: The 2PC protocol involves multiple rounds of communication and can introduce latency, impacting performance in high-transaction environments.
- **Single Point of Failure**: The coordinator is a critical component, and its failure can disrupt the transaction process.