UNIT 4 DISTRIBUTED DATABASES

Structure

4.0 INTRODUCTION

In a distributed database system, the database is stored on several computers. The computers in a distributed system communicate with each other through various communication media, such as high-speed buses or telephone line. They do not share main memory, nor do they share a clock.

The processors in a distributed system may vary in size and function. They may include small microcomputers, work station, minicomputers, and large general-purpose computer system. These processors are referred to by a number of different names such as sites, nodes, computers, and so on, depending on the context in which they are mentioned. We mainly use the term site, in order to emphasize 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 centralized and distributed database systems is that, in the former, the data resides in one single location, while in the latter, the data resides in several locations. As we shall see, this distribution of data is the cause of many difficulties that will be addressed in this chapter.

4.1 OBJECTIVES

After going through this unit you may able to :

- Differentiate DDBMS and conventional DBMS
- Discuss Network topology for DDBMS
- Discuss advantages and disadvantages of DDBMS
- Distinguish between horizontal and vertial fragmentation.

4.2 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 is several sites. The execution of global transactions requires communication among the sites.

The sites in the 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 configurations are depicted in Figure 1. 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 the failure of some links or sites.

As we shall see, 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 the all Indian states), or over a small geographical area such as a single building or a number of adjacent buildings). The former type of network is referred to as a long-haul network, while the latter is referred to as a local-area network.

Since the sites in long-haul 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 long-haul links are telephone lines, microwave links, and satellite channels. In contrast, since all the sites in local-area networks are close to each other, communication links are of higher speed and lower error rate than their counterparts in long-haul networks. The most common links are twisted pair, baseband coaxial, broadband coaxial, and fiber optics.

Let us illustrate these concepts by considering a banking system consisting of four branches located in four 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 branches of the bank. Suppose that the database systems at the various sites are based on the relational model. Thus, each branch maintains (among others) the relation deposite (Deposit-scheme) where

Deposite-scheme = (branch-name, account-number, customer-name, balance)

site containing information about the four branches maintains the relation branch (Branch-scheme), where

Branch-scheme = (branch-name, assets, branch-city)

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

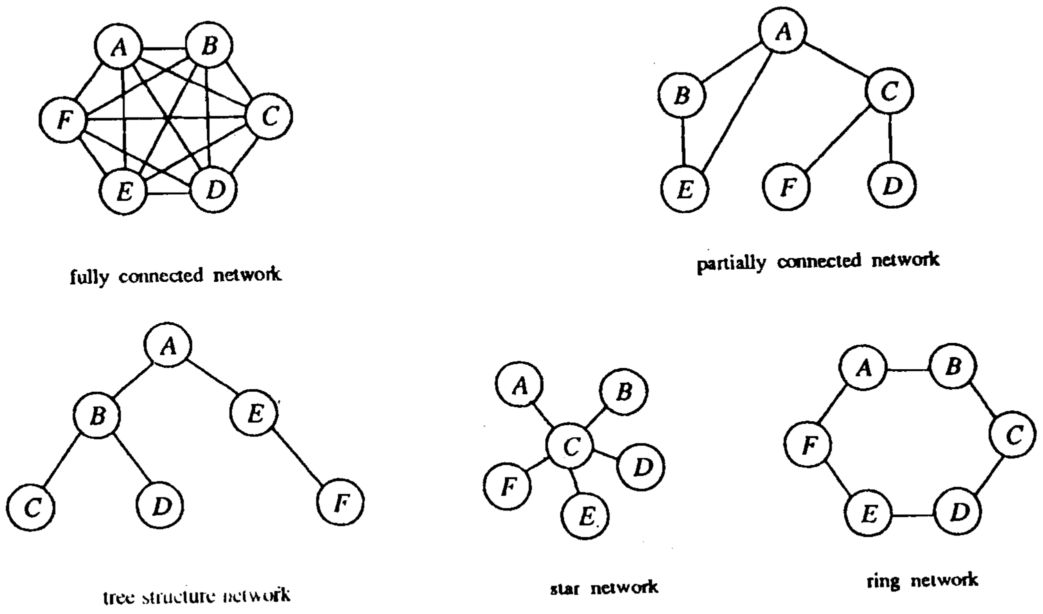

Figure 1 : Network topology

A local transaction is a transaction that accesses accounts in the one single site, at which the transaction was initiated. A global transaction, on the other hand is one which either access 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 $ 50 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 $ 50 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.

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

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

4.3 TRADE-OFFS IN DISTRIBUTING THE DATABASE

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 overhead. In this section, we shall elaborate briefly on each of these.

4.3.1 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

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. For example, in the distributed banking system described in Section 4.2, it is possible for a user in one branch to access data in another branch. Without this capability, a user wishing to transfer funds from one branch to another would have to resort to some external mechanism for such a transfer. This external mechanism would, in effect, be a single centralized database.

The primary advantage to accomplishing data sharing by means of data distribution is that each site is able to retain a degree of control over data stored locally. In a centralized system, the database administrator of the central site controls the database. In a distributed system, there is a global database administrator responsible for the entire system. A part of these responsibilities is delegated to the local database administrator for each site. Depending upon the design of the distributed database system, each local administrator may have a different degree of autonomy is often a major advantage of distributed databases.

Reliability and Availability

If one site fails in distributed system, the remaining sites may be able to continue operating. In particular, if data are replicated in several sites, transaction needing a particular data item may find it in 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 service 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.

Although recovery from failure is more complex in distributed systems than in centralized system, the ability of most of the system to continue to operate despite the failure of one site results in increased availability. Availability is crucial for database systems used for real-time applications. Loss of access to data by, for example, an airline may result in the loss of potential ticket buyers to competitors.

Speedup Query Processing

If a query involves data at several sites, it may be possible to split the query into subqueries that can be executed in parallel by several sites. Such parallel computation allows for faster processing of a user's query. In those cases in which data is replicated, queries may be directed by the system to the least heavily loaded sites.

4.3.2 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 :

- **Software development cost :** It is more difficult to implement a distributed database system and, thus, more costly.
- **Greater potential for bugs :** Since the sites that comprise the distributed system operate in parallel, it is harder to ensure the correctness of algorithms. The potential exists for extremely subtle bugs. The art of constructing distributed algorithms remains an active and important area or research.
- **Increased processing overhead :** The exchange of messages and the additional computation required to achieve intersite coordination is a form of overhead that does not arise in centralized systems.
- In choosing the design for a database system, the designer must balance the advantages against the disadvantages of distribution of data design ranging from fully distributed designs to designs which included large degree of centralization.

4.4 DESIGN OF DISTRIBUTED DATABASES

The principles of database design that we discussed earlier apply to distributed databases as well. In this section, we focus on those design issues that are specific to distributed databases.

Consider a relation that is to be stored in the database. There are several issues involved in storing this relation in the distributed database, including:

- **Replication :** The system maintains several identical replicas (copies) of the relation. Each replica is stored in a different site, resulting in data replication. The alternative to replication is to store only one copy of relation.
- **Fragmentation :** The relation is partitioned into several fragments. Each fragment is stored in a different site.
- **Replication and Fragmentation :** This is a combination of the above two notions. The relation is partitioned into several fragments. The system maintains several identical replicas of each such fragment.

In the following subsections, we elaborate on each of these.

4.4.1 Data Replication

If relation r is replicated, a copy of relation r is stored in two or more sites. In the most extreme case, we have full replication, in which a copy is stored in every site in the system.

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 be found in another site. Thus, the system may continue to process queries involving r despite the failure of one site.
- **Increased parallelism :** In the case where the majority of access to the relation r results in only the reading of the relation, the several sites can process queries involving r in parallel. The more replicas of r there are, the greater the chance that the needed data is found in the site where the transaction is executing. Hence, data replication minimizes movement of data between sites.
- **Increase overhead on update :** The system must ensure that all replicas of a relation r are consistent since 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 overhead. For example, in a banking system, where account information is replicated in various sites, it is necessary that transactions assure that the balance in a particular account agrees in all sites.

In general, replication enhances the performance of read operations and increases the availability of data to read transactions. However, update transactions incur greater overhead. The problem of controlling concurrent updates by several transactions to replicated data is more complex than the centralized approach to concurrency control. We may simplify the management of replicas of relation r by choosing one of them as the primary copy of r. For example, in a banking system, an account may be associated with the site in which the account has been opened. Similarly, in an airline reservation system, a flight may be associated with the site at which the flight originates.

4.4.2 Data Fragmentation

If the relation r if fragmented, r is divided into a number of fragments $r_1, r_2, \ldots, r_n$. These fragments contain sufficient information to reconstruct the original relation r. As we shall see, this reconstruction can take place through the application of either the union operation or a special type of join operation on the various fragments. There are two different schemes for fragmenting a relation: horizontal fragmentation and vertical fragmentation. Horizontal fragmentation splits the relation by assigning each tuple of r to one or more fragments. Vertical fragmentation splits the relation by decomposing the scheme R of relation r in a special way that we shall discuss. These two schemes can be applied successively to the same relation, resulting in a number of different fragments. Note that some information may appear in several fragments.

Below we discuss the various ways for fragmenting a relation. We shall illustrate these by fragmenting the relation deposit, with scheme:

Deposite-scheme = (branch-name, account-name, customer-name, balance)

The relation deposite (deposite-scheme) is shown in Figure 2.

branch-name	*account number*	*customer-name*	*balance*
Bombay	305	Lowman	500
Bombay	226	Camp	336
Delhi	117	Camp	205
Delhi	402	Khan	10000
Bombay	155	Khan	62
Delhi	408	Khan	1123
Delhi	639	Green	750

Figure 2 : Sample deposite relation

Horizontal Fragmentation

The relation r is partitioned into a number of subsets, $r_1, r_2, \ldots, r_n$. Each subset consists of a number of tuples of relation r. Each tuple of relation r must belong to one of the fragments, so that the original relation can be reconstructed, if needed.

A fragment may be defined as a selection on the global relation r. That is, a predicate P1 is used to construct fragment ri as follows:

$$r_i = \sigma_{P_i}(r)$$

The reconstruction of the relation r can be obtained by taking the union of all fragments, that is,

$$r = \bigcup_{i=1}^{n} r_i$$

To illustrate this, suppose that the relation r is the deposite relation of Figure 2. This relation can be divided into n different fragments, each of which consists of tuples of accounts belonging to a particular branch. If the

branch-name	*account-number*	*customer-name*	*balance*
Bombay	305	Lowman	500
Bombay	226	Camp	336
Bombay	155	Khan	62

(a)

branch-name	*account-number*	*customer-name*	*balance*
Delhi	177	Camp	205
Delhi	402	Khan	10000
Delhi	408	Khan	1123
Delhi	639	Green	750

(b)

Figure 3 : Horizontal fragmentation of relation of *deposit*

banking system has only two branches, Bombay and Delhi, then there are two different fragments:

$$deposit_1 = \sigma_{branch\text{-}name\ =\ \text{"Hillside"}}(deposit)$$
$$deposit_2 = \sigma_{branch\text{-}name\ =\ \text{"Valleyview"}}(deposit)$$

these two fragments are shown in Figure 3. Fragment deposit 1 is stored in the Bombay site. Fragment deposit 2 is stored in the Delhi site.

In our example, the fragments are disjoint. By changing the selection predicates used to construction the fregments, we may have a particular tuple of r appear in more than one of the r_1. This is a form of data replication about which we shall say more at the end of this section.

Vertical Fragmentation

In its most simple form, vertical fragmentation is the same as decomposition. Vertical fragmentation of r(R) involves the definition of several subsets $R_1, R_2, \ldots, R_n$ of R such that $r_i = r$.

Each fragment ri of r is defined by:

$$r_i = \Pi_{R_i}(r)$$

relation r can be reconstructed from the fragments by taking the natural join:

$$r = r_1 \infty \ r_2 \infty \ r_3 \infty \ \ldots \infty \ r_n$$

branch-name	account-number	customer-name	balance	tuple-id
Bombay	305	Lowman	500	1
Bombay	226	Camp	336	2
Delhi	177	Camp	205	3
Delhi	402	Kahn	10000	4
Bombay	155	Kahan	62	5
Delhi	408	Khan	1123	6
Delhi	639	Green	750	7

Figure 4 : The deposite relation of Figure 4.2 with tuple-ids

More generally, vertical fragmentation is accomplished by adding a special attribute called a tuple-id to the scheme R. A tuple-id is a physical or logical address for a tuple. Since each tupe in r must have a unique address, the tuple-id attribute is a key for the augmented scheme.

In Figure 4, we show the relation deposit', the deposit relation of Figure 2 with tuple-ids added. Figure 5 shows a vertical decomposition of the scheme Deposit-scheme tuple-id into:

Deposit-scheme-3 = (branch-name, customer-name, tuple-id)
Deposit-scheme-4 = (account-number, balance, tuple-id)

The two relation shown in figure 5 result from computing:

$$deposit_3 = \Pi_{Deposit\text{-}scheme\text{-}3}(Deposit')$$
$$deposit_4 = \Pi_{Deposit\text{-}scheme4\text{-}}(Deposit')$$

branch-name	*customer-name*	*tuple-id*
Bombay	Lowman	1
Bombay	Camp	2
Delhi	Camp	3
Delhi	Khan	4
Bombay	Khan	5
Delhi	Khan	6
Delhi	Green	7

(a)

account-number	*balance*	*tuple-id*
305	500	1
226	336	2
177	205	3
402	10000	4
155	62	5
408	1123	6
639	750	7

(b)

Figure 5 : Vertical fragmentation of relation *deposit*

To reconstruct the original deposit relation from the fragments, we compel

$$\Pi_{Deposit\text{-}scheme}\,(deposit_3 \infty\ deposit_4)$$

Note that the expression

$$deposit_3 \infty\ deposit_4$$

is special form of natural join. The join attribute is tuple-id. Since the *tupled-id* value represents an address, it is possible to pair a tuple of deposit 3 with the corresponding tuple of deposit 4 by using the address given by the tuple-id value. This address allows direct retrieval of the tuple without the need for an index. Thus, this natural join may be computed much more efficiently than typical natural joins.

Although the *tuple-id* attribute is important in the implementation of vertical partitioning, it is important that this attribute not be visible to users. If users are given access to tuple-ids, it becomes impossible for the system to change tuple addresses. Furthermore, the accessibility of internal addresses violates the notion of data independence, one of the main virtues of the relational model.

4.5 SUMMARY

A distributed database system consists of a collection of sites, each of which maintain a local database system. Each site is able to process local transaction, those transaction 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 n 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 overhead. The primary disadvantage of distributed database systems in 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.

4.6 FURTHER READING

Henry F. Korth, Abraham Silberschatz, *Database System Concepts*, McGraw Hill International Edition.