Chapter I
Introduction

1.1 Distributed Database

A distributed database is a collection of multiple, logically interrelated databases distributed over a computer network (figure 1.1). A distributed database management system (distributed DBMS) is then defined as the software system that permits the management of the distributed database and makes the distribution transparent to the users. Sometimes distributed database system (DDBS) is used to refer jointly to the distributed database and the distributed DBMS. Distribution leads to increased complexity in the system design and implementation. A DDBS must be able to provide some additional functionality, as given below, compared to a central DBMS (Ozsu and Valduriez, 2011).

- To access remote sites and transmit queries and data among the various sites via a communication network.
- To keep track of the data distribution and replication in the DDBMS catalog.
- To devise execution strategies for queries and transactions that access data from more than one site.
- To decide on which copy of a replicated data item to access.
- To maintain the consistency of copies of a replicated data item.
- To maintain the global conceptual schema of the distributed database.
- To recover from individual site crashes and from new types of failures such as failure of a communication link.
1.1.1 Features of DDBS

- **Transparent Management of Distributed and Replicated Data**

  Transparency refers to separation of the higher-level semantics of a system from lower-level implementation issues. In other words, a transparent system hides the implementation details from users. The advantage of a fully transparent DBMS is the high level of support that it provides for the development of complex applications.

- **Reliability Through Distributed Transactions**

  Distributed DBMSs are intended to improve reliability since they have replicated components and, thereby eliminate single points of failure. The failure of a single site, or the failure of a communication link which makes one or more sites unreachable, is not sufficient to bring down the entire system. In the case of a distributed database, this means that some of the data may be unreachable, but with proper care, users may be permitted to access other parts of the distributed database. The proper care comes in the form of support for distributed transactions and application protocols.
• **Improved Performance**

The case for the improved performance of distributed DBMSs is typically made based on two points. First, a distributed DBMS fragments the conceptual database, enabling data to be stored in close proximity to its points of use (also called data localization). This has two potential advantages: firstly, since each site handles only a portion of the database, contention for CPU and I/O services is not as severe as for centralized databases and secondly, localization reduces remote access delays that are usually involved in wide area networks.

• **Easier System Expansion**

In a distributed environment, it is much easier to accommodate increasing database sizes. Major system overhauls are seldom necessary; expansion can usually be handled by adding processing and storage power to the network. Obviously, it may not be possible to obtain a linear increase in power, since this also depends on the overhead of distribution. However, significant improvements are still possible.

1.1.2 Types of Distributed Databases

**Homogeneous Distributed Database**

In a homogeneous distributed database

- All sites have identical software and are aware of each other and agree to cooperate in processing user requests.
- Each site surrenders part of its autonomy in terms of right to change schemas or software.
- Appears to user as a single system.
Heterogeneous Distributed Database

In a heterogeneous distributed database

- Different sites may use different schemas and software.
- Difference in schema is a major problem for query processing.
- Difference in software is a major problem for transaction processing.
- Sites may not be aware of each other and may provide only limited facilities for cooperation in transaction processing.

1.1.3 Components of a Distributed DBMS

The detailed components of a distributed DBMS are given (figure 1.2) in this section. One component handles the interaction with users, and another deals with the storage. The first major component, which is called the user processor consists of four elements (Ozsu and Valduriez, 2011):

- The user interface handler is responsible for interpreting user commands as they come in, and formatting the result data as it is sent to the user.
- The semantic data controller uses the integrity constraints and authorizations that are defined as part of the global conceptual schema to check if the user query can be processed. This component is also responsible for authorization and other functions.
- The global query optimizer and decomposer determines an execution strategy to minimize a cost function, and translate the global queries into local ones using the global and local conceptual schemas as well as the global directory. The global query optimizer is responsible for generating the best strategy to execute distributed join operations.
- The distributed execution monitor coordinates the distributed execution of a user request. The execution monitor is also called the distributed transaction manager.
executing queries in a distributed fashion, the execution monitors at various sites may communicate with one another.

Figure 1.2. Components of a distributed DBMS.
The second major component of a distributed DBMS is the data processor and consists of three elements:

- The local query optimizer, which actually acts as the access path selector, is responsible for choosing the best access path to access any data item.
- The local recovery manager is responsible for making sure that the local database remains consistent even when failures occur.
- The run-time support processor physically accesses the database according to the physical commands in the schedule generated by the query optimizer. The run-time support processor is the interface to the operating system and contains the database buffer (or cache) manager, which is responsible for maintaining the main memory buffers and managing the data accesses.

1.2 Need of Replication

Replication is the key characteristic in improving the availability of data in distributed systems. Replicated data is stored at multiple server sites so that it can be accessed by the users even when some of the copies are not available due to server or site failures. A Major restriction to using replication is that replicated copies must behave like a single copy, i.e. mutual consistency as well internal consistency must be preserved, synchronization techniques for replicated data in distributed database systems are needed in order to increase the degree of consistency and to reduce the possibility of transactions rollback. The major advantages of replication are as follows (Sameh et al., 2005):

- Improve data availability and scalability.
- Provide fail safe backup.
- Provide load spreading.
• Provide disconnected operations without sacrificing performance, security and application usability.

Some examples of usage of replication are given below.

• Enhance availability of data on different locations by distributing data by providing a mirror.

• Split workload to different servers by distributing data by providing a mirror.

• Merge data from different locations (consolidation).

• Provide a mirror for disaster recovery.

• Emulate a multi-company application by sharing a limited number of tables through replication.

• Facilitate Mobile Solutions: Applications can be made independent of network availability (off line).

• Application Integration: Connect different applications without a common interface.

• Provide Data Transformation: Change data in a controlled way.

• Audit or analyze database usage.

1.3 Database Replication

Replication in database systems is done mainly for performance reasons. The objective is to access data locally in order to improve response times and eliminate the overhead of having to communicate with other sites. Database replication is the process of copying a database from one database server to another server, and then keeping the two copies in synchronization, so that they behave as near as possible. Replication
copies data or changes of data from one database to another. Accessing the first database or the second, it does not matter because they are the same, i.e. they are in synchronized. Replication is not limited to two databases; any number of databases can take place into the replication model. It does not mean that the replicated databases are exactly the same, because replication can be configured that only a subset of tables or columns or rows will be replicated, i.e. partial replication. However, replication ensures that those specific configured objects are kept in sync between the different databases. With replication, maintenance of multiple copies of data to different locations is required. Because each location has its own database, each location works essentially locally with maximum performance and security; databases don't have to be accessed outside the local network or workstation.

Replication is a process which needs no interaction from the user. In that way, replication is a low level process, which should be completely invisible for the application users. This means that replication by copying databases (through backup & restore) or by using export and import scripts are not viable replication mechanisms. In essence, replication is the method of sharing information so as to make sure of consistency between redundant resources, such as software or hardware components, to improve reliability, fault-tolerance, or accessibility. Figure 1.3 shows the basic replication model. In this model user or client does not know that multiple physical copies of data exists. Data replication is a combination of database and distributed system (Moiz et al., 2011). Using replication, any directory tree or sub-tree (stored in its own database) can be copied between servers. The Directory Server that holds the master copy of the information automatically copies any updates to all replicas. A computational job is typically replicated in space, i.e. executed on separate devices, or
it could be replicated in time, if it is executed repeatedly on a single device. The access to a replicated entity is typically uniform with access to a single, non-replicated entity. The replication itself should be transparent to an external user. The important features of database replication are discussed below:

- **Database Locality**

  This feature of database replication maintains the database locally so that geographically far distance users can access data with high speed. These users can access data from local servers instead of far distance servers because data access speed will be much higher compared to a distant area network.

![Figure 1.3. Basic data replication model.](image)

- **Performance**

  Database replication typically focuses on improving both read and write performances. When an application is widely used across a large network but the database is stored at a single server, the database server can be a bottleneck of that system and the whole system slows down, i.e. slow response time and low request
throughput capacity. Multiple replicas offer the system the data in parallel and as a result there is an increase in performance.

- **Availability and Fault Tolerance**

  High availability of database requires low downtime of a database system. In a database system there exist two downtimes the first is planned and the other is unplanned. Planned downtime is incurred during the maintenance operation of all the software and hardware. Unplanned downtime can strike at any time and it is due to predictable or unpredictable failures such as hardware failures, software bugs, human error, etc. Downtime is usually the primary optimization area of database replication to increase the database availability. If a database item is stored at a single server and that server does not respond or is down or it might have crashed. In such cases database replication is the solution of this problem, which also has the capability to provide a fault tolerant database system. The replica of a database server can provide the data items to the users when a server failure occurs. Database replication can be done in at-least the following three different ways (Stockinger et al., 1999).

**1.3.1 Snapshot Replication**

Data on one database server is plainly copied to another database server, or to another database on the same server. The snapshot replication method functions by periodically sending data in bulk format. Usually it is used when the subscribing servers can function in read-only environment (figure 1.4), and also when the subscribing server can function for some time without updated data. Functioning without updated data for a period of time is referred to as latency. Snapshot replication works by reading the published database and creating files in the working folder on the distributor. These
files are called snapshot files and contain the data from the published database as well as some additional information that will help to create the initial copy on the subscription server (Linsenbardt et al, 2000). Snapshot replication is helpful when:

- Data is mostly static and does not change often.
- It is acceptable to have copies of data that are out of date for a period of time.

![Figure 1.4. Schematic of the snapshot replication.](image)

**1.3.2 Merger Replication**

Data from two or more databases is combined into a single database. Merge replication is the process of distributing data from Publisher to Subscribers, allowing the Publisher and Subscribers to make updates while connected or disconnected, and then merging the updates between sites when they are connected (figure 1.5). Merge
replication allows various sites to work autonomously and at a later time merge the updates into a single, uniform result. Merge replication includes default and custom choices for conflict resolution. When a conflict occurs, a resolve is invoked by the Merge Agent and determines which data will be accepted and propagated to other sites (Moiz et al., 2011). Merge Replication is helpful when:

- Multiple Subscribers need to update data at various times and propagate those changes to the Publisher and to other Subscribers.
- Subscribers need to receive data, make changes offline, and later synchronize changes with the Publisher and other Subscribers.

1.3.3 Transactional Replication

It enables users to obtain a complete initial copy of a database and then obtain periodic updates as data changes. In transactional replication, each committed transaction is replicated to a subscriber as it occurs. The replication process cannot be controlled so that it will accumulate transactions and send them at timed intervals, or transmit all changes as they occur. This type of replication is used in environments having a lower degree of latency and higher bandwidth connections (Moiz et al., 2011). Transactional replication requires a continuous and reliable connection (figure 1.5), because the Transaction Log will grow quickly if the server is unable to connect for replication and might become unmanageable. Transactional replication begins with a snapshot that sets up the initial copy. That copy is then later updated by the copied transactions. There could be a choice that how often to update the snapshot, or choose not to update the snapshot after the first copy. Once the initial snapshot has been copied, transactional replication uses the Log Reader agent to read the Transaction Log of the published database and stores new transactions in the distribution Database. The
Distribution agent then transfers the transactions from the publisher to the subscriber (Moiz et al., 2011). Transactional replication is helpful when:

- The incremental changes are to be propagated to Subscribers as they occur.
- Subscribers are reliably and frequently connected to the Publisher.

Figure 1.5. Schematic of the merge replication.

To cope with the complexity of replication, the notion of group (of servers) and group communication primitives are required (Birman et al., 1993). The notion of group acts as a logical addressing mechanism, allowing a client to ignore the degree of replication and the identity of the individual server processes of a replicated service.
Group communication primitives provide one-to-many communication with various powerful semantics. These semantics hide much of the complexity of maintaining the consistency of replicated servers. The two main group communication primitives are Atomic Broadcast (or ABCAST) and View Synchronous Broadcast (or VSCAST) (A. Schiper et al. 1993). Informal definitions of these primitives are given below.

**Atomic Broadcast (ABCAST)**

Atomic Broadcast provides atomicity and total order. Let m and m' be two messages that are ABCAST to the same group g of servers. The atomicity property ensures that if one member of g delivers m (respt. m'), then all (not crashed) members of g eventually deliver m (respt. m'). The order property ensures that if two members of g deliver both m and m', they deliver them in the same order.

**View Synchronous Broadcast (VSCAST)**

The definition of View Synchronous Broadcast is more complex. It is defined in the context of a group g, and is based on the notion of a sequence of views v₀(g), v₁(g), . . . , vᵢ(g), . . . of group g. Each view vᵢ(g) defines the composition of the group at same time t, i.e. the members of the group that are perceived as being correct at time t. Whenever a process p in some view vᵢ(g) is suspected to have crashed, or some process q wants to join, a new view vᵢ₊₁(g) is installed, which reflects the membership change. Roughly speaking, VSCAST of message m by some member of the group g currently in view vᵢ(g) ensures the following property: if one process p in vᵢ(g) delivers m before installing view vᵢ₊₁(g), than no process installs view vᵢ₊₁(g) before having first delivered m.
1.4 Synchronous and Asynchronous Replication

The capabilities and performance characteristics vary from one type of replication to another. The updates at a primary site can be propagated to the secondary sites by Synchronous and Asynchronous methods ([Moiz et al., 2011](#)). These methods are discussed in this section with their pros and cons ([Brouwer, P., 2011](#)).

1.4.1 Synchronous Replication

In synchronous replication method, disk writes are replicated to the target disk within the boundary of same transaction as writes to the source disk. All disk writes must occur and be acknowledged on both the source and target disks before a host can move on to the next disk write. Due to this behavior, application performance requirements must be carefully considered when deploying synchronous replication. Distance from source side typically plays a role in determining if Synchronous replication is implemented ([Moiz et al., 2011](#)). On the positive side, synchronous replication continuously provides a real-time copy of replicated data, allowing for a complete recovery at all times. Synchronous Replication works on the principle of Two-Phase commit protocol. In a two-phase commit protocol, when an update to the master database is requested, the master system connects to all other systems (slave databases), locks those databases at the record level and then updates them simultaneously. If one of the slaves is not available, the data may not be updated. The consistency of data is preserved; however it requires availability of all sites at the time of propagation of updates ([Brouwer, P., 2011](#)).

**Advantages**

- Can provide real-time data copy at remote location.
- Consistency is guaranteed.
- Consistency is supported by internal distributed transaction mechanism.
- No visible propagation latency.

**Disadvantage**

- Performance can be adversely impacted.
- Higher bandwidth requirements.
- May not be cost-effective.
- Target database availability impacts source/primary database transactions.
- May require manual involvement during distributed transactions (two-phase commit) failures.
- Replication into heterogeneous databases requires external distributed transaction broker.
- It may significantly slow down transactions in the source database if a target database is remote.

**1.4.2 Asynchronous Replication**

In asynchronous method, data is replicated to the target storage without requiring an acknowledgment before additional writes can occur. This improves performance but introduces more risk because the remote copy is not always current while the data is being transmitted across the network to the remote location. Asynchronous can also be implemented as point-in-time snapshots where the data is completely copied to the target storage at predetermined intervals. There exist two variations of Asynchronous replication i.e. Periodic and Aperiodic. In Periodic replication, the updates to data items are done at specific intervals and in Aperiodic replication the updates are propagated...
only when necessary (Moiz et al., 2011). The time at which the copies are inconsistent is an adjustable parameter which is application dependent.

**Advantages**

- Can provide near real-time data copy at remote location.
- More cost-effective.
- Creates little impact on local database.
- Does not slow down transactions/DMLs in the source database.
- Source database is not dependent on availability of a target database.

**Disadvantages**

- Some data loss may occur.
- Point-in-time consistency between primary and target databases is not guaranteed.
- Data is synchronized with some latency.
- Requires to store data changes from primary database.

1.5 Design Issues in Distributed Database Replication

In replicated database systems, copies of the data items can be stored at multiple servers and at a number of places. The potential of data replication for high data availability and improved read performance is crucial to distributed real time database system. In contrast, data replication introduces its own problems. Access to a data item is no longer controlled exclusively by a single server; instead, the access control is distributed across the servers each storing the copy of the data item. It is necessary to ensure that mutual consistency of the replicated data is provided; it
must fulfill the ACID properties of database (Kenme et al., 1999).

While data copying can provide users with local and much quicker access to data, the problem is to provide these copies to users so that the overall systems operate with the same integrity and management capacity that is available within a centralized model. Managing a replicated data is significantly more complicated than running against a single location database. It deals with all of the design and implementation issues of a single location and additionally with complexity of distribution, network latency and remote administration. A proven architectural approach should be followed to create nearly-identical copies of the data and to manage the integrity of the copies if they can be updated at both the source and target within a replication interval (Raynal et al., 1996). Various important issues related to data replication in distributed databases are discussed below.

- **Replication Set Size**

  It decides whether to replicate an entire table, a subset of a table or data from more than one table. This is a tradeoff among the amount of data that changes, the overall table size, and the complexity of the link.

- **Transmission Volume**

  Choose the right amount of data to transmit. The decision between sending all changes for any one row, or just the net effect of all the changes, is a key one.

- **Replication Set Data Changes at the Target**

  If these have to occur and if the source wants to see the changes, then try to
make the changes naturally non-conflicting to avoid the need for conflict detection and resolution.

- **Replication Frequency**
  Decides the appropriate timing of the replication for the requirements and optimizes the use of computing resources.

- **Replication Unit**
  A replication set consists of a group of replication units. Identify the unit of data that will be transmitted from the source to the target. In the extreme requirements, this will be a transaction as it has been executed on the source. A less precise but easier to achieve requirement is to move a changed row. For environments with a high risk of conflicts, it can also be an individual change in a cell within a record.

- **Initiator**
  Decides whether the source pushes the data or the target pulls it, and makes sure that throughout a replication topology, these decisions do not cause later replication links to have problems meeting their operational requirements.

- **Locking Issues**
  Verify that the locking impact of the replication on the source can be accepted. If not, verify that a slight decrease in consistency at a point in time is acceptable for the targets so that lock conflict can be avoided.
• **Security**

Ensure that the replicated data is treated with the right level of security at the target given the source security conditions. Also, verify that a replication link is secure enough in the overall topology requirements.

• **Key Updates**

Verify whether the source allows updates to the key of records belonging to the replication set. If so, special care must be taken for a consistent replication of such operations. Key updates are SQL updates to the columns of the physical key within a replication set. Such key updates must be handled specially by the replication.

• **Referential Integrity**

Verify whether the target has implemented referential integrity. If so, rules are needed to prevent changes from the replication link being applied twice if the change triggers a target change in another replicated table.

• **Bandwidth**

Low bandwidth between data nodes can slow data node recovery. In normal operation, the available bandwidth can limit the maximum system throughput. If link saturation causes latency on individual links to increase, then node failures, and potentially cluster failure could occur.

• **Latency and Performance**

Synchronously committing transactions over a wide area increases the latency of operation execution and commit, therefore individual operations are slowed. To
maintain the same overall throughput, higher client concurrency is required. With the same client concurrency level, throughput will decrease relative to a lower latency configuration.

- **Data Loss**

  Speedy local replication, including synchronous and near synchronous methods minimize data loss due to storage failures and configuration errors. Fully synchronous replication is not needed for most systems even at the local level but replication does need to be quite fast to ensure local replicas are up-to-date. One of the big issues for avoiding local data loss is to configure systems carefully.

- **Network Performance Management**

  Managing performance of networks involves optimizing the way networks function in an effort to maximize capacity, minimize latency and offer high reliability regardless of bandwidth available and occurrence of failures. Network performance management consists of tasks like measuring, modeling, planning and optimizing networks to ensure that they carry traffic with the speed, capacity and reliability that is expected by the applications using the network or required in a particular scenario.

- **QoS Guarantees**

  The term Quality of Service, in the field of networking, refers to control procedures that can provide a guaranteed level of performance to data flows in accordance to requests from an application or user of the network. A network that provides QoS usually agrees on a traffic contract with an application and reserves a finite capacity in the network nodes, based on the contract, during the session.
establishment phase. While the session is in progress, the network strives to adhere to the contract by monitoring and ensuring that the QoS guarantees are met. The reserved capacities are released subsequently after the session.

1.6 Justification

Database replication has been used for many years very successfully. Now as the internet traffic is growing rapidly, database replication is a complex activity, dependent on the various network parameters. In database replication generally larger size files are replicated, hence they occupy the network for a while. Therefore the parameters like throughput and latency are very important. Database management systems are among the most important software systems driving the information age. In many Internet applications, a large number of users that are geographically dispersed may routinely query and update the same database. In this environment, a centralized database is exposed to several significant risks as performance degrades due to high server load, data-loss risk due to server crashes, high latency for queries issued by remote clients, and availability issues due to lack of network connectivity or server downtime. The apparent solution to these problems would be to replicate the database server on a set of peer servers. In such a system queries can be answered by any of the servers, without any additional communication; however, in order for the system to remain consistent, all the transactions that update the database need to be disseminated and synchronized at all the replicas.
1.7 Objectives

1. To analyze the existing replication control protocols for distributed databases.

2. To design and develop an efficient replication control protocol for a distributed database and implement the same.

3. To evaluate the performance of the proposed protocol.