Lecture 7: Link | Syllabus
**Class Week 7/15*

Database Systems by Coronel & Morris

Chapter 12: Distributed Database Management Systems

* 12.1 *The Evolution of Distributed Database Management Systems*

A distributed database management system (DDBMS) governs the storage and processing of logically related data over interconnected computer systems in which both data and processing are distributed among several sites.
The dynamic business environment and the centralized database's shortcomings spawned a demand for applications based on accessing data from different sources at multiple locations. Such a multiple-source/multiple-location database environment is best managed by a DDBMS.

* 12.2 *DDBMS Advantages and Disadvantages*

Distributed databases are being used successfully in many web staples such as Google and Amazon, but they still have a long way to go before they yield the full flexibility and power they theoretically possess.

DISTRIBUTED DBMS ADVANTAGES AND DISADVANTAGES

ADVANTAGES	DISADVANTAGES
Data is located near the site of greatest demand. The data in a distributed database system is dispersed to match business requirements.	***Complexity of management and control.*** Applications must recognize data location, and they must be able to stitch together data from various sites. Database administrators must have the ability to coordinate database activities to prevent database degradation due to data anomalies.
Faster data access. End users often work with only the nearest stored subset of the data.	***Technological difficulty.*** Data integrity, transaction management, concurrency control, security, backup, recovery, and query optimization must all be addressed and resolved.
Faster data processing. A distributed database system spreads out the system's workload by processing data at several sites.	***Security.*** The probability of security lapses increases when data is located at multiple sites. The responsibility of data management will be shared by different people at several sites.
Growth facilitation. New sites can be added to the network without affecting the operations of other sites.	***Lack of standards.*** There are no standard communication protocols at the database level. For example, different database vendors employ different and often incompatible techniques to manage the distribution of data and processing in a DDBMS environment.
Improved communications. Because local sites are smaller and located closer to customers, local sites foster better communication among departments and between customers and company staff.	***Increased storage and infrastructure requirements.*** Multiple copies of data are required at different sites, thus requiring additional storage space.
Reduced operating costs. It is more cost-effective to add nodes to a network than to update a mainframe system. Development work is done more cheaply and quickly on low-cost PCs than on mainframes.	***Increased training cost.*** Training costs are generally higher in a distributed model than they would be in a centralized model, sometimes even to the extent of offsetting operational and hardware savings.
User-friendly interface. PCs and workstations are usually equipped with an easy-to-use graphical user interface (GUI). The GUI simplifies training and use for end users.	***Costs.*** Distributed databases require duplicated infrastructure to operate, such as physical location, environment, personnel, software, and licensing.
Less danger of a single-point failure. When one of the computers fails, the workload is picked up by other workstations. Data is also distributed at multiple sites.	
Processor independence. The end user can access any available copy of the data, and an end user's request is processed by any processor at the data location.	

* 12.3 *Distributed Processes and Distributed Databases*

In distributed processing, a database's logical processing is shared among two or more physically independent sites that are connected through a network. For example, the data input/output (I/O), data selection, and data validation might be performed on one computer, and a report based on that data might be created on another computer. A distributed database stores a logically related database over two or more physically independent sites. The sites are connected via a computer network. In contrast, the distributed processing system uses only a single-site database but shares the processing chores among several sites. In a distributed database system, a database is composed of several parts known as database fragments. The database fragments are located at different sites and can be replicated among various sites. Each database fragment is, in turn, managed by its local database process.

* 12.4 *Characteristics of Distributed Database Management Systems*

A DBMS must have at least the following functions to be classified as distributed :

- *Application interface* to interact with the end user, application programs, and other DBMSs within the distributed database
- *Validation* to analyze data requests for syntax correctness
- *Transformation* to decompose complex requests into atomic data request components
- *Query optimization* to find the best access strategy (which database fragments must be accessed by the query, and how must data updates, if any, be synchronized?)
- *Mapping* to determine the data location of local and remote fragments
- *I/O* interface to read or write data from or to permanent local storage
- *Formatting* to prepare the data for presentation to the end user or to an application program
- *Security* to provide data privacy at both local and remote databases
- *Backup and recovery* to ensure the availability and recoverability of the database in case of a failure
- *DB administration* features for the database administrator
- *Concurrency control* to manage simultaneous data access and to ensure data consistency across database fragments in the DDBMS
- *Transaction management* to ensure that the data moves from one consistent state to another; this activity includes the synchronization of

local and remote transactions as well as transactions across multiple distributed segments

A fully distributed database management system must perform all of the functions of a centralized DBMS, as follows :

1. Receive the request of an application or end user
2. Validate, analyze, and decompose the request. The request might include mathematical and logical operations such as the following :
   a. Select all customers with a balance greater than $1,000. The request might require data from only a single table, or it might require access to several tables.
3. Map the request's logical-to-physical data components
4. Decompose the request into several disk I/O operations
5. Search for, locate, read, and validate the data
6. Ensure database consistency, security, and integrity
7. Validate the data for the conditions, if any, specified by the request
8. Present the selected data in the required format

* 12.5 *DDBMS Components*

The DDBMS must include at least the following components :

- *Computer workstations* or remote devices (sites or nodes) that form the network system. The distributed database system must be independent of the computer system hardware.
- *Network hardware* and software components that reside in each workstation or device. The network components allow all sites to interact and exchange data. Because the components—computers, operating systems, network hardware, and so on—are likely to be supplied by different vendors, it is best to ensure that distributed database functions can be run on multiple platforms
- *Communications media* that carry the data from one node to another. The DDBMS must be communications media-independent; that is, it must be able to support several types of communications media
- The transaction processor (TP) is the software component found in each computer or device that requests data. The transaction processor receives and processes the application's remote and local data requests. The TP is also known as the application processor (AP) or the transaction manager (TM)
- The data processor (DP) is the software component residing on each computer or device that stores and retrieves data located at the site. The DP is also known as the data manager (DM). A data processor may even be a centralized DBMS

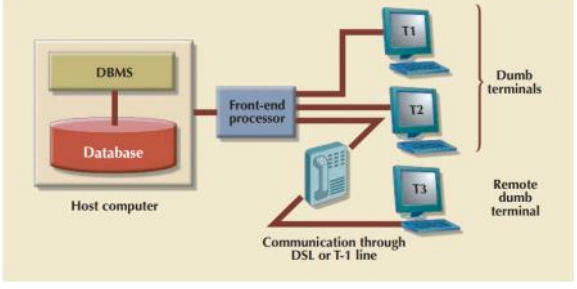
* 12.6 *Levels of Data and Process Distributed*

Current database systems can be classified on the basis of how process distribution and data distribution are supported. For example, a DBMS may store data in a single site (using a centralized DB) or in multiple sites (using a distributed DB), and it may support data processing at one or more sites.

* 12.6-a *Single-Site Processing, Single-Site Data*

In the single-site processing, single-site data (SPSD) scenario, all processing is done on a single host computer, and all data is stored on the host computer's local disk system. Processing cannot be done on the end user's side of the system. Such a scenario is typical of most mainframe and midrange UNIX/Linux server DBMSs. The DBMS is on the host computer, which is accessed by terminals connected to it. This scenario is also typical of the first generation of single-user microcomputer databases.

FIGURE 12.6 SINGLE-SITE PROCESSING, SINGLE-SITE DATA (CENTRALIZED)

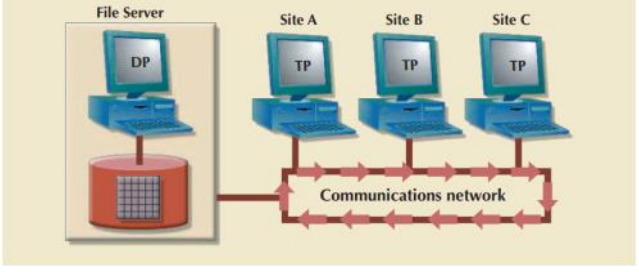


* Using Figure 12.6 as an example, you can see that the functions of the TP and DP are embedded within the DBMS on the host computer. The DBMS usually runs under a time-sharing, multitasking operating system, which allows several processes to run concurrently on a host computer accessing a single DP. All data storage and data processing are handled by a single host computer.

* 12.6-b *Multiple-Site Processing, Single-Site Data*

Under the multiple-site processing, single-site data (MPSD) scenario, multiple processes run on different computers that share a single data repository. Typically, the MPSD scenario requires a network file server running conventional applications that are accessed through a network. Many multiuser accounting applications running under a personal computer network fit such a description (see Figure 12.7)

FIGURE 12.7 MULTIPLE-SITE PROCESSING, SINGLE-SITE DATA



As you examine Figure 12.7, note that :

- The TP on each workstation acts only as a redirector to route all network data requests to the file server.
- The end user sees the file server as just another hard disk. Because only the data storage input/output (I/O) is handled by the file server's computer, the MPSD offers limited capabilities for distributed processing
- The end user must make a direct reference to the file server to access remote data. All record- and file-locking activities are performed at the end-user location.
- All data selection, search, and update functions take place at the workstation, thus requiring that entire files travel through the network for processing at the workstation. Such a requirement increases

end-user location.

- All data selection, search, and update functions take place at the workstation, thus requiring that entire files travel through the network for processing at the workstation. Such a requirement increases network traffic, slows response time, and increases communication costs.

*** 12.6-c** *Multiple-Site Processing, Multiple-Site Data*

The multiple-site processing, multiple-site data (MPMD) scenario describes a fully distributed DBMS with support for multiple data processors and transaction processors at multiple sites. Depending on the level of support for various types of databases, DDBMSs are classified as either homogeneous or heterogeneous.

Homogeneous DDBMSs integrate multiple instances of the same DBMS over a network—for example, multiple instances of Oracle 11g running on different platforms. In contrast, heterogeneous DDBMSs integrate different types of DBMSs over a network, but all support the same data model. For example, Table 12.3 lists several relational database systems that could be integrated within a DDBMS. A fully heterogeneous DDBMS will support different DBMSs, each one supporting a different data model, running under different computer systems.

*** 12.7** *Distributed Database Transparency Features*

A distributed database system should provide some desirable transparency features that make all the system's complexities hidden to the end user. In other words, the end user should have the sense of working with a centralized DBMS.

For this reason, the minimum desirable DDBMS transparency features are :

- Distribution transparency allows a distributed database to be treated as a single logical database.
  If a DDBMS exhibits distribution transparency, the user does not need to know :
  - The data is partitioned—meaning the table's rows and columns are split vertically or horizontally and stored among multiple sites.
  - The data is geographically dispersed among multiple sites.
  - The data is replicated among multiple sites.
- Transaction transparency allows a transaction to update data at more than one network site. Transaction transparency ensures that the transaction will be either entirely completed or aborted, thus maintaining database integrity.
- Failure transparency ensures that the system will continue to operate in the event of a node or network failure. Functions that were lost because of the failure will be picked up by another network node. This is a very important feature, particularly in organizations that depend on web presence as the backbone for maintaining trust in their business.
- Performance transparency allows the system to perform as if it were a centralized DBMS. The system will not suffer any performance degradation due to its use on a network or because of the network's platform differences. Performance transparency also ensures that the system will find the most cost-effective path to access remote data. The system should be able to "scale out" in a transparent manner or increase performance capacity by adding more transaction or data-processing nodes, without affecting the overall performance of the system.
- Heterogeneity transparency allows the integration of several different local DBMSs (relational, network, and hierarchical) under a common, or global, schema. The DDBMS is responsible for translating the data requests from the global schema to the local DBMS schema.

*** 12.8** *Distributed Transparency*

Distribution transparency allows a physically dispersed database to be managed as though it were a centralized database. The level of transparency supported by the DDBMS varies from system to system.

Three levels of distribution transparency are recognized :

- Fragmentation transparency is the highest level of distribution transparency. The end user or programmer does not need to know that a database is partitioned. Therefore, neither fragment names nor fragment locations are specified prior to data access.
- Location transparency exists when the end user or programmer must specify the database fragment names but does not need to specify where those fragments are located.
- Local mapping transparency exists when the end user or programmer must specify both the fragment names and their locations.

12.9 *Transaction Transparency*

Transaction transparency is a DDBMS property that ensures database transactions will maintain the distributed database's integrity and consistency. Remember that a DDBMS database transaction can update data stored in many different computers connected in a network. Transaction transparency ensures that the transaction will be completed only when all database sites involved in the transaction complete their part of the transaction.

Distributed database systems require complex mechanisms to manage transactions and ensure the database's consistency and integrity. To understand how the transactions are managed, you should know the basic concepts governing remote requests, remote transactions, distributed transactions, and distributed requests

12.9-a *Distributed Requests and Distributed Transactions*

Whether or not a transaction is distributed, it is formed by one or more database requests. The basic difference between a no-distributed transaction and a distributed transaction is that the distributed transaction can update or request data from several different remote sites on a network.

* A remote request lets a single SQL statement access the data that are to be processed by a single remote database processor. In other words, the SQL statement (or request) can reference data at only one remote site. Similarly, a remote transaction, composed of several requests, accesses data at a single remote site.

* A distributed transaction can reference several different local or remote DP sites. Although each single request can reference only one local or remote DP site, the transaction as a whole can reference multiple DP sites because each request can reference a different site. A distributed request lets a single SQL statement reference data located at several different local or remote DP sites. Because each request (SQL statement) can access data from more than one local or remote DP site, a transaction can access several sites.

The ability to execute a distributed request provides fully distributed database processing because you can :

- Partition a database table into several fragments.
- Reference one or more of those fragments with only one request. In other words, there is fragmentation transparency.

12.9-b *Distributed Concurrency Control*

Concurrency control becomes especially important in distributed databases because multisite, multiple-process operations are more likely to create data inconsistencies and deadlocked transactions than single-site systems. For example, the TP component of a DDBMS must ensure that all parts of the transaction are completed at all sites before a final COMMIT is issued to record the transaction

12.9-c *Two-Phase Commit Protocol*

Centralized databases require only one DP. All database operations take place at only one site, and the consequences of database operations are immediately known to the DBMS. In contrast, distributed databases make it possible for a transaction to access data at several sites. A final COMMIT must not be issued until all sites have committed their parts of the transaction. The two-phase commit protocol (2PC) guarantees that if a portion of a transaction operation cannot be committed, all changes made at the other sites participating in the transaction will be undone to maintain a consistent database state.

The two-phase commit protocol defines the operations between two types of nodes :

The coordinator and one or more subordinates, or *cohorts*. The participating nodes agree on a coordinator. Generally, the coordinator role is assigned to the node that initiates the transaction. However, different systems implement various, more sophisticated election methods. The protocol is implemented in two phases, as illustrated in the following sections.

12.10 *Performance and Failure Transparency*

One of the most important functions of a database is its ability to make data available. Web-based distributed data systems demand high availability, which means not only that data is accessible but that requests are processed in a timely. For example, the average Google search has a sub second response time. When was the last time you entered a Google query and waited more than a couple of seconds for the results?

Performance transparency allows a DDBMS to perform as if it were a centralized database. In other words, no performance degradation should be incurred due to data distribution. Failure transparency ensures that the system will continue to operate in the case of a node or network failure. Although these are two separate issues, they are interrelated in that a failing node or congested network path could cause performance problems. Therefore, both issues are addressed in this section

12.11 *Distributed Database Design*

Whether the database is centralized or distributed, the design principles and concepts described in Chapters 3, 4, and 6 are still applicable.

However, the design of a distributed database introduces three new issues :

- How to partition the database into fragments
- Which fragments to replicate
- Where to locate those fragments and replicas

Data fragmentation and data replication deal with the first two issues, and data allocation deals with the third issue. Ideally, data in a distributed database should be evenly distributed to maximize performance, increase availability (reduce bottlenecks), and provide location awareness, which is an ever-increasing requirement for mobile applications.

12.11-a *Data Fragmentation*

Data fragmentation allows you to break a single object into two or more segments, or fragments. The object might be a user's database, a system database, or a table. Each fragment can be stored at any site over a computer network. Information about data fragmentation is stored in the distributed data catalog (DDC), from which it is accessed by the TP to process user requests.

- Horizontal fragmentation refers to the division of a relation into subsets (fragments) of tuples (rows). Each fragment is stored at a different node, and each fragment has unique rows. However, the unique rows all have the same attributes (columns). In short, each fragment represents the equivalent of a SELECT statement, with the WHERE clause on a single attribute.
- Vertical fragmentation refers to the division of a relation into attribute (column) subsets. Each subset (fragment) is stored at a different node, and each fragment has unique columns—with the exception of the key column, which is common to all fragments. This is the equivalent of the PROJECT statement in SQL.
- Mixed fragmentation refers to a combination of horizontal and vertical strategies. In other words, a table may be divided into several horizontal subsets (rows), each one having a subset of the attributes (columns).

12.11-b *Data Replication*

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

Three replication 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. In this case, all database fragments are replicated. A fully replicated database can be impractical due to the amount of overhead it imposes on the system.
- A partially replicated database stores multiple copies of some database fragments at multiple sites. Most DDBMSs are able to handle the partially replicated database well.
- An unreplicated database stores each database fragment at a single site. Therefore, there are no duplicate database fragments.

12.11-c *Data Allocation*

Data allocation describes the process of deciding where to locate data.

Data allocation strategies are as follows :

- With centralized data allocation, the entire database is stored at one site.
- With partitioned data allocation, the database is divided into two or more disjointed parts (fragments) and stored at two or more sites.
- With replicated data allocation, copies of one or more database fragments are stored at several sites.

12.11-b *The CAP Theorem*

In a 2000 symposium on distributed computing, Dr. Eric Brewer stated in his presentation that "in any highly distributed data system there are three commonly desirable properties: consistency, availability, and partition tolerance. However, it is impossible for a system to provide all three properties at the same time."

The initials CAP stand for the three desirable properties. Consider these three properties in more detail :

- *Consistency*. In a distributed database, consistency takes a bigger role. All nodes should see the same data at the same time, which means that the replicas should be immediately updated. However, this involves dealing with latency and network partitioning delays, as you learned in Section 12-10.
- *Availability*. Simply speaking, a request is always fulfilled by the system. No received request is ever lost. If you are buying tickets online, you do not want the system to stop in the middle of the operation. This is a paramount requirement of all web-centric organizations.
- *Partition tolerance*. The system continues to operate even in the event of a node failure. This is the equivalent of failure transparency in distributed databases (see Section 12-7). The system will fail only if all nodes fail.

* Transaction management consistency is different from CAP Theorem; Transition Management refers to the result when executing a transaction yields a database that complies with all integrity constraints. Consistency in CAP is based on the assumption that all transaction operations take place at the same time in all nodes, as if they were executing in a single-node database. ("All nodes see the same data at the same time.")

DISTRIBUTED DATABASE SPECTRUM

DBMS TYPE	CONSISTENCY	AVAILABILITY	PARTITION TOLERANCE	TRANSACTION MODEL	TRADE-OFF
Centralized DBMS	High	High	N/A	ACID	No distributed data processing
Relational DBMS	High	Relaxed	High	ACID (2PC)	Sacrifices availability to ensure consistency and isolation.
NoSQL DDBMS	Relaxed	High	High	BASE	Sacrifices consistency to ensure availability