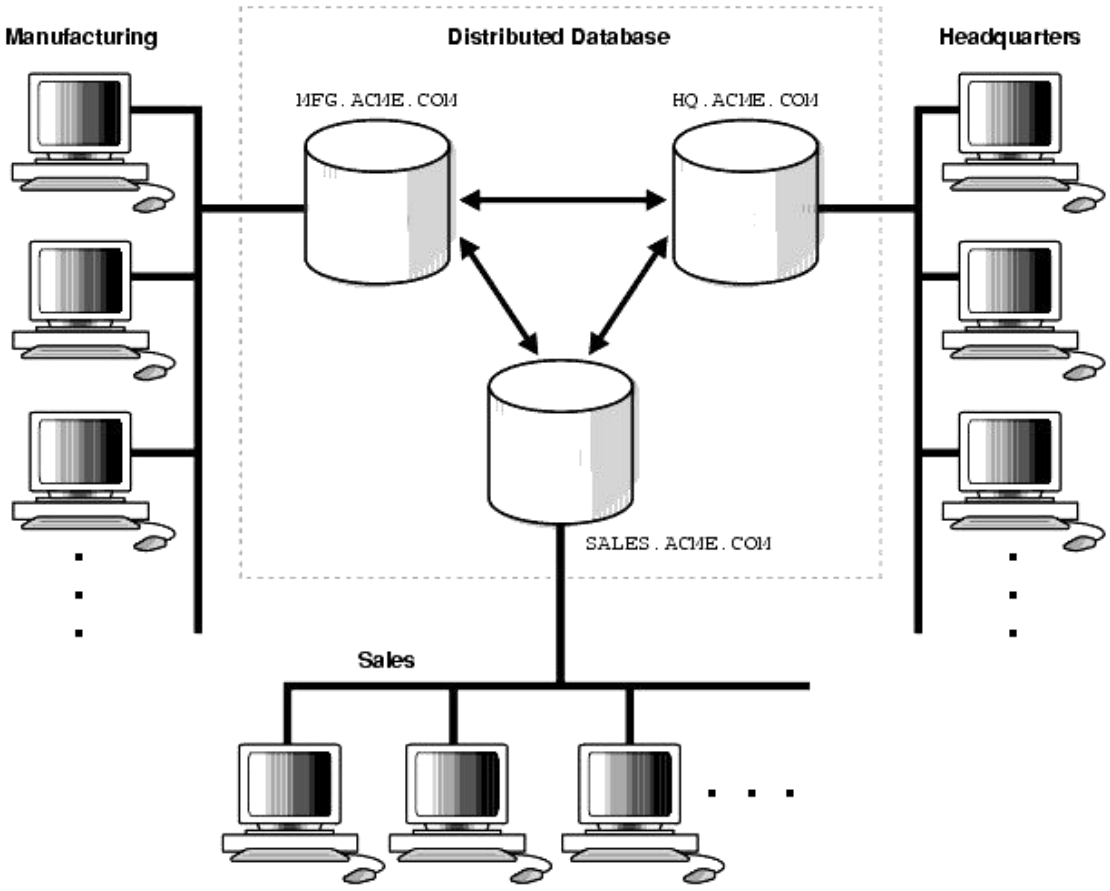


Distributed Database

A distributed database is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network.

Architecture

A database user accesses the distributed database through:

Local applications: Applications which do not require data from other sites.

Global applications: Applications which do require data from other sites.

A **homogeneous** distributed database has identical software and hardware running all databases instances, and may appear through a single interface as if it were a single database.

A **heterogeneous** distributed database may have different hardware, operating systems, database management systems, and even data models for different databases.

Distributed Data Storage

A distributed data store is a computer network where information is stored on more than one node, often in a replicated fashion. It is usually specifically used to refer to either a distributed database where users store information on a number of nodes, or a computer network in which users store information on a number of peer network nodes.

Distributed Transaction

A distributed transaction includes one or more statements that, individually or as a group, update data on two or more distinct nodes of a distributed database. A distributed transaction is a type of transaction with two or more engaged network hosts. Generally, hosts provide resources, and a transaction manager is responsible for developing and handling the transaction.

Commit Protocols

In a distributed system, the transaction manager should convey the decision to commit to all the servers in the various sites where the transaction is being executed and uniformly enforce the decision. When processing is complete at each site, it reaches the partially committed transaction state and waits for all other transactions to reach their partially committed states. When it receives the message that all the sites are ready to commit, it starts to commit. In a distributed system, either all sites commit or none of them does.

The different distributed commit protocols are −

- **One-phase commit**
- **Two-phase commit**
- **Three-phase commit**

Distributed One-phase Commit

Distributed one-phase commit is the simplest commit protocol. Let us consider that there is a controlling site and a number of slave sites where the transaction is being executed. The steps in distributed commit are −

- After each slave has locally completed its transaction, it sends a “DONE” message to the controlling site.
- The slaves wait for “Commit” or “Abort” message from the controlling site. This waiting time is called window of vulnerability.

- When the controlling site receives “DONE” message from each slave, it makes a decision to commit or abort. This is called the commit point. Then, it sends this message to all the slaves.
- On receiving this message, a slave either commits or aborts and then sends an acknowledgement message to the controlling site.

Distributed Two-phase Commit

Distributed two-phase commit reduces the vulnerability of one-phase commit protocols. The steps performed in the two phases are as follows −

Phase 1: Prepare Phase

- After each slave has locally completed its transaction, it sends a “DONE” message to the controlling site. When the controlling site has received “DONE” message from all slaves, it sends a “Prepare” message to the slaves.
- The slaves vote on whether they still want to commit or not. If a slave wants to commit, it sends a “Ready” message.
- A slave that does not want to commit sends a “Not Ready” message. This may happen when the slave has conflicting concurrent transactions or there is a timeout.

Phase 2: Commit/Abort Phase

- After the controlling site has received “Ready” message from all the slaves −
  - The controlling site sends a “Global Commit” message to the slaves.
  - The slaves apply the transaction and send a “Commit ACK” message to the controlling site.
  - When the controlling site receives “Commit ACK” message from all the slaves, it considers the transaction as committed.
- After the controlling site has received the first “Not Ready” message from any slave −
  - The controlling site sends a “Global Abort” message to the slaves.
  - The slaves abort the transaction and send a “Abort ACK” message to the controlling site.
  - When the controlling site receives “Abort ACK” message from all the slaves, it considers the transaction as aborted.

Distributed Three-phase Commit

The steps in distributed three-phase commit are as follows −

Phase 1: Prepare Phase

- The steps are same as in distributed two-phase commit.

Phase 2: Prepare to Commit Phase

- The controlling site issues an “Enter Prepared State” broadcast message.
- The slave sites vote “OK” in response.

Phase 3: Commit / Abort Phase

- The steps are same as two-phase commit except that “Commit ACK”/”Abort ACK” message is not required.

Concurrency Control in Distributed Systems

There are various approaches for concurrency control in distributed systems.

Distributed Two-phase Locking Algorithm

The basic principle of distributed two-phase locking is same as the basic two-phase locking protocol. However, in a distributed system there are sites designated as lock managers. A lock manager controls lock acquisition requests from transaction monitors. In order to enforce co-ordination between the lock managers in various sites, at least one site is given the authority to see all transactions and detect lock conflicts.

Depending upon the number of sites who can detect lock conflicts, distributed two-phase locking approaches can be of three types −

1. Centralized two-phase locking- In this approach, one site is designated as the central lock manager. All the sites in the environment know the location of the central lock manager and obtain lock from it during transactions.
2. Primary copy two-phase locking- In this approach, a number of sites are designated as lock control centers. Each of these sites has the responsibility of managing a defined set of locks. All the sites know which lock control center is responsible for managing lock of which data table/fragment item.
3. Distributed two-phase locking − In this approach, there are a number of lock managers, where each lock manager controls locks of data items stored at its local site. The location of the lock manager is based upon data distribution and replication.

Distributed Timestamp Concurrency Control

In a centralized system, timestamp of any transaction is determined by the physical clock reading. But, in a distributed system, any site’s local physical/logical clock readings cannot be used as global timestamps, since they are not globally unique. So, a timestamp comprises of a combination of site ID and that site’s clock reading.

Conflict Graphs

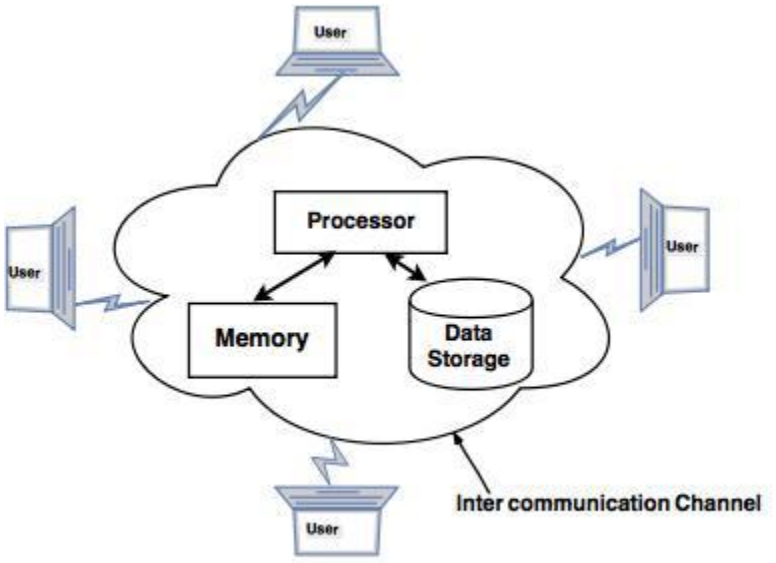
Another method is to create conflict graphs. For this transaction classes are defined. A transaction class contains two set of data items called read set and write set. A transaction belongs to a particular class if the transaction's read set is a subset of the class' read set and the transaction's write set is a subset of the class' write set. In the read phase, each transaction issues its read requests for the data items in its read set. In the write phase, each transaction issues its write requests.

Distributed Optimistic Concurrency Control Algorithm

Distributed optimistic concurrency control algorithm extends optimistic concurrency control algorithm. For this extension, two rules are applied −

Rule 1 − According to this rule, a transaction must be validated locally at all sites when it executes. If a transaction is found to be invalid at any site, it is aborted. Local validation guarantees that the transaction maintains serializability at the sites where it has been executed.

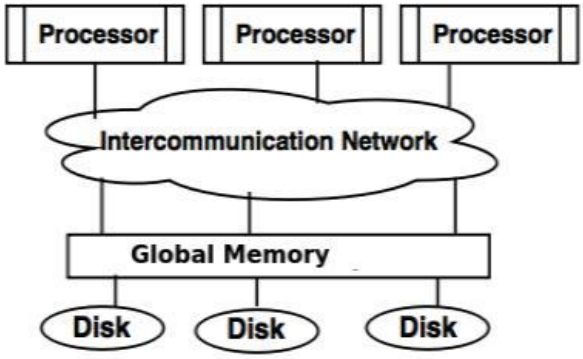
Rule 2 − According to this rule, after a transaction passes local validation test, it should be globally validated. Global validation ensures that if two conflicting transactions run together at more than one site, they should commit in the same relative order at all the sites they run together. This may require a transaction to wait for the other conflicting transaction, after validation before commit. This requirement makes the algorithm less optimistic since a transaction may not be able to commit as soon as it is validated at a site.



Parallel database system

Parallel Databases

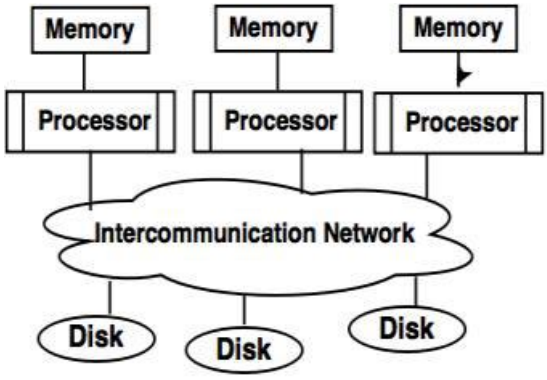
- Parallel database system improves performance of data processing using multiple resources in parallel, like multiple CPU and disks are used parallely.
- It also performs many parallelization operations like, data loading and query processing.
- Parallel databases can be roughly divided into two groups, the first group of architecture is the multiprocessor architecture, the alternatives of which are the following:



Shared Memory System in Parallel Databases

Shared memory architecture

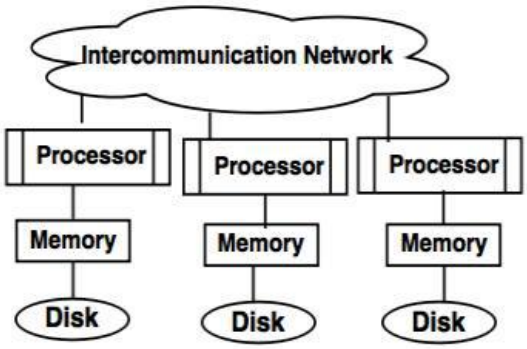
Where multiple processors share the main memory (RAM) space but each processor has its own disk (HDD). If many processes run simultaneously, the speed is reduced, the same as a computer when many parallel tasks run and the computer slows down.



Shared disk system in Parallel Databases

Shared disk architecture

Where each node has its own main memory, but all nodes share mass storage, usually a storage area network. In practice, each node usually also has multiple processors.



Shared nothing disk system in Parallel Databases

Shared nothing architecture

Where each node has its own mass storage as well as main memory.

The other architecture group is called hybrid architecture, which includes:

Non-Uniform Memory Architecture (NUMA), which involves the non-uniform memory access.

Cluster (shared nothing + shared disk: SAN/NAS), which is formed by a group of connected computers.

In this switches hubs are used to connect different computers. It is most cheapest way and simplest way only simple topologies are used to connect different computers much smarter if switches are implemented.

Input/output parallelism

In its simplest form, I/O parallelism refers to reducing the time required to retrieve relations from disk by partitioning the relations on multiple disks.

When there are multiple containers for a table space, the database manager can use parallel I/O. Parallel I/O refers to the process of writing to, or reading from, two or more I/O devices simultaneously; it can result in significant improvements in throughput.

Query parallelism

There are two types of query parallelism: interquery parallelism and intraquery parallelism.

Inter-query Parallelism

It is a form of parallelism where many different Queries or Transactions are executed in parallel with one another on many processors.

Advantages

- This technique allows to run multiple queries on different processors simultaneously.
- Transaction throughput can be increased by this form of parallelism.

Intra-Query Parallelism

It is the form of parallelism where Single Query is executed in parallel on many processors.

Advantages

- In this technique query is divided in sub queries which can run simultaneously on different processors, this will minimize the query evaluation time.
- Intra query parallelism improves the response time of the system.

In summary, the execution of a single query can be parallelized in two ways:

Intraoperation parallelism: We can speed up processing of a query by parallelizing the execution of each individual operation, such as sort, select, project, and join.

Interoperation parallelism: We can speed up processing of a query by executing in parallel the different operations in a query expression.

Design of Parallel Systems

Parallel processing divides a large task into many smaller tasks, and executes the smaller tasks concurrently on several nodes. As a result, the larger task completes more quickly.

Characteristics of a Parallel System

A parallel processing system has the following characteristics:

- Each processor in a system can perform tasks concurrently.
- Tasks may need to be synchronized.
- Nodes usually share resources, such as data, disks, and other devices.

Some issues in the design of parallel systems:

- Parallel loading of data from external sources is needed in order to handle large volumes of incoming data.
- Resilience to failure of some processors or disks.
- On-line reorganization of data and schema changes must be supported.