Distributed Databases: The Basic Concepts

ACS 575: Database Systems

Instructor: Dr. Jin Soung Yoo, Professor
Department of Computer Science
Purdue University Fort Wayne

References

- Elmasri et al., Fundamentals of Database Systems, Ch 23
- W. Lemanhieu, et al., Principles of Database Management: The Practical Guide to Storing, Managing and Analyzing Big and Small Data, Ch 16

Outline

- Distributed Systems and Distributed Databases
- Distributed Database Concepts
- Data Fragmentation, Replication and Allocation Techniques
- Concurrency Control and Recovery in Distributed Databases
- Query Processing and Optimization in Distributed Databases
- Distributed Database Architectures

Distributed Computing Systems

- A **distributed computing system**
  - is composed of multiple nodes or processing sites that are interconnected by a computer network, each with a degree of autonomy, collaborating to execute complex tasks.
  - breaks down complicate problems into smaller, manageable segments, facilitating easier handling of complexity.
  - spreads data and the associated retrieval processes across various sources and locations.
- The **objective** of distributed computing system is to offer a unified perspective of this distributed data while maintaining transparency in data access and manipulation.

Outline

- Distributed Systems and Distributed Databases
- ☞ **Distributed Database Concepts**
- Data Fragmentation, Replication and Allocation Techniques
- Concurrency Control and Recovery in Distributed Databases
- Query Processing and Optimization in Distributed Databases
- Distributed Database Architectures

Distributed Database Concepts

- Distributed Database and DDBMS
- Transparency
- Availability and Reliability
- Scalability and Partition Tolerance
- Autonomy
- Advantages of Distributed Databases

Distributed Databases and DBMS

- A **distributed database (DDB)** is a collection of multiple logically interrelated databases distributed over a computer network.

- A **distributed database management system** (DDBMS) is a software system that manages a distributed database

Characteristics of Distributed Databases

- **Networked Database Nodes:**
  - Sites interconnected through a network infrastructure.
- **Logical Database Interconnection:**
  - Cohesive logical relationships across the networked databases.
- **Heterogeneity Across Nodes:**
  - Potential for variations in data, as well as hardware and software platforms, without a requirement for uniformity.

Transparency

- **Transparency** ensures that applications and users interact with a unified logical database, shielded from distribution complexities—an extension of data independence.
  - End users/application developers requires little or no awareness of underlying details – distribution complexity as well as has logical and physical data independence.

Types of Transparency

- **Data Organization Transparency** (also known as **Distribution** or **Network Transparency**)
  - **Location Transparency**: Users are unaware of the actual location of data. The command used to perform a task is independent of the location of the data and location of the node where the command was issued.
  - **Naming Transparency**: implies that once a name is associated with an object, the named objects can be accessed unambiguously without additional specification as to where the data is located.
- **Replication Transparency**
  - Ensures consistent replication across nodes; updates to one replica automatically synchronize with others.
  - Makes the user unaware of the existence of these copies.

Types of Transparency (cont.)

- **Fragmentation Transparency**:
  - Makes the user unaware of the existence of data fragmentation
  - Global queries are executable without user awareness of the underlying distribution and necessary combination of data fragments.
- **Access Transparency**:
  - Allows uniform querying and access, despite variations in database systems and APIs.
- **Transaction Transparency**:
  - Distributed transactions are managed seamlessly, akin to transactions on a single database system.
  - Makes the user unaware where a transaction executes.

Availability and Reliability

- **Availability**
  - Is defined as the probability that the system is continuously available during a time interval
- **Reliability**
  - Is broadly defined as the probability that the system is running (not down) at a certain time point
- Both directly related to faults, errors, and failures
  - **Faults** are the root causes of errors, and **errors** are specific system states that lead to failures.
- ***Fault-tolerant*** - A common approach to construct a system that is reliable.
  - Recognize that faults will occur, and remove faults before they can result in a system failure.

Scalability

- **Scalability** refers to the system's ability to increase its capacity and maintain seamless operations without interruption.
- Types of scalability
  - **Horizontal scalability**
    - Expanding the number of nodes in a distributed system
  - **Vertical scalability**
    - Expanding capacity of the individual nodes, such as the storage capacity or processing power.

Autonomy

- **Autonomy** determines extent to which individual nodes can operate independently
- **Design autonomy**
  - refers to independence of data model usage and transaction management techniques among nodes
- **Communication autonomy**
  - is the degree to which each node can independently control the sharing of its information with other nodes
- **Execution autonomy**
  - refers to the ability of users to operate independently according to their preferences

Advantages of Distributed Databases

- **Improved ease and flexibility of application development**
  - Development is easier and more flexible, especially for geographically dispersed organizations, due to data distribution transparency.
- **Increased availability**
  - System resilience is enhanced by confining faults to their origin site, allowing other nodes to continue functioning.
  - Data replication across sites further supports availability.

Advantages of Distributed Databases

- **Improved performance**
  - Placing data closer to where it's used reduces both contention for resources and network delay.
  - Smaller databases at each site and reduced transaction loads improve local query performance.
  - Parallel query execution across sites also boosts efficiency.
- **Easier expansion via scalability**
  - Adding data, increasing database sizes, or expanding the number of nodes is simpler in DDBs compared to centralized systems, facilitating growth and system expansion.

Outline

- Distributed Systems and Distributed Databases
- Distributed Database Concepts
- ☞ **Data Fragmentation, Replication and Allocation Techniques**
- Concurrency Control and Recovery in Distributed Databases
- Query Processing and Optimization in Distributed Databases
- Distributed Database Architectures

Data Fragmentation

- **Fragments**
  - Are logical units of the database, which may be assigned for storage at the various nodes.
  - Enhance localized data processing and system performance
  - Facilitate system scalability and fault isolation
  - Ensure data availability and system reliability
- Importance of database fragmentation
  - Critical for distributed database architecture
  - Impacts system performance and expansion capability
  - Careful planning required for effective use
- Two Types: **Horizontal** (by rows) and **Vertical** (by columns)

Horizontal Fragmentation (Sharding)

- **Horizontal fragment** or **shard** of a relation
  - Is a subset of tuples defined by conditions
    - E.g., horizontal fragment on EMPLOYEE with condition, DNO=5
  - Can be specified using SELECT operations (row selection)
  - Complete and disjoint: All tuples included, no overlaps
- **Complete horizontal fragmentation**
  - Apply UNION operation to the fragments to reconstruct the relation

Vertical Fragmentation

- **Vertical fragmentation**
  - Divides relation vertically by attributes
  - Each fragment holds different attribute subset
  - Includes primary key for reassembly
  - Specified by PROJECT operations
- **Complete vertical fragmentation**
  - Attribute lists cover all original attributes
  - Only primary key shared across fragments
  - Reconstructed with OUTER UNION or FULL OUTER JOIN
  - Necessary for maintaining data integrity

Mixed (Hybrid) Fragmentation

- **Mixed (hybrid) fragmentation**
  - Combination of horizontal and vertical fragmentations

- Fragmentation schema
  - Defines a set of fragments that includes all attributes and tuples in the database
- Allocation schema
  - Describes the allocation of fragments to nodes of the DDBS

Fragmentation Examples

CustomerID	FirstName	LastName	Country	Year of birth	Gender
10023	Bart	Baesens	Belgium	1975	M
10098	Charlotte	Bobson	USA	1968	F
10233	Donald	McDonald	UK	1960	M
10299	Heiner	Pilzner	Germany	1973	M
10351	Simonne	Toutdroit	France	1981	F
10359	Seppe	vanden Broucke	Belgium	1989	M
10544	Bridget	Charlton	UK	1992	F
11213	Angela	Kissinger	USA	1969	F
11349	Henry	Dumortier	France	1987	M
11821	Wilfried	Lemahieu	Belgium	1970	M
12111	Tim	Pope	UK	1956	M
12194	Naomi	Leary	USA	1999	F

Horizontal fragmentation

Vertical fragmentation

Illustration of Various Fragmentation

CustomerID	FirstName	LastName
10023	Bart	Baesens
10098	Charlotte	Bobson
10233	Donald	McDonald
10299	Heiner	Pilzner
10351	Simonne	Toutdroit
10359	Seppe	vanden Broucke
10544	Bridget	Charlton
11213	Angela	Kissinger
11349	Henry	Dumortier
11821	Wilfried	Lemahieu
12111	Tim	Pope
12194	Naomi	Leary

CustomerID	Country	Year of birth	Gender
10023	Belgium	1975	M
10098	USA	1968	F
10233	UK	1960	M
10299	Germany	1973	M
10351	France	1981	F
10359	Belgium	1989	M
10544	UK	1992	F
11213	USA	1969	F
11349	France	1987	M
11821	Belgium	1970	M
12111	UK	1956	M
12194	USA	1999	F

(a) Vertical fragmentation

CustomerID	FirstName	LastName	Country	Year of birth	Gender
10023	Bart	Baesens	Belgium	1975	M
10359	Seppe	vanden Broucke	Belgium	1989	M
11821	Wilfried	Lemahieu	Belgium	1970	M
10351	Simonne	Toutdroit	France	1981	F
11349	Henry	Dumortier	France	1987	M

CustomerID	FirstName	LastName	Country	Year of birth	Gender
10299	Heiner	Pilzner	Germany	1973	M

CustomerID	FirstName	LastName	Country	Year of birth	Gender
10544	Bridget	Charlton	UK	1992	F
10233	Donald	McDonald	UK	1960	M
12111	Tim	Pope	UK	1956	M

CustomerID	FirstName	LastName	Country	Year of birth	Gender
11213	Angela	Kissinger	USA	1969	F
10098	Charlotte	Bobson	USA	1968	F
12194	Naomi	Leary	USA	1999	F

(b) Horizontal fragmentation

CustomerID	FirstName	LastName
10023	Bart	Baesens
10359	Seppe	vanden Broucke
11821	Wilfried	Lemahieu
10351	Simonne	Toutdroit
11349	Henry	Dumortier

CustomerID	FirstName	LastName
10299	Heiner	Pilzner

CustomerID	FirstName	LastName
10544	Bridget	Charlton
10233	Donald	McDonald
12111	Tim	Pope

CustomerID	FirstName	LastName
11213	Angela	Kissinger
10098	Charlotte	Bobson
12194	Naomi	Leary

CustomerID	Country	Year of birth	Gender
10023	Belgium	1975	M
10098	USA	1968	F
10233	UK	1960	M
10299	Germany	1973	M
10351	France	1981	F
10359	Belgium	1989	M
10544	UK	1992	F
11213	USA	1969	F
11349	France	1987	M
11821	Belgium	1970	M
12111	UK	1956	M
12194	USA	1999	F

(c) Mixed fragmentation

Data Replication and Allocation

- **Fully replicated distributed database**
  - The entire database is replicated across every site within the distributed system.
  - Significantly enhances data availability.
  - Updates may be slower due to the need for synchronization across sites.
- **Nonredundant allocation (no replication)**
  - Each data fragment exists at only one site, preventing redundancy.

Data Replication and Allocation (cont.)

- **Partial replication**
  - Selected fragments are replicated, while others remain unreplicated, according to a predefined replication schema.
- **Data allocation** (**data distribution**)
  - Each fragment is strategically placed at a specific site within the distributed system based on system performance and availability objectives.

Outline

- Distributed Systems and Distributed Databases
- Distributed Database Concepts
- Data Fragmentation, Replication and Allocation Techniques
- ☞ **Concurrency Control and Recovery in Distributed Databases**
- Query Processing and Optimization in Distributed Databases
- Distributed Database Architectures

Unique Challenges in Distributed Database Systems

- Handling multiple data replicas across different locations.
- Managing system reliability amidst site failures.
- Ensuring communication integrity despite potential network issues.
- Executing distributed transactions with guaranteed atomicity.
- Resolving distributed deadlocks efficiently.

Distributed Concurrency Control via Distinguished Data Copies

- Selects a unique copy of each data element, known as the **distinguished copy**, as the reference point for synchronization.
- Locking mechanisms are centralized around the distinguished copy to manage data access and ensure consistency.
- Utilizes the **primary site approach**, where a single site holds all distinguished copies for streamlined control.
- Employs a **dual-site strategy** for robustness, where both the primary and a designated backup site maintain essential locking details.
- The primary copy methodology efficiently spreads the responsibility of managing locks across multiple sites, optimizing resource use and system responsiveness.

Distributed Concurrency Through Voting Mechanism

- Implements a voting protocol for concurrency control without a designated primary copy.
- Lock requests are disseminated across all sites housing a replica of the data.
- Each data replica independently controls its own locking mechanism.
- Transactions must win the majority of votes from replicas to secure a lock across the system.
- A time-out mechanism is in place to prevent indefinite lock waiting.
- The method inherently increases communication overhead due to frequent voting messages.

Distributed Recovery

- Determining the status of distributed sites is complex and typically requires extensive inter-site communication.
- The distributed commit process requires a transaction to secure confirmation from all involved sites before it can be finalized to prevent data loss.
- The **two-phase commit protocol** is a commonly employed mechanism to guarantee the consistency and integrity of a distributed transaction across multiple sites.

Transaction Management in Distributed Database Systems

- The **global transaction manager** oversees distributed transactions and is generally operated by the originating site of the transaction.
- This manager collaborates with transaction managers at various sites to orchestrate the execution of the distributed transaction.
- Database operations and pertinent details are forwarded by the manager to the concurrency controller.
- The concurrency controller is tasked with obtaining and relinquishing locks necessary to ensure data integrity during transaction processing.

Commit Protocols in Distributed Databases

- **Two-phase Commit Protocol (2PC)**
  - A coordinator node preserves necessary recovery data in collaboration with local recovery managers at each site.
  - The protocol ensures a transaction commits only if all involved sites agree.
- **Three-phase Commit Protocol (3PC)**
  - Enhances the two-phase commit by splitting the commit phase into two separate stages:
  - Introduces a '**Prepare-to-commit**' phase that notifies nodes of the impending commit based on the voting outcome.
  - The final '**Commit**' phase follows the traditional two-phase commit procedure to finalize the transaction.

Outline

- Distributed Systems and Distributed Databases
- Distributed Database Concepts
- Data Fragmentation, Replication and Allocation Techniques
- Concurrency Control and Recovery in Distributed Databases
- ☞ **Query Processing and Optimization in Distributed Databases**
- Distributed Database Architectures

Stages of a Distributed Database Query

- **Query mapping**
  - Translates queries to align with the global conceptual schema.
- **Localization**
  - Breaks down the distributed query into subqueries for specific data fragments.
- **Global query optimization**
  - Chooses the most efficient execution strategy from several possibilities.
- **Local query optimization**
  - Conducted at all nodes, it enhances the performance of query execution locally.

Data Transfer Costs in Distributed Query Processing

- Data Movement Overhead:
  - Encompasses the expense of moving interim and outcome datasets across the network.
- Optimization Goal:
  - Focus on minimizing the volume of data being transferred to enhance query efficiency.

Semijoin Strategy in Distributed Query Processing

- **Semijoin Mechanism**:
  - Optimizes data transfer by reducing relation tuples prior to transmission between sites.
- Process Execution:
  - Attributes for the join from one dataset R are sent to the site with a related dataset S. There, the semijoin is executed and the relevant attributes, along with the results, are returned to the originating site of dataset R.
- Optimization Advantage:
  - An effective approach for data transfer reduction during distributed query processing.

Decomposing Queries and Updates in Distributed Databases

- Query Centralization Illusion:
  - Users can operate as though the DBMS is centralized, assuming the system supports full distribution, fragmentation, and replication transparency.
- **Query Decomposition Mechanism**:
  - Disassembles a global query into component subqueries for distribution to various sites.
  - Generates a coherent strategy for reassembling the results from subqueries.
- Catalog Utilization:
  - Uses a catalog to define and store attribute lists and/or guard conditions for efficient query processing.

Outline

- Distributed Systems and Distributed Databases
- Distributed Database Concepts
- Data Fragmentation, Replication and Allocation Techniques
- Concurrency Control and Recovery in Distributed Databases
- Query Processing and Optimization in Distributed Databases
- ☞ **Distributed Database Architectures**

Classifying Distributed Database Systems

- Determining Factors for DDBMS Classification:
  - DDBMS software Homogeneity
    - Homogeneous: Uniform DDBMS software across nodes
    - Heterogeneous: Diverse DDBMS software across nodes
  - Autonomy Level
    - Centralized Control: No autonomy, centralized oversight
    - Full Independence: Each database system maintains complete autonomy, as seen in multidatabase setups.

Federated Database Management Systems

- **Federated Database Systems (FDBS)** operate with a unified global schema, facilitating application access to the collective data of the federation.

Challenges in Federated Database Management Systems

- Data Model Discrepancies:
  - Diverse data model structures across participating databases.
- Constraint Variation:
  - Disparate constraints that may affect data integrity and operations.
- Query Language Divergence:
  - Variations in query syntax and capabilities across different systems.
- Semantic Heterogeneity:
  - Varied interpretations and uses of similar or identical data due to differences in context or schema design.

Autonomy in Database Design

- Universe of Discourse Definition:
  - Scope and domain of the data encompassed by the system.
- Data Representation and Naming Conventions:
  - Establishing standard formats and naming rules for data elements.
- Semantic Understanding:
  - Clarifying data meaning, and ensuring consistent interpretation.
- Transactional and Policy Constraints:
  - Specifying rules and restrictions governing data transactions.
- Summary Data Derivation:
  - Outlining methods for aggregating and summarizing data details.

Autonomy in Database Design (cont.)

- Communication Autonomy:
  - Freedom to choose if and how to engage in data exchanges with other databases.
- Execution Autonomy:
  - Independence in performing local operations without external influences.
  - Control over the sequence in which operations are executed.
- Association Autonomy:
  - Discretion in determining the extent of sharing system capabilities and resources.

Distributed Database Architectures

- Multiprocessor System Classifications:
  - **Shared Memory** Architectures:
    - Systems where processors have direct access to common memory spaces.
  - **Shared Disk** Architectures:
    - Configurations in which multiple processors can access the same disk storage, but each operates its own local memory.
  - **Shared-Nothing** Architectures:
    - Each processor has exclusive access to its own memory and disk, coordinating with others as needed.

Core Components of Pure Distributed Database Systems

- **Global Query Compiler:**
  - Interprets queries against a global schema, ensuring adherence to constraints listed in the global system catalog.
- **Global Query Optimizer:**
  - Converts global queries into efficient local query plans for execution at various sites.
- **Global Transaction Manager:**
  - Manages and synchronizes multi-site transaction execution in concert with local transaction managers.

Strategies for Distributed Catalog Management

- **Centralized Catalogs:**
  - Single-site storage of the full catalog data.
  - Simplifies implementation but introduces a single point of failure.
- **Fully Replicated Catalogs:**
  - Exact catalog copies stored at every site.
  - Enhances read operations, but write operations require synchronization.
- **Partially Replicated Catalogs:**
  - Each site houses a full catalog for locally held data.
  - Balances data access speed with update complexity.