**5. Transaction States**

A transaction progresses through different states:

* **Active state:** Beginning of execution.
* **Partially committed state:** End of read/write operations, but changes are not yet permanent.
* **Committed state:** All changes are permanently saved.
* **Failed state:** Transaction is aborted due to a failure.
* **Terminated State:** Transaction leaves the system.

**6. Why Concurrency Control is Needed**

While concurrent execution offers advantages, it can lead to problems if uncontrolled:

* **Problems with Uncontrolled Concurrent Execution:** "If this concurrent execution is uncontrolled, it may lead to problems, such as an inconsistent database."
* **Main Problems:Reading uncommitted data (WR conflict - "dirty read"):** Occurs when a transaction reads data updated by another transaction that subsequently fails.
* **Unrepeatable read (RW conflict):** Occurs when a transaction reads data and then, when attempting to read the same data again, finds it has been modified or deleted by another concurrent transaction. "This inconsistency can lead to unexpected and incorrect results."
* **The Lost Update Problem (WW conflict):** Occurs when two transactions update the same data concurrently, resulting in one update being overwritten and lost.

**7. Schedules and Serializability**

* **Schedule:** "a schedule refers to the sequential or chronological order in which instructions of concurrent transactions are executed." A valid schedule must include all instructions and preserve their order within each transaction.
* **Types of Schedule:Serial Schedule:** Operations of each transaction are executed consecutively without interleaving from other transactions. "Inconsistency not present here."
* **Concurrent Schedule:** Operations from different transactions are interleaved.
* **Serializability:** "Serializability is a property of a database transaction that ensures that the final state of the database, after executing a set of transactions concurrently, [is the same] as if the transactions had in fact executed one-at-a-time." The objective of concurrency control is to produce serializable schedules.
* **Conflict Serializability:** A schedule is conflict serializable if it can be converted into a serial schedule by swapping non-conflicting operations.
* **View Serializability:** A schedule is view serializable if it is view equivalent to a serial schedule, considering initial reads, final writes, and intermediate reads.

**8. Concurrency Control Techniques**

Concurrency control techniques aim to ensure isolation and preserve database consistency:

* **Purpose of Concurrency Control:**"To enforce Isolation (through mutual exclusion) among conflicting transactions."
* "To preserve database consistency through consistency preserving execution of transactions."
* "To resolve read-write and write-write conflicts."
* **Main Techniques:**Locking
* Timestamping
* Multiversion
* Optimistic methods (Validation or Certification)

**9. Locking**

* **Locking Operation:** "Locking is an operation which secures (a) permission to Read (b) permission to Write a data item for a transaction."
* **Unlocking Operation:** "Unlocking is an operation which removes these permissions from the data item."
* **Lock Modes:Shared mode (read):** Multiple transactions can hold a shared lock for reading. "Reads cannot conflict, so more than one transaction can hold shared locks simultaneously on same item."
* **Exclusive mode (write):** Only one transaction can hold an exclusive lock for both reading and writing. "Exclusive lock gives transaction exclusive access to that item."
* **Lock-based protocols:** "Lock based protocol insure conflict serialazblity. It insure consisitency."
* **Basic Rules:** A transaction requests access by issuing lock\_Item(X). If the item is already locked, the transaction waits. If not, the lock is granted, and the transaction can access the item. Unlock\_Item releases the lock.

**10. Limitations of Lock-based Protocol**

* **Deadlock:** "It is a state that may result when two or more transactions are each waiting for locks held by the other to be released." This creates a cycle in the wait-for-graph.
* **Starvation:** "Starvation occurs when a particular transaction consistently waits or restarted and never gets a chance to proceed further while other transaction continue normally." This can happen due to waiting methods or victim selection algorithms.

**11. Dealing with Deadlock and Starvation**

* **Deadlock Prevention:Conservative two-phase locking:** Transactions lock all data items before execution, preventing waits but restricting concurrency.
* **Transaction Timestamp:** Assigning priorities based on timestamps, ensuring lower-priority transactions don't wait for higher-priority ones. "The lower the timestamp, the higher the transaction's priority."
* **Deadlock Detection and Resolution:** Allows deadlocks to happen and uses a wait-for-graph to detect cycles. A victim transaction is selected and aborted to break the cycle.
* **Timeouts:** Aborts a transaction if it waits for a lock for a longer period than a predefined timeout. "It has lower overhead cost and it is simple."
* **Starvation Solutions:** Using FIFO queue for waiting transactions or giving higher priority to transactions that have been aborted multiple times.

**12. Two-Phase Locking Techniques (2PL)**

* **Principle:** "Transaction follows 2PL protocol if all locking operations precede the first unlock operation in the transaction." Transactions have a growing phase (acquiring locks) and a shrinking phase (releasing locks).
* **Types of Two Phase Locking:Simple Two phase lock:** Basic 2PL.
* **Strict Two phase locking:** A transaction holds write locks until it commits or aborts.
* **Rigorous two phase locking:** A transaction holds all locks (exclusive and shared) until it commits or aborts.
* **Locking Conversion:** Allows upgrading (S to X) in the growing phase and downgrading (X to S) in the shrinking phase.

**13. Other Concurrency Control Techniques**

* **Timestamp-based concurrency control algorithm:** Uses timestamps to serialize execution without using locks, thus avoiding deadlocks. Conflicts are ordered based on timestamps.
* **Multiversion Concurrency Control Techniques:** Maintains multiple versions of data items. Read operations are never rejected as they are given an appropriate version. "This algorithm uses the concept of view serilazability than conflict serialiazability." A side effect is the need for significantly more storage.
* **Validation (Optimistic) Concurrency Control Schemes:** Transactions proceed asynchronously and serializability is checked only at commit time. Transactions are aborted if validation fails. It has three phases: Read, Validation, and Write. This technique is good when there is little interference among transactions.

**14. Lock Granularity**

* **Definition:** "A lockable unit of data defines its granularity." It can range from a field to the entire database.
* **Impact on Concurrency:** "Data item granularity significantly affects concurrency control performance." Coarse granularity (e.g., entire database) leads to low concurrency, while fine granularity (e.g., a field) leads to high concurrency. The optimal granularity depends on the transaction's access patterns.

This briefing summarizes the essential points from the provided source regarding database transactions, concurrent processing, the ACID properties, concurrency control challenges, and various techniques used to address them, with a particular focus on locking.

Chapter 5

Briefing Document: Distributed Databases

This briefing document provides an overview of distributed databases based on the provided excerpts from "ADB Chapter 5 - Distributed Databases.pdf". It highlights key concepts, advantages, disadvantages, functions, data management techniques, types of distributed systems, and challenges in query processing, concurrency control, and recovery.

**I. Fundamental Concepts**

* **Distributed Computing System:** Defined as a system with multiple, potentially heterogeneous, processing elements interconnected by a network that cooperate on tasks.
* **Distributed Database (DDB):** A collection of multiple, logically interrelated databases spread across a computer network.
* **Distributed Database Management System (DDBMS):** Software that manages a distributed database while making the distribution transparent to the user.

**II. Advantages of Distributed Databases**

Distributed databases offer significant benefits over centralized systems:

* **Management of distributed data with different levels of transparency:** This is a crucial aspect, hiding the complexities of data distribution from the user. Key types of transparency include:
* **Data Transparency:** Users access data as if it were in a single location, regardless of its physical distribution.
* **Location Transparency:** The system hides the physical location of data.
* **Replication Transparency:** The system manages data replication across nodes automatically and transparently.
* **Failure Transparency:** The system handles node failures transparently, allowing continued data access.
* **Scalability Transparency:** The system scales out to handle increasing data and user load without user intervention.
* **Distribution or Network Transparency:** Users are shielded from network operational details. This includes:
* **Location Transparency:** Freedom to issue commands from any location without affecting operation.
* **Naming Transparency:** Access to named objects (files, relations, etc.) from any location.
* **Fragmentation Transparency:** Users are unaware of the existence of data fragments.
* **Increased reliability and availability:** "A distributed database system has multiple nodes (computers) and if one fails then others are available to do the job." Reliability is the probability the system is running at a certain time, while availability is the probability it's continuously accessible.
* **Improved performance:** By fragmenting and locating data closer to where it's most needed, a DDBMS reduces data management time.
* **Easier expansion (scalability):** New nodes can be added easily without changing the entire system structure.

**III. Disadvantages of Distributed Databases**

Despite the advantages, DDBMSs also present challenges:

* **Complexity:** Managing data replication, failure recovery, and the network makes DDBMSs more complex than centralized systems.
* **Cost:** Requires more personnel and hardware, leading to potentially higher costs.
* **Problem of Connecting Dissimilar Machines:** Requires additional operating system software to translate and coordinate data flow between different machine types.
* **Data integrity and security problems:** "Because data maintained by distributed systems can be accessed at any locations in the network, controlling the integrity of a database can be difficult."

**IV. Functions of a DDBMS**

A DDBMS performs several key functions:

* Keeping track of data distribution, fragmentation, and replication through an expanded catalog.
* Distributed query processing: Accessing remote sites and transmitting queries and data over the network.
* Distributed transaction management: Devising strategies for transactions accessing data from multiple sites and synchronizing access while maintaining integrity.
* Replicated data management: Deciding which copy of replicated data to access and maintaining consistency.
* Distributed database recovery: Recovering from individual site crashes and communication link failures.
* Security: Managing data security and user authorization/access privileges for distributed transactions.
* Distributed directory (catalog) management: Managing metadata about data, which can be global or local.

**V. Data Management Techniques**

Two primary techniques for storing data in a distributed database are:

* **Data Fragmentation:** Breaking down the database into logically related units called fragments.
* **Horizontal Fragmentation:** Dividing a relation by rows based on selection conditions. "divides a relation 'horizontally' by grouping rows to create subsets of tuples, where each subset has a certain logical meaning."
* **Vertical Fragmentation:** Dividing a relation by columns. "divides a relation 'vertically' by columns." Each fragment must include the primary key.
* **Mixed (Hybrid) Fragmentation:** A combination of horizontal and vertical fragmentation.
* **Fragmentation Rules:** Completeness (data item in at least one fragment), Reconstruction (original relation can be rebuilt), and Disjointness (data item in only one fragment).
* **Data Replication:** Storing copies of the database or parts of it at multiple sites.
* **Full Replication:** The entire database is replicated at all sites, improving availability and global query performance but potentially slowing down updates.
* **Partial Replication:** Selected parts of the database are replicated at some sites.
* Data distribution or allocation is the process of assigning fragments or copies to specific sites.

**VI. Types of Distributed Systems**

Distributed database systems can be categorized based on their setup and autonomy:

* **Homogeneous:** All sites use identical database system software and may have limited local autonomy.
* **Heterogeneous:** Sites may run different database systems.
* **Federated:** Each site runs a different DBMS, but data access is managed through a single conceptual schema with minimal local autonomy and potentially a global schema.
* **Multidatabase:** No single conceptual global schema; schemas are constructed dynamically. Each server is an independent and autonomous centralized DBMS with high local autonomy.

**VII. Query Processing in Distributed Databases**

Query processing in a DDBMS involves significant optimization challenges, primarily due to the cost of data transfer over the network.

* **Issues:** The high cost of transferring data (files and results) necessitates optimization.
* **Optimization Criteria:** Minimizing data transfer (communication cost) is the preferred optimization criteria. The source provides examples illustrating how choosing the site for a join operation can significantly impact the amount of data transferred.

**VIII. Concurrency Control and Recovery**

Distributed databases introduce unique concurrency control and recovery problems:

* **Challenges:**Dealing with multiple copies of data items while maintaining global consistency (all nodes having the same view of data).
* Handling individual site failures without affecting database availability.
* Managing communication link failures, which can create network partitions.
* Distributed commit: Ensuring transactions updating data at multiple sites are either fully committed or rolled back across all sites. This often requires two or three-phase commit protocols.
* Distributed deadlock: Deadlocks can occur across multiple sites and are more difficult to detect and resolve than in single-node systems.
* **Distributed Deadlock Detection Methods:**Centralized Deadlock Detection (central coordinator examines system state).
* Distributed Deadlock Detection (nodes exchange information to detect deadlocks).
* Timeouts (processes wait for a specified time before assuming a deadlock).
* **Distributed Deadlock Resolution Strategies:**Aborting involved processes.
* Preempting resources held by involved processes.
* Rolling back involved transactions.
* **Distributed Concurrency Control based on a Distinguished Copy:**A "distinguished copy" is the authoritative or primary copy of a data item. Updates are made to this copy first and then propagated.
* **Primary Site Technique:** All distinguished copies are kept at a single primary site that coordinates transaction management. This is a simple extension of centralized two-phase locking.
* *Advantages:* Simple implementation, data items locked at one site but accessible from any.
* *Disadvantages:* Primary site can become overloaded, failure of primary site makes the entire system inaccessible. Backup sites are often used for recovery.
* **Primary Copy Technique:** Instead of a site, a data item partition is designated as the primary copy. Locking a data item only requires locking its primary copy.
* *Advantages:* No single site is overloaded with locking requests as primary copies are distributed.
* *Disadvantages:* Identifying primary copies is complex, requiring a distributed directory.
* **Distributed Concurrency Control based on Voting:**No primary copy or coordinator.
* Each copy maintains its own lock.
* A transaction sends lock requests to all sites with the data item.
* The transaction gets the data item if a majority of sites grant the lock.
* Timeouts are used to abort transactions if voting information isn't received within a defined period.

**IX. Distributed Recovery**

Distributed recovery presents challenges in determining site status and managing distributed commits:

* **Major Problems:**Difficulty in determining if a site is down without extensive message exchange. Various factors can cause communication failures.
* Distributed commit: Ensuring transactions updating data at multiple sites are fully committed and their effects are not lost on any site.
* **Recovery from Coordinator Failure:Primary site approach with no backup site:** Aborts and restarts all active transactions at all sites. All updates are made at the primary site, with no fallback option.
* **Primary site approach with backup site:** Suspends active transactions, designates the backup site as the new primary, and identifies a new backup. Primary and backup sites synchronize data regularly.
* **Primary and backup sites fail or no backup site:** Requires an election process to select a new coordinator site. A site failing to communicate with the coordinator initiates the election by proposing itself as the new coordinator and requesting a majority of "yes" votes from other running sites.

**X. Concluding Remarks**

Distributed databases offer significant advantages in terms of scalability, availability, and performance but introduce considerable complexity in management, query processing, concurrency control, and recovery. The various techniques for data fragmentation, replication, and concurrency control (like the primary site/copy techniques and voting) are designed to address these complexities and ensure data consistency and system resilience in a distributed environment. Handling failures of individual sites, communication links, and coordinators are critical aspects of distributed database design and require robust recovery mechanisms.

DATA base

**1. Transaction Processing**

* **Single-User System**: One user at a time (e.g., PCs).
* **Multiuser System**: Multiple concurrent users (e.g., banking, airlines).
* **Concurrency**:
  + **Interleaved**: Single CPU switches processes, avoids idle time.
  + **Parallel**: Multiple CPUs process simultaneously.

**2. Transactions**

* **Definition**: Logical unit with read/write/insert/update/delete operations.
* **Examples**: ATM, reservations, billing.
* **Consistency**: Database remains valid before/after transactions.
* **Boundaries**: Marked by Begin/End statements.

**3. Database Operations**

* **Granularity**: Data size (field, record, block).
* **Operations**:
  + **read\_item(X)**: Fetch X from disk to memory.
  + **write\_item(X)**: Update X on disk via buffer.
* **Buffers**: Manage disk blocks; modified buffers written back.

**4. ACID Properties**

* **Atomicity**: All or no operations executed.
* **Consistency**: Valid state transitions.
* **Isolation**: No interference until commit.
* **Durability**: Committed changes persist.

**5. Transaction States**

* **Active**: Execution starts.
* **Partially Committed**: Operations done, not saved.
* **Committed**: Changes permanent.
* **Failed**: Aborted due to errors.
* **Terminated**: Exits system.

**6. Concurrency Control**

* **Purpose**: Ensures data integrity in concurrent execution.
* **Benefits**: Faster response, better resource use.
* **Issues**:
  + **Dirty Read**: Reading uncommitted data.
  + **Unrepeatable Read**: Data changes mid-transaction.
  + **Lost Update**: Overwritten updates.

**7. Schedules**

* **Serial Schedule**: Sequential execution, no issues.
* **Serializable Schedule**: Concurrent but equivalent to serial.
* **Conflict Serializability**: Non-conflicting operations swapped.

**8. Concurrency Control Techniques**

* **Locking**:
  + **Shared**: Multiple reads, no writes.
  + **Exclusive**: Single read/write.
  + **Two-Phase Locking (2PL)**: Growing (lock) and shrinking (unlock) phases.
* **Timestamp-Based**: Orders by start time, no deadlocks.
* **Multiversion**: Multiple data versions, no read rejection.
* **Optimistic**: Validates serializability at commit.
* **Granularity Locking**: Coarse (database) to fine (record) locks.

**9. Deadlock & Starvation**

* **Deadlock**: Transactions wait for each other’s locks.
  + **Solutions**: Prevention (e.g., conservative 2PL), detection (wait-for graph), timeouts.
* **Starvation**: Transaction repeatedly delayed.
  + **Solutions**: FIFO, prioritize long-waiting/aborted transactions.