Distributed Databases and Client/Server Architecture

Objectives

- Understand the advantages and disadvantages of distributed databases
- Know the design issues involved in distributed databases
- Understand the purpose and some of the techniques used for data fragmentation, replication and allocation
- Appreciate the importance of query optimization as applied to distributed databases
- Understand distributed concurrency control and recovery techniques
- Identify and explain the interaction between different components of client/server architecture as applied to database systems.

Distributed Database

- A **Distributed Database** is a set of logically interrelated databases that are stored on computers at several geographically different sites and are linked by means of a computer network.
- These interrelated databases work together to perform certain specific tasks.
- Distributed computer systems work by splitting a large task into a number of smaller ones that can then be solved in a coordinated fashion.
- Each processing element can be managed independently and can develop its own applications.

Data Fragmentation

- **Data Fragmentation** use to split our database into logical units.

Example

- Consider the University database. We could make three fragments from the DEPARTMENT relation by specifying "DeptNo = 111" for the first, "DeptNo = 666" for the second and "DeptNo =999" for the third.
- We could partition the relation UNIT along similar lines.

DEPARTMENT

DeptName	DeptNo	OfficePhoneExt
Computer Science	111	4211
Accountancy	666	2114
Economics	999	3125

Horizontal Fragment

- A Horizontal Fragment is defined by:
  - $\sigma_{Ci}(R)$
- where C_i stands for one of the conditions.
- If we have a set of conditions C_1, C_2, …C_i,…C_n such that all the tuples in the relation R will be included in one condition or another (every tuple in R satisfies {C_1 or C_2 or ….or C_n}) is called **Complete**.
- If no tuple in R satisfies {C_i and C_j} for any i not equal to j then the fragmentation is called **Disjoint**.
- To reconstruct the relation from a complete horizontal fragmentation, we apply the UNION operation to the fragments.

Vertical Fragmentation

- A vertical fragment is defined by:
  - $\pi_{Li}(R)$
- where L_i stands for a list of one or more attributes of the relation.
- If we have a vertical fragmentation with lists L_1, L_2, …L_i,…L_n such that all the attributes of R are included yet the lists only have the primary key attribute of R in common, then this is called **Complete**.
- So for a complete vertical fragmentation:
  - $L_1 \cup L_2 \cup \ldots\ldots L_n = Attributes(R)$
    $L_i \cap L_j = PK\ (R)$, $i \neq j$
- where *PK (R)* is the primary key of *R*.

Vertical Fragmentation (cont')

- To reconstruct the relation from a complete vertical fragmentation use the **OUTER UNION** operation applied to the vertical fragments (That is, provided there is no horizontal fragmentation).
- If we have a complete vertical fragmentation with some horizontal fragmentation as well, we need to use the **FULL OUTER JOIN**.

Mixed Fragmentation

- We can combine horizontal and vertical fragmentation in what is called **Mixed Fragmentation**.

Determination of Fragment

- A ***Fragment*** of a relation *R* is defined by
  - $\pi_L(\sigma_C(R))$
- where *L* is a list of attributes of *R* and *C* is a condition on the tuples of *R*.
- If *C* = TRUE and *L* is not equal to Attributes(*R*), we have a ***Vertical Fragment***.
- If *C* is not TRUE, so it will select a proper subset of the tuples, and *L* equals Attributes(*R*) we have a ***Horizontal Fragment***.
- If *C* is not TRUE, so it will select a proper subset of the tuples, and *L* is not equal to Attributes(*R*), we have a ***Mixed Fragment***.

Fragmentation Schema

- A **Fragmentation Schema** is a definition of a set of fragments which will include all attributes and all tuples of the database and for which the whole database can be reconstructed by applying a sequence of **OUTER UNION** (or OUTER JOIN) and **UNION** operations.

Data Replication and Allocation

- Allocation strategies are used to decide where to place each fragment so that the maximum benefits can be obtained.

Non Redundant Best Fit Allocation

- The non-redundant best-fit method determines the single most likely site to allocate a fragment based on maximum benefit.

Example

- Suppose that we have three fragments of relations F1, F2 and F3 and five sites S1, S2, S3, S4 and S5.
- The following table specifies the user transactions in terms of the sites and the transaction access to the relations as well as their frequency.

Transaction	Site(s)	Frequency	Relation Access
T1	S1, S4, S5	1	4 to F1 and 2 to F2
T2	S2, S4	2	2 to F1 and 4 to F3
T3	S3, S5	3	4 to F2 and 2 to F3

Example (cont')

- We want to allocate the fragments F1, F2 and F3 to the sites so that we can get maximum utilization.
- The transaction data for T1 suggests that we would want to allocate F1 to S1 or S4 or S5, that for T2 narrows this down to S4.
- The transaction data for T3 suggests that we would want to allocate F2 to S3 or S5, that for T1 narrows this down to S5.
- The transaction data for T2 suggests that we would want to allocate F3 to S2 or S4, while that for T3 suggests S3 or S5.
- In terms of frequency times access, S2 or S4 will win out but there is no reason to choose one rather than the other.

Data Replication

- Data replication refers to the storage of data copies at multiple sites served by a computer network.
- Fragmented copies can be stored at several sites to serve specific information requirements.
- Existence of fragment copies can enhance data availability and response time, also reduce communication and total query costs.

Data Replication

- Suppose a database A is divided into two fragments A1 and A2. It is possible to store fragment A1 at sites S1 and S2, while fragment A2 is stored at sites S2 and S3.
- Three possible scenarios exist: a database can be fully replicated, partially replicated or unreplicated.
  - **A Fully Replicated Database** stores multiple copies of each database fragment at multiple sites.
  - **A Partially Replicated Database** stores multiple copies of some database fragments at multiple sites.
  - **An Unreplicated Database** stores each database fragment at a single site.

Query Optimization

- For a distributed database system
  - The database may be portioned into several fragments and so a decision will be needed as to which fragment to access.
  - The same data may be stored at different sites, which raises the question of which copy of the data should be queried.
- A distributed database system uses query optimization techniques to deal with these problems and thus to guarantee acceptable database performance.
- A query optimization routine is designed to minimize the total cost associated with the execution of a request.

Costs Calculation

- The total costs are a function of the:
  - **Access Time (I/0) Cost** involved in accessing the physical data stored on disk.
  - **Communication Cost** associated with the transmission of data among nodes in distributed database systems.
  - **CPU Time Cost** associated with the processing overhead of managing distributed transactions.

Query Optimization Evaluation

- Despite the fact that we separate costs into either **Communication** or **Processing** costs, in practice we can have difficulty doing this.
- Partly this is because not all query optimization algorithms use the same parameters, and even when they do, the algorithms do not all assign the same weight to each parameter.
- Some algorithms minimize total time, others minimize the communication time, and still others do not factor in the CPU time at all, considering it insignificant relative to other cost sources.

Query Optimization Evaluation

- When we try to evaluate a query optimization process, we need to remember that the transaction processor receives data, synchronizes it, assembles the answer, and presents it to the end user or an application.
- The big difference in the distributed case is that a particular query *may be executed at any one of several different sites*.
- The response time associated with remote sites may not be easily predetermined, because *some nodes are able to finish their part of the query in less time than others.*

Query Optimization Evaluation

- Most of the algorithms proposed for query optimization are based on two principles:
  - The selection of the optimum execution order.
  - The selection of sites to be accessed is made to minimize communication costs.
- Within these two principles, a query optimization algorithm can be evaluated on the basis of its operation mode or the timing of its optimization.

Query Optimization Evaluation

- Operation modes can be classified as **Manual** or **Automatic**.
  - **Automatic Query Optimization** means that the DBMS finds the most cost-effective access path without user intervention.
  - **Manual Query Optimization** requires that the optimization be selected and scheduled by the end user or programmer.
- It is clear that automatic query optimization will be much more desirable from the end user's point of view, but it does impose a higher cost.

Query Optimization Classification

- Query optimization algorithms can also be classified according to when the optimization is done and are then designated either as **Static** or **Dynamic**.

Static Query Optimization

- Takes place at compilation time.
- Chosen when the query is compiled by the DBMS.
- Often used when SQL statements are embedded in procedural programming languages.
  - When the program is submitted to the DBMS for compilation, the system creates the plan necessary to access the database.
  - When the program is executed, the DBMS uses that plan to access the database.

Dynamic Query Optimization

- Takes place at execution time.
- The database access strategy is defined when the program is executed.
- Access strategy is dynamically determined by the DBMS at run time, using the most up-to-date information about the database.
- Although dynamic query optimization is efficient, its cost is measured in terms of an increased run-time processing overhead.
- The best strategy is determined every time the query is executed and this could happen several times in the same program.

Optimization Algorithm

- The actual strategies used for query optimization are also categorized by the type of information that is used to design the algorithm that optimizes the query.
- This yields the two most common kinds of algorithm
  - Statistically Based Algorithms
  - Rule-based Algorithms

Statistically based Query Optimization Algorithm

- A statistically based query optimization algorithm uses statistical information about the database.
- The statistics provide information about various database characteristics such as size, number of records, average access time, number of requests serviced, number of users with access rights, and so on.
- These statistics are then used by the DBMS to determine the best access strategy.

Rule-based Query Optimization Algorithm

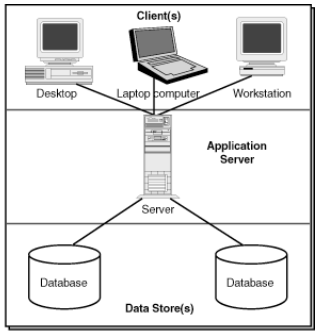
- A rule-based query optimization algorithm is based on a set of user-defined rules to determine the best query access strategy.
- Normally these rules are entered by the end user or database administrator, and often they are very general in nature.

Overview of Concurrency Control and Recovery

- Concurrency control is very important in the distributed database environment because once we are using many sites and operations are involving many processes the possibility of data inconsistency and deadlocked transactions becomes much greater.
- Most of them work by extending techniques used for centralized databases in some way.
  - E.g. If we restrict ourselves to the concurrency control technique of centralized locking we can extend that to the distributed case by designating one copy of the database as a distinguished copy and then applying our locking protocols to that copy.

Client/Server Architecture

- Databases built on the client-server architecture are quite common, so it is useful to review that architecture and how it applies to the field of databases.



Review of Client-Server Architecture

- The client/server architecture is based on the hardware and software components that interact to form a system.
- The system includes three main components: **Clients**, **Servers**, and **Communications Middleware**.

Client

- The client is any computer process that requests services from the server.
- The client is also known as the **Front-end Application**.

Server

- The server is any computer process providing services to the clients.
- The server is also known as the **Back-end Application**.

Communication Middleware

- The communication middleware is any computer process through which clients and servers communicate and is also known as **Communications Layer**.

Client Components

- The client application or front end, runs on top of the operating system and connects with the middleware to access services available in the network.
- Several **Third-Generation Language** (3GL) and **Fourth-Generation Language** (4GL) can be used to create the front-end applications.
- Most front-end applications are GUI-based to hide the complexity of the Client/Server components from the end users.

Server Components

- The server application, or back end, runs on top of operating system and interacts with the middleware to “listen” for client’s requests for services.
- Unlike front-end client process, the server process need not be GUI-based.

Database Middleware Components

- The middleware software is divided into three main components:
  - Applications Programming Interface (API)
  - Database Translator

Application Programming Interface (API)

- API is public to the client application.
- The programmer interacts with middleware through the API provided by the middleware software.
- The middleware API allows the client process to be database-server-independent.
- Means that the server can be changed without requiring that the client applications be completely rewritten.

Database Translator

- Translates the SQL requests into the specific database server syntax.
- The database translator takes the generic SQL request and maps it to the database server’s SQL protocol.

Network Translator

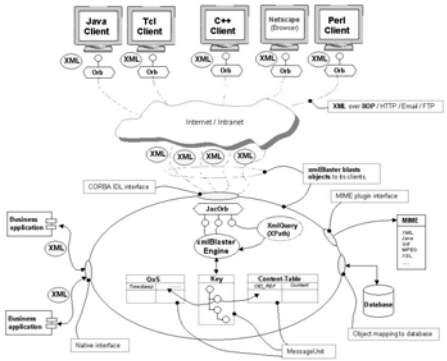
- Manages the network communications protocols.
- Database servers can use any of the network protocols, such as TCP/IP, IPX/SPX or Net BIOS.
- The network layer handles all the communications details of each database transparently to the client application.

Middleware Classifications

- Database middleware software can be classified according to the way clients and servers communicate across the network.
- The middleware software is usually classified as:
  - Message-Oriented Middleware (MOM)
  - Remote-Procedure-Call-based (RPC-based) Middleware
  - Object-based Middleware

Message-Oriented Middleware (MOM)

- Message-oriented middleware is generally more efficient in local area networks with limited bandwidth and in applications in which data integrity is not quite so critical.



Remote-Procedure-Call-based (RPC-based) Middleware

- RPC-based middleware is probably most suited to highly integrated systems in which data integrity is critical, as well as high-throughput networks.

Object-based Middleware

- Object-based middleware is an emerging technology based on object-oriented concepts.

Client/Server Databases

- A **Database Management System** (DBMS) lies at the center of most client/server systems in use today.
- A client/server DBMS reduces the network traffic as only the rows that match the query are returned.
- Client/server DBMS differ from other DBMS in terms of where the processing takes place and what data are sent over the network to the client computer.
- When the data are stored in multiple sites, client/server databases are closely related to distributed databases.

Client/Server Databases

- To function properly, the client/server DBMS must be able to:
  - Provide transparent data access to multiple and heterogeneous clients, regardless of hardware, software and network platform used by the client application.
  - Allow client requests to the database server over the network.
  - Processes client data request at the local server.
  - Send only the SQL results to the clients over the network.

Client/Server vs. Traditional Data Processing

- The client/server computing has introduced some major changes from traditional data processing:
  - Proprietary to open systems.
  - Maintenance-oriented to analysis, design and service.
  - Data collection to data deployment.
  - Centralized to a more distributed style of data management.
  - Vertical, inflexible organizational style to a more horizontal, flexible organizational style.