UNIT 4 DISTRIBUTED AND CLIENT SERVER DATABASES

Structure **Page Nos.**

4.0 INTRODUCTION

In the previous units, we have discussed the basic issues relating to Centralised database systems. This unit discusses the distributed database systems which are primarily relational and one important implementation model: the client server model. This unit focuses on the basic issues involved in the design of distributed database designs. The unit also discusses some very basic concepts; in respect of client server computing including the basic structure, advantages/disadvantages etc. It will be worth mentioning here that almost all the commercial database management systems follow such a model of implementation. Although in client server model one of the concepts is the concept of distribution but it is primarily distribution of roles on server computers or rather distribution of work. So a client server system may be a centralised database.

4.1 OBJECTIVES

After going through this unit, you should be able to:

- differentiate DDBMS and conventional DBMS;
- define Network topology for DDBMS;
- define the concepts of horizontal and vertical fragmentation;
- define the concepts and rates of client server computing, and
- identify the needs of all these systems.

4.2 NEED FOR DISTRIBUTED DATABASE SYSTEMS

A *distributed database* is a set of database stored on multiple computers that appears to applications as a single database. As a result, an application can simultaneously access and modify the data in several databases in a network. Each database in the

system is controlled by its local server but cooperates to maintain the consistency of the global distributed database. The computers in a distributed system communicate with each other through various communication media, such as high-speed buses or telephone line. They don't share main memory, nor a clock, although, to work properly many applications on different computers might have to synchronise their clocks. In some cases the absolute time might be important and the clocks might have to be synchronised with atomic clocks.

The processors in a distributed system may vary in size and function such as small microcomputers, workstation, minicomputers, and large general-purpose computer system. These processors are referred to by sites, nodes, computers, and so on, depending on the context in which they are mentioned. We mainly use the term site, in order to emphasise the physical distribution of these systems.

A distributed database system consists of a collection of sites, each of which may participate in the execution of transactions, which access data at one site, or several sites. The main difference between centralised and distributed database systems is that, in the former, the data resides in one single Centralised control, while in the latter, the data resides in several sets under the control of local distributed DBMS components which are under the control of one DBMS. As we shall see, this distribution of data is the cause of many difficulties that will be addressed in this unit.

Independent or decentralised systems were normally used in earlier days. There was duplication of hardware and other facilities. Evolution of computer systems also led to incompatible procedures and lack of management control. A centralised database system was then evolved. In a centralised database the DBMS and data reside at a single database instance. Although for recovery purposes we keep redundant database information yet it is under the control of a single DBMS. A further enhancement of the centralised database system may be to provide access to centralised data from a number of distributed locations through a network. In such a system a site failure except the central site will not result in total system failure. Although, communication technology has greatly improved yet the centralised approach may create problems for an organization that has geographically dispersed operations and data has to be accessed from a centralised database. Some of the problems may be:

a. loss of messages between a local and central site;
b. failure of communication links between local and central site. This would make the system unreliable;
c. excessive load on the central site would delay queries and accesses. A single site would have to bear a large number of transactions and hence would require large computing systems.

The problems as above could be addressed using distributed database systems. It improves the reliability and sharability of data and the efficiency of data access. Distributed Database Systems can be considered a system connected to intelligent remote devices each of which can itself act as a local database repository. All data is accessible from each site. The distributed system increases the efficiency of access because multiple of sites can co-ordinate efficiently to respond to a query and control & processing is limited to this DBMS.

4.3 STRUCTURE OF DISTRIBUTED DATABASE

A distributed database system consists of a collection of sites, each of which maintains a local databases system. Each site is able to process local transactions, those transactions that access data only in that single site. In addition, a site may

participate in the execution of global transactions, those transactions that access data at several sites. The architecture of Distributed Database systems is given in *Figure 1*

Figure 1: **Three different database system architectures. (a) No database sharing architecture. (b) A networked architecture with a centralised database. (c) A distributed database architecture**

The execution of global transactions on the distributed architecture requires communication among the sites. *Figure 2* illustrates a representative distributed database system architecture having transactions. Please note that both the transactions shown are global in nature as they require data from both the sites.

Figure 2: Distributed Database Schema and Transactions

Some of the key issues involving DDBMS are:

- How the data is distributed?
- Is this data distribution hidden from the general users?
- How is the integration of data and its control including security managed locally and globally?
- How are the distributed database connected?

Well we will provide the basic answers to most of these questions during the course of this unit. However, for more details, you may refer to further readings.

The sites in a distributed system can be connected physically in a variety of ways. The various topologies are represented as graphs whose nodes correspond to sites. An edge from node A to node B corresponds to a direct connection between the two sites.

Some of the most common connection structures are depicted in *Figure 3*. The major differences among these configurations involve:

- **Installation cost.** The cost of physically linking the sites in the system
- **Communication cost.** The cost in time and money to send a message from site A to site B.

- **Reliability.** The frequency with which a link or site fails.

- **Availability.** The degree to which data can be accessed despite failure of some links or sites.

These differences play an important role in choosing the appropriate mechanism for handling the distribution of data The sites of a distributed database system may be distributed physically either over a large geographical area (such as all over India), or over a small geographical area such as a single building or a number of adjacent buildings). The former type of network is based on wide area network, while the latter uses local-area network.

Figure 3: Some interconnection Networks

Since the sites in wide area networks are distributed physically over a large geographical area, the communication links are likely to be relatively slow and less reliable as compared with local-area networks. Typical wide area links are telephone lines, microwave links, and satellite channels. Many newer enhanced communication technologies including fiber optics are also used for such links. The local-area network sites are close to each other, communication links are of higher speed and lower error rate than their counterparts in wide area networks. The most common links are twisted pair, base band coaxial, broadband coaxial, and fiber optics.

A Distributed Transaction

Let us illustrate the concept of a distributed transaction by considering a banking system consisting of three branches located in three different cities. Each branch has its own computer with a database consisting of all the accounts maintained at that branch. Each such installation is thus a site. There also exists one single site which maintains information about all the other branches of the bank. Suppose that the database systems at the various sites are based on the relational model. Each branch maintains its portion of the relation: DEPOSIT (DEPOSIT-BRANCH) where

DEPOSIT-BRANCH = (branch-name, account-number, customer-name, balance)

A site containing information about the four branches maintains the relation branch-details, which has the schema as:

BRANCH-DETAILS (branch-name, Financial_ health, city)

There are other relations maintained at the various sites which are ignored for the purpose of our example.

A local transaction is a transaction that accesses accounts in one single site, at which the transaction was initiated. A global transaction, on the other hand, is one which either accesses accounts in a site different from the one at which the transaction was initiated, or accesses accounts in several different sites. To illustrate the difference between these two types of transactions, consider the transaction to add Rs.5000 to account number 177 located at the Delhi branch. If the transaction was initiated at the Delhi branch, then it is considered local; otherwise, it is considered global. A transaction to transfer Rs.5000 from account 177 to account 305, which is located at the Bombay branch, is a global transaction since accounts in two different sites are accessed as a result of its execution. A transaction finding the total financial standing of all branches is global.

What makes the above configuration a distributed database system are the facts that:

- The various sites may be locally controlled yet are aware of each other.
- Each site provides an environment for executing both local and global transactions.

However, a user need not know whether on underlying application is distributed or not.

4.4 ADVANTAGES AND DISADVANTAGES OF DDBMS

There are several reasons for building distributed database systems, including sharing of data, reliability and availability, and speedup of query processing. However, along with these advantages come several disadvantages, including software development

cost, greater potential for bugs, and increased processing overheads. In this section, we shall elaborate briefly on each of these.

4.4.1 Advantages of Data Distribution

The primary advantage of distributed database systems is the ability to share and access data in a reliable and efficient manner.

Data sharing and Distributed Control

The geographical distribution of an organization can be reflected in the distribution of the data; if a number of different sites are connected to each other, then a user at one site may be able to access data that is available at another site. The main advantage here is that the user need not know the site from which data is being accessed. Data can be placed at the site close to the users who normally use that data. The local control of data allows establishing and enforcement of local policies regarding use of local data. A global database administrator (DBA) is responsible for the entire system. Generally, part of this responsibility is given to the local administrator, so that the local DBA can manage the local DBMS. Thus in the distributed banking system, it is possible for a user to get his/her information from any branch office. This external mechanism would, in effect to a user, look to be a single centralised database.

The primary advantage of accomplishing data sharing by means of data distribution is that each site is able to retain a degree of control over data stored locally. Depending upon the design of the distributed database system, each local administrator may have a different degree of autonomy. This is often a major advantage of distributed databases. In a centralised system, the database administrator of the central site controls the database and, thus, no local control is possible.

Reflects organisational structure

Many organizations are distributed over several locations. If an organisation has many offices in different cities, databases used in such an application are distributed over these locations. Such an organisation may keep a database at each branch office containing details of the staff that work at that location, the local properties that are for rent, etc. The staff at a branch office will make local inquiries to such data of the database. The company headquarters may wish to make global inquiries involving the access of data at all or a number of branches.

Improved Reliability

In a centralised DBMS, a server failure terminates the operations of the DBMS. However, a failure at one site of a DDBMS, or a failure of a communication link making some sites inaccessible, does not make the entire system inoperable. Distributed DBMSs are designed to continue to function despite such failures. In particular, if data are replicated in several sites, a transaction needing a particular data item may find it at several sites. Thus, the failure of a site does not necessarily imply the shutdown of the system.

The failure of one site must be detected by the system, and appropriate action may be needed to recover from the failure. The system must no longer use the services of the failed site. Finally, when the failed site recovers or is repaired, mechanisms must be available to integrate it smoothly back into the system. The recovery from failure in distributed systems is much more complex than in a centralised system.

Improved availability

The data in a distributed system may be replicated so that it exists at more than one site. Thus, the failure of a node or a communication link does not necessarily make the data inaccessible. The ability of most of the systems to continue to operate despite the failure of one site results in increased availability which is crucial for database

systems used for real-time applications. For example, loss of access to data in an airline may result in the loss of potential ticket buyers to competitors.

Improved performance

As the data is located near the site of its demand, and given the inherent parallelism due to multiple copies, speed of database access may be better for distributed databases than that of the speed that is achievable through a remote centralised database. Furthermore, since each site handles only a part of the entire database, there may not be the same contention for CPU and I/O services as characterized by a centralised DBMS.

Speedup Query Processing

A query that involves data at several sites can be split into sub-queries. These sub-queries can be executed in parallel by several sites. Such parallel sub-query evaluation allows faster processing of a user's query. In those cases in which data is replicated, queries may be sent to the least heavily loaded sites.

Economics

It is now generally accepted that it costs less to create a system of smaller computers with the equivalent power of a single large computer. It is more cost-effective to obtain separate computers. The second potential cost saving may occurs where geographically remote access to distributed data is required. In such cases the economics is to minimise cost due to the data being transmitted across the network for data updates as opposed to the cost of local access. It may be more economical to partition the application and perform the processing locally at application site.

Modular growth

In distributed environments, it is easier to expand. New sites can be added to the network without affecting the operations of other sites, as they are somewhat independent. This flexibility allows an organisation to expand gradually. Adding processing and storage power to the network can generally result in better handling of ever increasing database size. A more powerful system in contrast, a centralised DBMS, would require changes in both the hardware and software with increasing size and more powerful DBMS to be procured.

4.4.2 Disadvantages of Data Distribution

The primary disadvantage of distributed database systems is the added complexity required to ensure proper coordination among the sites. This increased complexity takes the form of:

- **Higher Software development cost:** Distributed database systems are complex to implement and, thus, more costly. Increased complexity implies that we can expect the procurement and maintenance costs for a DDBMS to be higher than those for a centralised DBMS. In addition to software, a distributed DBMS requires additional hardware to establish a network between sites. There are ongoing communication costs incurred with the use of this network. There are also additional maintenance costs to manage and maintain the local DBMSs and the network.

- **Greater potential for bugs:** Since the sites of a distributed system operate concurrently, it is more difficult to ensure the correctness of algorithms. The art of constructing distributed algorithms is an active and important area of research.

- **Increased processing overhead:** The exchange of messages and the additional computation required to achieve coordination among the sites is an overhead that does not arise in centralised systems.

- **Complexity:** A distributed DBMS that is reliable, available and secure is inherently more complex than a centralised DBMS. Replication of data discussed in the next section, also adds to complexity of the distributed DBMS. However, adequate data replication is necessary to have availability, reliability, and performance.

- **Security:** In a centralised system, access to the data can be easily controlled. However, in a distributed DBMS not only does access to replicated data have to be controlled in multiple locations, but also the network needs to be secured. Networks over a period have become more secure, still there is a long way to go.

- **Lack of standards and experience:** The lack of standards has significantly limited the potential of distributed DBMSs. Also, there are no tools or methodologies to help users convert a centralised DBMS into a distributed DBMS.

- **General Integrity control more difficult:** Integrity is usually expressed in terms of constraints or the rules that must be followed by database values. In a distributed DBMS, the communication and processing costs that are required to enforce integrity constraints may be very high as the data is stored at various sites. However, with better algorithms we can reduce such costs.

- **Purpose:** General-purpose distributed DBMSs have not been widely accepted. We do not yet have the same level of experience in industry as we have with centralised DBMSs.

- **Database design more complex:** Besides the normal difficulties of designing a centralised database, the design of a distributed database has to take account of fragmentation of data, allocation of fragments to specific sites, and data replication. The designer of distributes systems must balance the advantages against the disadvantages of distribution of data.

☞ Check Your Progress 1

1) What are the advantages of distributed databases over centralised Databases?

……

……

……

2) Differntiate between global and local transaction.

……

……

……

4.5 DESIGN OF DISTRIBUTED DATABASES

The distributed databases are primarily relational at local level. So a local database schema is the same as that of a centralised database design. However, a few more dimensions have been added to the design of distributed database. These are:

- **Replication:** It is defined as a copy of a relation. Each replica is stored at a different site. The alternative to replication is to store only one copy of a relation which is not recommended in distributed databases.

- **Fragmentation:** It is defined as partitioning of a relation into several fragments. Each fragment can be stored at a different site.

The distributed database design is a combination of both these concepts. Let us discuss them in more detail in the following subsections.

4.5.1 Data Replication

"If a relation R has its copies stored at two or more sites, then it is considered replicated".

But why do we replicate a relation?

There are various advantages and disadvantages of replication of relation

- **Availability:** A site containing the copy of a replicated relation fails, even then the relation may be found in another site. Thus, the system may continue to process at least the queries involving just read operation despite the failure of one site. Write can also be performed but with suitable recovery algorithm.

- **Increased parallelism:** Since the replicated date has many copies a query can be answered from the least loaded site or can be distributed. Also, with more replicas you have greater chances that the needed data is found on the site where the transaction is executing. Hence, data replication can minimise movement of data between sites.

- **Increased overheads on update:** On the disadvantage side, it will require the system to ensure that all replicas of a relation are consistent. This implies that all the replicas of the relation need to be updated at the same time, resulting in increased overheads. For example, in a banking system, if account information is replicated at various sites, it is necessary that the balance in a particular account should be the same at all sites.

The problem of controlling concurrent updates by several transactions to replicated data is more complex than the centralised approach of concurrency control. The management of replicas of relation can be simplified by choosing one of the replicas as the primary copy. For example, in a banking system, the primary copy of an account may be the site at which the account has been opened. Similarly, in an airline reservation system, the primary copy of the flight may be associated with the site at which the flight originates. The replication can be classified as:

Complete replication: It implies maintaining of a complete copy of the database at each site. Therefore, the reliability and availability and performance for query response are maximized. However, storage costs, communication costs, and updates are expensive. To overcome some of these problems relation, snapshots are sometimes used. A snapshot is defined as the copy of the data at a given time. These copies are updated periodically, such as, hourly or weekly. Thus, snapshots may not always be up to date.

Selective replication: This is a combination of creating small fragments of relation and replicating them rather than a whole relation. The data should be fragmented on need basis of various sites, as per the frequency of use, otherwise data is kept at a centralised site. The objective of this strategy is to have just the advantages of the other approach but none of the disadvantages. This is the most commonly used strategy as it provides flexibility.

4.5.2 Data Fragmentation

"Fragmentation involves decomposing the data in relation to non-overlapping component relations".

Why do we need to fragment a relation? The reasons for fragmenting a relation are:

Use of partial data by applications: In general, applications work with views rather than entire relations. Therefore, it may be more appropriate to work with subsets of relations rather than entire data.

Increases efficiency: Data is stored close to most frequently used site, thus retrieval would be faster. Also, data that is not needed by local applications is not stored, thus the size of data to be looked into is smaller.

Parallelism of transaction execution: A transaction can be divided into several sub-queries that can operate on fragments in parallel. This increases the degree of concurrency in the system, thus allowing transactions to execute efficiently.

Security: Data not required by local applications is not stored at the site, thus no unnecessary security violations may exist.

But how do we carry out fragmentation? Fragmentation may be carried out as per the following rules:

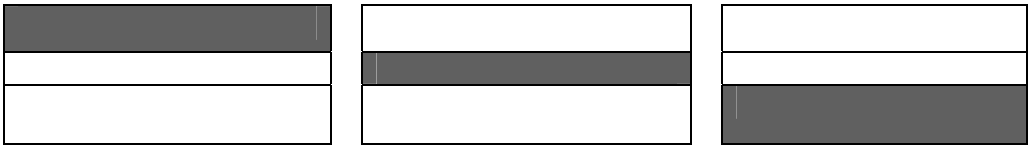
a) **Completeness:** This rule ensures that there is no loss of data during fragmentation. If a relation is decomposed into fragments, then each data item must appear in at least one fragment.

b) **Reconstruction:** This rule ensures preservation of functional dependencies. It should be possible to define a relational operation that will reconstruct the relation from its fragments.

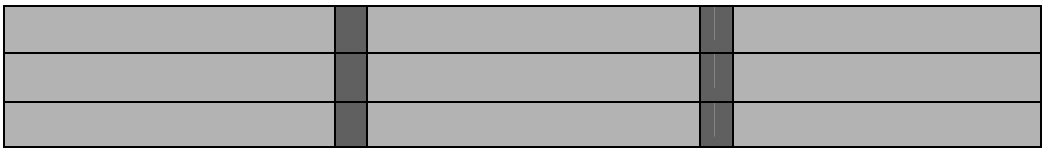
c) **Disjointness:** *A* data item that appears in a fragment should not appear in any other fragment. However, a typical fragmentation called vertical fragmentation is an exception to this rule. In vertical fragmentation the primary key attributes must be repeated to allow reconstruction of original relation. This rule ensures minimization of data redundancy.

What are the different types of fragmentation?

There are two main types of fragmentation: **horizontal** and **vertical**. Horizontal fragments, as the name suggests are subsets of tuples and vertical fragments are subsets of attributes (refer to Figure 4). There are also two other types of fragmentation: mixed, and derived–a type of horizontal fragmentation, are just introduced. A detailed discussion on them is beyond the scope of this unit.



(a) Horizontal Fragmentation



(b) Vertical Fragmentation

Figure 4: Horizontal and Vertical Fragmentation

Horizontal Fragmentation

Horizontal fragmentation groups together the tuples in a relation that are collectively used by the important transactions. A horizontal fragment is produced by specifying a WHERE clause condition that performs a restriction on the tuples in the relation. It can also be defined using the *Selection* operation of the relational algebra.

Example:

Let us illustrate horizontal fragmentation with the help of an example.

DEPOSIT (branch-code, account-number, customer-name, balance)

A sample relation instance of the relation DEPOSIT is shown in *Figure 5*.

Branch-code	Account number	Customer name	Balance
1101	3050	Suresh	5000
1101	2260	Swami	3360
1102	1170	Swami	2050
1102	4020	Khan	10000
1101	1550	Khan	620
1102	4080	Khan	1123
1102	6390	Khan	7500

Figure 5: Sample DEPOSIT relation

Mathematically a fragment may be defined as a selection on the global relation R. The reconstruction of the relation R can be obtained by taking the union of all fragments.

So let us decompose the table in Figure 5 into horizontal fragments. Let us do these fragments on the branch-code as 1101 and 1102

DEPOSIT1 obtained by selection on branch-code as 1101			
Branch-code	**Account number**	**Customer name**	**Balance**
1101	3050	Suresh	5000
1101	2260	Swami	3360
1101	1550	Khan	620
DEPOSIT2 obtained by selection on branch- code as 1102			
Branch-code	**Account number**	**Customer name**	**Balance**
1102	1770	Swami	2050
1102	4020	Khan	10000
1102	4080	Khan	1123
1102	6390	Khan	7500

Figure 6: Horizontal fragmentation of relation DEPOSIT

The two fragments can be defined in relational algebra as:

$$\text{DEPOSIT1} = \sigma_{\text{branch-code= 1101}} (\text{DEPOSIT})$$

$$\text{DEPOSIT2} = \sigma_{\text{branch-code= 1102}} (\text{DEPOSIT})$$

These two fragments are shown in *Figure 6*. Fragment 1 can be stored in the branch whose code is 1101 while the second fragment can be stored at branch 1102.

In our example, the fragments are disjoint. However, by changing the selection predicates used to construct the fragments; we may have overlapping horizontal fragments. This is a form of data replication.

Vertical Fragmentation

Vertical fragmentation groups together only those attributes in a relation that are used jointly by several important transactions. A vertical fragment is defined using the **Projection** operation of the relational algebra. In its most simple form, vertical

fragmentation is the same as that of decomposition. In general, a relation can be constructed on taking Natural join of all vertical fragments.

More generally, vertical fragmentation is accomplished by adding a special attribute called a tuple-number to the scheme R. A tuple-number is a physical or logical address for a tuple. Since each tuple in R must have a unique address, the tuple-number attribute is a key to the new fragments obtained (please refer to *Figure 7*).

Branch-code	Account number	Customer name	Balance	Tuple-number
1101	3050	Suresh	5000	1
1101	2260	Swami	3360	2
1102	1170	Swami	2050	3
1102	4020	Khan	10000	4
1101	1550	Khan	620	5
1102	4080	Khan	1123	6
1102	6390	Khan	7500	7

Figure 7: The relation DEPOSIT of figure 5 with tuple- numbers

This relation now can be decomposed into two fragments as: shows a vertical decomposition of the scheme Deposit-scheme tuple number into:

$$\text{DEPOSIT3} = \prod_{\text{(branch-code. customer-name, tuple-number)}} (\text{DEPOSIT})$$

$$\text{DEPOSIT4} = \prod_{\text{(account-number, balance, tuple-number)}} (\text{DEPOSIT})$$

The example of Figure7 on this basis would become:

DEPOSIT3		
Branch-code	**Customer-name**	**Tuple-number**
1101	Suresh	1
1101	Swami	2
1102	Swami	3
1102	Khan	4
1101	Khan	5
1102	Khan	6
1102	Khan	7
DEPOSIT4		
Account number	**Balance**	**Tuple-number**
3050	5000	1
2260	3360	2
1170	2050	3
4020	10000	4
1550	620	5
4080	1123	6
6390	7500	7

Figure 8: Vertical fragmentation of relation DEPOSIT

How can we reconstruct the original relation from these two fragments? By taking natural join of the two vertical fragments on tuple-number. The tuple number allows direct retrieval of the tuples without the need for an index. Thus, this natural join may be computed much more efficiently than typical natural join.

However, please note that as the tuple numbers are system generated, therefore they should not be visible to general users. If users are given access to tuple-number, it becomes impossible for the system to change tuple addresses.

Mixed fragmentation

Sometimes, horizontal or vertical fragmentation of a database schema by itself is insufficient to adequately distribute the data for some applications. Instead, **mixed** or **hybrid** fragmentation is required. Mixed fragmentation consists of a horizontal fragment that is vertically fragmented, or a vertical fragment that is then horizontally fragmented.

Derived horizontal fragmentation

Some applications may involve a join of two or more relations. If the relations are stored at different locations, there may be a significant overhead in processing the join. In such cases, it may be more appropriate to ensure that the relations, or fragments of relations, that are joined together are at the same location. We can achieve this using derived horizontal fragmentation.

After going through the basis of concepts relating to distributed database systems, let us sketch the process of design of distributed database systems.

A step-wise distributed database design methodology

Following is a step-wise methodology for distributed database design.

(1) Examine the nature of distribution. Find out whether an organisation needs to have a database at each branch office, or in each city, or possibly at a regional level. It has direct implication from the viewpoint of fragmentation. For example, in case database is needed at each branch office, the relations may fragment on the basis of branch number.
(2) Create a detailed global E-R diagram, if so needed and identify relations from entities.
(3) Analyse the most important transactions in the system and identify where horizontal or vertical fragmentation may be desirable and useful.
(4) Identify the relations that are not to be fragmented. Such relations will be replicated everywhere. From the global ER diagram, remove the relations that are not going to be fragmented.
(5) Examine the relations that are on one-side of a relationship and decide a suitable fragmentation schema for these relations. Relations on the many-side of a relationship may be the candidates for derived fragmentation.
(6) During the previous step, check for situations where either vertical or mixed fragmentation would be needed, that is, where the transactions require access to a subset of the attributes of a relation.

☞ Check Your Progress 2

1) What are the rules while fragmenting data?

……

……

……

……

2) What is Data Replication? What are its advantages & disadvantages?

……

……

……

……

……

……

……

4.6 CLIENT SERVER DATABASES

The concept behind the Client/Server systems is concurrent, cooperative processing. It is an approach that presents a single systems view from a user's viewpoint. It involves processing on multiple, interconnected machines. It provides coordination of activities in a manner transparent to end-users. Remember, client-server database is distribution of activities into clients and a server. It may have a centralised or distributed database system at the server backend. It is primarily a very popular commercial database implementation model.

4.6.1 Emergence of Client Server Architecture

Some of the pioneering work that was done by some of the relational database vendors allowed the computing to be distributed on multiple computers on network using contemporary technologies involving:

- Low Cost, High Performance PCs and Servers
- Graphical User Interfaces
- Open Systems
- Object-Orientation
- Workgroup Computing
- EDI and E-Mail
- Relational Databases
- Networking and Data Communication.

4.6.2 Need for Client Server Computing

Client/Server (C/S) architecture involves running the application on multiple machines in which each machine with its component software handles only a part of the job. Client machine is basically a PC or a workstation that provides presentation services and the appropriate computing, connectivity and interfaces while the server machine provides database services, connectivity and computing services to multiple users. Both client machines and server machines are connected to the same network. As the number of users grows, client machines can be added to the network while as the load on the database server increases more servers can be connected through the network. Thus, client-server combination is a scalable combination. Server machines are more powerful machines database services to multiple client requests.

The client-server systems are connected though the network. This network need not only be the Local Area Network (LAN). It can also be the Wide Area Network (WAN) across multiple cities. The client and server machines communicate through standard application program interfaces (called API), and remote procedure calls (RPC). The language through which RDBMS based C/S environment communicate is the structured query language (SQL).

4.6.3 Structure of Client Server Systems

In client/server architecture, clients represent users who need services while servers provide services. Both client and server are a combination of hardware and software. Servers are separate logical objects that communicate with clients over a network to perform tasks together. A client makes a request for a service and receives a reply to that request. A server receives and processes a request, and sends back the required response. The client/server systems may contain two different types of architecture - 2-Tier and 3-Tier Client/Server Architectures

Every client/server application contains three functional units:

- Presentation logic which provides the human/machine interaction (the user interface). The presentation layer handles input from the keyboard, mouse, or

other input devices and provides output in the form of screen displays. For example, the ATM machine of a bank provides such interfaces.

- Business logic is the functionality provided to an application program. For example, software that enables a customer to request to operate his/her balance on his/her account with the bank is business logic. It includes rules for withdrawal, for minimum balance etc. It is often called business logic because it contains the business rules that drive a given enterprise.

- The bottom layer provides the generalized services needed by the other layers including file services, print services, communications services and database services. One example of such a service may be to provide the records of customer accounts.

These functional units can reside on either the client or on one or more servers in the application:

Figure 9: A client-server system

In general two popular client/server systems are:

- 2-Tier client Server Models

- 3-Tier client server Model

2-Tier Client/Server Models

Initial two-tier (client/server) applications were developed to access large databases available on the server side and incorporated the rules used to manipulate the data with the user interface into the client application. The primary task of the server was simply to process as many requests for data storage and retrieval as possible.

Two-tier client/server provides the user system interface usually on the desktop environment to its users. The database management services are usually on the server that is a more powerful machine and services many clients. Thus, 2-Tier client-server architecture splits the processing between the user system interface environment and the database management server environment. The database management server also provides stored procedures and triggers. There are a number of software vendors that provide tools to simplify the development of applications for the two-tier client/server architecture.

In 2-tier client/server applications, the business logic is put inside the user interface on the client or within the database on the server in the form of stored procedures. This results in division of the business logic between the client and server. File servers and database servers with stored procedures are examples of 2-tier architecture.

The two-tier client/server architecture is a good solution for distributed computing. Please note the use of words distributed computing and not distributed databases. A 2-tier client server system may have a centralised database management system or distributed database system at the server or servers. A client group of clients on a LAN can consist of a dozen to 100 people interacting simultaneously. A client server system does have a number of limitations. When the number of users exceeds 100, performance begins to deteriorate. This limitation is a result of the server maintaining a connection via communication messages with each client, even when no work is being done.

A second limitation of the two-tier architecture is that implementation of processing management services using vendor proprietary database procedures restricts flexibility and choice of DBMS for applications. The implementations of the two-tier architecture provides limited flexibility in moving program functionality from one server to another without manually regenerating procedural code.

Figure 10: 2-tier and 3 tier client server systems

Some of the major functions performed by the client of a two-tier application are: present a user interface, gather and process user input, perform the requested processing, report the status of the request.

This sequence of commands can be repeated as many times as necessary. Because servers provide only access to the data, the client uses its local resources to perform most of the processing. The client application must contain information about where the data resides and how it is organised in the server database. Once the data has been retrieved, the client is responsible for formatting and displaying it to the user.

3-tier architecture

As the number of clients increases the server would be filled with the client requests. Also, because much of the processing logic was tied to applications, changes in business rules lead to expensive and time-consuming alterations to source code. Although the ease and flexibility of two-tier architecture tools continue to drive many small-scale business applications, the need for faster data access and rapid developmental and maintenance timelines has persuaded systems developers to seek out a new way of creating distributed applications.

The three-tier architecture emerged to overcome the limitations of the two tier architecture (also referred to as the multi-tier architecture). In the three-tier architecture, a middle tier was added between the user system interface client environment and the database management server environment. The middle tier may consist of transaction processing monitors, message servers, or application servers.

The middle-tier can perform queuing, application execution, and database staging. For example, on a middle tier that provides queuing, the client can deliver its request to the middle layer and simply gets disconnected because the middle tier will access the data and return the answer to the client. In addition the middle layer adds scheduling and prioritisation for various tasks that are currently being performed.

The three-tier client/server architecture has improved performance for client groups with a large number of users (in the thousands) and improves flexibility when compared to the two-tier approach. Flexibility is in terms of simply moving an application on to different computers in three-tier architecture. It has become as simple as “drag and drop” tool. Recently, mainframes have found a new use as servers in three-tier architectures:

In 3-tier client/server applications, the business logic resides in the middle tier, separate from the data and user interface. In this way, processes can be managed and deployed separately from the user interface and the database. Also, 3-tier systems can integrate data from multiple sources.

4.6.4 Advantages of Client Server Computing

Client/Server systems have following advantages:

- They provide low cost and user-friendly environment
- They offer expandability that ensures that the performance degradation is not so much with increased load.
- They allow connectivity with the heterogeneous machines deals with real time data feeders like ATMs, Numerical machines.
- They allow enforcement of database management activities including security, performance, backup, integrity to be a part of the database server machine. Thus, avoiding the requirement to write a large number of redundant piece of code dealing with database field validation and referential integrity.

- One major advantage of the client/server model is that by allowing multiple users to simultaneously access the same application data, updates from one computer are instantly made available to all computers that had access to the server.

Client/server systems can be used to develop highly complex multi-user database applications being handled by any mainframe computer.

Since PCs can be used as clients, the application can be connected to the spreadsheets and other applications through Dynamic Data Exchange (DDE) and Object Linking and Embedding (OLE). If the load on database machine grows, the same application can be run on a slightly upgraded machine provided it offers the same version of RDBMSs.

A more detailed discussion on these topics is beyond the scope of this unit. You can refer to further readings for more details.

☞ Check your Progress 3

1) What are the advantages of Client/Server Computing?

……

……

2) Describe the Architecture of Client/Server system.

……

……

4.7 SUMMARY

A distributed database system consists of a collection of sites, each of which maintains a local database system. Each site is able to process local transactions–those transactions that access data only at that single site. In addition, a site may participate in the execution of global transactions–those transactions that access data at several sites. The execution of global transactions requires communication among the sites.

There are several reasons for building distributed database systems, including sharing of data, reliability and availability, and speed of query processing. However, along with these advantages come several disadvantages, including software development cost, greater potential for bugs, and increased processing overheads. The primary disadvantage of distributed database systems is the added complexity required to ensure proper co-ordination among the sites.

There are several issues involved in storing a relation in the distributed database, including replication and fragmentation. It is essential that the system minimise the degree to which a user needs to be aware of how a relation is stored.

Companies that have moved out of the mainframe system to Client/Server architecture have found three major advantages:

- Client/Server technology is more flexible and responsive to user needs
- A significant reduction in data processing costs
- An increase in business competitiveness as the market edge turns towards merchandising.

4.8 SOLUTIONS/ANSWERS

Check Your Progress 1

1) The advantages of DDBMS are:

- The primary advantages of distributed database systems is the ability to share and access data in a reliable and efficient manner. A user at one site may be able to access data that is available at another site.
- Each site is able to retain a degree of control over data stored locally.
- It reflects organisational structure
- If distributed over several locations then improved availability of DBMS. A failure of a communication link making some sites inaccessible does not make the entire system inoperable. The ability of most of the systems to continue to operate despite failure of onesite results in increased availability, which is crucial for database systems used for real-time applications.
- Speed of database access may be better than that achievable from a remote centralised database. Parallel computation allows faster processing of a user's query.
- It costs much less to create a system of smaller computers with lower equivalent a single large computer. It is much easier to expand. New sites can be added to the network without affecting the operations of other sites.

2)

Global Transactions	**Local Transactions**
Involves multiple sites	Can be performed locally
Uses data communication links	Performed locally
Takes longer to execute, in general	Faster
Performance depends on global schema objects available	Performance is governed by local schema objects like local indexes
Just 20-25 % of total	Most transactions are local

Check Your Progress 2

1) Fragmentation cannot be carried out haphazardly. There are three rules that must be followed during fragmentation.

a) Completeness: If a relation R is decomposed into fragments R1, R2,.. Rn, each data item that can be found in R must appear in at least one fragment.

b) Reconstruction: It must be possible to define a relational operational that will reconstruct the relation R from the fragments.

c) Disjointness: If a data item d appears in fragment R1 then it should not appear in any other fragment. Vertical fragmentation is the exception to this rule, where primary key attributes must be repeated to allow reconstruction.

2) If relation R is replicated a copy of relation R is stored in two or more sites. There are a number of advantages and disadvantages to replication.

Availability: If one of the sites containing relation R fails, then the relation R may found in another site.

Increased parallelism: In cases where the majority of access to the relation R results in only the reading of the relation, several sites can process queries involving R in parallel.

Increasing overhead on update: The system must ensure that all replicas of a relation R are consistent, otherwise erroneous computations may result. This implies that whenever R is updated, this update must be propagated to all sites containing replicas, resulting in increased overheads.

Check Your Progress 3

1) Advantages of client/server computing are as follows:

- C/S computing caters to low-cost and user-friendly environment.
- It offers expandability.
- It ensures that the performance degradation is not so much with increased load.
- It allows connectivity with the heterogeneous machines and also with real time data feeders.
- It allows server enforced security, performance, backup, integrity to be a part of the database machine avoiding the requirement to write a large number of redundant piece of code dealing with database field validation and referential integrity, allows multiple users to simultaneously access the same application data.
- Updates from one computer are instantly made available to all computers that had access to the server.
- It can be used to develop highly complex multi-user database applications being handled by any mainframe computer. If the load on database machine grows, the same application can be run on a slightly upgraded machine.

2) **2-Tier client/Server Models**

With two-tier client/server, the user system interface is usually located in the user's desktop environment and the database management services are usually in a server. Processing management is spilt between the user system interface environment and the database management server environment. The database management server provides stored procedures and triggers. In 2-tier client/server applications, the business logic is buried inside the user interface on the client or
within the database on the server in the form of stored procedures.
Alternatively, the business logic can be divided between client and server.

3-tier architecture

The three-tier architecture (also referred to as the multi-tier architecture) emerged to overcome the limitations of the two-tier architecture. In the three-tier architecture, a middle tier was added between the user system interface client environment and the database management server environment. There are a variety of ways of implementing this middle tier, such as transaction processing monitors, message servers, or application servers. The middle tier can perform queuing, application execution, and database staging. In addition the middle layer adds scheduling and prioritization for work in progress. The three-tier client/server architecture has been shown to improve performance for groups with a large number of users (in the thousands) and improves flexibility when compared to the two-tier approach.

In 3-tier client/server application, the business logic resides in the middle tier, separate from the data and user interface. In this way, processes can be managed and deployed separately from the user interface and the database. Also, 3-tier systems can integrate data from multiple sources.