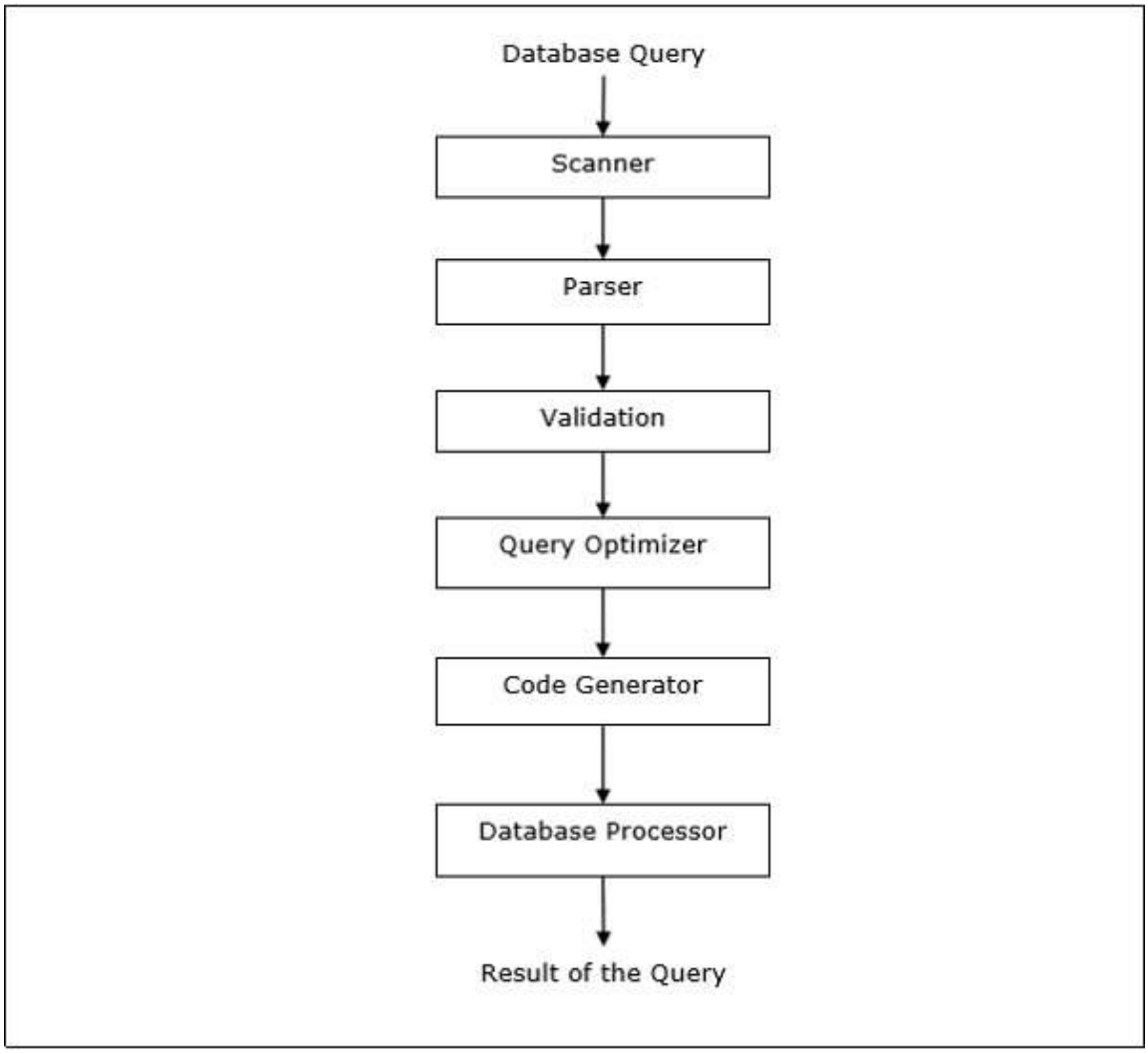
UNIT-II

Query processing and decomposition: Query processing objectives, characterization of query processors, layers of query processing, query decomposition, localization of distributed data.
Distributed query Optimization: Query optimization, centralized query optimization, distributed query optimization algorithms.

Query processing and decomposition: Query processing objectives

Query processing is a set of all activities starting from query placement to displaying the results of the query. The steps are as shown in the following diagram –



The main **objectives** of **query processing** in a distributed environment is to form a high level query on a distributed database, which is seen as a single database by the users, into an

efficient execution strategy expressed in a low level language in local databases. ... This is the time elapsed for executing the query.

Distributed query processing is the procedure of answering queries (which means mainly read operations on large data sets) in a distributed environment where data is managed at multiple sites in a computer network.

The main **objectives** of query processing in a distributed environment is to form a high level query on a distributed database, which is seen as a single database by the users, into an efficient execution strategy expressed in a low level language in local databases.

Ø An important point of query processing is query optimization. Because many execution strategies are correct transformations of the same high level query the one that optimizes (minimizes) resource consumption should be retained.

Ø The good measure of resource consumption are:

o The total cost that will be incurred in processing the query. It is the dome of all times incurred in processing the operations of the query at various sites and intrinsic communication.

o The resource time of the query. This is the time elapsed for executing the query. Since operations can be executed in parallel at different sited, the response time of a query may be significantly less than its cost.

Characterization of query processors:

-Languages

-Types of optimization

-Optimization Timing

-Statistics

-Decision sites

-Exploitation of network Topology

-Exploitation of Replicated Fragments

-Use of Semi-join

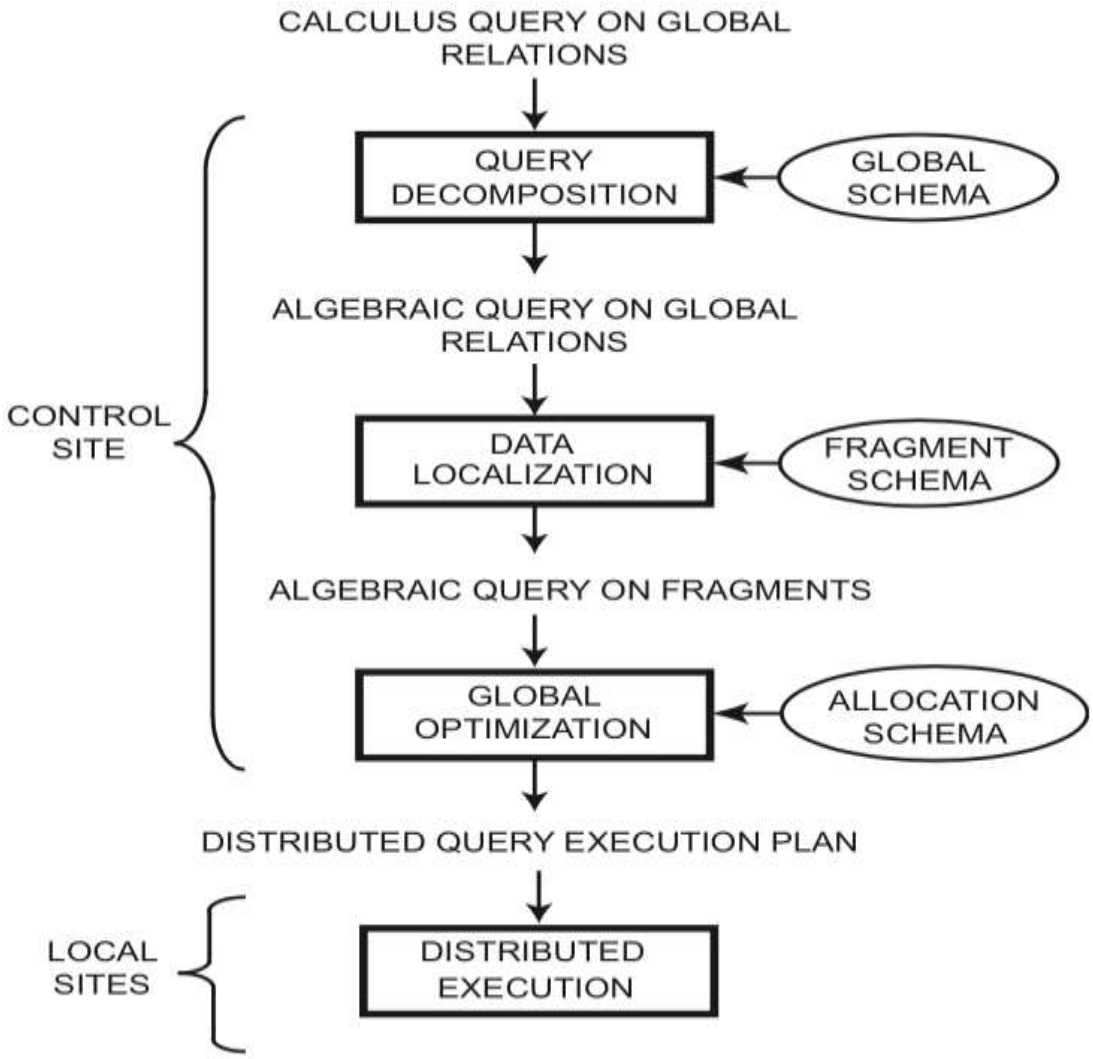
Layers of query processing:-

Query Decomposition. The first layer decomposes the calculus query into an algebraic query on global relations.

Data Localization The input to the second layer is an algebraic query on global relations. .

Global Query Optimization The input to the third layer is an algebraic query on fragments.

Distributed Query Execution



The problem of query processing can itself be decomposed into several subproblems, corresponding to various layers. A generic layering scheme for query processing is shown where each layer solves a well-defined subproblem. To simplify the discussion, let us assume a static and semicentralized query processor that does not exploit replicated fragments. The input is a query on global data expressed in relational calculus. This query is posed on global (distributed) relations, meaning that data distribution is hidden. Four main layers are involved in distributed query processing. The first three layers map the input query into an optimized distributed query execution plan. They perform the functions of query decomposition, data localization, and global query optimization. Query decomposition and data localization correspond to query rewriting. The first three layers are performed by a central control site

and use schema information stored in the global directory. The fourth layer performs distributed query execution by executing the plan and returns the answer to the query. It is done by the local sites and the control site.

Query Decomposition

The first layer decomposes the calculus query into an algebraic query on global relations. The information needed for this transformation is found in the global conceptual schema describing the global relations. However, the information about data distribution is not used here but in the next layer. Thus the techniques used by this layer are those of a centralized DBMS.

Query decomposition can be viewed as four successive steps. First, the calculus query is rewritten in a normalized form that is suitable for subsequent manipulation. Normalization of a query generally involves the manipulation of the query quantifiers and of the query qualification by applying logical operator priority.

Second, the normalized query is analyzed semantically so that incorrect queries are detected and rejected as early as possible. Techniques to detect incorrect queries exist only for a subset of relational calculus. Typically, they use some sort of graph that captures the semantics of the query.

Third, the correct query (still expressed in relational calculus) is simplified. One way to simplify a query is to eliminate redundant predicates. Note that redundant queries are likely to arise when a query is the result of system transformations applied to the user query. Such transformations are used for performing semantic data control (views, protection, and semantic integrity control).

Fourth, the calculus query is restructured as an algebraic query. That several algebraic queries can be derived from the same calculus query, and that some algebraic queries are “better” than others. The quality of an algebraic query is defined in terms of expected performance. The traditional way to do this transformation toward a “better” algebraic specification is to start with an initial algebraic query and transform it in order to find a “good” one. The initial algebraic query is derived immediately from the calculus query by translating the predicates and the target statement into relational operators as they appear in the query. This directly translated algebra query is then restructured through transformation rules. The algebraic query generated by this layer is good in the sense that the worse executions are typically avoided. For instance, a relation will be accessed only once, even if there are several select predicates. However, this query is generally far from providing an optimal execution, since information about data distribution and fragment allocation is not used at this layer.

Data Localization

The input to the second layer is an algebraic query on global relations. The main role of the second layer is to localize the query’s data using data distribution information in the fragment

schema. We saw that relations are fragmented and stored in disjoint subsets, called fragments, each being stored at a different site. This layer determines which fragments are involved in the query and transforms the distributed query into a query on fragments. Fragmentation is defined by fragmentation predicates that can be expressed through relational operators. A global relation can be reconstructed by applying the fragmentation rules, and then deriving a program, called a localization program, of relational algebra operators, which then act on fragments. Generating a query on fragments is done in two steps. First, the query is mapped into a fragment query by substituting each relation by its reconstruction program (also called materialization program). Second, the fragment query is simplified and restructured to produce another "good" query. Simplification and restructuring may be done according to the same rules used in the decomposition layer. As in the decomposition layer, the final fragment query is generally far from optimal because information regarding fragments is not utilized.

Global Query Optimization

The input to the third layer is an algebraic query on fragments. The goal of query optimization is to find an execution strategy for the query which is close to optimal. Remember that finding the optimal solution is computationally intractable. An execution strategy for a distributed query can be described with relational algebra operators and communication primitives (send/receive operators) for transferring data between sites. The previous layers have already optimized the query, for example, by eliminating redundant expressions. However, this optimization is independent of fragment characteristics such as fragment allocation and cardinalities. In addition, communication operators are not yet specified. By permuting the ordering of operators within one query on fragments, many equivalent queries may be found.

Query optimization consists of finding the "best" ordering of operators in the query, including communication operators that minimize a cost function. The cost function, often defined in terms of time units, refers to computing resources such as disk space, disk I/Os, buffer space, CPU cost, communication cost, and so on. Generally, it is a weighted combination of I/O, CPU, and communication costs. Nevertheless, a typical simplification made by the early distributed DBMSs, as we mentioned before, was to consider communication cost as the most significant factor. This used to be valid for wide area networks, where the limited bandwidth made communication much more costly than local processing. This is not true anymore today and communication cost can be lower than I/O cost. To select the ordering of operators it is necessary to predict execution costs of alternative candidate orderings. Determining execution costs before query execution (i.e., static optimization) is based on fragment statistics and the formulas for estimating the

cardinalities of results of relational operators. Thus the optimization decisions depend on the allocation of fragments and available statistics on fragments which are recorder in the allocation schema.

An important aspect of query optimization is join ordering, since permutations of the joins within the query may lead to improvements of orders of magnitude. One basic technique for optimizing a sequence of distributed join operators is through the semijoin operator. The main value of the semijoin in a distributed system is to reduce the size of the join operands and then the communication cost. However, techniques which consider local processing costs as well as communication costs may not use semijoins because they might increase local processing costs. The output of the query optimization layer is a optimized algebraic query with communication operators included on fragments. It is typically represented and saved (for future executions) as a distributed query execution plan.

Distributed Query Execution

The last layer is performed by all the sites having fragments involved in the query. Each subquery executing at one site, called a local query, is then optimized using the local schema of the site and executed. At this time, the algorithms to perform the relational operators may be chosen. Local optimization uses the algorithms of centralized systems.

The goal of distributed query processing may be summarized as follows: given a calculus query on a distributed database, find a corresponding execution strategy that minimizes a system cost function that includes I/O, CPU, and communication costs. An execution strategy is specified in terms of relational algebra operators and communication primitives (send/receive) applied to the local databases (i.e., the relation fragments). Therefore, the complexity of relational operators that affect the performance of query execution is of major importance in the design of a query processor.

Query decomposition:

The **query decomposition** is the first phase of query processing whose aims are to transform a high-level query into a relational algebra query and to check whether that query is syntactically and semantically correct. The SQL is then decomposed into query blocks (low-level operations), which form the basic units.

Query decomposition can be viewed as four successive steps:

--Normalization --Analysis
--Elimination of redundancy --Rewriting

Localization of distributed data:-

The main role of **data localization** is to localize the query's data using data distribution information. The localization layer translates an algebraic query on global relations into an algebraic query expressed on physical fragments. Localization uses information stored in the fragment schema.

The initial algebraic query generated by the query decomposition step is input to the second step: data localization. The initial algebraic query is specified on global relations irrespective of their fragmentation or distribution. The main role of data localization is to localize the query's data using data distribution information. In this step, the fragments that are involved in the query are determined and the query is transformed into one that operates on fragments rather than global relations. As indicated earlier, fragmentation is defined through fragmentation rules that can be expressed as relational operations (horizontal fragmentation by selection, vertical fragmentation by projection). A distributed relation can be reconstructed by applying the inverse of the fragmentation rules. This is called a localization program. The localization program for a horizontally (vertically) fragmented query is the union (join) of the fragments. Thus, during the data localization step each global relation is first replaced by its localization program, and then the resulting fragment query is simplified and restructured to produce another "good" query. Simplification and restructuring may be done according to the same rules used in the decomposition step. As in the decomposition step, the final fragment query is generally far from optimal; the process has only eliminated "bad" algebraic queries. We do not consider the fact that data fragments may be replicated, although this can improve the performance. A naïve way to localize a distributed query is to generate a query where each global relation is substituted by its localization program. This can be viewed as replacing the leaves of the operator tree of the distributed query with subtrees corresponding to the localization programs. The query obtained by this way is called localized query. In the reminder of this section for primary horizontal fragmentation we present reduction techniques that generate simpler queries. The horizontal fragmentation function distributes a relation based on selection predicates. The following example is used in subsequent discussions. Consider the employees relation EMP(ID, MGR_ID, NAME, SALARY). The ID column is a primary key for the EMP table and MGR_ID is the ID of the manager of the corresponding employee. The EMP relation is partitioned into three fragments: EMP1 = σ SALARY>100,000 (EMP) EMP2 = σ 50,000.
EMP2 = σ 50,000 <SALARY ≤100,000 (EMP)

EMP3 = σ SALARY≤50,000 (EMP)

Distributed query Optimization: Query optimization

Distributed query optimization refers to the process of producing a plan for the processing of a query to a distributed database system. The plan is called a query execution plan. In a distributed database system, schema and queries refer to logical units of data.

In a relational distributed relation database system, for instance, logical units of data are relations. These units may be be fragmented at the underlying physical level. The fragments,

which can be redundant and replicated, are allocated to different database servers in the distributed system.

Distributed query optimization requires evaluation of a large number of query trees each of which produce the required results of a query. This is primarily due to the presence of large amount of replicated and fragmented data. Hence, the target is to find an optimal solution instead of the best solution.

The main issues for distributed query optimization are −

Optimal utilization of resources in the distributed system.
Query trading.
Reduction of solution space of the query.

Steps for Query Optimization

There are three steps for query optimization. They are -

Step 1 − Query Tree Generation

A relational algebra expression is represented by a tree data structure known as a query tree. Leaf nodes represent the tables of the query. The internal nodes represent the relational algebra operations and the complete query is represented by a root.

When the operand table is made available, the internal node is executed. The result table replaces the node and the process is continued until the result table replaces the root node.

Step 2 − Query Plan Generation

The query plan is prepared once the query tree is generated. All the operations of the query tree are included with access paths which are known as query plan. The relational operations on the performance of the tree are specified by the access paths. For instance, the access path for a selection operation provides information on the use of B+ tree index.

The information about the intermediate tables that are required to be passed from one operator to another is provided by a query plan. The information about the usage of temporary tables, combining of the operations is provided by the query plan.

Step 3− Code Generation

The final step of query optimization is generation of the code. The type of the underlying operating system determines the form of the query. The results are produced by running the query code thus generated by the Execution Manager.

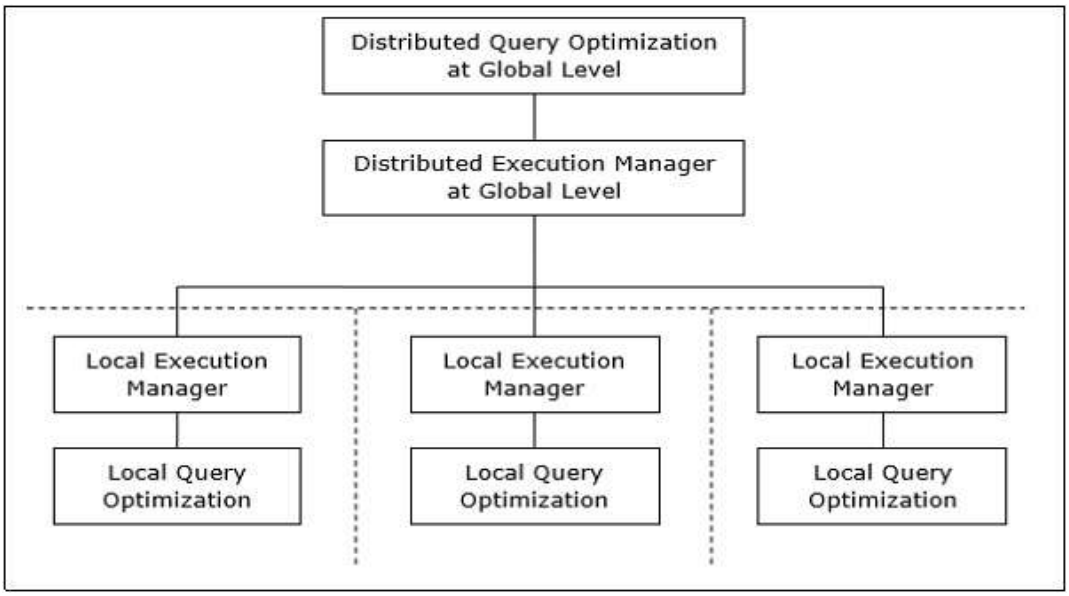
Centralized query optimization:

Once the alternative access paths for computation of a relational algebra expression are derived, the optimal access path is determined. 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.

Distributed query optimization algorithms:

Distributed query optimization refers to the process of producing a plan for the processing of a **query** to a **distributed database** system. The plan is called a **query** execution plan. In a **distributed database** system, schema and **queries** refer to logical units of data.



Distributed query optimization requires evaluation of a large number of query trees each of which produce the required results of a query. This is primarily due to the presence of large amount of replicated and fragmented data. Hence, the target is to find an optimal solution instead of the best solution.

The main issues for distributed query optimization are −

- Optimal utilization of resources in the distributed system.
- Query trading.
- Reduction of solution space of the query.

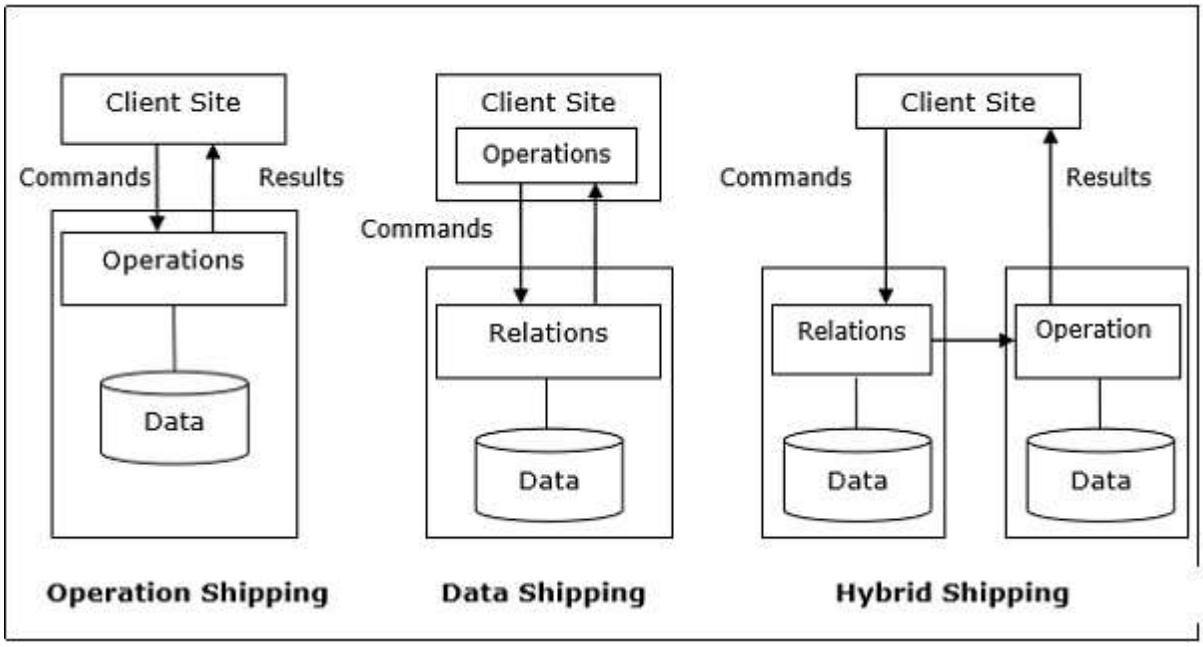
Optimal Utilization of Resources in the Distributed System

A distributed system has a number of database servers in the various sites to perform the operations pertaining to a query. Following are the approaches for optimal resource utilization −

Operation Shipping − In operation shipping, the operation is run at the site where the data is stored and not at the client site. The results are then transferred to the client site. This is appropriate for operations where the operands are available at the same site. Example: Select and Project operations.

Data Shipping − In data shipping, the data fragments are transferred to the database server, where the operations are executed. This is used in operations where the operands are distributed at different sites. This is also appropriate in systems where the communication costs are low, and local processors are much slower than the client server.

Hybrid Shipping − This is a combination of data and operation shipping. Here, data fragments are transferred to the high-speed processors, where the operation runs. The results are then sent to the client site.



Query Trading

In query trading algorithm for distributed database systems, the controlling/client site for a distributed query is called the buyer and the sites where the local queries execute are called sellers. The buyer formulates a number of alternatives for choosing sellers and for reconstructing the global results. The target of the buyer is to achieve the optimal cost.

The algorithm starts with the buyer assigning sub-queries to the seller sites. The optimal plan is created from local optimized query plans proposed by the sellers combined with the communication cost for reconstructing the final result. Once the global optimal plan is formulated, the query is executed.

Reduction of Solution Space of the Query

Optimal solution generally involves reduction of solution space so that the cost of query and data transfer is reduced. This can be achieved through a set of heuristic rules, just as heuristics in centralized systems.

Following are some of the rules −

- Perform selection and projection operations as early as possible. This reduces the data flow over communication network.
- Simplify operations on horizontal fragments by eliminating selection conditions which are not relevant to a particular site.
- In case of join and union operations comprising of fragments located in multiple sites, transfer fragmented data to the site where most of the data is present and perform operation there.
- Use semi-join operation to qualify tuples that are to be joined. This reduces the amount of data transfer which in turn reduces communication cost.
- Merge the common leaves and sub-trees in a distributed query tree.