Distributed Databases & Client - Structure Architecture

Contents

- What is It
- Disadvantages
- Functionality
- Where is it Used
- Advantages

Future
**Distributed databases:**

**Definition**—A distributed database is a single logical database that is spread physically across computers in multiple locations that are connected by a data communications network.

Two types of distributed database management systems:

- Homogeneous
- Heterogeneous
Homogeneous

All data centers have the same software

Much easier to design and manage

The user sees it as a single system
Heterogeneous

Different data centers run different DBMS products with possibly different underlying data models.

Occurs when sites have already implemented their own databases and integration is considered later.

Product translation is required to allow for:

- Different hardware
- Different DBMS products

An example would be one site using SQL software while the others are using Oracle.
Heterogenous

Advantages:

· Massive amounts of data can be stored in one global center from different data centers

· Remote access is done using the global schema

· Different DBMS may be used at each node

Disadvantages

· Difficult to design and manage
Database fragmentation

Decomposing a database into multiple smaller units called fragments, which are logically related and correct parts

- Must be complete
- Must be possible to reconstruct the original database from fragments
Database fragmentation

Fragmentation aims to improve:

- Reliability
- Performance
- Balanced storage capacity and costs
- Communication costs
- Security
Database Fragmentation

Three types:

- Horizontal
- Vertical
- Mixed
Database Fragmentation

Horizontal

A horizontal subset of a relation in which it contains those of tuples which satisfy selection conditions

Specified in the SELECT operation of the relational algebra on single or multiple attributes
Database Fragmentation

Primary Horizontal Fragmentation:
   Fragmentation of primary relation

Derived Horizontal Fragmentation:
   Fragmentations of the secondary relations that are dependent on the primary relation; related with Foreign keys

Complete Horizontal Fragmentation:
   Horizontal fragments have each and every tuple of the original relation
   Each tuple of the original relation will belong to at least one partition

Disjoint horizontal fragmentation:
   No two fragments EVER have common tuples
   Every one tuple of the original relation belongs to one fragment
Database Fragmentation

Reconstruction of original relation (horizontal fragmentation)

Apply the UNION operation to the fragments

Original Relation ← (Fragment 1) U (Fragment 2) U (Fragment 3)
Database Fragmentation

Vertical Fragmentation

Used when all attributes of a relation are not needed, divide “vertically” by columns

Subset of a relation which is created by a subset of columns
Database Fragmentation

Complete Vertical Fragmentation:

- A set of vertical fragments whose projection lists L1, L2, ..., Ln include all the attributes in R but share only the primary key of R

- Projection lists satisfy the following two conditions:
  
  o $L_1 \cup L_2 \cup ... \cup L_n = \text{ATTRS}(R)$

  o $L_i \cap L_j = \text{PK}(R)$ for any $i$ \& $j$, where $\text{ATTRS}(R)$ is the set of attributes of R and $\text{PK}(R)$ is the primary key of R
Database Fragmentation

Reconstruction of original relation (vertical fragmentation)

- An OUTER UNION is applied to reconstruct R from complete vertical fragments
Database Fragmentation

Mixed Fragmentation

A combination of Vertical and Horizontal fragmentation

Achieved by SELECT-PROJECT operations

- Represented by $\pi L_1(\sigma C_i(R))$
Database Replication

- Useful in improving the availability of data by copying data at multiple sites
- Either a relation or a fragment can be replicated at one or more sites
- Fully redundant databases are those in which every site contains a copy of the entire database
- The replication of fragments is described as a replication schema
Database Replication

Depending on the availability and redundancy factor there are three types of replications:

- Full Replication
- No Replication
- Partial Replication
Database Replication

Full Replication:

Replication of the whole database at every site in the distributed system

Can improve availability remarkably because the system can continue to operate as long as at least one site is up

Improves performance for retrieval of global queries as the result can be obtained locally at any client

Disadvantage is it slows the update process as a single update must be performed at different databases to keep the copies consistent
Database Replication

No Replication:

Each fragment is stored at exactly one site

All fragments must be disjoint, except for the repetition of primary keys among vertical fragments

Also known as nonredundant allocation
Database Replication

Partial Replication

Some fragments are replicated while others are not

The number of copies of each fragment can range from one up to the total number of sites in the distributed system

An example of this are workers on the go who would carry partially replicated databases on their laptops and mobile devices and periodically synchronize with the server databases
Allocation

Each fragment or fragment copy must be assigned to a particular site in the distributed system.

Choice of sites and the degree of replication depend on the performance and availability goals of the system and on the types and frequencies of transactions submitted at each site.

If high availability is required, transactions can be submitted at any site, and most transactions are retrieval only, a fully replicated database is a good choice.

If certain transactions that access particular parts of the database are mostly submitted at a particular site, the corresponding set of fragments can be allocated at that site only.

Data that is accessed at multiple site can be replicated at those sites. If many updates are performed, it may be useful to limit replication.
User Access Locks

A **lock** is used when multiple users need to access a database concurrently. This prevents data from being corrupted or invalidated when multiple users try to write to the database.

Any single user can only modify those database records (that is, items in the database) to which they have applied a lock that gives them exclusive access to the record until the lock is released. Locking not only provides exclusivity to write but also prevents (or controls) reading of unfinished modifications.
Where Are Distributed Databases Used?

Facilities in modern organizations are often geographically distributed, and often across national boundaries. Often each unit has the authority to create its own information systems, and often these units want local data over which they can have control. Business mergers and acquisitions often create this environment.

- Data sharing:
  - Even moderately complex business decisions require sharing data across business units, so it must be convenient to consolidate data across local databases on demand.
Data communications costs and reliability

The cost to ship large quantities of data across a communications network or to handle a large volume of transactions from remote sources can still be high.

It is more economical to locate data and applications close to where they are needed. Dependence on data communications always involves an element of risk.

Keeping local copies or fragments of data can be a reliable way to support the need for rapid access to data across the organization.
How Long Have They Been Around?

The ability to create a distributed database has existed since at least the 1980s. As you might expect, a variety of distributed database options exist (Bell and Grimson, 1992).
Distributed Database Environments

Homogeneous
The same DBMS is used at each node.
A. Autonomous
Each DBMS works independently, passing messages back and forth to share data updates.
B. Nonautonomous
A central, or master, DBMS coordinates database access and updates across the nodes.
II. Heterogeneous
Potentially different DBMSs are used at each node.
Distributed Database Environments Continued

II. Heterogeneous
Potentially different DBMSs are used at each node.
A. Systems
Supports some or all of the functionality of one logical database.
1. Full DBMS functionality
Supports all of the functionality of a distributed database, as discussed in the remainder of this chapter.
Chapter 12

2. Partial-multidatabase
Supports some features of a distributed database, as discussed in the remainder of this chapter.
a. Federated
Supports local databases for unique data requests.
i. Loose integration
Many schemas exist, for each local database, and each local DBMS must communicate with all local schemas.

ii. Tight integration
One global schema exists that defines all the data across all local databases.

b. Unfederated
Requires all access to go through a central coordinating module.
Continued

B. Gateways
Simple paths are created to other databases, without the benefits of one logical database.

A homogeneous distributed database environment is depicted in Figure 12-2. This environment is typically defined by the following characteristics (related to the nonautonomous category described previously):

• Data are distributed across all the nodes.
• The same DBMS is used at each location.
• All data are managed by the distributed DBMS (so there are no exclusively local data).
a distributed database management system (DDBMS) is a software system that manages a distributed database while making the distribution transparent to the user.
Transparency

Data organization transparency (also known as *distribution* or *network transparency*). This refers to freedom for the user from the operational details of the network and the placement of the data in the distributed system. It may be divided into location transparency and naming transparency.

**Location transparency** refers to the fact that the command used to perform a task is independent of the location of the data and the 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.** As we show in Figure 25.1, copies of the same data objects may be stored at multiple sites for better availability, performance, and reliability. Replication transparency makes the user unaware of the existence of these copies.

- **Fragmentation transparency.** Two types of fragmentation are possible. **Horizontal fragmentation** distributes a relation (table) into subrelations
Autonomy
determines the extent to which individual nodes or DBs in a connected DDB can operate independently. A high degree of autonomy is desirable for increased flexibility and customized maintenance of an individual node. Autonomy can be applied to design, communication, and execution. Design autonomy refers to independence of data model usage and transaction management techniques among nodes. Communication autonomy determines the extent to which each node can decide on sharing of information with other nodes. Execution autonomy refers to independence of users to act as they please.
Advantages of Distributed Databases

1. **Improved ease and flexibility of application development.** Developing and maintaining applications at geographically distributed sites of an organization is facilitated owing to transparency of data distribution and control.

2. **Increased reliability and availability.** This is achieved by the isolation of faults to their site of origin without affecting the other databases connected to the network. When the data and DDBMS software are distributed over several sites, one site may fail while other sites continue to operate. Only the data and software that exist at the failed site cannot be accessed. This improves both reliability and availability. Further improvement is achieved by judiciously replicating data and software at more than one site. In a centralized system, failure at a single site makes the whole system unavailable to all users. In a distributed database, some of the data may be unreachable, but users may still be able to access other parts of the database. If the data in the failed site had been replicated at another site prior to the failure, then the user will not be affected at all.
Advantages

3. Improved performance. A distributed DBMS fragments the database by keeping the data closer to where it is needed most. Data localization reduces the contention for CPU and I/O services and simultaneously reduces access delays involved in wide area networks. When a large database is distributed over multiple sites, smaller databases exist at each site. As a result, local queries and transactions accessing data at a single site have better performance because of the smaller local databases. In addition, each site has a smaller number of transactions executing than if all transactions are submitted to a single centralized database. Moreover, interquery and intraquery parallelism can be achieved by executing multiple queries at different sites, or by breaking up a query into a number of subqueries that execute in parallel. This contributes to improved performance.
Advantages

4. **Easier expansion.** In a distributed environment, expansion of the system in terms of adding more data, increasing database sizes, or adding more processors is much easier. Total transparency provides the global user with a view of the entire DDBS as if it is a single centralized system. Transparency is provided as a complement to **autonomy**, which gives the users tighter control over local databases. Transparency features may be implemented as a part of the user language, which may translate the required services into appropriate operations. Additionally, transparency impacts the features that must be provided by the operating system and the DBMS.
Additional Functions of Distributed Databases

- **Keeping track of data distribution.** The ability to keep track of the data distribution, fragmentation, and replication by expanding the DDBMS catalog.
- **Distributed query processing.** The ability to access remote sites and transmit queries and data among the various sites via a communication network.
- **Distributed transaction management.** The ability to devise execution strategies for queries and transactions that access data from more than one site and to synchronize the access to distributed data and maintain the integrity of the overall database.
- **Replicated data management.** The ability to decide which copy of a replicated data item to access and to maintain the consistency of copies of a replicated data item.
- **Distributed database recovery.** The ability to recover from individual site crashes and from new types of failures, such as the failure of communication links.
Security. Distributed transactions must be executed with the proper management of the security of the data and the authorization/access privileges of users.

Distributed directory (catalog) management. A directory contains information (metadata) about data in the database. The directory may be global for the entire DDB, or local for each site. The placement and distribution of the directory are design and policy issues. These functions themselves increase the complexity of a DDBMS over a centralized DBMS. Before we can realize the full potential advantages of distribution, we must find satisfactory solutions to these design issues and problems. Including all this additional functionality is hard to accomplish, and finding optimal solutions is a step beyond that.
Three-Tier Client-Server Architecture

1. **Presentation layer (client).** This provides the user interface and interacts with the user. The programs at this layer present Web interfaces or forms to the client in order to interface with the application. Web browsers are often utilized, and the languages and specifications used include HTML, XHTML, CSS, Flash, MathML, Scalable Vector Graphics (SVG), Java, JavaScript, Adobe Flex, and others. This layer handles user input, output, and navigation by accepting user commands and displaying the needed information, usually in the form of static or dynamic Web pages. The latter are employed when the interaction involves database access. When a Web interface is used, this layer typically communicates with the application layer via the HTTP protocol.

2. **Application layer (business logic).** This layer programs the application logic. For example, queries can be formulated based on user input from the client, or query results can be formatted and sent to the client for presentation. Additional application functionality can be handled at this layer, such as security checks, identity verification, and other functions. The application layer can interact with one or more databases or data sources as needed by connecting to the database using ODBC, JDBC, SQL/CLI, or other database access techniques.
3-tier client-server architecture

**Figure 25.7**
The three-tier client-server architecture.

- **Client**
  - User interface or presentation tier
    - (Web browser, HTML, JavaScript, Visual Basic, ...)
  - HTTP Protocol

- **Application server**
  - Application (business) logic tier
    - (Application program, JAVA, C/C++, C#, ...) 
  - ODBC, JDBC, SQL/CLI, SQLJ

- **Database server**
  - Query and transaction processing tier
    - (Database access, SQL, PSM, XML, ...)

3. Database server. This layer handles query and update requests from the application layer, processes the requests, and sends the results. Usually SQL is used to access the database if it is relational or object-relational and stored database procedures may also be invoked. Query results (and queries) may be formatted into XML when transmitted between the application server and the database server.
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.
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.
The client is any computer process that requests services from the server.

The client is also known as the Front-end Application.
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.
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
- Network 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 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

- 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.
Continued...

- 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.