UNIT-II
Database System architectures

Centralized Database Management System

A centralized database is stored at a single location such as a mainframe computer.

It is maintained and modified from that location only and usually accessed using an internet connection such as a LAN or WAN.

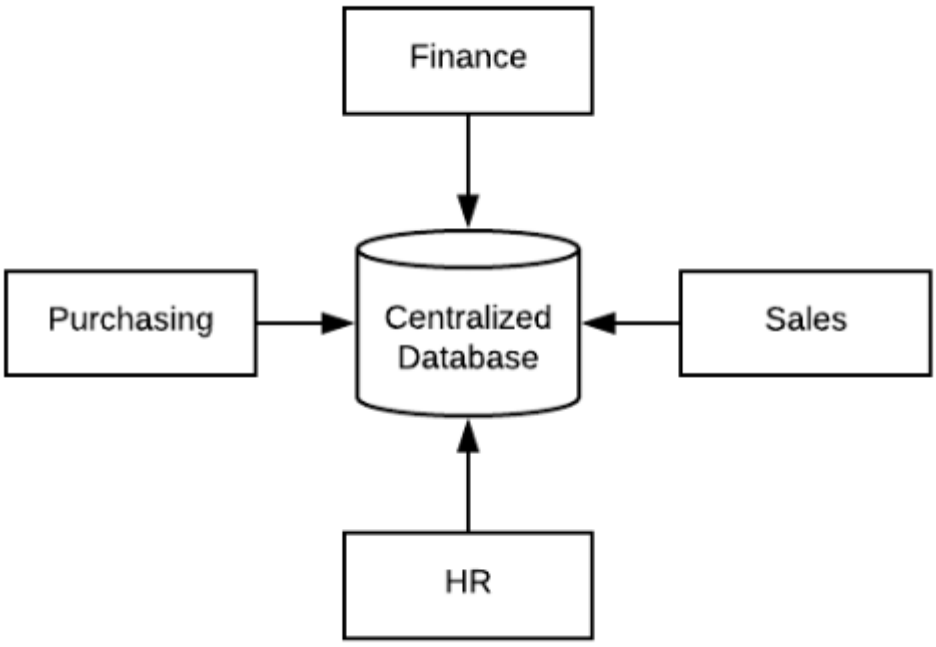
The centralized database is used by organisations such as colleges, companies, banks etc.

The centralized database mostly used in colleges, banks, hospitals and in small companies.

In a centralized database system one computer act as a server for storing whole data.

In a centralized database system, client/server architecture is used it is the very simplest form of a database system in which one client sent a request to the server.

The server will receive a request and will be a response. Many small organizations use a centralized database system.



As can be seen from the above diagram, all the information for the organisation is stored in a single database.

This database is known as the centralized database.

Advantages:

Some advantages of Centralized Database Management System are −

- The data integrity is maximised as the whole database is stored at a single physical location.
- This means that it is easier to coordinate the data and it is as accurate and consistent as possible.
- The data redundancy is minimal in the centralised database.
- All the data is stored together and not scattered across different locations. So, it is easier to make sure there is no redundant data available.
- Since all the data is in one place, there can be stronger security measures around it.
- So, the centralised database is much more secure.
- Data is easily portable because it is stored at the same place.
- The centralized database is cheaper than other types of databases as it requires less power and maintenance.
- All the information in the centralized database can be easily accessed from the same location and at the same time.

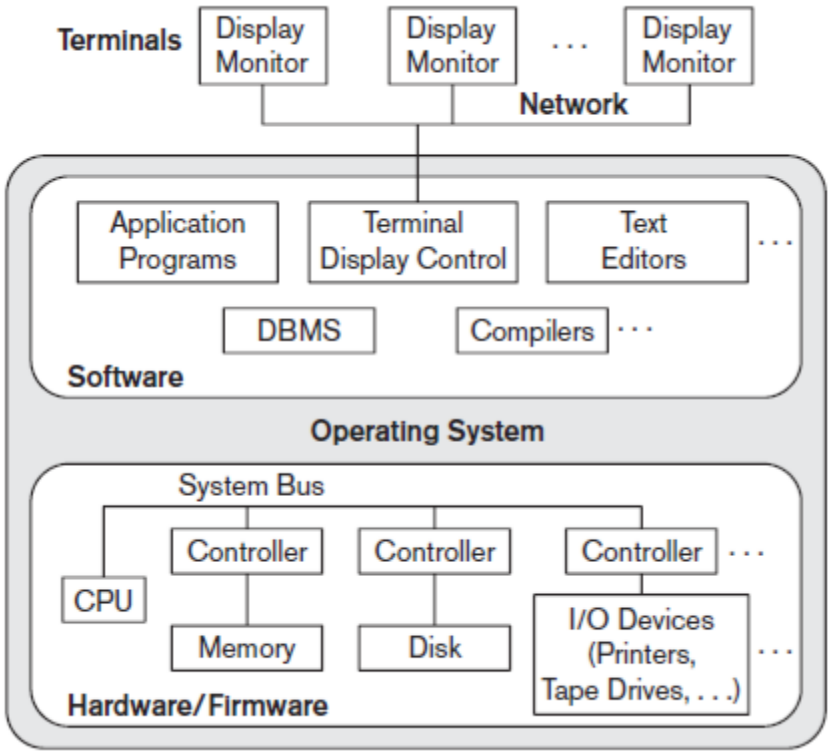
Disadvantages:

- Some disadvantages of Centralized Database Management System are −
- Since all the data is at one location, it takes more time to search and access it. If the network is slow, this process takes even more time.

There is a lot of data access traffic for the centralized database. This may create a bottleneck situation.

Since all the data is at the same location, if multiple users try to access it simultaneously it creates a problem. This may reduce the efficiency of the system.

If there are no database recovery measures in place and a system failure occurs, then all the data in the database will be destroyed.

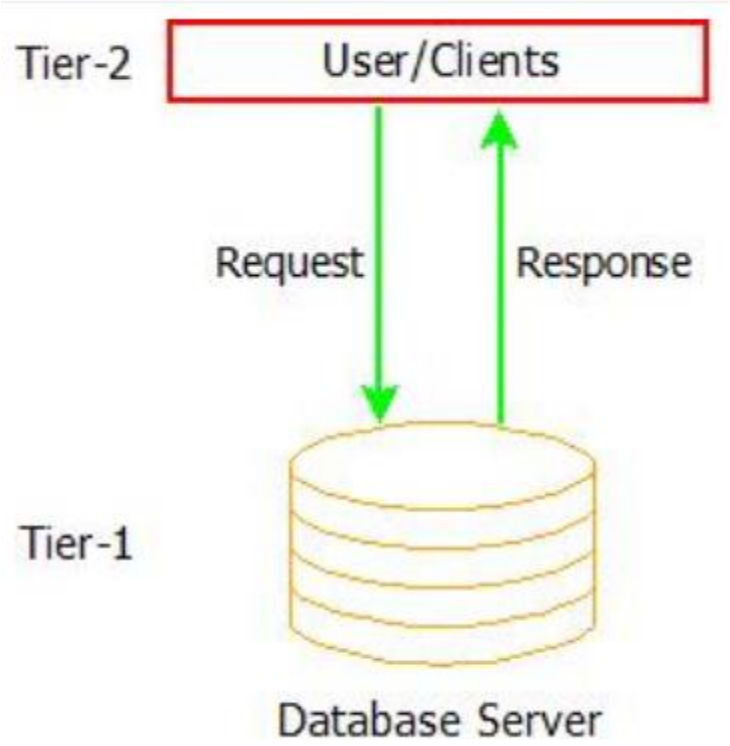


Centralized Architecture for DBMS

- **Client/Server Architecture:**
- The concept of client/server architecture assumes an underlying framework that consists of many PCs as well as a smaller number of mainframe machines, connected via LANs and other types of computer networks.
- A client in this framework is typically a user machine that provides user interface capabilities and local processing.
- When a client requires access to additional functionality, such as database access which does not exist at that machine, it connects to a server that provides the needed functionality.
- A Server is a system which contains both Hardware and Software which provides services to client Machines like file access, printing and database access.

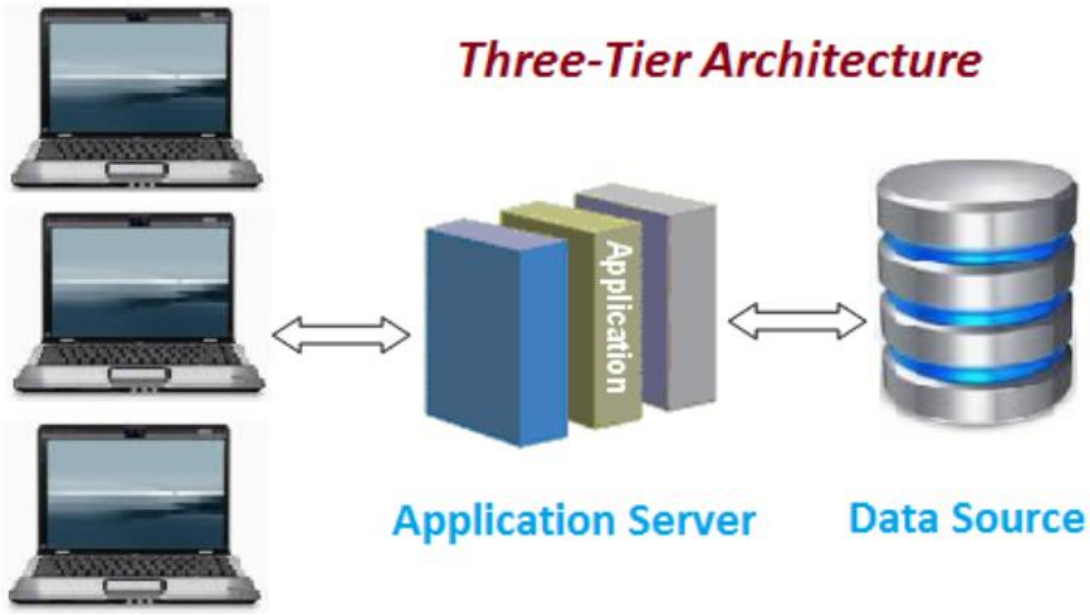
1) Two Tier Client/Server Architecture for DBMS:

- Here Two-tier means that our Architecture has two layers, which are client layer and Data layer.
- In Client layer we have several Client machines which can have the access to the database server.
- The API present on the client machine will establish the connection between the machine and the Database server through JDBC somthing else.
- This is because Clients and Database Server may be at different different locations.
- Once this connection gets established, the Interface present on the client machine contains an Application Program on the back-side which contains a query.
- This query will be processed by the Database server and in turn the queried information will be sent to the client machine.
- For example if we query the database to retrieve some information, the query will be Processed by Database server and that information will be sent to the client by Database server itself!!!



- Two-Tier Architecture for DBMS

- **2) Three-Tier client/server Architecture for DBMS:**
- Here there is an additional layer which acts as an intermediate between Client layer and Data layer called Business logic layer. Business logic layer is the layer where the Application Programs are processed.
- Here the Application Programs are processed in the Application server itself, which makes it different from Two-tier Architecture where queries are processed in the database server.
- Simply the Client machines will contact Application Server which in turn processes our Application Programs and fetches the Required Data from Database and then sends this Information back to the client machine in the suitable format only.



- Three-Tier client/server Architecture for DBMS
- Now we may think that Two-Tier Architecture is easy to use and maintain and why we should go for Three-Tier.
- The Reason is Three-Tier Architecture is Scalable and more secured.
- Even it is easy to maintain Two-Tier Architecture of DBMS it is still not scalable when we have large number of clients and also not secure because the clients are having direct access to database server.
- But Three-Tier Architecture ensures Scalability and Security of the data because of the presence of this Intermediate layer which processes the queries and it just retrieves data from server instead of processing in the server to take place.

Server System Architectures

Server systems can be broadly categorized as transaction servers and data servers.

- **Transaction-server systems**, also called query-server systems, provide an interface to which clients can send requests to perform an action, in response to which they execute the action and send back results to the client.
  Usually, client machines ship transactions to the server systems, where those transactions are executed, and results are shipped back to clients that are in charge of displaying the data.
  Requests may be specified by using SQL, or through a specialized application program interface.
- **Data-server systems** allow clients to interact with the servers by making requests to read or update data, in units such as files or pages.
  For example, file servers provide a file-system interface where clients can create, update, read, and delete files.
  Data servers for database systems offer much more functionality; they support units of data such as pages, tuples, or objects that are smaller than a file.
  They provide indexing facilities for data, and provide transaction facilities so that the data are never left in an inconsistent state if a client machine or process fails.

Of these, the transaction-server architecture is by far the more widely used architecture. We shall elaborate on the transaction-server and data-server architectures .

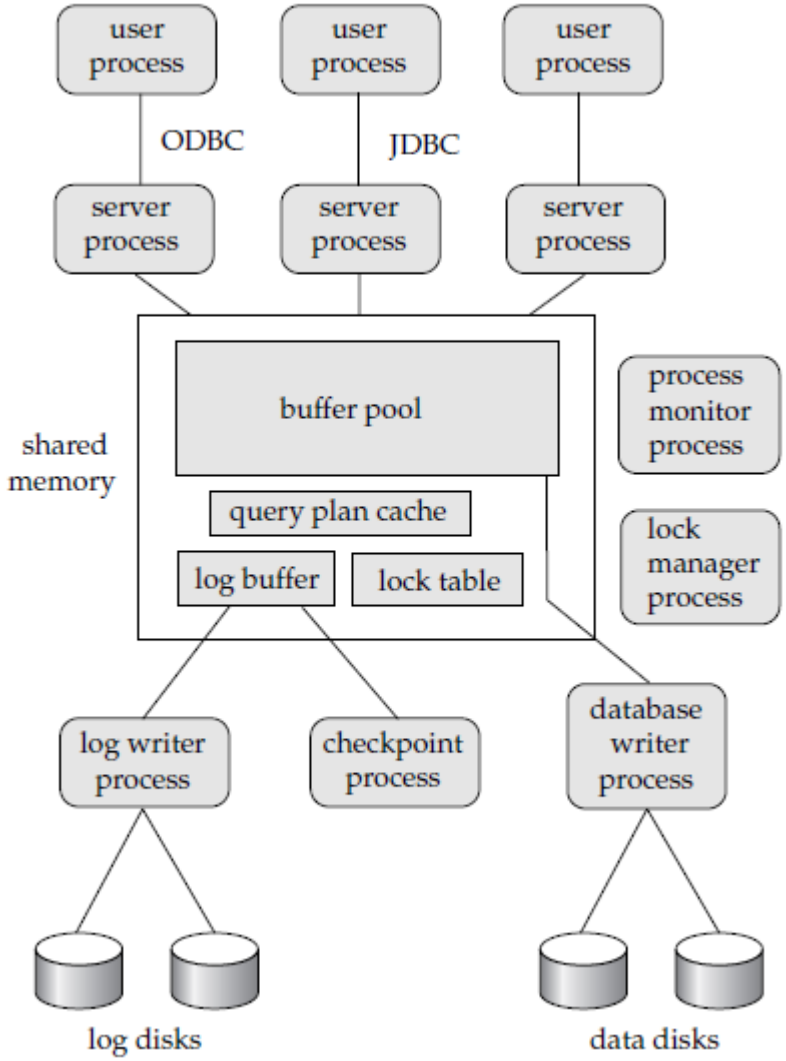
Transaction Server Process Structure

A typical transaction server system today consists of multiple processes accessing data in shared memory, as in Figure . The processes that form part of the database system include

- **Server processes:** These are processes that receive user queries (transactions), execute them, and send the results back.
  The queries may be submitted to the server processes from a a user interface, or from a user process running embedded SQL, or via JDBC, ODBC, or similar protocols.
  Some database systems use a separate process for each user session, and a few use a single database process for all user sessions, but with multiple threads so that multiple queries can execute concurrently.
  (A thread is like a process, but multiple threads execute as part of the same process, and all threads within a process run in the same virtual memory space. Multiple threads within a process can execute concurrently.)

Many database systems use a hybrid architecture, with multiple processes, each one running multiple threads.

- **Lock manager process:** This process implements lock manager functionality, which includes lock grant, lock release, and deadlock detection.
- **Database writer process:** There are one or more processes that output modified buffer blocks back to disk on a continuous basis.



Shared memory and process structure.

- **Log writer process:** This process outputs log records from the log record buffer to stable storage. Server processes simply add log records to the log record buffer in shared memory, and if a log force is required, they request the log writer process to output log records.
- **Checkpoint process:** This process performs periodic checkpoints.

- **Process monitor process:** This process monitors other processes, and if any of them fails, it takes recovery actions for the process, such as aborting any transaction being executed by the failed process, and then restarting the process.

The shared memory contains all shared data, such as:

- Buffer pool
- Lock table
- Log buffer, containing log records waiting to be output to the log on stable storage
- Cached query plans, which can be reused if the same query is submitted again
  All database processes can access the data in shared memory.
  Since multiple processes may read or perform updates on data structures in shared memory, there must be a mechanism to ensure that only one of them is modifying any data structure at a time, and no process is reading a data structure while it is being written by others.

Such mutual exclusion can be implemented by means of operating system functions called semaphores.

Alternative implementations, with less overheads, use special atomic instructions supported by the computer hardware;

one type of atomic instruction tests a memory location and sets it to 1 atomically. Further implementation details of mutual exclusion can be found in any standard operating system textbook.

The mutual exclusion mechanisms are also used to implement latches.

To avoid the overhead of message passing, in many database systems, server processes implement locking by directly updating the lock table (which is in shared memory), instead of sending lock request messages to a lock manager process.

The lock request procedure executes the actions that the lock manager process would take on getting a lock request.

The actions on lock request and release are like those but with two significant differences:

- Since multiple server processes may access shared memory, mutual exclusion must be ensured on the lock table.
- If a lock cannot be obtained immediately because of a lock conflict, the lock request code keeps monitoring the lock table to check when the lock has been granted.
  The lock release code updates the lock table to note which process has been granted the lock.

To avoid repeated checks on the lock table, operating system semaphores can be used by the lock request code to wait for a lock grant notification.

The lock release code must then use the semaphore mechanism to notify waiting transactions that their locks have been granted.

Even if the system handles lock requests through shared memory, it still uses the lock manager process for deadlock detection.

Data Servers

Data-server systems are used in local-area networks, where there is a high-speed connection between the clients and the server, the client machines are comparable in processing power to the server machine, and the tasks to be executed are computation intensive.

In such an environment, it makes sense to ship data to client machines, to perform all processing at the client machine (which may take a while), and then to ship the data back to the server machine.

Note that this architecture requires full back-end functionality at the clients. Data-server architectures have been particularly popular in object-oriented database systems.

Interesting issues arise in such an architecture, since the time cost of communication between the client and the server is high compared to that of a local memory reference (milliseconds, versus less than 100 nano seconds):

- Page shipping versus item shipping. The unit of communication for data can be of coarse granularity, such as a page, or fine granularity, such as a tuple (or an object, in the context of object-oriented database systems).
- We use the term item to refer to both tuples and objects.
- If the unit of communication is a single item, the overhead of message passing is high compared to the amount of data transmitted.
- Instead, when an item is requested, it makes sense also to send back other items that are likely to be used in the near future.
- Fetching items even before they are requested is called prefetching.
- Page shipping can be considered a form of prefetching if multiple items reside on a page, since all the items in the page are shipped when a process desires to access a single item in the page.
- Locking. Locks are usually granted by the server for the data items that it ships to the client machines.
- A disadvantage of page shipping is that client machines may be granted locks of too coarse a granularity a lock on a page implicitly locks all items contained in the page.
- Even if the client is not accessing some items in the page, it has implicitly acquired locks on all pre fetched items.
- Other client machines that require locks on those items may be blocked unnecessarily.
- Techniques for lock de-escalation, have been proposed where the server can request its clients to transfer back locks on prefetched items.
- If the client machine does not need a prefetched item, it can transfer locks on the item back to the server, and the locks can then be allocated to other clients.
- Data caching. Data that are shipped to a client on behalf of a transaction can be cached at the client, even after the transaction completes, if sufficient storage space is available. Successive transactions at the same client may be able to make use of the cached data. However, cache coherency is an issue: Even if a transaction finds cached data, it must make sure that those data are up to date, since they may have been updated by a different client after they were cached. Thus, a message must still be exchanged with the server to check validity of the data, and to acquire a lock on the data.
- Lock caching. If the use of data is mostly partitioned among the clients, with clients rarely requesting data that are also requested by other clients, locks can also be cached at the client machine. Suppose that a client finds a data item in the cache, and that it also finds the lock required for an access to the data item in the cache. Then, the access can proceed without any communication with the server. However, the server must keep track of cached locks; if a client requests

a lock from the server, the server must call back all conflicting locks on the data item from any other client machines that have cached the locks. Thetask becomes more complicated when machine failures are taken into account.

This technique differs from lock de-escalation in that lock caching takes place across transactions; otherwise, the two techniques are similar.

Parallel Systems

Parallel systems improve processing and I/O speeds by using multiple CPUs and disks in parallel.

Parallel machines are becoming increasingly common, making the study of parallel database systems correspondingly more important.

The driving force behind parallel database systems is the demands of applications that have to query extremely large databases (of the order of terabytes that is, 1012 bytes) or that have to process an extremely large number of transactions per second (of the order of thousands of transactions per second).

Centralized and client–server database systems are not powerful enough to handle such applications.

In parallel processing, many operations are performed simultaneously, as opposed to serial processing, in which the computational steps are performed sequentially. A coarse-grain parallel machine consists of a small number of powerful processors; a massively parallel or fine-grain parallel machine uses thousands of smaller processors.

Most high-end machines today offer some degree of coarse-grain parallelism: Two or four processor machines are common.

Massively parallel computers can be distinguished from the coarse-grain parallel machines by the much larger degree of parallelism that they support.

Parallel computers with hundreds of CPUs and disks are available commercially.

There are two main measures of performance of a database system:

(1) throughput, the number of tasks that can be completed in a given time interval, and

(2) response time, the amount of time it takes to complete a single task from the time it is submitted.

A system that processes a large number of small transactions can improve throughput by processing many transactions in parallel.
A system that processes large transactions can improve response time as well as throughput by performing subtasks of each transaction in parallel.

Speedup and Scaleup

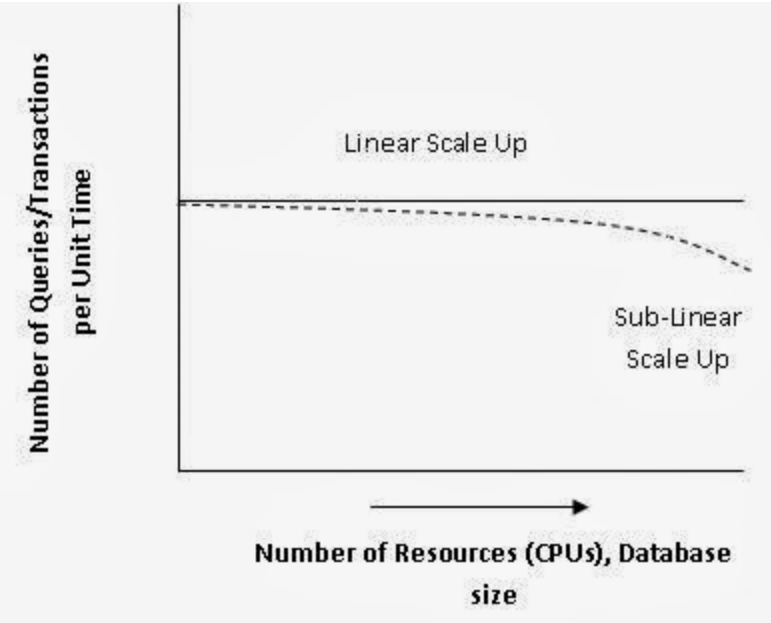
Two important issues in studying parallelism are speedup and scaleup.

Running a given task in less time by increasing the degree of parallelism is called speedup.

Handling larger tasks by increasing the degree of parallelism is called scaleup.
Consider a database application running on a parallel system with a certain number of processors and disks.

Now suppose that we increase the size of the system by increasing the number or processors, disks, and other components of the system.

The goal is to process the task in time inversely proportional to the number of processors and disks allocated.



Scaleup relates to the ability to process larger tasks in the same amount of time by providing more resources.

Scaleup in Parallel (Systems) Database

Scaleup is the ability to keep the same performance levels (response time) when both workload (transactions) and resources (CPU, memory) increase proportionally.

Scaleup = (Small system small problem elapsed time[single machine])/(large system large problem elapsed time[parallel machine])

For example, if 50 users consume close to 100 percent of the CPU during normal processing, then adding more users would cause the system to slow down due to contention for limited CPU cycles. However, by adding more CPUs, we can support extra users without degrading performance.

Linear Scaleup

Scaleup is linear if the resources increase in proportional with the problem size (it is very rare). According to the equation given above, if small system small problem elapsed time is equal to large system large problem elapsed time, then Scaleup = 1 which is linear.

Sub-linear Scaleup

Scaleup is sub-linear if large system large problem elapsed time is greater than small system small problem elapsed time, then the scaleup is sub-linear.

Further useful discussions:

- If scaleup is 1, i.e Linear, then performance of the system is excellent.
- If scaleup is Sub-linear and the value falls between 0 and 1, then we need extra care to choose our plan for parallel execution. For example, if small system small problem elapsed time is 5 seconds, and large system large problem elapsed time is 5 seconds. This clearly shows Linear. That is, 5/5 = 1. For other values of denominator, especially low values (not possible beyond a limit), we would say that the system performance is excellent. But, for higher values of the denominator, say 6, 7, 8 and so on, the scale up value falls below 1 which needs much attention for better workload redistribution.

Speedup in Parallel (Systems) Database

Speedup is the effect of applying an increasing number of resources to a fixed amount of work to achieve a proportional reduction in execution times:

Speedup = (Small system elapsed time[single machine])/(large system elapsed time[parallel machine])

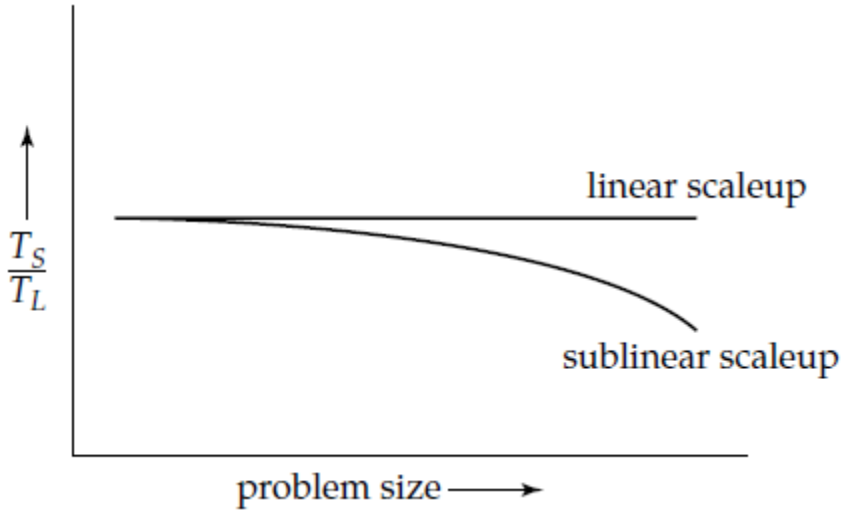
Speedup results in resource availability for other tasks. For example, if queries usually take ten minutes to process in one CPU and running in parallel in more CPUs reduces the time, then additional queries can run without introducing the contention that might occur were they to run concurrently.

Speedup reduces the Response Time, i.e, the time to complete a task is reduced.

Figure illustrates linear and sublinear scaleups (where the resources increase proportional to problem size).

There are two kinds of scaleup that are relevant in parallel database systems, depending on how the size of the task is measured:

- In batch scale up, the size of the database increases, and the tasks are large jobs whose runtime depends on the size of the database.
- An example of such a task is a scan of a relation whose size is proportional to the size of the database.
- Thus, the size of the database is the measure of the size of the problem.
- Batch scaleup also applies in scientific applications, such as executing a query at an N-times finer resolution or performing an N-times longer simulation.
- In transaction scaleup, the rate at which transactions are submitted to the database increases and the size of the database increases proportionally to the transaction rate.
- This kind of scale up is what is relevant in transaction processing systems where the transactions are small updates for example, a deposit or withdrawal from an account and transaction rates grow as more accounts are created. Such transaction processing is especially well adapted for parallel execution, since transactions can run concurrently and independently on separate processors, and each transaction takes roughly the same amount of time, even if the database grows.



Scaleup with increasing problem size and resources.

Parallel Database Architectures:

A parallel DBMS is a DBMS that runs across multiple processors or CPUs and is mainly designed to execute query operations in parallel, wherever possible. The parallel DBMS link a number of smaller machines to achieve the same throughput as expected from a single large machine.

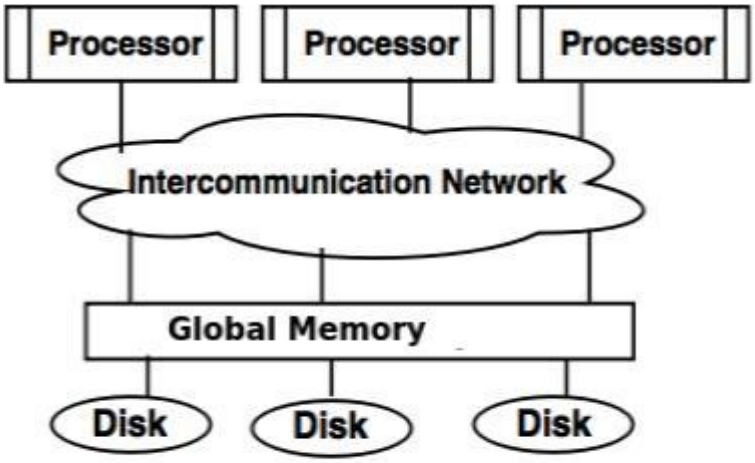
- **Shared memory**. All the processors share a common memory.
- **Shared disk.** All the processors share a common set of disks. Shared-disk systems are sometimes called clusters.
- **Shared nothing.** The processors share neither a common memory nor common disk .
- **Hierarchical**. This model is a hybrid of the preceding three architecture.

Techniques used to speed up transaction processing on data-server systems, such as data and lock caching and lock de-escalation, can also be used in shared-disk parallel databases as well as in shared-nothing parallel databases.

In fact, they are very important for efficient transaction processing in such systems.

Shared memory system

- Shared memory system uses multiple processors which is attached to a global shared memory via intercommunication channel or communication bus.
- Shared memory system have large amount of cache memory at each processors, so referencing of the shared memory is avoided.
- If a processor performs a write operation to memory location, the data should be updated or removed from that location.
    - A processor can send messages to other processors much faster by using memory writes (which usually take less than a microsecond) than by sending a message through a communication mechanism.
    - The downside of shared-memory machines is that the architecture is not scalable beyond 32 or 64 processors because the bus or the interconnection network becomes a bottleneck (since it is shared by all processors).



Shared Memory System in Parallel Databases

Advantages of Shared memory system

- Data is easily accessible to any processor.
- One processor can send message to other efficiently.

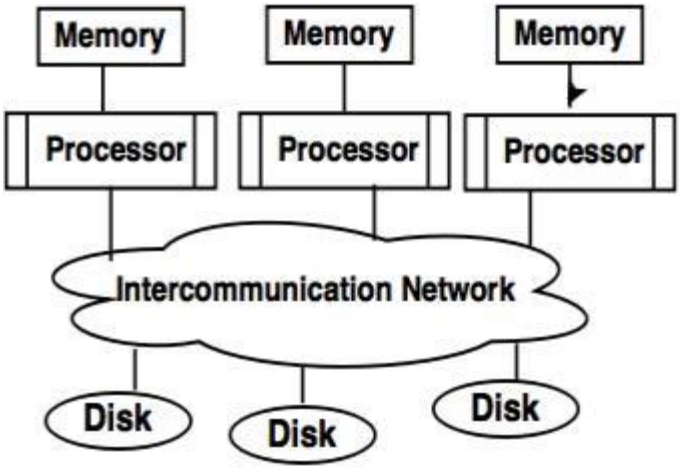
Disadvantages of Shared memory system

- Waiting time of processors is increased due to more number of processors.
- Bandwidth problem.

Shared Disk System

- Shared disk system uses multiple processors which are accessible to multiple disks via intercommunication channel and every processor has local memory.
- Each processor has its own memory so the data sharing is efficient.
- The system built around this system are called as clusters.
    - In the shared-disk model, all processors can access all disks directly via an interconnection network, but the processors have private memories.
    - There are two advantages of this architecture over a shared-memory architecture. First, since each processor has its own memory, the memory bus is not a bottleneck.
    - Second, it offers a cheap way to provide a degree of fault tolerance: If a processor (or its memory) fails, the other processors can take over its tasks,

since the database is resident on disks that are accessible from all processors. We can make the disk subsystem itself fault tolerant by using a RAID architecture. The shared-disk architecture has found acceptance in many applications.



Shared disk system in Parallel Databases

Advantages of Shared Disk System

- Fault tolerance is achieved using shared disk system.

  Fault tolerance: If a processor or its memory fails, the other processor can complete the task. This is called as fault tolerance.

Disadvantage of Shared Disk System

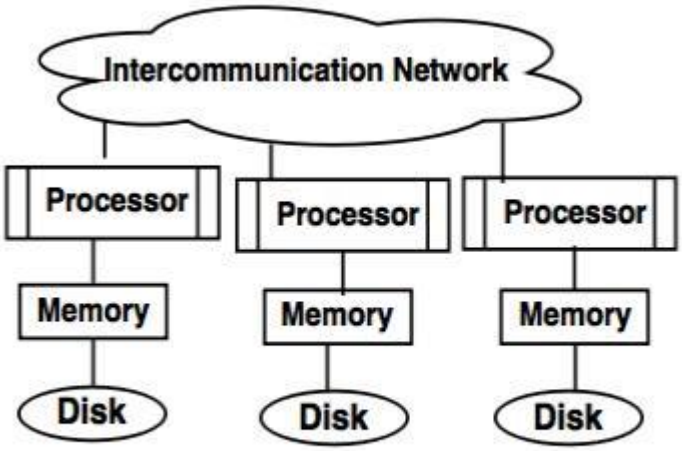
- Shared disk system has limited scalability as large amount of data travels through the interconnection channel.
- If more processors are added the existing processors are slowed down.

Applications of Shared Disk System

Digital Equipment Corporation(DEC): DEC cluster running relational databases use the shared disk system and now owned by Oracle.

Shared nothing disk system

- Each processor in the shared nothing system has its own local memory and local disk.
- Processors can communicate with each other through intercommunication channel.
- Any processor can act as a server to serve the data which is stored on local disk.



Shared nothing disk system in Parallel Databases

Advantages of Shared nothing disk system

- Number of processors and disk can be connected as per the requirement in share nothing disk system.
- Shared nothing disk system can support for many processor, which makes the system more scalable.

Disadvantages of Shared nothing disk system

- Data partitioning is required in shared nothing disk system.
- Cost of communication for accessing local disk is much higher.

Applications of Shared nothing disk system

- Tera data database machine.
- The Grace and Gamma research prototypes.

Hierarchical System or Non-Uniform Memory Architecture

- Hierarchical model system is a hybrid of shared memory system, shared disk system and shared nothing system.

- Hierarchical model is also known as **Non-Uniform Memory Architecture (NUMA).**
- In this system each group of processor has a local memory. But processors from other groups can access memory which is associated with the other group in coherent.
- **NUMA** uses local and remote memory(Memory from other group), hence it will take longer time to communicate with each other.
    - The hierarchical architecture combines the characteristics of shared-memory, shared disk, and shared-nothing architectures.
    - At the top level, the system consists of nodes connected by an interconnection network, and do not share disks or memory with one another.
    - Thus, the top level is a shared-nothing architecture. Each node of the system could actually be a shared-memory system with a few processors.
    - Alternatively, each node could be a shared-disk system, and each of the systems sharing a set of disks could be a shared-memory system.
    - Thus, a system could be built as a hierarchy, with shared-memory architecture with a few processors at the base, and a shared nothing architecture at the top, with possibly a shared-disk architecture in the middle.

Advantages of NUMA

- Improves the scalability of the system.
- Memory bottleneck(shortage of memory) problem is minimized in this architecture.

Disadvantages of NUMA

The cost of the architecture is higher compared to other architectures.

Distributed Systems

In a distributed database system, the database is stored on several computers.

The computers in a distributed system communicate with one another through various communication media, such as high-speed networks or telephone lines.

They do not share main memory or disks. The computers in a distributed system may vary in size and function, ranging from workstations up to mainframe systems.

The computers in a distributed system are referred to by a number of different names, such as sites or nodes, depending on the context in which they are mentioned.

We mainly use the term site, to emphasize the physical distribution of these systems. The general structure of a distributed system appears in Figur .

The main differences between shared-nothing parallel databases and distributed databases are that distributed databases are typically geographically separated, are separately administered, and have a slower interconnection.

Another major difference is that, in a distributed database system, we differentiate between local and global transactions.

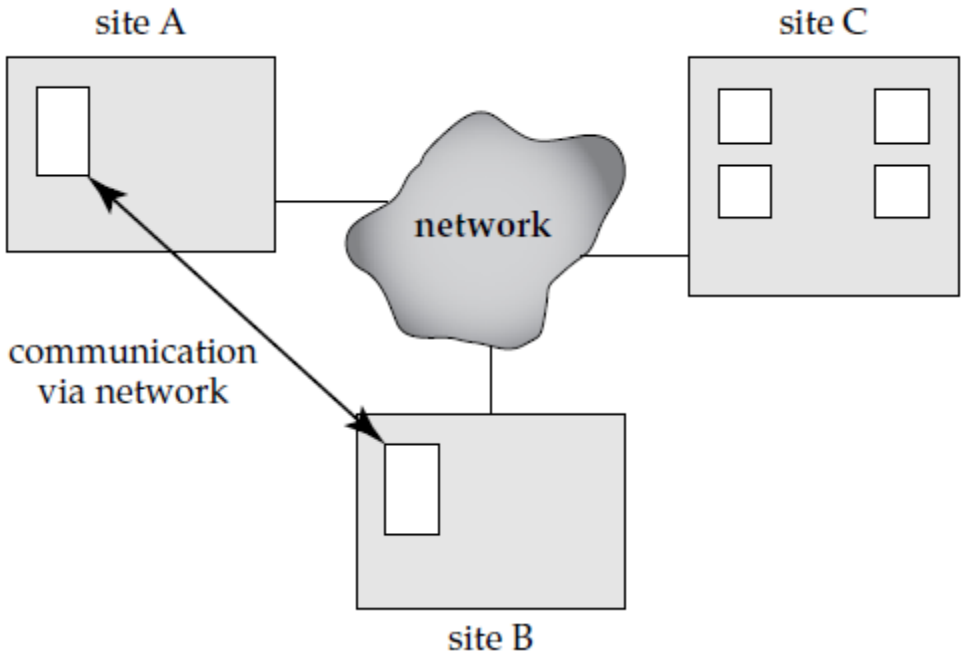
A local transaction is one that accesses data only from sites where the transaction was initiated. A global transaction, on the other hand, is one that either accesses data in a site different from the one at which the transaction was initiated, or accesses data in several different sites.

There are several reasons for building distributed database systems, including sharing of data, autonomy, and availability.

- **Sharing data.** The major advantage in building a distributed database system is the provision of an environment where users at one site may be able to access the data residing at other sites.
  For instance, in a distributed banking system, where each branch stores data related to that branch, 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 that would couple existing systems.

- **Autonomy.** The primary advantage of sharing data by means of data distribution is that each site is able to retain a degree of control over data that are 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 on the design of the distributed database system, each administrator may have a different degree of local autonomy.
  The possibility of local autonomy is often a major advantage of distributed databases.



A distributed system.

- **Availability.** If one site fails in a distributed system, the remaining sites may be able to continue operating.
  In particular, if data items are replicated in several sites, a transaction needing a particular data item may find that item in any of 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 services 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 systems, 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,

How Distributed Systems Work

The most important functions of distributed computing are:

- **Resource sharing** - whether it's the hardware, software or data that can be shared
- **Openness** - how open is the software designed to be developed and shared with each other
- **Concurrency** - multiple machines can process the same function at the same time
- **Scalability** - how do the computing and processing capabilities multiply when extended to many machines
- **Fault tolerance** - how easy and quickly can failures in parts of the system be detected and recovered
- **Transparency** - how much access does one node have to locate and communicate with other nodes in the system.

An Example of a Distributed Database

Consider a banking system consisting of four branches in four different cities.

Each branch has its own computer, with a database of all the accounts maintained at that branch. Each such installation is thus a site.

There also exists one single site that maintains information about all the branches of the bank.

Each branch maintains (among others) a relation account(Account-schema),

where Account-schema = (account-number, branch-name, balance)

The site containing information about all the branches of the bank maintains the relation branch(Branch-schema), where

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

There are other relations maintained at the various sites.

Important Questions:

1. Explain Centralized system architecture.
2. Explain parallel system speed up and scale up.
3. Explain an Example of Distributed Database.
4. Explain Parallel Database Architectures.
5. Explain Server System Architectures.