**Technical problem areas**

1. **Distributed Database Design**

It addresses how the database and the applications that run against it should be placed across the sites. There are two basic alternatives to placing data: partitioned (or non-replicated) and replicated. In the partitioned scheme the database is divided into a number of disjoint partitions each of which is placed at a different site. Replicated designs can be either fully replicated (also called fully Duplicated) where the 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.

1. **Distributed Directory Management**

A directory contains information (such as descriptions and locations) about data

items in the database.

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.

1. **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. The factors to be considered are the distribution of data, communication costs, and lack of sufficient locally-available information.

1. **Distributed Concurrency Control**

Concurrency control involves the synchronization of accesses to the distributed database, such that the integrity of the database is maintained. The concurrency control problem in a distributed context is somewhat different than 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.

1. **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.

1. **Reliability of Distributed DBMS**

one of the potential advantages of distributed systems is improved reliability and availability. This, however, is not a feature that comes automatically. 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.

1. **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 transaction completes, or they may be lazy so that the transaction updates one copy (called the master) from which updates are propagated to the others after the transaction completes.

1. **Heterogeneous databases**

If there is no homogeneity among the DBs at various sites either in terms of the way data is logically structured (data model) or in terms of the access mechanisms

(data language), it becomes necessary to provide translation mechanisms between database systems. translation mechanisms involves translating data manipulation instruction.

1. **Relationship among Problems**