6. Design Issues of Distributed DBMS

6.1 Distributed Database Design

One of the main questions that is being addressed is how database and the applications that run against it should be placed across the sites. There are two basic alternatives to placing data: *partitioned (or no-replicated)* and *replicated*. In the partitioned scheme the database is divided into a number of disjoint partitions each of which is placed at different site. Replicated designs can be either *fully replicated* (also called *fully duplicated*) where entire database is stored at each site, or *partially replicated* (or *partially duplicated*) where each partition of the database is stored at more than one site, but not at all the sites. The two fundamental design issues are fragmentation, the separation of the database into partitions called *fragments*, and distribution, the optimum distribution of fragments.
The research in this area mostly involve mathematical programming in order to minimize the combined cost of storing the database, processing transactions against it, and message communication among site.

6.2 Distributed Directory Management

A directory contains information (such as descriptions and locations) about data items in the database. Problems related to directory management are similar in nature to the database placement problem discussed in the preceding section. A directory may be global to the entire DDBS or local to each site; it can be centralized at one site or distributed over several sites; there can be a single copy or multiple copies.

6.3 Distributed Query Processing

Query processing deals with designing algorithms that analyze queries and convert them into a series of data manipulation operations. The problem is how to decide on a strategy for executing each query over the network in the most cost-effective way, however cost is defined. The factors to be considered are the distribution of data, communication cost, and lack of sufficient locally-available information. The objective is to optimize where the inherent parallelism is used to improve the performance of executing the transaction, subject to the above-mentioned constraints.

6.4 Distributed Concurrency Control

Concurrency control involves the synchronization of access to the distributed database, such that the integrity of the database is maintained. It is, without any doubt, one of the most extensively studied problems in the DDBS field. The concurrency control problem in a distributed context is somewhat different that in a centralized framework. One not only has to worry about the integrity of a single database, but also about the consistency of multiple copies of the database. The condition that requires all values of multiple copies of every data item to converge to the same value is called *mutual consistency*. Let us only mention that the two general classes are *pessimistic,* synchronizing the execution of the user request before the execution starts, and *optimistic,* executing requests and then checking if the execution has compromised the consistency of the database. Two fundamental primitives that can be used with both approaches are *locking,* which is based on the mutual exclusion of access to data items, and *timestamping,* where transactions executions are ordered based on timestamps. There are variations of these schemes as well as hybrid algorithms that attempt to combine the two basic mechanisms.

6.5 Distributed Deadlock Management

The deadlock problem in DDBSs is similar in nature to that encountered in operating systems. The competition among users for access to a set of resources (data, in this case) can result in a deadlock if the synchronization mechanism is based on locking. The well-known alternatives of prevention, avoidance, and detection/recovery also apply to DDBSs.

6.6 Reliability of Distributed DBMS

It is important that mechanisms be provided to ensure the consistency of the database as well as to detect failures and recover from them. The implication for DDBSs is that when a failure occurs and various sites become either inoperable or inaccessible, the databases at the operational sites remain consistent and up to date. Furthermore, when the computer system or network recovers from the failure, the DDBSs should be able to recover and bring the databases at the failed sites up-to-date. This may be especially difficult in the case of network partitioning, where the sites are divided into two or more groups with no communication among them.

6.7 Replication

If the distributed database is (partially or fully) replicated, it is necessary to implement protocols that ensure the consistency of the replicas, i.e. copies of the same data item have the same value. These protocols can be *eager* in that they force the updates to be applied to all the replicas before the transactions completes, or they may be *lazy* so that the transactions updates one copy (called *the master*) from which updates are propagated to the others after the transaction completes.