CS6005-Advanced Database Systems
UNIT-I

What you meant by Parallel DBMS:

A DBMS running across multiple processors and disk that is designed to execute operations in parallel whenever possible in order to improve performance.

Parallel Database:

- A parallel database system seeks to improve performance through parallelization of various operations, such as loading data, building indexes and evaluating queries.
- Although data may be stored in a distributed fashion, the distribution is governed exclusively by performance considerations.
- Parallel databases improve processing and input/output speeds by using multiple CPUs and disks in parallel.
- Centralized and client–server database systems are not powerful enough to handle such applications.
- In parallel processing, many operations are performed simultaneously, as opposite to serial processing, in which the computational steps are performed sequentially.
- Parallel databases can be approximately divided into two groups,
- the first group of architecture is the multiprocessor architecture, the alternatives of which are the followings :

Shared memory architecture: where multiple processors share the main memory space, as well as mass storage (e.g. hard disk drives).

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 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.

Shared Memory Architecture:

- Shared memory is a tightly coupled architecture in which multiple processors within a single system share system memory.

- Known as symmetric multiprocessing (SMP), this approach has become popular on platforms ranging from personal workstations that support a few microprocessors in parallel.
- This architecture provides high-speed data access for a limited number of processors, but it is not scalable beyond about 64 processors when the interconnection network becomes a bottleneck.

Advantages:

- Simple implementation
- Establishes effective communication between processors through single memory addresses space.
- Above point leads to less communication overhead.

Disadvantages:

- Higher degree of parallelism cannot be achieved due to the reason that all the processors share the same interconnection network to connect with memory. This causes Bottleneck in interconnection network (Interference), especially in the case of Bus interconnection network.
- Addition of processor would slow down the existing processors..
- Degree of Parallelism is limited. More number of parallel processes might degrade the performance.

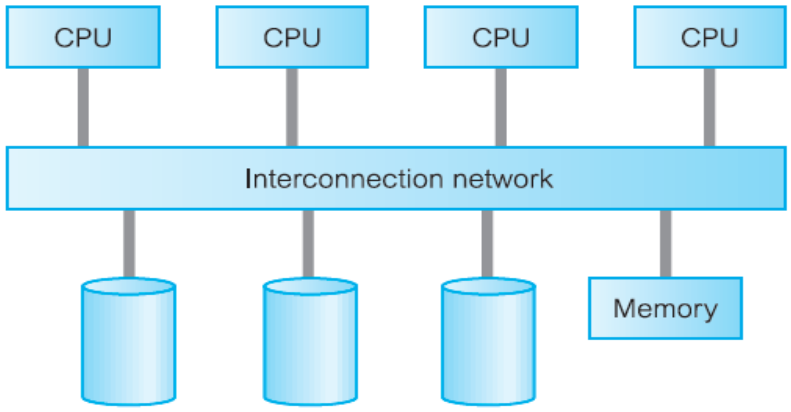


Figure Shared Memory Architecture

Shared Disk Architecture:

- Shared disk is a loosely-coupled architecture optimized for applications that are inherently centralized and require high availability and performance.
- Each processor can access all disks directly, but each has its own private memory.
- Like the shared nothing architecture, the shared disk architecture eliminates the shared memory performance bottleneck.

- Unlike the shared nothing architecture, however, the shared disk architecture eliminates this bottleneck without introducing the overhead associated with physically partitioned data.
- Shared disk systems are sometimes referred to as clusters.

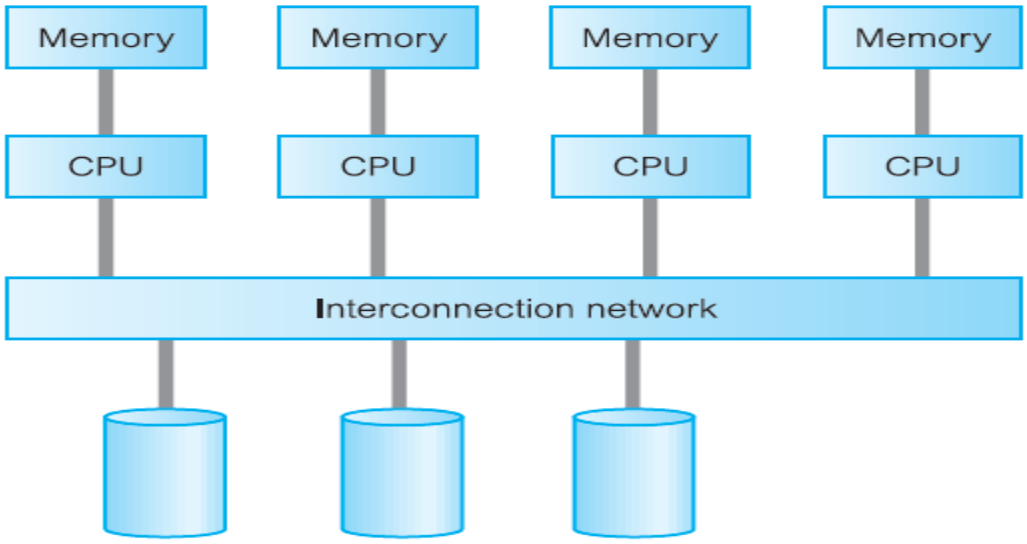


Figure Shared Disk Architecture

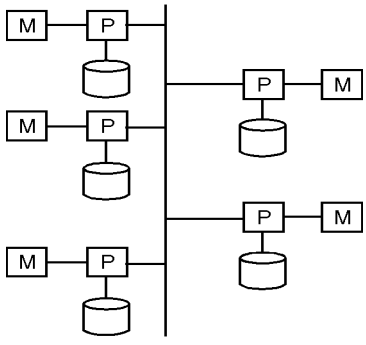
Advantages

- Failure of any processors would not stop the entire system (Fault tolerance)
- Interconnection to the memory is not a bottleneck. (It was bottleneck in Shared Memory architecture)
- Support larger number of processors (when compared to Shared Memory architecture)

Disadvantages:

- Interconnection to the disk is bottleneck as all processors share common disk setup.
- Inter-processor communication is slow.
- The reason is, all the processors have their own memory. Hence, the communication between processors need reading of data from other processors' memory which needs additional software support.

Shared Nothing The processors share neither a common memory nor common disk.



- In a shared-nothing system, each node of the machine consists of a processor, memory, and one or more disks.
- The processors at one node may communicate with one another processor at another node by a high-speed interconnection network.
- A node functions as the server for the data on the disk or disks that the node owns. Since local disk references are serviced by local disks at each processor.

Advantages

- The shared-nothing model overcomes the disadvantage of requiring all I/O to go through a singly interconnection network; only queries, accesses to non local disks, and result relations pass through the network.
- Moreover, the interconnection networks for shared nothing systems are usually designed to be scalable, so that their transmission capacity increases as more nodes are added.
- Consequently, shared-nothing architectures are more scalable, and can easily support a large number of processors.

Disadvantage

- The main drawback of shared nothing systems is the costs of communication and of nonlocal disk access, which are higher than in a shared memory or shared-disk architecture since sending data involves software interaction at both ends.

Hierarchical

- 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 shared-disk architecture in the middle.
- Fig. illustrates a hierarchical architecture with shared-memory nodes connected together in a shared nothing architecture.
- Commercial parallel database systems today run on several of these architectures.
- Attempts to reduce the complexity of programming such systems have yielded distributed virtual memory architectures, where logically there is a single shared memory, but physically there are multiple disjoint memory systems; the virtual-memory-mapping hardware, coupled with system software, allows each processor *to* view the disjoint memories as a single virtual memory.

Since access speeds differ, depending whether the page is available locally or not, such architecture is also referred to as nonuniform memory architecture (NUMA).

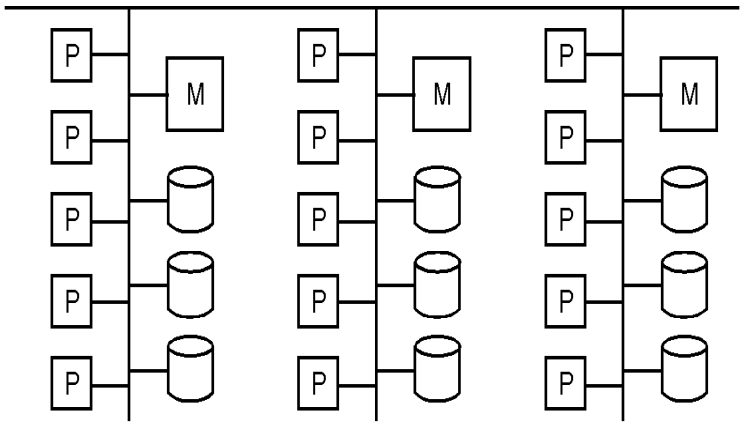


Figure Hierarchical architecture

INTER& INTRA QUERY PARALLELISM

Parallelism

- Components of a task, such as a database query, can be run in parallel to dramatically enhance performance.
- The nature of the task, the database configuration, and the hardware environment, all determine how DB2(R) Universal Database (DB2 UDB) will perform a task in parallel.
- These considerations are interrelated, and should be considered together when you work on the physical and logical design of a database.

The following types of parallelism are supported by DB2 UDB:

- **I/O** parallelism.
- **Query** parallelism.
- **Utility** parallelism.

Input/output parallelism

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

Query parallelism

– There are two types of query parallelism:

Inter query parallelism

- Different queries or transactions execute in parallel with one another.
- It increases scale up and throughput.

Intra query parallelism

- It refers to the execution of a single query in parallel on multiple processors and disk.
- It is important for speeding up long running queries.

1) Inter query parallelism refers to the ability of the database to accept queries from multiple applications at the same time.
2) Each query runs independently of the others, but DB2 UDB runs all of them at the same time.
3) DB2 UDB has always supported this type of parallelism.
4) Intra query parallelism refers to the simultaneous processing of parts of a single query, using either intra partition parallelism, inter partition parallelism, or both.

Intra partition parallelism:

Intra partition parallelism refers to the ability to break up a query into multiple parts. Some DB2 UDB utilities also perform this type of parallelism.

- Intra partition parallelism subdivides what is usually considered a single database operation,
- Such as index creation, database loading, or SQL queries into multiple parts, many or all of which can be run in parallel within a single database partition.
- The pieces are copies of each other.
- To utilize intra partition parallelism, you must configure the database appropriately.
- You can choose the degree of parallelism or let the system do it for you.
- The degree of parallelism represents the number of pieces of a query running in parallel.

Intra partition parallelism:

- Speed up processing of a query by parallelising the execution of each individual operation.

Intra partition parallelism:

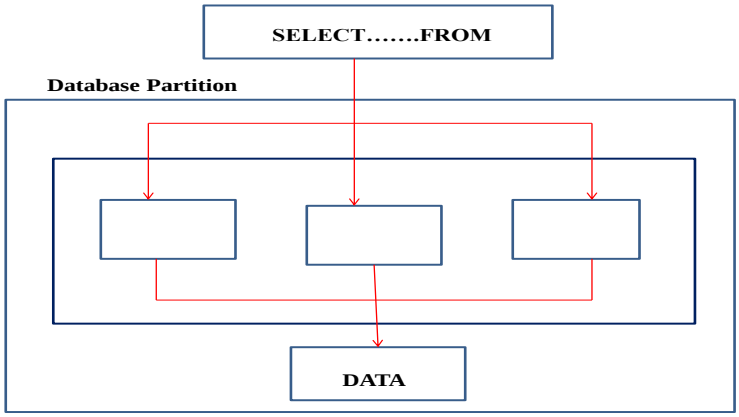


Figure1 Intra partition parallelism

1. Inter partition parallelism:

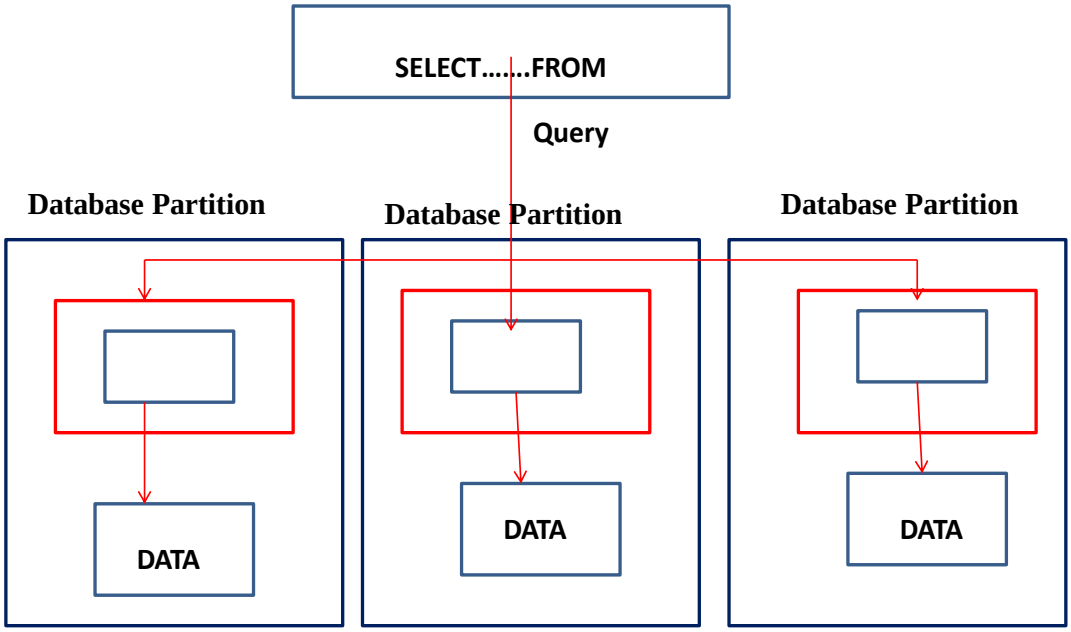
Inter partition parallelism refers to the ability to break up a query into multiple parts across multiple partitions of a partitioned database, on one machine or multiple machines.

1. The query is run in parallel.
2. Some DB2 UDB utilities also perform this type of parallelism.
3. Inter partition parallelism subdivides what is usually considered a single database operation.
4. such as index creation, database loading, or SQL queries into multiple parts, many or all of which can be run in parallel across multiple partitions of a partitioned database on one machine or on multiple machines.
5. The degree of parallelism is largely determined by the number of partitions you create and how you define your database partition groups.

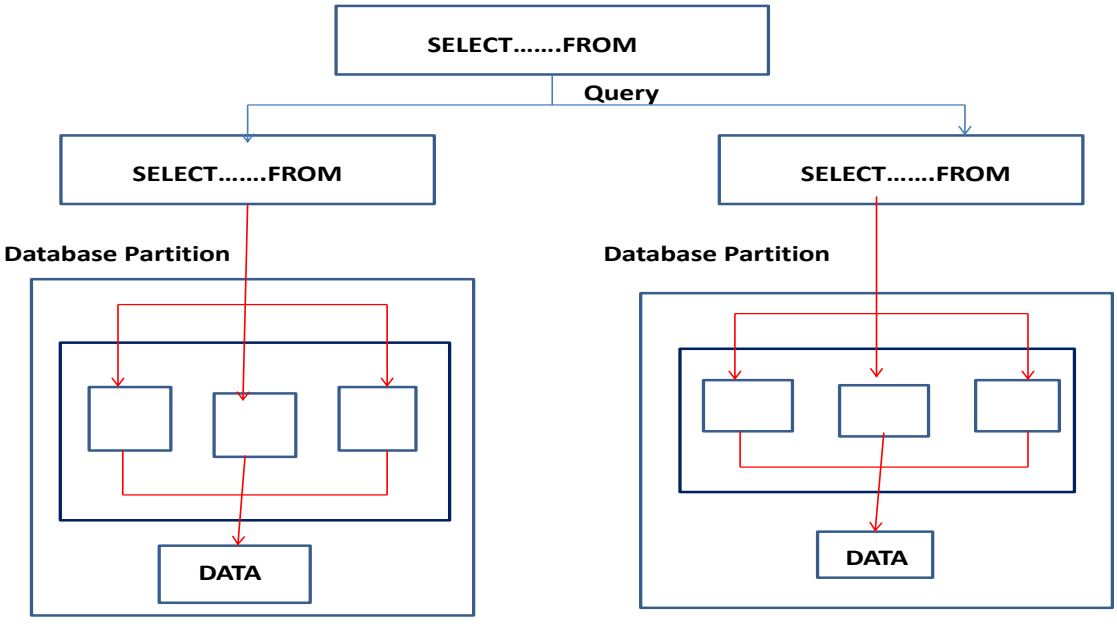
2. **Inter partition parallelism:**

1. Speed up processing of a query by executing in parallel the different operations in a query expression.

Inter partition parallelism:



Simultaneous inter partition and intra partition parallelism:



DISTRIBUTED DATABASE:

A logically interrelated collection of shared data (and a description of this data), physically distributed over a computer network.

DISTRIBUTED DBMS:

The software system that permits the management of distributed database and makes the distribution transparent to the user.

- A **Distributed Database Management System** (DDBMS) consists of a single logical database that is split into a number of **fragments**.
- Each fragment is stored on one or more computers under the control of a separate DBMS, with the computers connected by a communications network.
- Each site is capable of independently processing user requests that require access to local data (that is, each site has some degree of local autonomy) and is also capable of processing data stored on other computers in the network.
- Users access the distributed database via applications, which are classified as those that do not require data from other sites (local applications) and those that do require data from other sites (global applications).
- We require a DDBMS to have at least one global application. A DDBMS therefore has the following characteristics:

  1. Collection of logically-related shared data.
  2. Data split into fragments.
  3. Fragments may be replicated.
  4. Fragments/replicas allocated to sites.
  5. Sites linked by a communications network.
  6. Data at each site is under control of a DBMS.
  7. DBMSs handle local applications autonomously.
  8. Each DBMS participates in at least one global application.

It is not necessary for every site in the system to have its own local database, as illustrated by the topology of the DDBMS shown in Figure1.

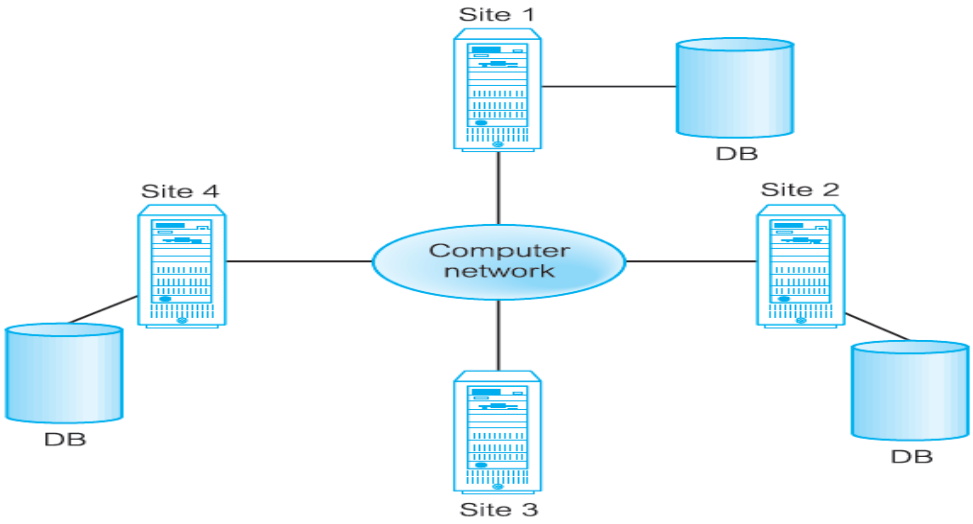


Figure 1 Distributed database management system.

Distributed processing

It is important to make a distinction between a distributed DBMS and distributed processing.

Distributed processing:

A centralized database that can be accessed over a computer network.

- The key point with the definition of a distributed DBMS is that the system consists of data that is physically distributed across a number of sites in the network.
- If the data is centralized, even though other users may be accessing the data over the network, we do not consider this to be a distributed DBMS, simply distributed processing.
- We illustrate the topology of distributed processing in Figure 2. Compare this figure, which has a central database at site 2, with Figure 1, which shows several sites each with their own database (DB).

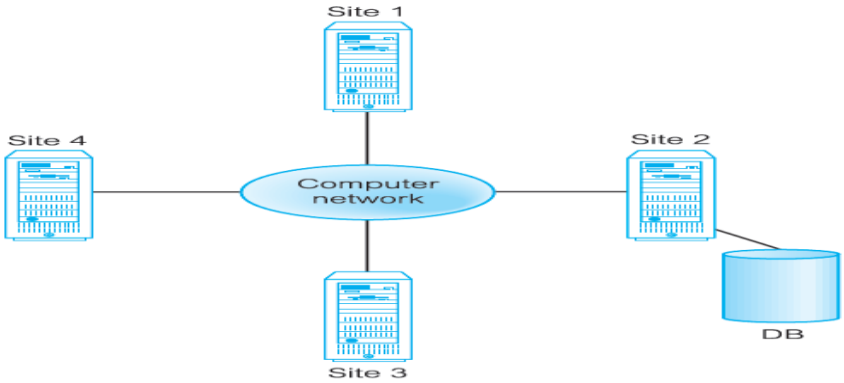


Figure 2 Distributed processing.

Advantages of DDBMSs:

1. Reflects organizational structure.
2. Improved shareability and local autonomy.
3. Improved availability.
4. Improved reliability.
5. Improved performance.
6. Economics.
7. Modular growth.

Disadvantages of DDBMSs:

1. Complexity.
2. Cost.
3. Security.
4. Integrity control more difficult.
5. Lack of standards.
6. Lack of experience.
7. Database design more complex.

Types of DDBMSs

- A DDBMS may be classified as homogeneous or heterogeneous. In a homogeneous system, all sites use the same DBMS product.
- In a heterogeneous system, sites may run different DBMS products, which need not be based on the same underlying data model, and so the system may be composed of relational, network, hierarchical, and object-oriented DBMSs.
- Homogeneous systems are much easier to design and manage.
- This approach provides incremental growth, making the addition of a new site to the DDBMS easy, and allows increased performance by exploiting the parallel processing capability of multiple sites.
- Heterogeneous systems usually result when individual sites have implemented their own databases and integration is considered at a later stage.
- In a heterogeneous system, translations are required to allow communication between different DBMSs.
- To provide DBMS transparency, users must be able to make requests in the language of the DBMS at their local site.
- The system then has the task of locating the data and performing any necessary translation.
  - **Homogeneous DDBMS.**
  - **Heterogeneous DDBMS.**

Homogeneous DDBMS:

1. All sites use same dbms product.
2. Much easier to design and manage.
3. Approach provides incremental growth.

4. Allows increased performance.

Heterogeneous DDBMS:

- Sites may run different DBMS products, with possibly different underlying data models.
- When individual sites have implemented their own databases and integration is considered later stage.

Translations are required to allow communication between different DBMS.

Data may be required from another site that may have:

- Different hardware.
- Different DBMS products.
- Different hardware and different DBMS products.

Open Database Access and Interoperability

- Open Group formed a Working Group to provide specifications that will create a database infrastructure environment where there is:
- Common SQL API that allows client applications to be written that do not need to know vendor of DBMS they are accessing.
- Common database protocol that enables DBMS from one vendor to communicate directly with DBMS from another vendor without the need for a gateway.
- A common network protocol that allows communications between different DBMSs.
- Most ambitious goal is to find a way to enable transaction to span DBMSs from different vendors without use of a gateway.
- Group has now evolved into DBIOP Consortium and are working in version 3 of DRDA (Distributed Relational Database Architecture) standard.

Overview of networking:

An interconnected collection of autonomous computers that are capable of exchanging

information

Table 22.2 Summary of WAN and LAN characteristics.

WAN	LAN
Distances up to thousands of kilometers	Distances up to a few kilometers
Link autonomous computers	Link computers that cooperate in distributed applications
Network managed by independent organization (using telephone or satellite links)	Network managed by users (using privately owned cables)
Data rate up to 33.6 kbit/s (dial-up via modem), 45 Mbit/s (T3 circuit)	Data rate up to 2500 Mbit/s (ATM)
Complex protocol	Simpler protocol
Use point-to-point routing	Use broadcast routing
Use irregular topology	Use bus or ring topology
Error rate about $1{:}10^{5}$	Error rate about $1{:}10^{9}$

Network protocols

A set of rules that determines how messages between computers are sent, interpreted, and processed.

TCP/IP (Transmission Control Protocol/Internet Protocol)

- TCP/IP is a routable protocol, which means that all messages contain not only the address of the destination station, but also the address of a destination network.
- This allows TCP/IP messages to be sent to multiple networks within an organization or around the world, hence its use in the Internet.

SPX/IPX (Sequenced Packet Exchange/Internetwork Package Exchange)

- Novell created SPX/IPX as part of its NetWare operating system. Similar to TCP,SPX ensures that an entire message arrives intact but uses NetWare's IPX protocol as itsdelivery mechanism.
- Like IP, IPX handles routing of packets across the network. UnlikeIP, IPX uses an 80-bit address space, with a 32-bit network portion and a 48-bit host portion(this is much larger than the 32-bit address used by IP)

WAP (Wireless Application Protocol)

- A standard for providing cellular phones, pagers, and other handheld devices with secureaccess to e-mail and text-based Web pages.
- Introduced in 1997 by Phone.com (formerlyUnwired Planet), Ericsson, Motorola, and Nokia, WAP provides a complete environmentfor wireless applications that includes a wireless counterpart of TCP/IP and a frameworkfor telephony integration such as call control and phone book access.

Functions and Architectures of a DDBMS

- Extended communication services to provide access to remote sites and allow the transfer of queries and data among the site using a network.
- Extended system catalog to store data distribution details.
- Distributed query processing including query optimization and remote data access.
- Extended security control to maintain appropriate authorization/access privileges to the distributed data.
- Extended concurrency control to maintain consistency of replicated data.

Reference Architecture for a DDBMS

Reference architecture consists of:

- Set of global external schemas.
- Global conceptual schema (GCS).
- Fragmentation schema .
- Allocation schema.
- Set of schemas for each local DBMS conforming to 3-level ANSI/SPARC.

- Some levels may be missing, depending on levels of transparency supported.

Global conceptual schema:

- The global conceptual schema is logical description of the whole database.
- It provides physical data independence from the distributed environment.

Global External Schema:

- It provides logical data independence from the distributedenvironment.

Fragmentation schema :

- The fragmentation schema is a description of how the data is to be logically partitioned.

Allocation schema:

- The allocation schema is a description of where the data is to be located,taking account of any replication.

Local Schemas:

- Each local DBMS has it own set of Schemas.
- Its is DBMS independent and is the basis for supporting heterogeneous DBMS.

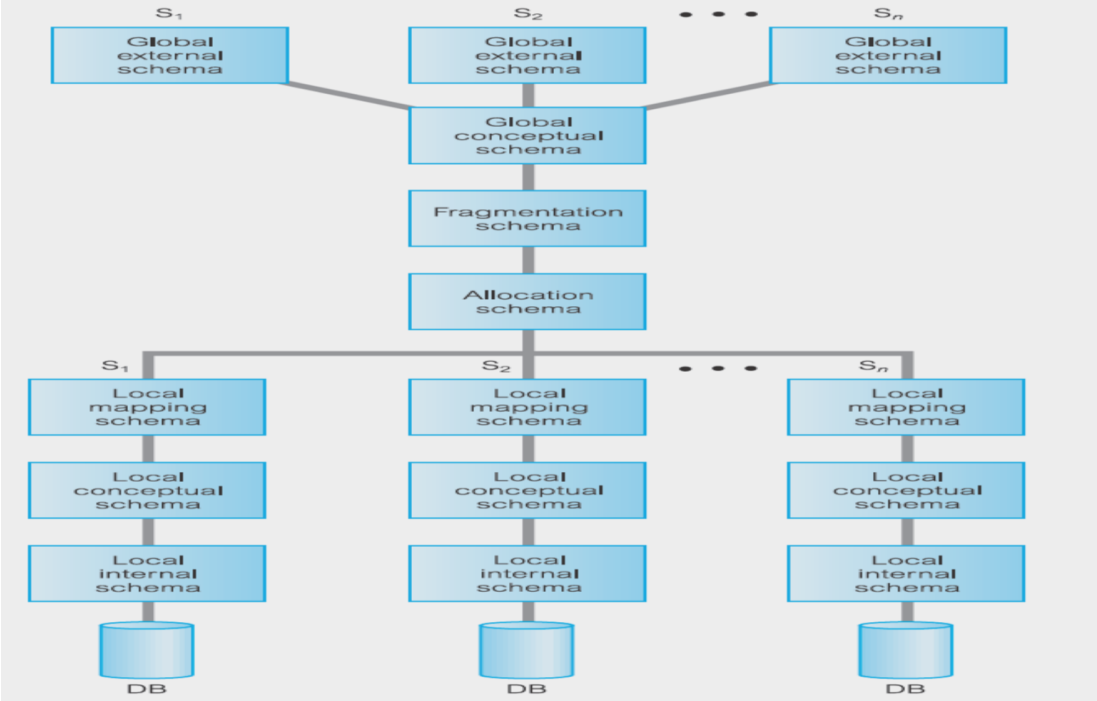
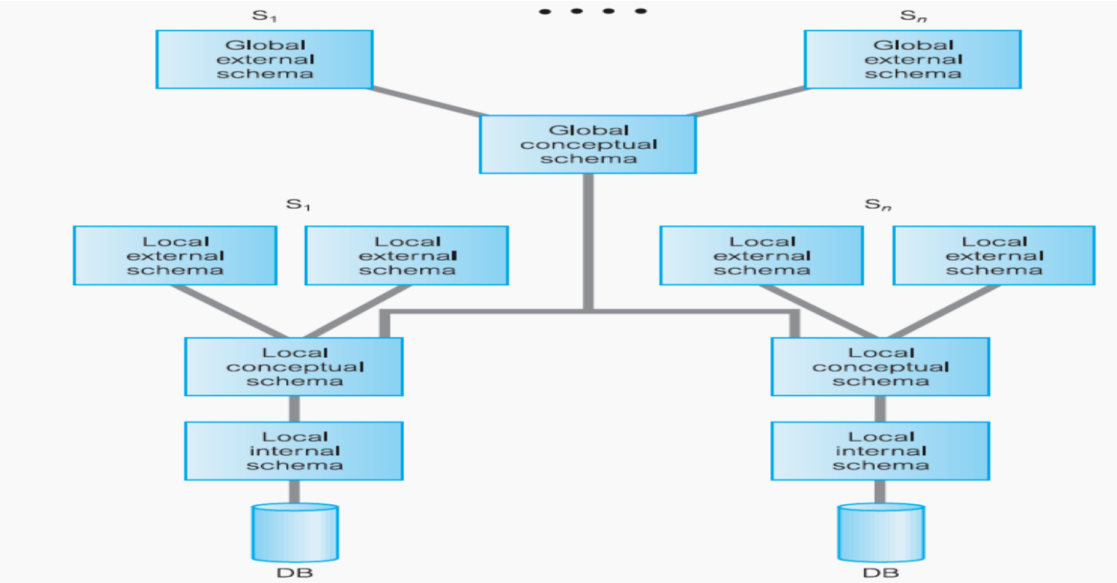


Figure Referencearchitecture fora DDBMS.

Reference Architecture for MDBS

- In DDBMS, GCS is union of all local conceptual schemas.
- In FMDBS, GCS is subset of local conceptual schemas (LCS), consisting of data that each local system agrees to share.

- GCS of tightly coupled system involves integration of either parts of LCSs or local external schemas.
- FMDBS with noGCS called loosely coupled.



Local DBMS component:

- The LDBMS component is a standard DBMS, responsible for controlling the local data at each site that has a database.
- It has its own local system catalog that stores information about the data held at that site.
- In a homogeneous system, the LDBMS component is the same product, replicated at each site.
- In a heterogeneous system, there would be at least two sites with different DBMS products and/or platforms.

Data communications component:

- The DC component is the software that enables all sites to communicate with each other.
- The DC component contains information about the sites and the links.

Global system catalog:

- The GSC has the same functionality as the system catalog of a centralized system.
- The GSC holds information specific to the distributed nature of the system, such as the fragmentation, replication, and allocation schemas.

Distributed DBMS component:

The DDBMS component is the controlling unit of the entire system.

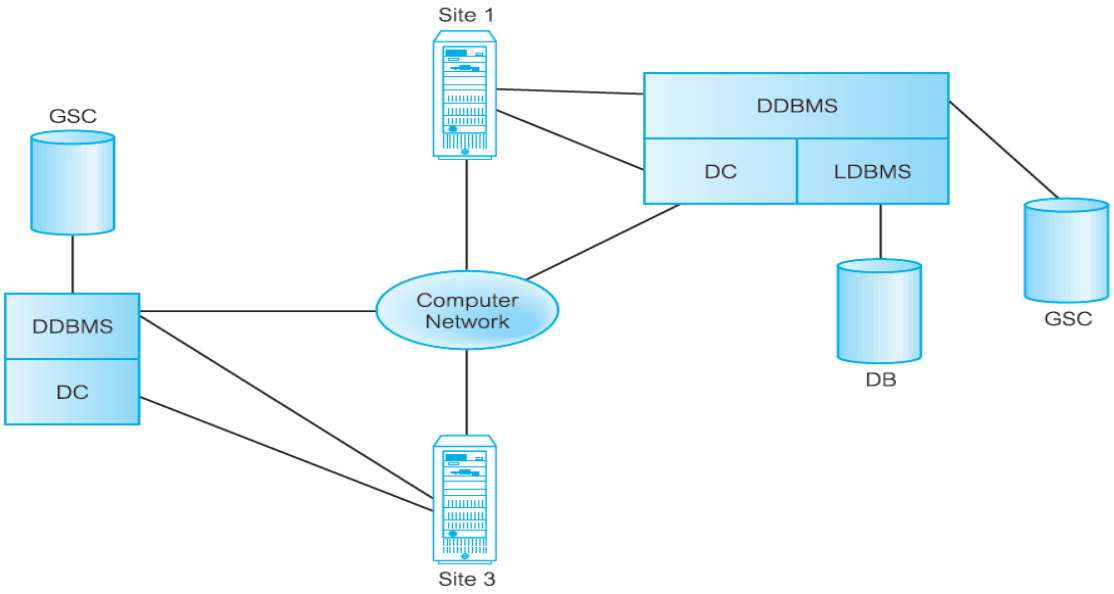
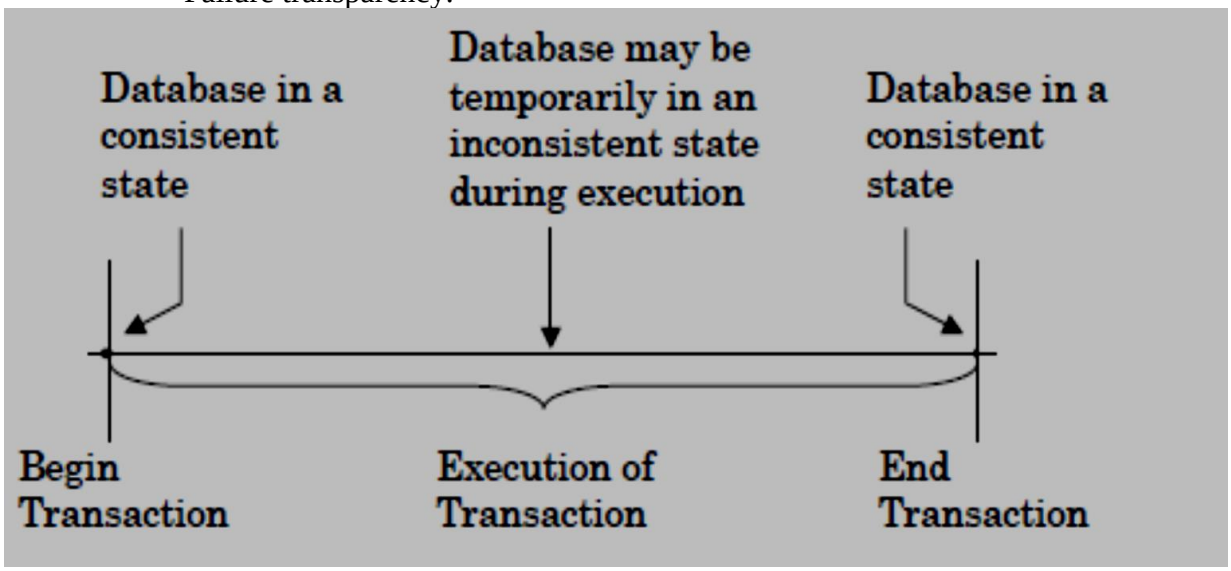


Figure: Components ofa DDBMS.

TRANSACTION PROCESSING

Transaction:

- A transaction is a collection of actions that make consistent transformations of system states while preserving system consistency.
  - Concurrency transparency.
  - Failure transparency.



DISTRIBUTED TRANSACTION:

- Consider a transaction T that prints out the names of all staff, using the fragmentation schema defined above as S1, S2, S21, S22, and S23.
- We can define three sub transactions TS3, TS5, and TS7 to represent the agents at sites 3, 5, and 7, respectively.
- Each sub transaction prints out the names of the staff at that site.

Time	T_{S_3}	T_{S_5}	T_{S_7}
t_1	begin_transaction	begin_transaction	begin_transaction
t_2	read(fName, lName)	read(fName, lName)	read(fName, lName)
t_3	print(fName, lName)	print(fName, lName)	print(fName, lName)
t_4	end_transaction	end_transaction	end_transaction

Classification of transactions:

- In DRDA, there are four types of transaction, each with a progressive level of complexity in the interaction between the DBMSs:
  (1) Remote request;
  (2) Remote unit of work;
  (3) Distributed unit of work;
  (4) Distributed request.

Remote request

- An application at one site can send a request (SQL statement) to some remote site for execution.
- The request is executed entirely at the remote site and can reference data only at the remote site.

Remote unit of work

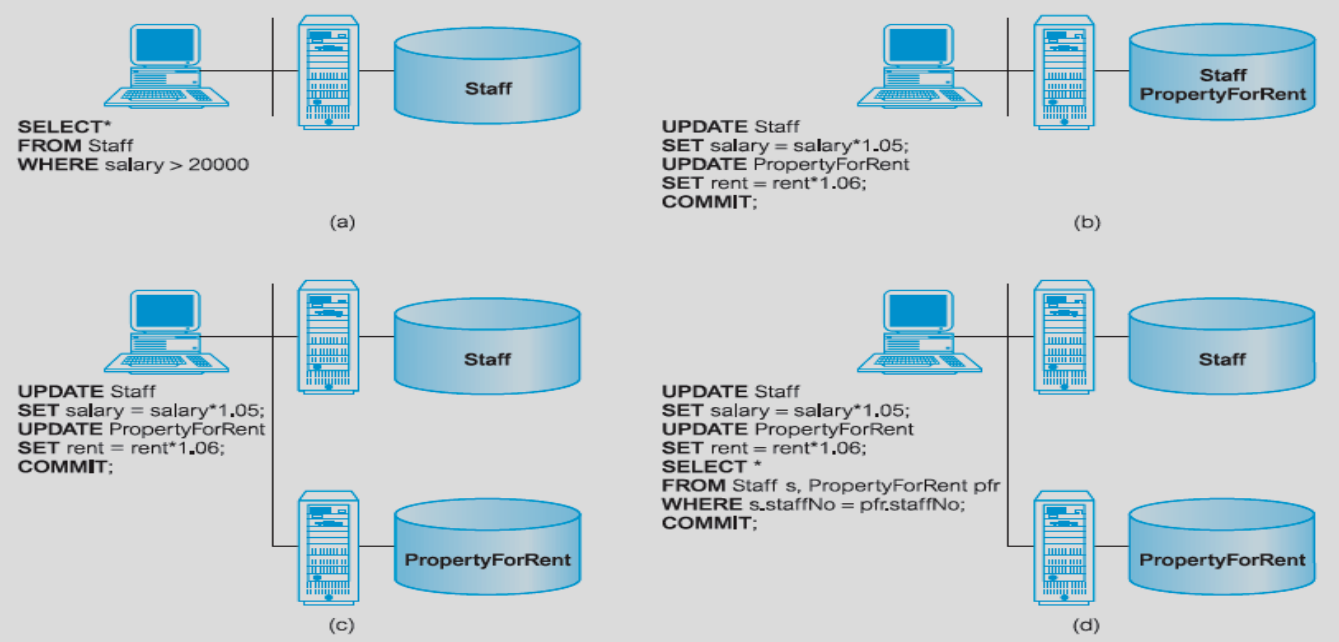
- An application at one (local) site can send all the SQL statements in a unit of work (transaction) to some remote site for execution.
- All SQL statements are executed entirely at the remote site and can only reference data at the remote site.

Distributed unit of work :

- An application at one (local) site can send some of or all the SQL statements in a transaction to one or more remote sites for execution.
- Each SQL statement is executed entirely at the remote site and can only reference data at the remote site.

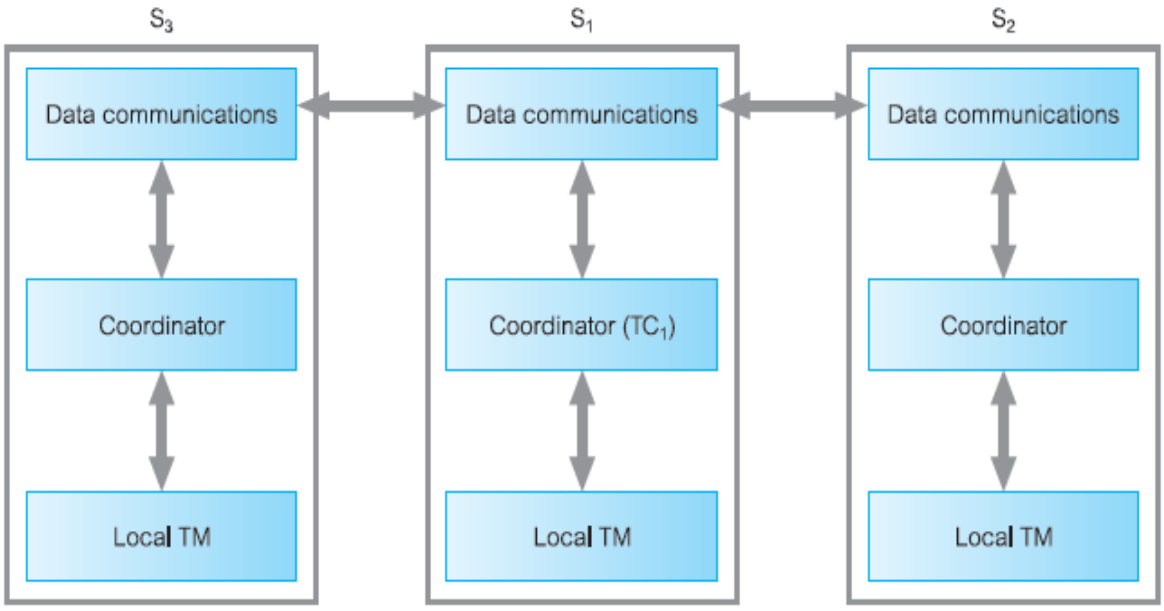
Distributed request:

- An application at one (local) site can send some of or all the SQL statements in a transaction to one or more remote sites for execution.



Distributed Transaction Management:

- The transaction coordinator (TC1) at site S1 divides the transaction into a number of
- subtransactions using information held in the global system catalog.
- The data communications component at site S1 sends the subtransactions to the appropriate sites, S2 and S3, say.
- The transaction coordinators at sites S2 and S3 manage these subtransactions.
- The results of subtransactions are communicated back to TC1 via the data communications components.



CONCURRENCY CONTROL

- The problem of synchronizing concurrent transactions such that the consistency of the database is maintained while, at the same time, maximum degree of concurrency is achieved.

Anomalies

- Lost updates:
  - The effects of some transactions are not reflected on the database.
- Inconsistent retrievals:
  - A transaction, if it reads the same data item more than once, should always read the same value.
- Extends centralised concurrency mechanisms
- Multiple copies of data items
  - maintain consistency
- failures in individual sites/network
  - continue operations, update and rejoin
- distributed commit
  - 2-phase protocol (local and global)

Distributed Locks

- The following protocols based on two-phase locking (2PL) that can be employed to ensure serializability for distributed DBMS.
- Just like centralised mechanisms.
- But we need to consider locks that manage replication and sub-transactions
  - **Centralized 2PL,**

- **Primary copy 2PL**
- **Distributed 2PL,**
- **Majority locking.**

Centralized 2PL:

- With the centralized 2PL protocol there is a single site that maintains all locking information.
- There is only one scheduler, or lock manager, for the whole of the distributed DBMS that can grant and release locks.
- The transaction coordinator at site S1 divides the transaction into a number of subtransactions, using information held in the global system cat log.
- The coordinator has responsibility for ensuring that consistency is maintained.
- If the transaction involves an update of a data item that is replicated, the coordinator must ensure that all copies of the data item are updated.
- The coordinator requests exclusive locks on all copies before updating each copy and releasing the locks.
- The local transaction managers involved in the global transaction request and release locks from the centralized lock manager using the normal rules for two-phase locking.
- The centralized lock manager checks that a request for a lock on a data item is compatible with the locks that currently exist

The advantage of centralized 2PL

- Deadlock detection is no more difficult than that of a centralized DBMS, because one lock manager maintains all lock information.

The disadvantages of centralized 2PL

- As all lock requests go to one central site, that site may become a bottleneck.
- The system may also be less reliable since the failure of the central site would cause major system failures.

Example

- A global update operation that has agents (subtransactions) at n sites may require a minimum of 2n + 3 messages with a centralized lock manager:
  - 1 lock request;
  - 1 lock grant message;
  - n update messages;

- n acknowledgements;

  1 unlock request

Primary copy 2PL:

- This protocol attempts to overcome the disadvantages of centralized 2PL by distributing the lock managers to a number of sites.
- Each lock manager is then responsible for managing the locks for a set of data items.
- For each replicated data item, one copy is chosen as the primary copy; the other copies are called slave copies.
- Only primary copy locked for updates, slaves updated later.

Distributed 2PL:

- This protocol again attempts to overcome the disadvantages of centralized 2PL, this time by distributing the lock managers to every site.
- Each lock manager is then responsible for managing the locks for the data at that site.
- If the data is not replicated, this protocol is equivalent to primary copy 2PL.
- distributed 2PL implements a Read-One-Write-All (ROWA) replica control protocol. This means that any copy of a replicated item can be used for a read operation, but all copies must be exclusively locked before an item can be updated.
- This scheme deals with locks in a decentralized manner, thus avoiding the drawbacks of centralized control.

Disadvantages:

- This approach are that deadlock handling is more complex owing to multiple lock managers and that communication costs are higher than primary copy 2PL, as all items must be locked before update.

Example:

- A global update operation that has agents at n sites, may require a minimum of 5n messages with this protocol:

  - **n lock request messages;**
  - **n lock grant messages;**
  - **n update messages;**
  - **n acknowledgements;**
  - **n unlock requests.**

Majority Locking:

  - Extension of distributed 2PL
  - Doesn't lock all copies before update
  - The system maintains a lock manager at each site to manage the locks for all data at that site.
- When a transaction wishes to read or write an data item that is replicated at n sites
- Needs more than half of locks on a copy to proceed
- If so, it informs other sites.
- Otherwise it cancels request.

- Only one transaction with an exclusive lock.
- Many transactions can hold a majority lock on a shared lock.

Timestamp Protocols:

<15,1>

15=Current Time.

1=Site number.

Distributed deadlock:

- Any locking-based concurrency control algorithm (and some timestamp-based algorithms that require transactions to wait) may result in deadlocks.
- In a distributed environment, deadlock detection may be more complicated if lock management is not centralized,

Example:

Consider three transactions T1, T2, and T3 with:

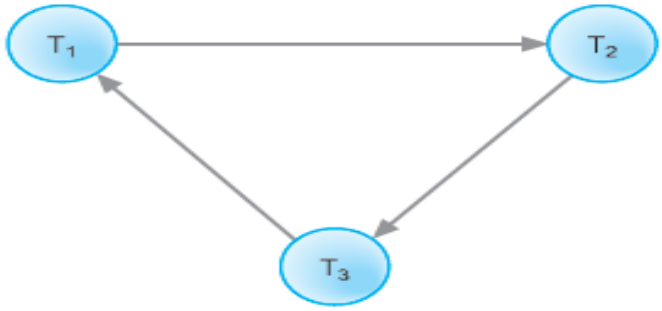
  - T1 initiated at site S1 and creating an agent at site S2;
  - T2 initiated at site S2 and creating an agent at site S3;
  - T3 initiated at site S3 and creating an agent at site S1.
- The transactions set shared (read) and exclusive (write) locks as illustrated below, where **read_lock(Ti, xj)** denotes a **shared lock** by transaction **Ti** on data item **xj** and **write_lock(Ti, xj)** denotes an **exclusive lock** by transaction **Ti** on data item **xj.**

TIME	S1	S2	S3
t1	read_lock(T1, x1)	write_lock(T2, y2)	read_lock(T3, z3)
t2	write_lock(T1, y1)	write_lock(T2, z2)	
t3	write_lock(T3, x1)	write_lock(T1, y2)	write_lock(T2, z3)

T1 → T2 → T3 → T1

- We can construct the wait-for graphs (WFGs) for each site.

Wait-for graphs for sites S1, S2, and S3.



Combined wait-for graphs for sites S1 S2, and S3.

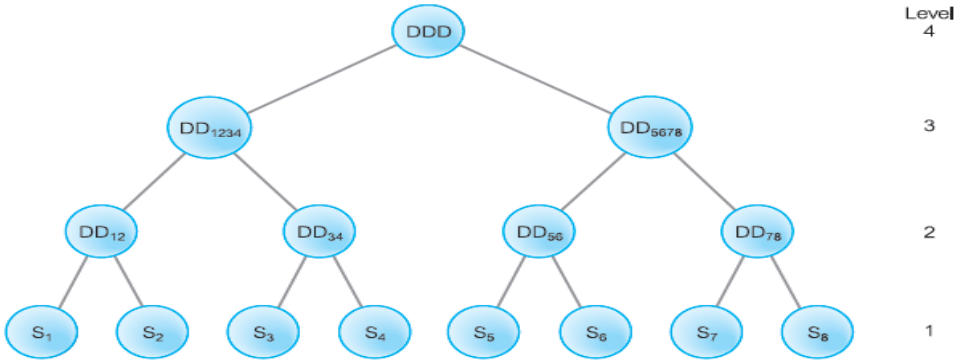
- There are three common methods for handling deadlock detection in DDBMSs:
  - **Centralized deadlock detection.**
  - **Hierarchical deadlock detection.**
  - **Distributed deadlock detection.**

Centralized deadlock detection:

- With centralized deadlock detection, a single site is appointed as the Deadlock Detection Coordinator (DDC).
- The DDC has the responsibility of constructing and maintaining the global WFG. Periodically, each lock manager transmits its local WFG to the DDC.
- The DDC builds the global WFG and checks for cycles in it.
- If one or more cycles exist, the DDC must break each cycle by selecting the transactions to be rolled back and restarted.

Hierarchical deadlock detection.

- With hierarchical deadlock detection, the sites in the network are organized into a hierarchy.
- Each site sends its local WFG to the deadlock detection site above it in the hierarchy.

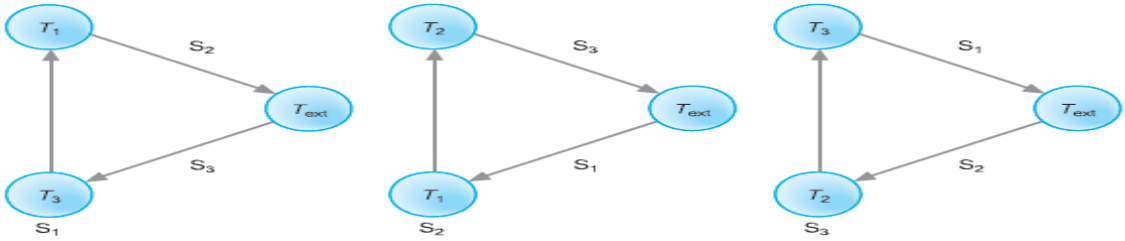


- Figure illustrates a possible hierarchy for eight sites,
- S1 to S8. The level 1 leaves are the sites themselves, where local deadlock detection is performed.
- The level 2 nodes DDij detect deadlock involving adjacent sites i and j.
- The level 3 nodes detect deadlock between four adjacent sites.
- The root of the tree is a global deadlock detector that would detect deadlock between, for example, sites S1 and S8.

Advantages:

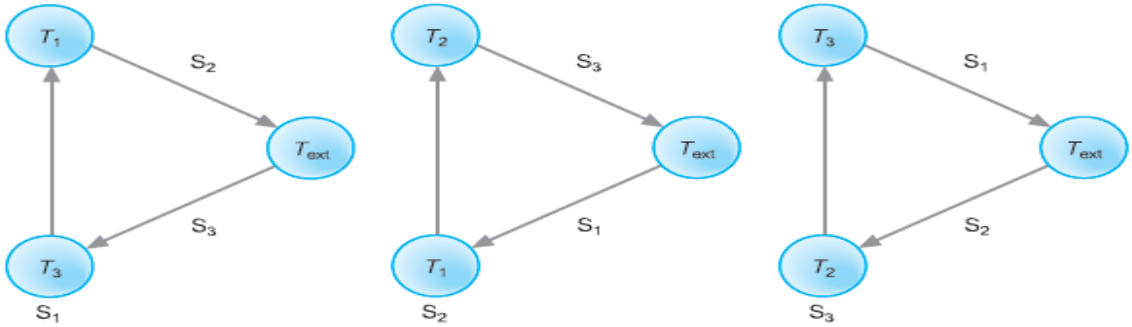
- The hierarchical approach reduces the dependence on a centralized detection site, thereby reducing communication costs.

Distributed deadlock detection:

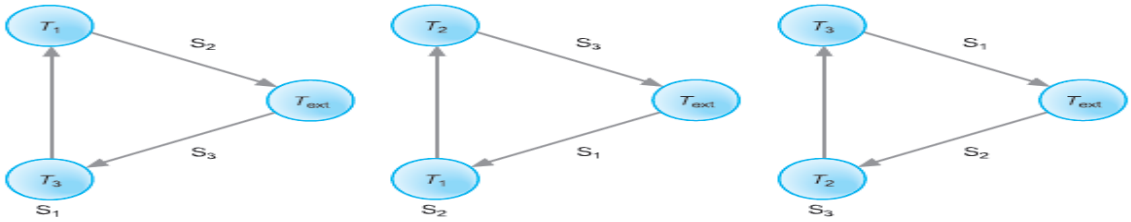


Distributed deadlock detection:

- The three local WFGs in Figure contain cycles:
  - S1: Text → T3 → T1 → Text
  - S2: Text → T1 → T2 → Text
  - S3: Text → T2 → T3 → Text



- In this example, we could transmit the local WFG for site S1 to the site for which transaction T1 is waiting: that is, site S2. The local WFG at S2 is extended to include this information and becomes:
- **S2: Text → T3 → T1 → T2 → Text**
- This still contains a potential deadlock, so we would transmit this WFG to the site for
- which transaction T2 is waiting: that is, site S3. The local WFG at S3 is extended to:
- **S3: Text → T3 → T1 → T2 → T3 → Text**



This global WFG contains a cycle that does not involve the Text node, so we can conclude that deadlock exists and an appropriate recovery protocol must be invoked.

QUERY OPTIMIZATION

In a centralized system, query processing is done with the following aim −

- Minimization of response time of query (time taken to produce the results to user's query).
- Maximize system throughput (the number of requests that are processed in a given amount of time).
- Reduce the amount of memory and storage required for processing.
- Increase parallelism.

Query Parsing and Translation

Initially, the SQL query is scanned. Then it is parsed to look for syntactical errors and correctness of data types. If the query passes this step, the query is decomposed into smaller query blocks. Each block is then translated to equivalent relational algebra expression.

Steps for Query Optimization

Query optimization involves three steps, namely query tree generation, plan generation, and query plan code generation.

Step 1 − Query Tree Generation

A query tree is a tree data structure representing a relational algebra expression. The tables of the query are represented as leaf nodes. The relational algebra operations are represented as the internal nodes. The root represents the query as a whole.

During execution, an internal node is executed whenever its operand tables are available. The node is then replaced by the result table. This process continues for all internal nodes until the root node is executed and replaced by the result table.

For example, let us consider the following schemas −

EMPLOYEE

EmpID	Ename	Salary	DeptNo	DateOfJoining

DEPARTMENT

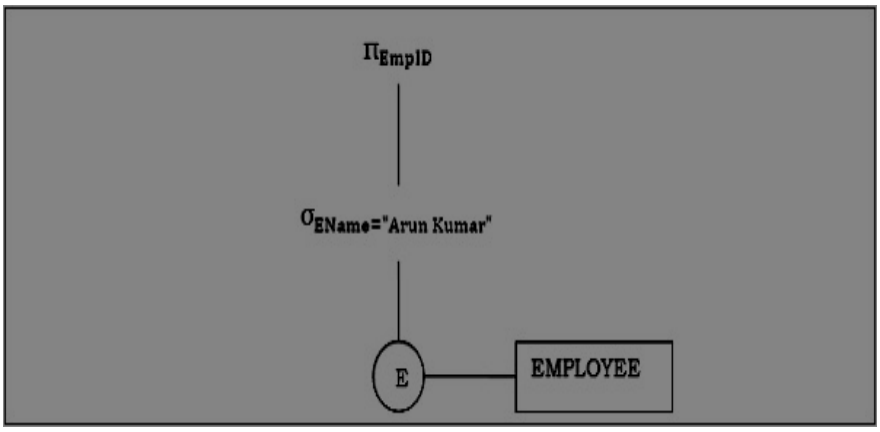
Dno	Dname	Location

Example 1

Let us consider the query as the following.

Π EmpID (σEName="ArunKumar"(EMPLOYEE))

The corresponding query tree will be −

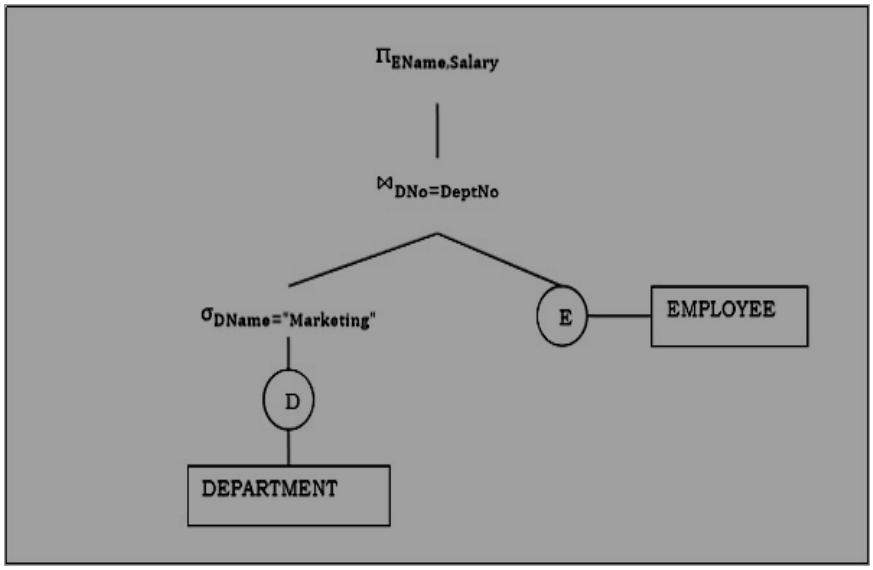


Example 2

Let us consider another query involving a join.

πEName, Salary(σDName="Marketing"(DEPARTMENT))⋈DNo=DeptNo(EMPLOYEE)

Following is the query tree for the above query.



Step 2 − Query Plan Generation

- After the query tree is generated, a query plan is made. A query plan is an extended query tree that includes access paths for all operations in the query tree. Access paths specify how the relational operations in the tree should be performed.
- For example, a selection operation can have an access path that gives details about the use of B+ tree index for selection.
- Besides, a query plan also states how the intermediate tables should be passed from one operator to the next, how temporary tables should be used and how operations should be pipelined/combined.

Step 3− Code Generation

- Code generation is the final step in query optimization. It is the executable form of the query, whose form depends upon the type of the underlying operating system. Once the query code is generated, the Execution Manager runs it and produces the results.

Approaches to Query Optimization

- Among the approaches for query optimization, exhaustive search and heuristics-based algorithms are mostly used.

Exhaustive Search Optimization

- In these techniques, for a query, all possible query plans are initially generated and then the best plan is selected. Though these techniques provide the best solution, it has an exponential time and space complexity owing to the large solution space. For example, dynamic programming technique.

Heuristic Based Optimization

- Heuristic based optimization uses rule-based optimization approaches for query optimization. These algorithms have polynomial time and space complexity, which is lower than the exponential complexity of exhaustive search-based algorithms. However, these algorithms do not necessarily produce the best query plan.

Some of the common heuristic rules are −

- Perform select and project operations before join operations. This is done by moving the select and project operations down the query tree. This reduces the number of tuples available for join.
- Perform the most restrictive select/project operations at first before the other operations.
- Avoid cross-product operation since they result in very large-sized intermediate tables.

MAPREDUCE MODEL

Introduction to MapReduce Algorithm

- MapReduce is a Distributed Data Processing Algorithm, introduced by Google in its MapReduce.
- MapReduce Algorithm is mainly inspired by Functional Programming model.
- MapReduce algorithm is mainly useful to process huge amount of data in parallel, reliable and efficient way in cluster environments.
- It uses Divide and Conquer technique to process large amount of data.
- It divides input task into smaller and manageable sub-tasks (They should be executable independently) to execute them in-parallel.

MapReduce Algorithm Steps

MapReduce Algorithm uses the following three main steps:

1. Map Function.
2. Shuffle Function.
3. Reduce Function.

Map Function:

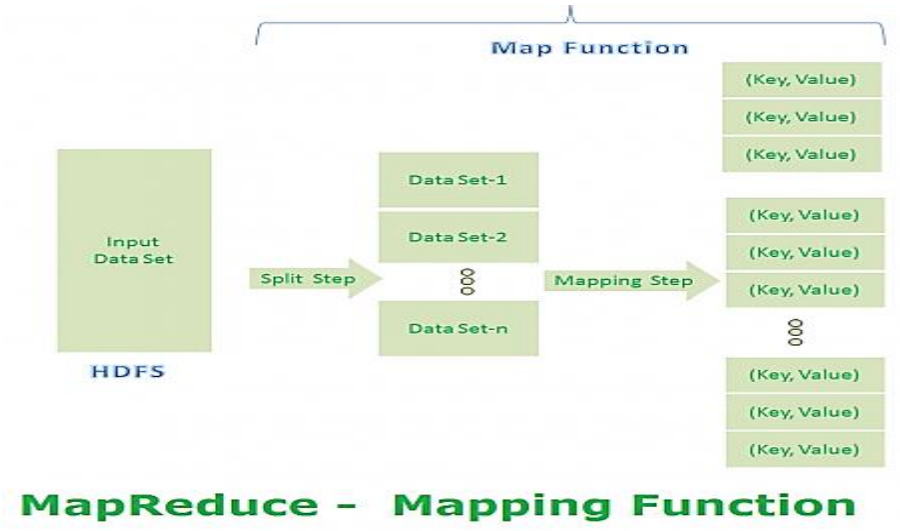
- Map Function is the first step in MapReduce Algorithm.
- It takes input tasks (say DataSets. I have given only one DataSet in below diagram.) and divides them into smaller sub-tasks.
- Then perform required computation on each sub-task in parallel.

This step performs the following two sub-steps:

1. Splitting
2. Mapping

- Splitting step takes input DataSet from Source and divide into smaller Sub-DataSets.
- Mapping step takes those smaller Sub-DataSets and perform required action or computation on each Sub-DataSet.

The output of this Map Function is a set of key and value pairs as <Key, Value> as shown in the below diagram.



MapReduce First Step Output:

ShuffleFunction

It is the second step in MapReduce Algorithm. Shuffle Function is also known as "Combine Function".

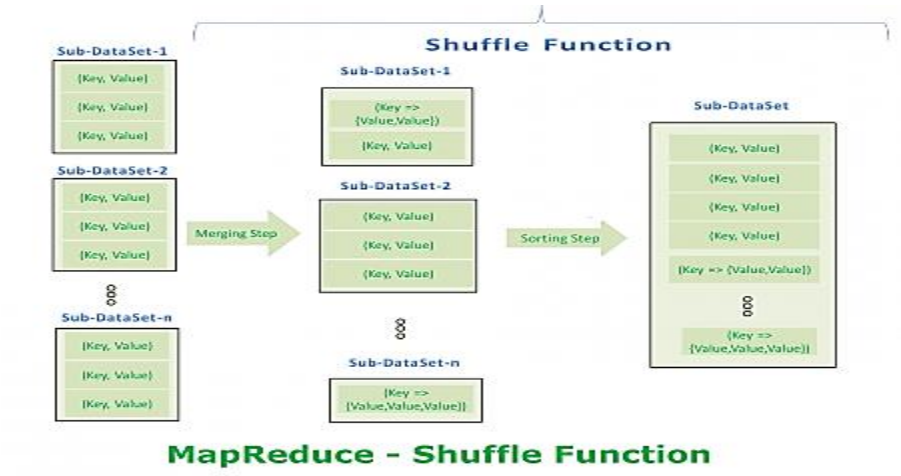
It performs the following two sub-steps:

1. Merging
2. Sorting

It takes a list of outputs coming from "Map Function" and perform these two sub-steps on each and every key-value pair.

- Merging step combines all key-value pairs which have same keys (that is grouping key-value pairs by comparing "Key"). This step returns <Key, List<Value>>.
- Sorting step takes input from Merging step and sort all key-value pairs by using Keys. This step also returns <Key, List<Value>> output but with sorted key-value pairs.

Finally, Shuffle Function returns a list of <Key, List<Value>> sorted pairs to next step.



MapReduce - Shuffle Function

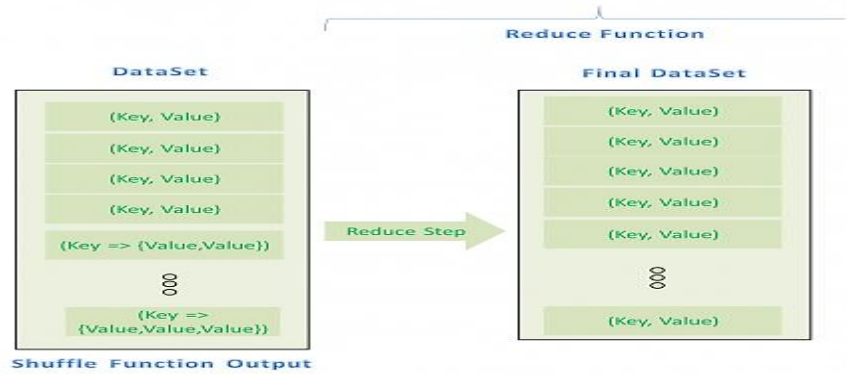
MapReduce Second Step Output:

Shuffle Function Output = List of <Key, List<Value>> Pairs

ReduceFunction:

It is the final step in MapReduce Algorithm. It performs only one step : Reduce step.

It takes list of <Key, List<Value>> sorted pairs from Shuffle Function and perform reduce operation as shown below.



MapReduce - Reduce Function

MapReduce Final Step Output:

Reduce Function Output = List of <Key, Value> Pairs

- Final step output looks like first step output. However final step <Key, Value> pairs are different than first step <Key, Value> pairs. Final step <Key, Value> pairs are computed and sorted pairs.

- We can observe the difference between first step output and final step output with some simple example. We will discuss same steps with one simple example in next section.
- That's it all three steps of MapReduce Algorithm.

How MapReduce Algorithm Works With WordCount Example

"How MapReduce Algorithm solves WordCount Problem" theoretically.

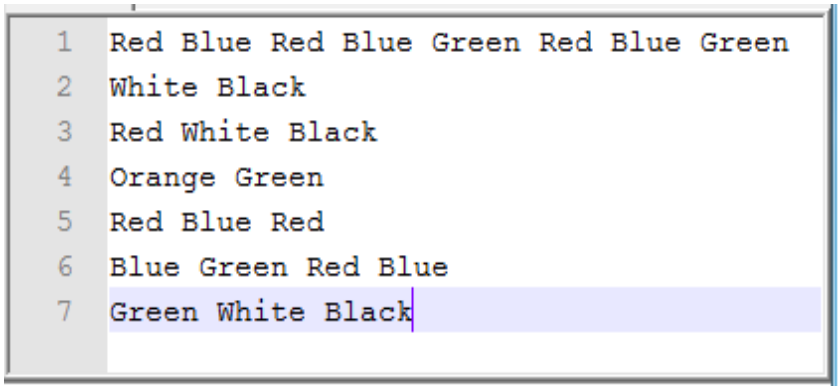
We will implement a Hadoop MapReduce Program and test it in my coming post.

ProblemStatement:

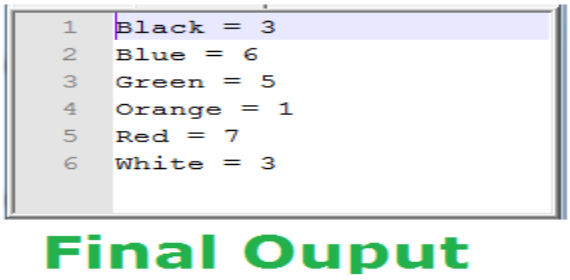
Count the number of occurrences of each word available in a DataSet.

InputDataSet

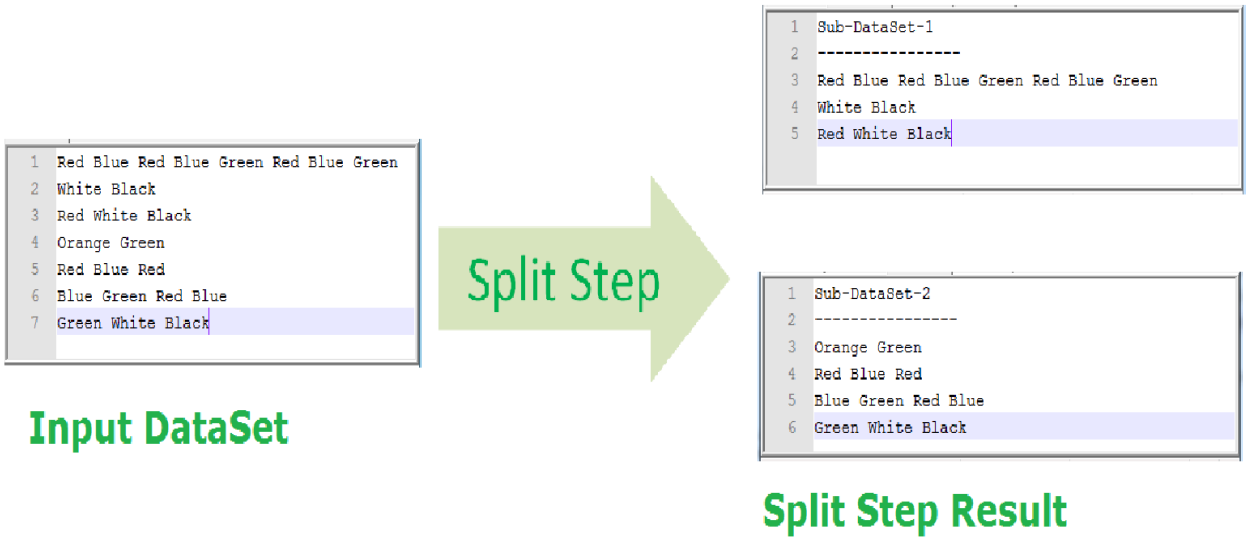
Please find our example Input DataSet file in below diagram. Just for simplicity, we are going to use simple small DataSet. However, Real-time applications use very huge amount of Data.



Client Required Final Result:

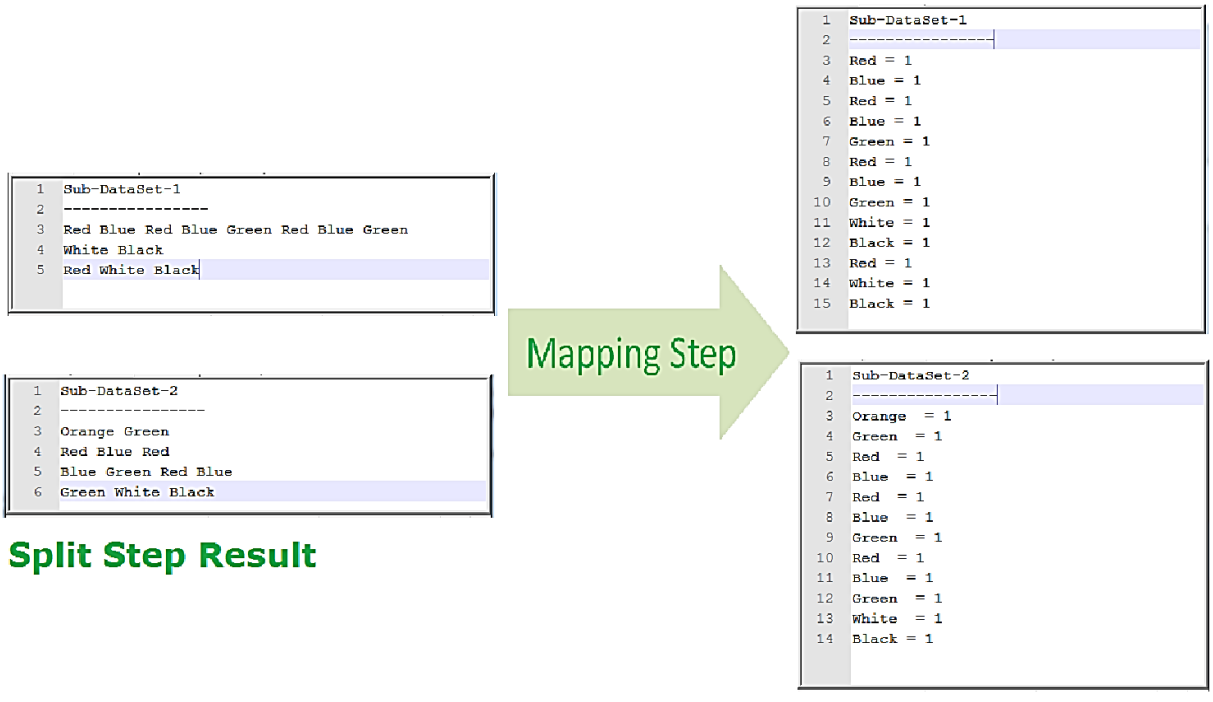


MapReduce – Map Function (Split Step)



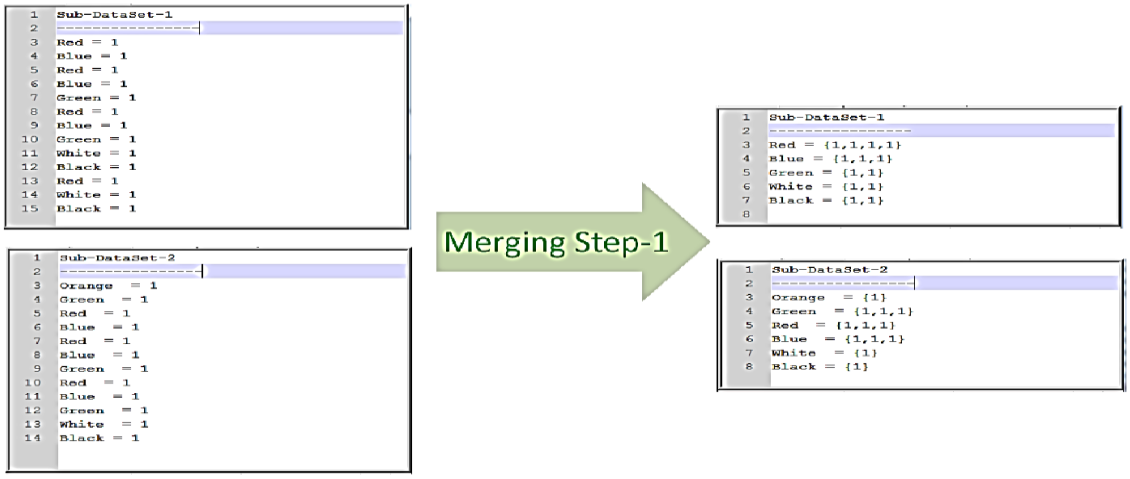
MapReduce - Split Step

MapReduce – Map Function (Mapping Step)

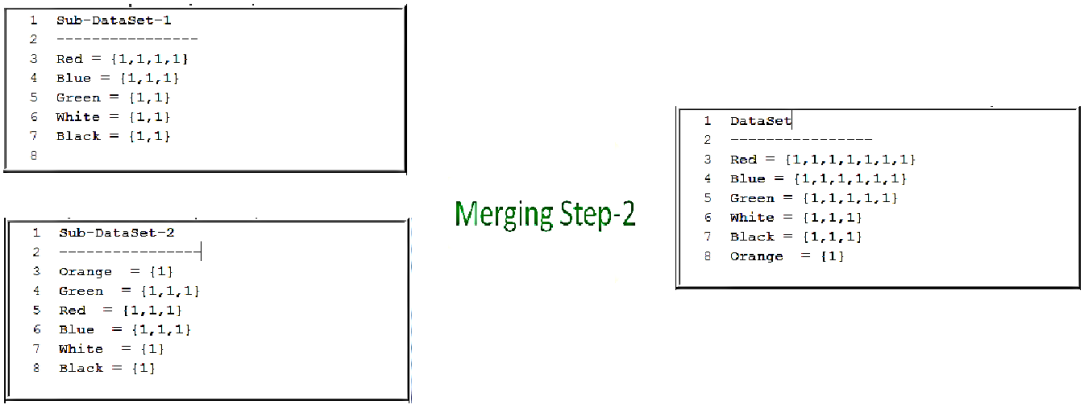


MapReduce - Mapping Step

MapReduce – Shuffle Function (Merge Step)



MapReduce - Merging Step 1

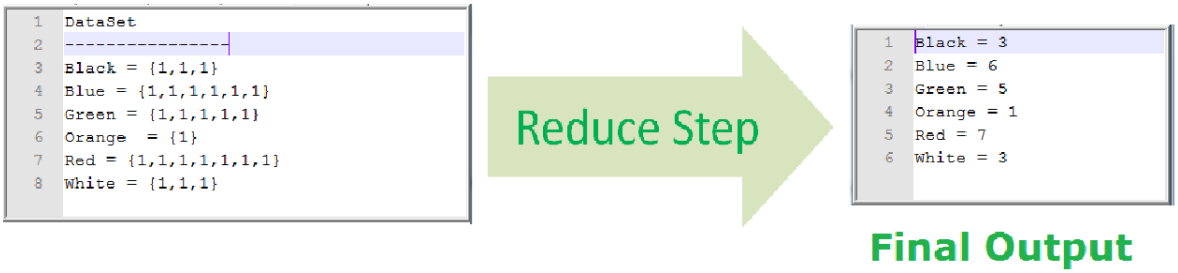


MapReduce - Merging Step 2

MapReduce – Shuffle Function (Sorting Step)

MapReduce - Sorting Step

MapReduce – Reduce Function (Reduce Step)



MapReduce - Reduce Step

What is Apache Pig?

- Apache Pig is an abstraction over Map Reduce. It is a tool/platform which is used to analyze larger sets of data representing them as data flows.
- Pig is generally used with Hadoop; we can perform all the data manipulation operations in Hadoop using Apache Pig.
- To write data analysis programs, Pig provides a high-level language known as Pig Latin. This language provides various operators using which programmers can develop their own functions for reading, writing, and processing data.
- To analyze data using Apache Pig, programmers need to write scripts using Pig Latin language.
- All these scripts are internally converted to Map and Reduce tasks.
- Apache Pig has a component known as Pig Engine that accepts the Pig Latin scripts as input and converts those scripts into Map Reduce jobs.

Why Do We Need Apache Pig?

Programmers who are not so good at Java normally used to struggle working with Hadoop, especially while performing any Map Reduce tasks.

Apache Pig is a boon for all such programmers.

- Using **Pig Latin**, programmers can perform Map Reduce tasks easily without having to type complex codes in Java.
- Apache Pig uses **multi-query approach**, thereby reducing the length of codes.
- For example, an operation that would require you to type 200 lines of code (LoC) in Java can be easily done by typing as less as just 10 LoC in Apache Pig. Ultimately Apache Pig reduces the development time by almost 16 times.
- Pig Latin is **SQL-like language** and it is easy to learn Apache Pig when you are familiar with SQL.
- Apache Pig provides many built-in operators to support data operations like joins, filters, ordering, etc.
- In addition, it also provides nested data types like tuples, bags, and maps that are missing from MapReduce.

Apache Pig	MapReduce
Apache Pig is a data flow language.	MapReduce is a data processing paradigm.
It is a high level language.	MapReduce is low level and rigid.
Performing a Join operation in Apache Pig is pretty simple.	It is quite difficult in MapReduce to perform a Join operation between datasets.
Any novice programmer with a basic knowledge of SQL can work conveniently with Apache Pig.	Exposure to Java is must to work with MapReduce.
Apache Pig uses multi-query approach, thereby reducing the length of the codes to a great extent.	MapReduce will require almost 20 times more the number of lines to perform the same task.
There is no need for compilation. On execution, every Apache Pig operator is converted internally into a MapReduce job.	MapReduce jobs have a long compilation process.

PIG	SQL
Pig Latin is a **procedural** language.	SQL is a **declarative** language.
In Apache Pig, **schema** is optional. We can store data without designing a schema (values are stored as $01, $02 etc.)	Schema is mandatory in SQL.
The data model in Apache Pig is **nested relational**.	The data model used in SQL **is flat relational**.
Apache Pig provides limited opportunity for **Query optimization**.	There is more opportunity for query optimization in SQL.

Features of Pig

Apache Pig comes with the following features −

- **Rich set of operators** − It provides many operators to perform operations like join, sort, filer, etc.
- **Ease of programming** − Pig Latin is similar to SQL and it is easy to write a Pig script if you are good at SQL.
- **Optimization opportunities** − The tasks in Apache Pig optimize their executionautomatically, so the programmers need to focus only on semantics of the language.
- **Extensibility** − Using the existing operators, users can develop their own functions to read, process, and write data.

In addition to above differences, Apache Pig Latin

- Allows splits in the pipeline.
- Allows developers to store data anywhere in the pipeline.
- Declares execution plans.
- Provides operators to perform ETL (Extract, Transform, and Load) function

Applications of Apache Pig

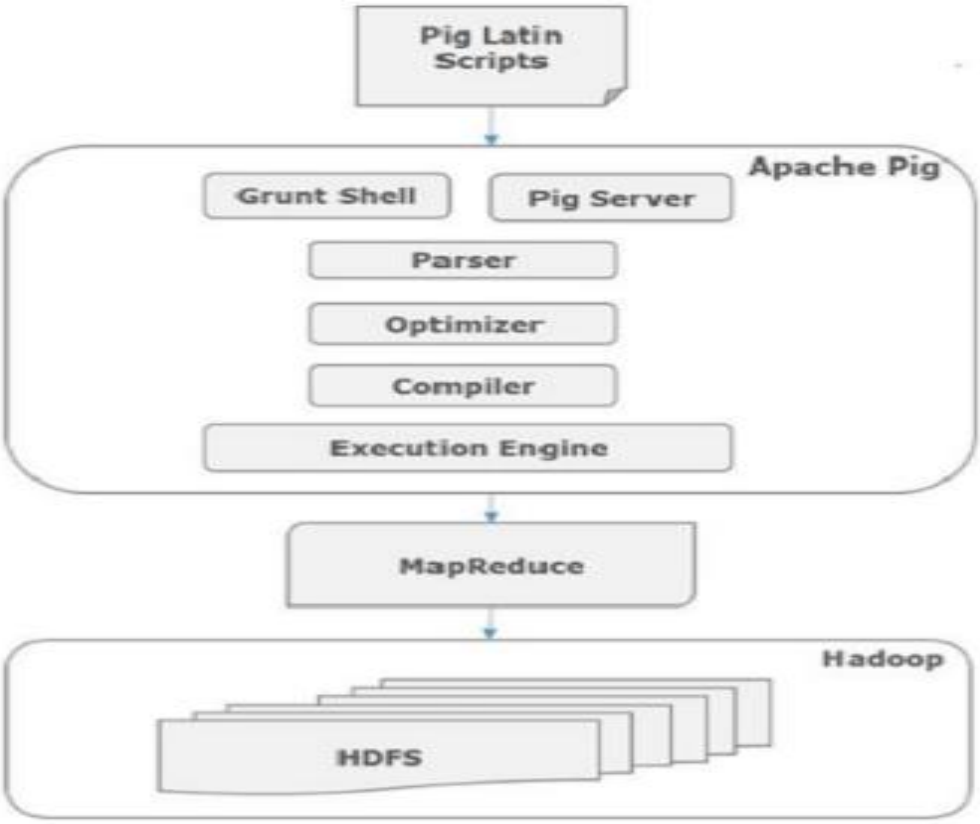
- Apache Pig is generally used by data scientists for performing tasks involving ad-hoc processing and quick prototyping.
- Apache Pig is used to process huge data sources such as web logs.
- To perform data processing for search platforms.
- To process time sensitive data loads.

Apache Pig – History

- In **2006**, Apache Pig was developed as a research project at Yahoo, especially to create and execute MapReduce jobs on every dataset.
- In **2007**, Apache Pig was open sourced via Apache incubator.
- In **2008**, the first release of Apache Pig came out. In **2010**, Apache Pig graduated as an Apache top-level project.

Apache Pig - Architecture

- The language used to analyze data in Hadoop using Pig is known as **Pig Latin**. It is a high level data processing language which provides a rich set of data types and operators to perform various operations on the data.
- To perform a particular task Programmers using Pig, programmers need to write a Pig script using the Pig Latin language, and execute them using any of the execution mechanisms (Grunt Shell, UDFs, and Embedded). After execution, these scripts will go through a series of transformations applied by the Pig Framework, to produce the desired output.
- Internally, Apache Pig converts these scripts into a series of MapReduce jobs, and thus, it makes the programmer's job easy. The architecture of Apache Pig is shown below.



Architecture of Apache Pig

Apache Pig Components

As shown in the figure, there are various components in the Apache Pig framework. Let us take a look at the major components.

Parser

- Initially the Pig Scripts are handled by the Parser. It checks the syntax of the script, does type checking, and other miscellaneous checks.
- The output of the parser will be a DAG (directed acyclic graph), which represents the Pig Latin statements and logical operators.
- In the DAG, the logical operators of the script are represented as the nodes and the data flows are represented as edges.

Optimizer

- The logical plan (DAG) is passed to the logical optimizer, which carries out the logical optimizations such as projection and pushdown.

Compiler

- The compiler compiles the optimized logical plan into a series of MapReduce jobs.

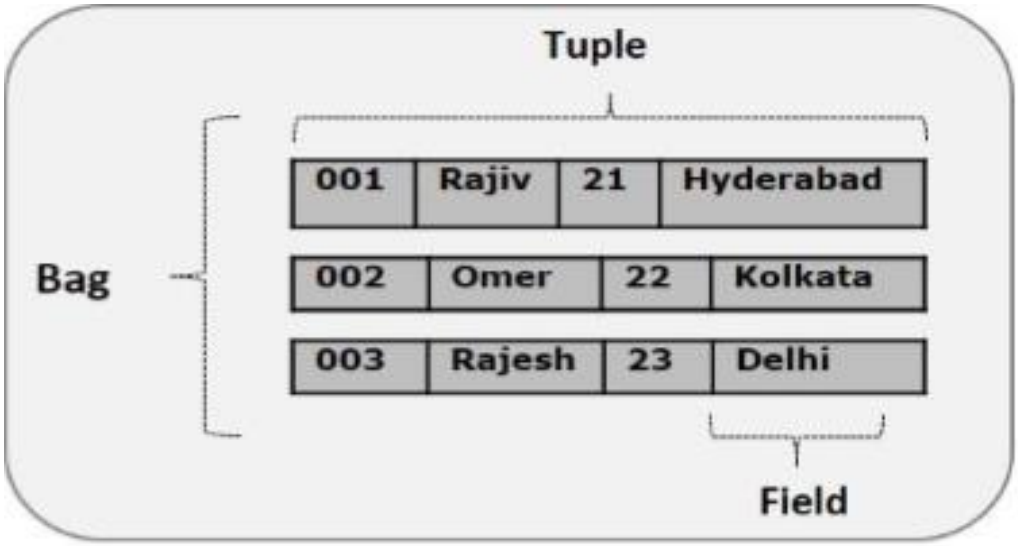
Execution engine

- Finally the MapReduce jobs are submitted to Hadoop in a sorted order. Finally, these MapReduce jobs are executed on Hadoop producing the desired results.

Pig Latin Data Model

- The data model of Pig Latin is fully nested and it allows complex non-atomic data types such map and tuple.

- Given below is the diagrammatical representation of Pig Latin's data model.



Atom

Any single value in Pig Latin, irrespective of their data, type is known as an **Atom**. It is stored as string and can be used as string and number. int, long, float, double, char array, and byte array are the atomic values of Pig. A piece of data or a simple atomic value is known as a **field**.

Example − 'raja' or '30'

Tuple

A record that is formed by an ordered set of fields is known as a tuple, the fields can be of any type. A tuple is similar to a row in a table of RDBMS. **Example** − (Raja, 30)

Bag

- A bag is an unordered set of tuples.
- In other words, a collection of tuples (non-unique) is known as a bag.
- Each tuple can have any number of fields (flexible schema).
- A bag is represented by '{}'.
- It is similar to a table in RDBMS, but unlike a table in RDBMS, it is not necessary that every tuple contain the same number of fields or that the fields in the same position (column) have the same type.

Example − {(Raja, 30), (Mohammad, 45)}

A bag can be a field in a relation; in that context, it is known as **inner bag**.

Example − {Raja, 30, **{9848022338, raja@gmail.com,}**}

Map

- A map (or data map) is a set of key-value pairs. The **key** needs to be of type chararray and should be unique. The **value** might be of any type. It is represented by '[]'
- **Example** − [name#Raja, age#30]

Relation

- A relation is a bag of tuples. The relations in Pig Latin are unordered (there is no guarantee that tuples are processed in any particular order).

Pig Latin – Data types

- Given below table describes the Pig Latin data types.

S.NO	Data Type	Description & Example
1	int	Represents a signed 32-bit integer. **Example** : 8
2	long	Represents a signed 64-bit integer. **Example** : 5L
3	float	Represents a signed 32-bit floating point. **Example** : 5.5F
4	double	Represents a 64-bit floating point. **Example** : 10.5
5	chararray	Represents a character array (string) in Unicode UTF-8 format. **Example** : 'welcome'
6	Byte array	Represents a Byte array (blob).
7	Boolean	Represents a Boolean value. **Example** : true/ false.
8	Date time	Represents a date-time. **Example** :1970-01-01T00:00:00.000+00:00
9	Big integer	Represents a Java BigInteger. **Example** : 60708090709

10	Big decimal	Represents a Java BigDecimal **Example** : 185.98376256272893883
Complex Types		
11	Tuple	A tuple is an ordered set of fields. **Example** : (raja, 30)
12	Bag	A bag is a collection of tuples. **Example** : {(raju,30),(Mohhammad,45)}
13	Map	A Map is a set of key-value pairs. **Example** : ['name'#'Raju', 'age'#30]

HIVE

What is Hive

- Hive is a data warehouse infrastructure tool to process structured data in Hadoop.
- It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy.
- Initially Hive was developed by Facebook; later the Apache Software Foundation took it up and developed it further as an open source under the name Apache Hive.
- It is used by different companies.
- For example, Amazon uses it in Amazon Elastic MapReduce.

Hive is not

- A relational database
- A design for OnLine Transaction Processing (OLTP)
- A language for real-time queries and row-level updates

Features of Hive

- It stores schema in a database and processed data into HDFS.
- It is designed for OLAP.
- It provides SQL type language for querying called HiveQL or HQL.
- It is familiar, fast, scalable, and extensible.

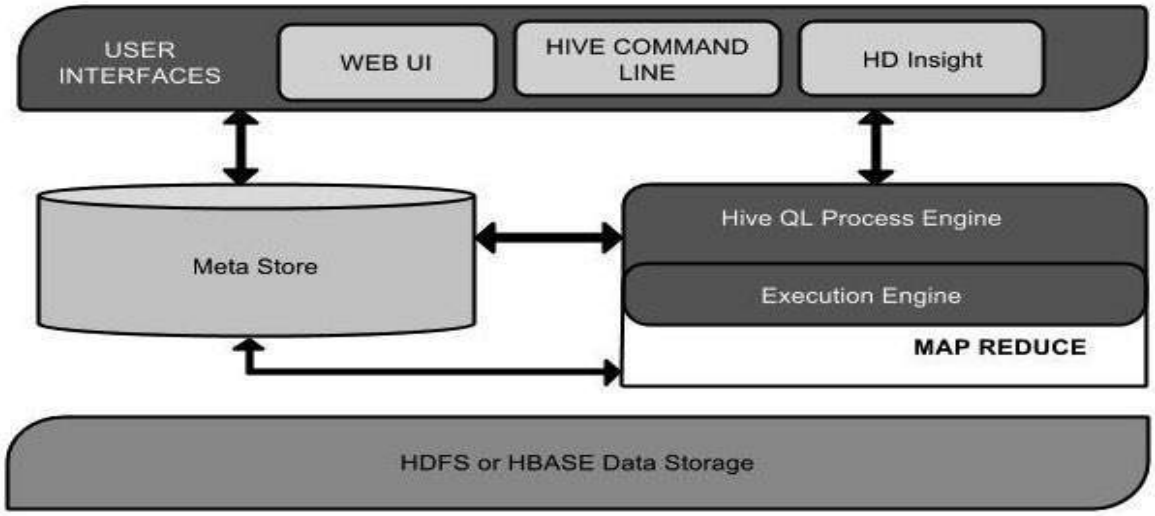
Architecture of Hive

- The following component diagram depicts the architecture of Hive:

User Interface

- Hive is a data warehouse infrastructure software that can create interaction between user and HDFS.

The user interfaces that Hive supports are Hive Web UI, Hive command line, and Hive HD Insight (In Windows server).



Meta Store

- Hive chooses respective database servers to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping.

HiveQL Process Engine

- HiveQL is similar to SQL for querying on schema info on the Metastore. It is one of the replacements of traditional approach for MapReduce program. Instead of writing MapReduce program in Java, we can write a query for MapReduce job and process it.

Execution Engine

- The conjunction part of HiveQL process Engine and MapReduce is Hive Execution Engine.
- Execution engine processes the query and generates results as same as MapReduce results. It uses the flavor of MapReduce.

HDFS or HBASE

- Hadoop distributed file system or HBASE are the data storage techniques to store data into file system.

Working of Hive

The following diagram depicts the workflow between Hive and Hadoop.

Execute Query

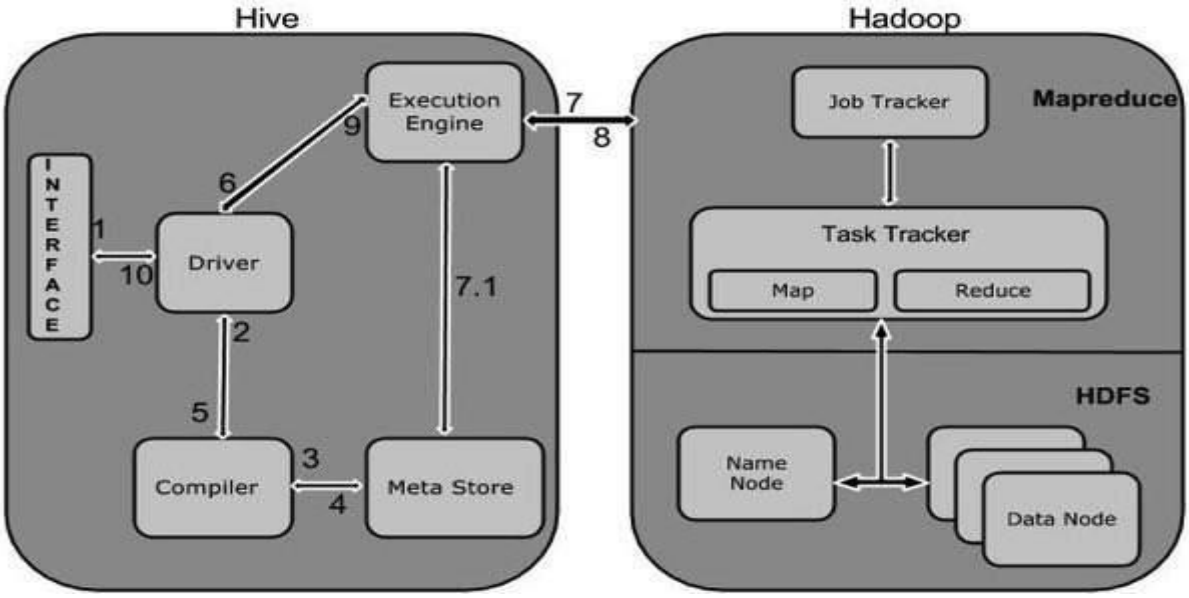
- The Hive interface such as Command Line or Web UI sends query to Driver (any database driver such as JDBC, ODBC, etc.) to execute.

Get Plan

- The driver takes the help of query compiler that parses the query to check the syntax and query plan or the requirement of query.

Get Metadata

- The compiler sends metadata request to Metastore (any database).



Send Metadata

- Metastore sends metadata as a response to the complier.

Send Plan

- The compiler checks the requirement and resends the plan to the driver. Up to here, the parsing and compiling of a query is complete.

Execute Plan

- The driver sends the execute plan to the execution engine.

Execute Job

- Internally, the process of execution job is a MapReduce job.
- The execution engine sends the job to JobTracker, which is in Name node and it assigns this job to TaskTracker, which is in Data node. Here, the query executes MapReduce job.

Metadata Ops

- Meanwhile in execution, the execution engine can execute metadata operations with Metastore.

Fetch Result

- The execution engine receives the results from Data nodes.

Send Results

- The execution engine sends those resultant values to the driver.

Send Results

- The driver sends the results to Hive Interfaces.

All the data types in Hive are classified into four types, given as follows:

- Column Types
- Literals
- Null Values
- Complex Types

Column Types

- Column type is used as column data types of Hive. They are as follows:

Integral Types

- Integer type data can be specified using integral data types, INT.

- When the data range exceeds the range of INT, you need to use BIGINT and if the data range is smaller than the INT, you use SMALLINT.
- TINYINT is smaller than SMALLINT.

The following table depicts various INT data types:

Type	Postfix	Example
TINYINT	Y	10Y
SMALLINT	S	10S
INT	-	10
BIGINT	L	10L

String Types

- String type data types can be specified using single quotes (' ') or double quotes (" ").
- It contains two data types: VARCHAR and CHAR. Hive follows C-types escape characters.

The following table depicts various CHAR data types:

Data Type	Length
VARCHAR	1 to 65355
CHAR	255

Dates

- DATE values are described in year/month/day format in the form {{YYYY-MM-DD}}.

Literals

- The following literals are used in Hive:

Floating Point Types

- Floating point types are nothing but numbers with decimal points. Generally, this type of data is composed of DOUBLE data type.

Decimal Type

- Decimal type data is nothing but floating point value with higher range than DOUBLE data type.
- The range of decimal type is approximately -10^{-308} to 10^{308}.

Complex Types

The Hive complex data types are as follows:

Arrays

- Arrays in Hive are used the same way they are used in Java.
- Syntax: ARRAY<data_type>

Maps

- Maps in Hive are similar to Java Maps.
- Syntax: MAP<primitive type, data_type>

Null Value

- Missing values are represented by the special value NULL.