Dynamic Programming Solution for Query Optimization in Homogeneous Distributed Databases

Ms. Anju Mishra
Department of Computer Application, IEC-CET,Greater Noida

Ms. Gunjan Nehru
Department of Computer Application, IEC-CET, Greater Noida

Mr. AshishPandey
Sapient Consulting, Gurgaon

Abstract

Due to new distributed database applications such as huge deductive database systems, the search complexity is constantly increasing and we need better algorithms to speedup traditional relational database queries. Now a days distributed database applications are applied on Heterogeneous distributed database systems and developing Homogeneous distributed databases. The "multiple query optimization" (MQO) tries to reduce the execution cost of a group of queries by performing common tasks only once, whereas traditional query optimization considersa single query at a time An optimal dynamic programming method for such high dimensional queries has the big disadvantage of its exponential order and thus we are interested in semi-optimal but faster approaches.
In this work we present a multiple query optimization on homogeneous distributed database application through dynamic programming for semi optimal solution.

Keywards: Distributedinformation retrieval (DIR), Multiple query optimization" (MQO)

1. Introduction

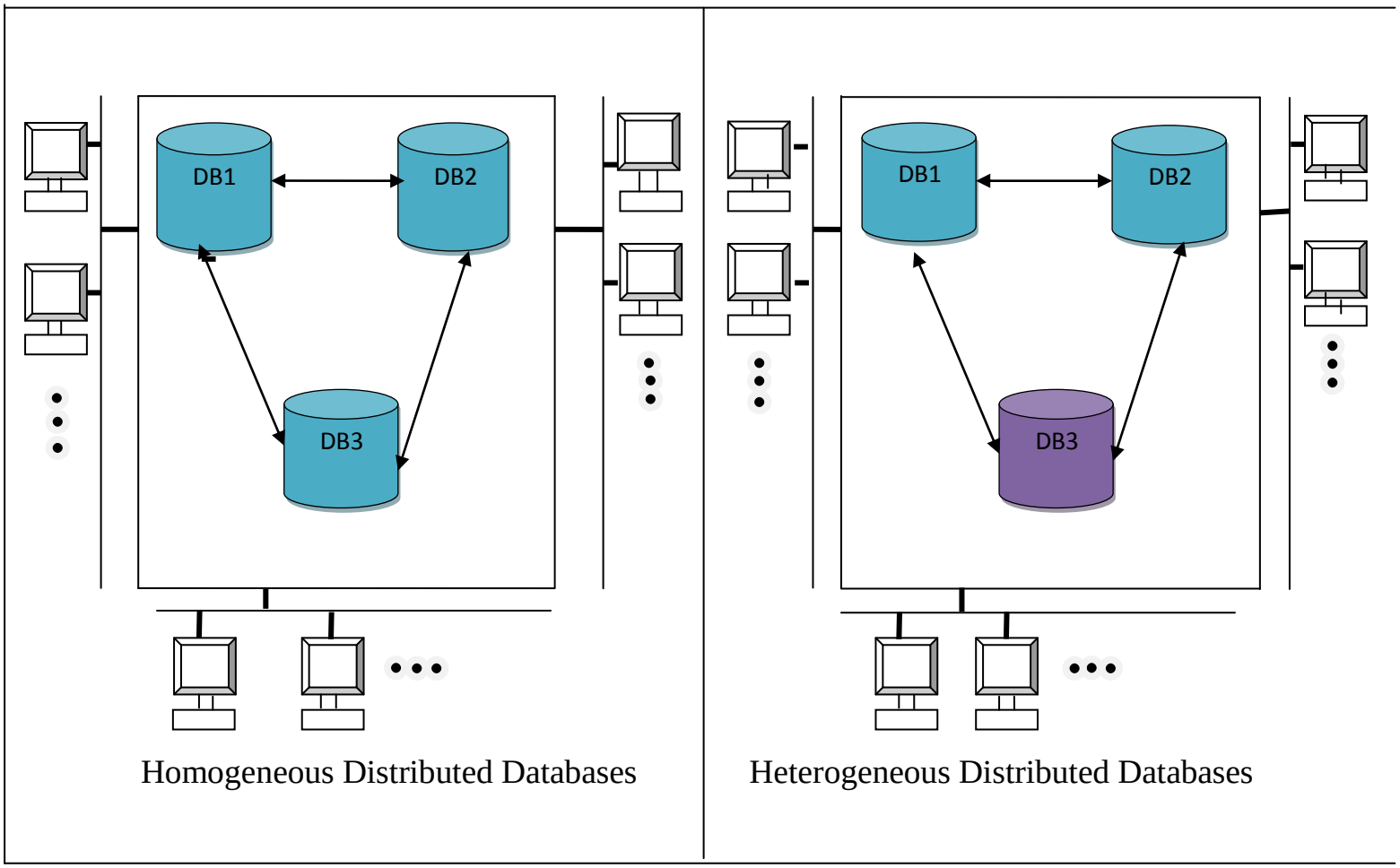
Distributed Systems is described as a partnershipamong independent cooperating centralized systems. Based on this concept the number of large scale applications has beeninvestigated during past decades among which distributedinformation retrieval (DIR) systems has developed to provide a singlesearch interface that provides access to the available databasesinvolving resource descriptions building for each database,choosing which databases to search for particular information,and merging retrieved results into a single result list [2]-[3].
In this work we proposed a dynamic programming approach to get the solution for query optimization in distributed database.

2. Distributed Database

A distributed database (DDB) is a collection of multiple,logically interrelated databases distributed over a computernetwork. This resource distribution improves performance,reliability, availability and modularity that are inherent indistributed systems. As with traditional centralized databases,distributed database systems (DDBS) must provide anefficient user interface that hides all of the underlying datadistribution details of the DDB from the users.

The use of a*relational query* allows the user to specify a description of thedata that is required without having to know where the data isphysically located [4].
Data retrieval from different sites in a DDB is known as*distributed query processing* (DQP).

Distributed Databases supports two types of distributed databases: homogenous and heterogeneous. In a homogenous distributed database system, each database is of same type. In a heterogeneous, distributed database system, at least one of the databases is of different type. The figure 1 below shows the illustration of homogeneous and heterogeneous distributed databases.



Homogeneous Distributed Databases

Heterogeneous Distributed Databases

Figure1. Homogeneous & Heterogeneous Distributed Databases

A homogenous distributed database system is a network of two or more Databases of same type that reside on one or more systems. Figure.2. illustrates a distributed system that connects three databases: HQ (Headquarter), MFG (Manufacturing), and SAL (Sales). An application can simultaneously access or modify the data in several databases in a single distributed environment. For example, a single query from a Manufacturing client on local database MFG can retrieve joined data from the table1 on the local database and the table2 on the remote database.

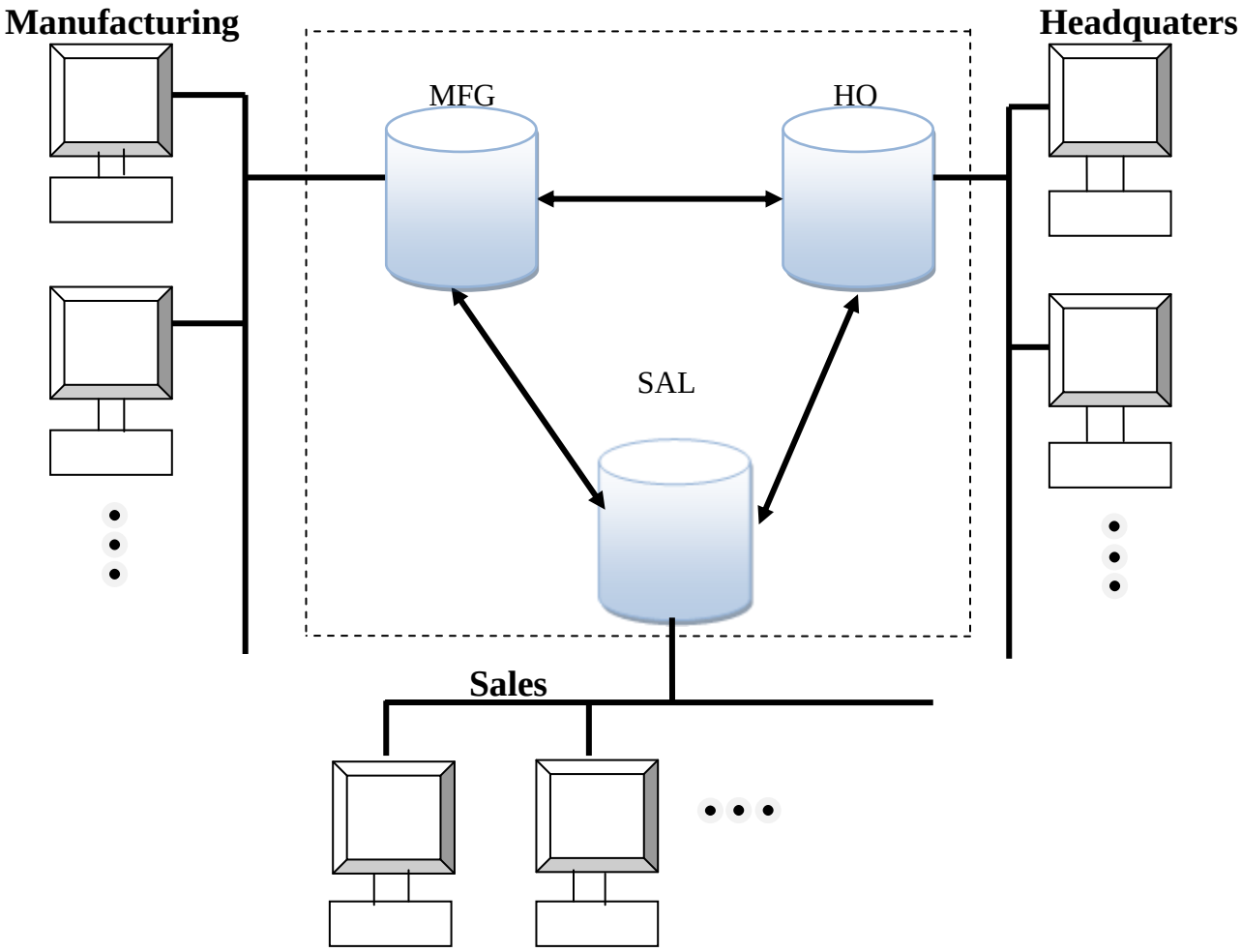


Figure 2. Distributed Database

For a client application, the location and platform of the databases are transparent by distribution transparency. You can also create synonyms for remote objects in the distributed system so that users can access them with the same syntax as local objects. For example, if you are connected to database MFG but want to access data on database HQ, creating a synonym on MFG for the remote T2 table enables you to issue this query:

```
SELECT * FROM T2;
```

In this way, a distributed system gives the appearance of native data access. Users on MFG do not have to know that the data they access resides on remote databases.

3. Query Processing

The path that a query traverses through a DBMS until its answer is generated is shown in Figure 3.The system modules through which it moves have the following functionality:

- The Query Parser checks the validity of the query and then translates it into an internalform, usually a relational calculus expression or something equivalent
- The Query Optimizer examines all algebraic expressions that are equivalent to the given queryand chooses the one that is estimated to be the cheapest.
- The Code Generator or the Interpreter transforms the access plan generated by the optimizer into calls to the query processor.
- The Query Processor actually executes the query.

Queries are posed to a DBMS by interactive users or by programs written in general-purpose Programming languages (e.g., C/C++, FORTRAN, PL-1) that have queries embedded in them. An Interactive (ad hoc) query goes through the entire path [8].

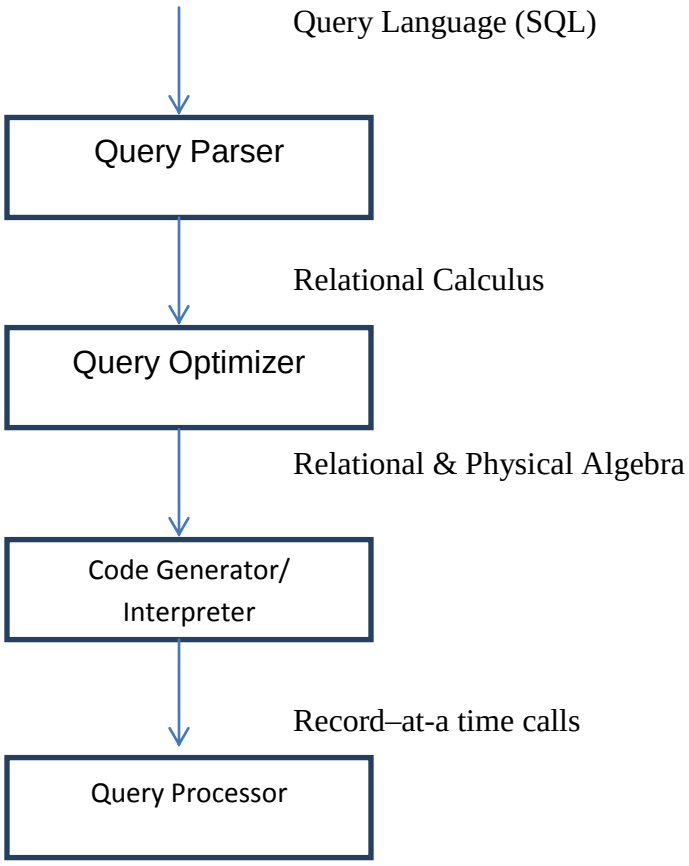


Figure 3: Query flow through a DBMS.

4. Query Optimization in Distributed Database

Global query management provides the ability to combine data from different local databases in a single retrieval operation. The necessity for global query management arises in an open, heterogeneous multidatabase system, since autonomy and heterogeneity of component databases have given rise to a number of new major issues regarding the global query optimization strategy and context mediation including data conversion and query translation. For instance, some local DBMSs never support semi join operator which has been proposed in order that data transmission between sites could be reduced. In this regard, the global query optimization strategies developed for *homogeneous* distributed database systems make extensive use of semi joins which are not attractive in the multidatabase context since this may increase the local processing time. Moreover, these do not consider the cost incurred as a result of data conversion and query translation[5].A major cost in executing queries in a distributed database system is the data transfer cost incurred in transferring relations (fragments) accessed by a query from different sites to the site where the query is initiated. The objective of a data allocation algorithm is to determine an assignment of fragments at different sites so as to minimize the total data transfer cost incurred in executing a set of queries. This is equivalent to minimizing the average query execution time, which is of primary importance in a wide class of distributed conventional as well as multimedia database systems.Our basic principle to get a high performance is that we decompose a global query to the finest level of subqueries in order to explore all possible execution plans.

5. Components and Problems of Distributed Query Optimization

There are three components of distributed queryoptimization [10][11]:

5.1 Access Method: In most RDBMS products, tables canbe accessed in one of two ways: by completely scanningthe entire table or by using an index. The best accessmethod to use will always depend upon the circumstances.For example, if 90 percent of the rows in the table aregoing to be accessed, you would not want to use an index. Scanning all of the rows would actually reduce I/O andoverall cost. Whereas, when scanning 10 percent of thetotal rows, an index will usually provide more efficientaccess. Of course, some products provide additionalaccess methods, such as hashing. Table scans and indexedaccess, however, can be found in all of the "Big Six"RDBMS products (i.e., DB2, Sybase, Oracle, Informix,Ingres, and Microsoft).

5.2 Join Criteria: If more than one table is accessed,the manner in which they are to be joined together must bedetermined. Usually the DBMS will provide severaldifferent methods of joining tables. For example, DB2provides three different join methods: merge scan join,nested loop join, and hybrid join. The optimizer mustconsider factors such as the order in which to join thetables and the number of qualifying rows for each joinwhen calculating an optimal access path. In a distributedenvironment, which site to begin with in joining the tablesis also a consideration.

5.3 Transmission Costs: If data from multiple sitesmust be joined to satisfy a single query, then the cost oftransmitting the results from intermediate steps needs to befactored into the equation. At times, it may be more costeffective simply to ship entire tables across the network toenable processing

to occur at a single site, therebyreducing overall transmission costs. This component ofquery optimization is an issue only in a distributedenvironment.

6. Dynamic Programming Solution

Dynamic programming is a method for solving complex problems by breaking them down into simpler subproblems. It is applicable to problems exhibiting the properties of overlapping subproblems which are only slightly smaller and optimal substructure (described below)whenapplicable; the method takes far less time than naive methods.

The key idea behind dynamic programming is quite simple. In general, to solve a given problem, we need to solve different parts of the problem (subproblems), and then combine the solutions of the subproblems to reach an overall solution. Often, many of these subproblems are really the same. The dynamic programming approach seeks to solve each subproblem only once, thus reducing the number of computations: once the solution to a given subproblem has been computed, it is stored or "memo-ized": the next time the same solution is needed, it is simply looked up. This approach is especially useful when the number of repeating subproblems grows exponentially as a function of the size of the input.

6.1 Elements of Dynamic Programing:

Optimal Substructure:A problem exhibits optimal substructure if an optimal solution to the problem contains within it optimal solutions to subproblems. It is often easy to show the optimal sub problem property as follows:-

1. Split problem into subproblems .

2. Sub problemsmust beoptimal, otherwise the optimal splitting would not have been optimal.

Overlapping Subproblem:In overlapping subproblems the space of subproblems must be small in the sense that a recursive algorithm for the problem solves the same subproblems over and over.But in dynamic programming same subproblem resolved once and result used in repeated subproblem.

Memoization: Dynamic programming algorithm typically take advantage of overlapping subproblem once and then storing the solution in a table where it can be looked up when needed.

Based on the study of the previous work, we proposed a scheme to reduce the search space of Dynamic Programming based on reuse of query plans among similar subqueries. The method generates the cover set of similar subgraphs present in the query graph and allows their corresponding subqueries to share query plans among themselves in the search space. Numerous variants to this scheme have been developed for enhanced memory efficiency and one of them has been found better suited to improve the performance of Iterative Dynamic Programming.

7. Need of Cost Factors

In a centralized DBMS, query execution “cost” is a single dimensional factor measured in conceptual units. In a distributed database, costs must be dividing into multiple dimensions under the control of single logical database. One proposal for a universal cost metric is hard currency, but typically there are

other costs that are valuable to expose orthogonally, including response time, data freshness, and accuracy of computations [12].

8. Need of Cost Estimation

A centralized optimizer cannot accurately estimate the costs of operations at many autonomous sites.Z. G. Ives and A. Tomasicperposed middleware systems in [13, 14] address this problem by involving site specific wrappers in the optimization process, but they do not consider the cost of communicating with these wrappers. This cost is not significant in these systems because the wrappers typically reside in the same address space as the optimizer. But in general, the execution costs may also depend on transient system issues including inter communication cost between two sites. Therefore cost estimation process must be distributed in a manner reflective of the query processing, with cost estimates being provided by the sites that would be doing the work. However, to the best of our knowledge, complete cost estimation, which requires the optimizer to communicate with the sites merely to find the cost of an operation, has not been studied before. In such a scenario, communication may become the dominant cost in the query optimization process. The high *cost of costing* raises a number of new design challenges, and adds additional factors to the complexity of distributed query optimization.

9. Query Graph Model

The Query Graph Model is an example of preferred internal representation of user query, and join graph is an example of preferred canonical form. A join graph denominates a user query representation having nodes connected by edge, where each node represents relation and edge represents a join predicate. A relation denominates a database table having tuples and column. A join predicates relates columns of two relations to be joined by specifying conditions on column values. The Cardinality of a relation denominates the number of tuples embraced by the relation and the Selectivity of a join predicate denominates the expected fraction of tuples for which the join column value in the relation satisfies the predicate. Query cardinality is the product of cardinalities of every relation in the query times the product of selectivity factor of the query predicates.

A hybrid technique for joining tables that selects a join execution plan from among the well-known “nested- loop” and “sort-merge” methods. A method for optimizing query execution that relies on measuring the degree of”clustering” (sortedness) in the stored relations.The testing of each join column to calculate the degree of clustering of the column values in their storage order to estimate the number of page accesses required for a partial index scan. These estimates are then used to calculate the cost of each proposed join execution plan.

Any user query can be recast as a join graph made up of some combination of “linear” and “star” subgraphs.”Linear queries” can be describe as a series of relation nodes each connected by predicate edges to no more than two other relation nodes. ”Star queries” describes as a group of relation nodes with a single central relation node connected by predicate edges to each of the other relation nodes. A join graph representing such a series of two way join as a canonical form of the query graph model(QGM). The practitioner may then either exhaustively enumerates all possible “feasible plans” for

join execution, using “dynamic programming” or may employ some “heuristically limited” search method to reduce the number of alternative plans in the search space considered by the optimizer. It is easily proven that dynamic programming never eliminates optimal plan, because all possible plans enumerated and evaluated. However any” heuristic” search method presents some non-zero probability of excluding superior execution plan without notice.There are usually many feasible plans for any given query and many practitioners use the exponential worst-case complexity argument to justify a priori search space truncation through heuristic search methods. Dynamic programming search space grows as N!, Here in, a search space denominates a set of executable query plans selected on the basis of some criteria related to primitive database operators.

Heuristic search method for query optimization is limiting the time and space complexity by truncating the enumeration of feasible query execution plans. Dynamic programming(DP) is the time honored method for optimizing join queries in relational database management systems and virtually all commercial optimizer rely on some abbreviated form of DP for this purpose. DP uses exhaustive enumeration with pruning to produce “optimal” execution plans without missing the best of these plans [7].

For a given query, the *search space* can be defined as theset of equivalent operator trees that can be produced usingtransformation rules. The example bellow illustrates 3equivalent join trees, which are obtained by exploiting theassociative property of binary operators. Join tree (c) whichstarts with a Cartesian product may have a much higher costthan other join trees [2].

```
SELECT ENAME, RESP
FROM EMP, ASG, PROJ
WHERE EMP.ENO=ASG.ENO
AND ASG.PNO=PROJ.PNO
```

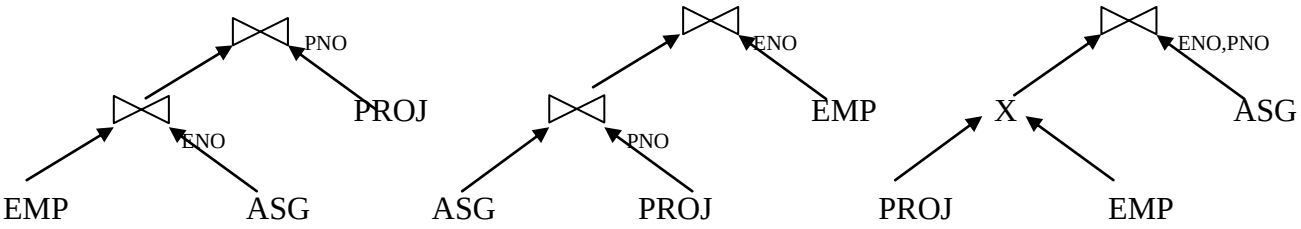
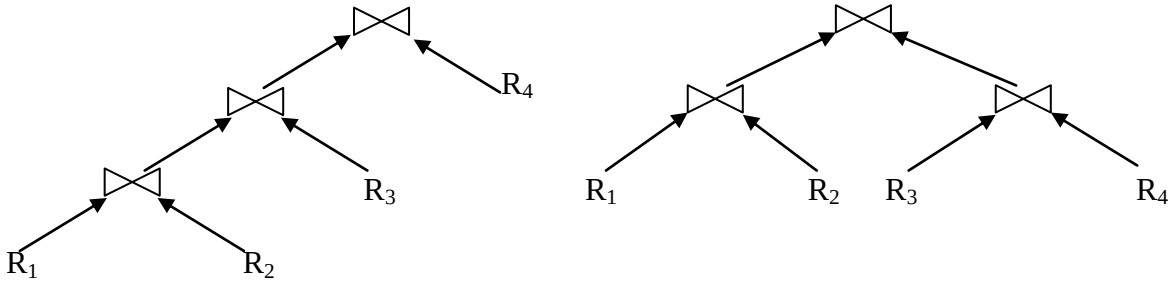


Figure 4.Query equivalent trees

Regarding different search spaces, there would be differentshape of the join tree. In a *linear tree*, at least one operand ofeach operand node is a base relation. However, a *bushy tree*might have operators whose both operands are intermediateoperators. In a distributed environment, bushy trees are usefulin exhibiting parallelism [18].



(a) linear join tree (b) bushy join tree

Figure 5. Linear vs. bushy join tree

In DP optimizer the cardinality of the join of N relations is the same regardless of join order, the "cost" of joining in different orders may vary substantially. Accordingly, for an N-way join query, there are N! Permutations of relation join orders embraced by the "search space" generated by DP [7].

10.Related Works

Three most common types of algorithms for join-orderingoptimization are deterministic, Genetic and randomizedalgorithms [15].

Deterministic algorithm, also known as exhaustive search*dynamic programming* algorithm, produces optimal left-deepprocessing trees with the big disadvantage of having anexponential running time. This means that for queries withmore than 10-15 joins, the running time and space complexityexplodes [15].Genetic and randomized algorithms [16]-[17] on the otherhand do not generally produce an optimal access plan. But inexchange they are superior to dynamic programming in termsof running time. Experiments have shown that it is possible toreach very similar results with both genetic and randomizedalgorithms depending on the chosen parameters. Still, thegenetic algorithm has in some cases proved to be slightlysuperior to randomized algorithms.Layers of distributed query optimization have beendepicted in Figure6.

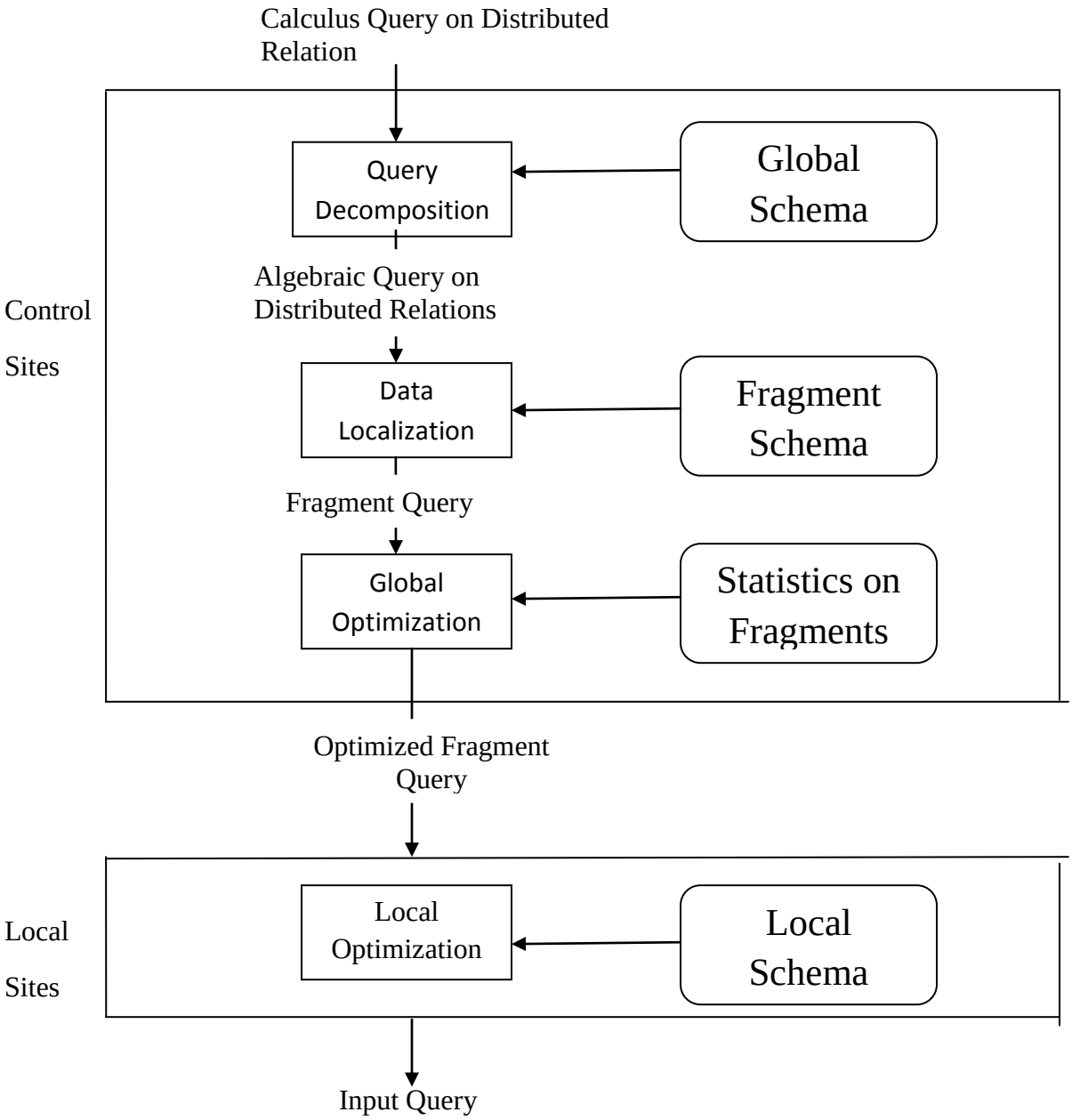


Figure 6: Distributed Query Optimization

There are number of Query Execution Plan for DDB suchas: row blocking, multi-cast optimization, multi-threadedexecution, joins with horizontal partitioning, Semi Joins, andTop n queries. In this paper we propose a dynamic programming algorithm for Query optimization in homogeneous distributed database systems [2].

11.COST MODEL

11.1Cost Model in DBMS

The cost model assigns an estimated cost to any partial or complete plan in the searchspace. It also determines the estimated size of the data stream for output of every operatorin the plan. It relies on:

a) A set of statistics maintained on relations and indexes, e.g., number of data pages in arelation, number of pages in an index, number of distinct values in a column
b) Formulas to estimate selectivity of predicates and to project the size of the output datastream for every operator node. For example, the size of the output of a join is estimatedby taking the product of the sizes of the two relations and then applying the jointselectivity of all applicable predicates.
c) Formulas to estimate the CPU and I/O costs of query execution for every operator.These formulas take into account the statistical properties of its input data streams,existing access methods over the input data streams, and any available order on the datastream (e.g., if a data stream is ordered, then the cost of a sort-merge join on that streammay be significantly reduced). In addition, it is also checked if the output data stream willhave any order.

The cost model gives a cost estimate for each operator tree. The cost refers to resourceconsumption, either space or time. Typically, query optimizers estimate the cost as timeconsumption [6].

10.2Cost Model in Distributed DBMS

In a distributed execution environment, there are two different time consumptionestimates to be considered: *total time* or *response time*. The former is the sum of the timeconsumed by each processor, regardless of concurrency, while the latter considers thatoperations may be carried out concurrently. Thus, response time is a more appropriateestimate, since it corresponds to the time the user has to wait for an answer to the query.In a distributed environment, the execution of an operator tree ***S*** is split into severalphases. Pipelined operations are executed in the same phase, whereas a storing indicationestablishes the boundary between one phase and the subsequent one. Resource contention is also another reason for splitting an operator treeinto different phases. For instance, if a sequence of operations that could be concurrentlyexecuted require more memory than available (e.g., if the memory is not sufficient tostore the entire hash tables for pipelined operations in the hash join algorithm), then it issplit into two or more phases. An operator tree is also split into different phases ifindependent operations (which, in principle, could remain in the same phase) should beexecuted at the same home site: in this case, the operations are not concurrently executedjust because the homes are the same and, accordingly, they are scheduled at differentphases [6].

An optimizer cost model includes cost functions to predictthe cost of operators, and formulas to evaluate the sizes ofresults. Cost functions can be expressed with respect to eitherthe total time, or the response time [18]-[19]. The total time isthe sum of all times and the response time is the elapsed timefrom the initiation to the completion of the query. The totaltime (*TT*) is computed as below, where $T_{CPU}$is the time of aCPU instruction, $T_{I/O}$the time of a disk I/O, $T_{MSG}$the fixedtime of initiating and receiving a message, and $T_{TR}$the time ittakes to transmit a data unit from one site to another, and $T_{DELAY}$the time of waiting for the producer to deliver the first result tuples

$$TT = T_{CPU} * \#insts + T_{I/O} * \#_{I/O} + T_{MSG} * \#msgs + T_{TR} * \#_{bytes} + T_{DELAY} * \#_{insts}$$

When the response time of the query is the objectivefunction of the optimizer, parallel local processing and parallelcommunications must also be considered. This response time(*RT*) is calculated as bellow:

$RT = T_{CPU} * seq\_\#insts + T_{I/O} * seq\_\#_{I/Os} + T_{MSG} * seq\_\#msgs + T_{TR} * seq\_\#_{bytes} + T_{DELAY} * seq\_\#_{insts}$

Most early distributed DBMSs designed for wide areanetworks have ignored the local processing cost andconcentrate on minimizing the communication cost. Considerthe following example:

$TT = 2 * T_{MSG} + 2*T_{DELAY} + T_{TR} * (x + y)$
$RT = max \{T_{MSG} + T_{DELAY} + T_{TR} * x, T_{MSG} + T_{DELAY} + T_{TR} * y \}$

Where, *x* and *y* considered to be two queries processing in parallel. In parallel transferring, response time is minimized byincreasing the degree of parallel execution. This does notimply that the total time is also minimized. On contrary, it canincrease the total time, for example by having more parallellocal processing (often includes synchronization overhead and it may increase the local processing time and comprising it will increase the total time)and transmissions. Minimizing the total time implies that theutilization of the resources improves, thus increasing thesystem throughput. In practice, a compromise between thetotal and response times is desired [2]

12.Dynamic Programming Approach

The Dynamic Programming(DP) approach is recursively called for larger subgraph to give the solution for join graph. The Dynamic Programming (DP) process for query graph subset of T_L or fewer relations, where T_L represents the predetermined size limits that may be arbitrarily selected to limit the solution space. The join graph is divided into subgraphs not more than T_L relation nodes [7]. Through dynamic programming we will get improved query optimization technique that can be applied immediately to any database

The following conventions are used in describing algorithm:

• s_c: candidate set of plans;

• S [i, j]: either empty set, representing no solutionfor the cost value *j;* or set of "candidate setof plans" obtained by using plans from the set {1, 2, . . ., i} and containing exactly one plan foreach query with a total cost j (that is, if S [i, j] is not empty, its sets represent solutions with cost $j$for all queries from 1 up to the query of a plan p_i);

• PS_i: starting plan number for the query q_i(that is,plans from $PS_i$to $PS_i + P_i$belongs to query q_i);

• Cost (s_c: candidate set of plans): summation of thecosts of the tasks in the tasks set obtained from theunion of the tasks of the plans (task sets) in s_c.

Candidate sets are obtained by adding new plans topreviously obtained candidate sets. The recurrence relationfor candidate sets S [i, j] is as follows:

S [i, j] =∪ {s_c∪ {p_i} | ∀scsuch that cost (s_c∪ {p_i})
= j, and query number of plan i is q
and, $s_c \in S$ [plans of previous query (PS_{q-1} to $PS_q - 1$), potentially usefulcosts ($j - C_i$to j)].

Using the inductive proof technique with an inductionon cost values from 0 to C and query numbersfrom 0 to Q, the correctness of the above recurrencerelation can easily be shown. The proof is based onthe following argument: if up to a certain cost value c,for queries from 1 to $q - 1$, all plan sets includingone plan for each query are known, then, these plansets can be extended with a plan for query q, producingcost values c, $c + 1$, . . . such that the cost valuesare calculated.

The DP algorithm implementing this recurrence relationin a bottom-up manner. Forthe base case of the recurrence relation S [0, 0] = {{ }}, represents that if there is no query, a plan containingno tasks (plan number 0) with total cost 0 is the solution. The starting indexes of theplans for each query, namely PS_q's.The candidate sets, S, are generated in a column wise manner for cost values startingfrom 1 and

considering plans from 1 to *P*. Sincethe candidates are sets of plans, and since plans may have common tasks, it is even possible to use a planset with total cost equal to the current cost, and extendit with the current plan and still obtain the same cost.This can occur if the current plan's tasks are commonwith the task set formed from the tasks of the plan set.Therefore, all the columns from "the current columnminus the cost of the current plan" (in case no commontask between the current plan and existing plan set'stasks) to "the current column" must be examined [1].

Dynamic Programming Psuedocode

Plan-based DP algorithm
Input: Join Graph G and size limit T_L
Output: s_c: solution set of plans that contains exactly one plan for each query

```
1        // initialization
2        S[∗, ∗]= { }
3        S[0, 0] = {{ }}
4        PS0= 0; P0= 1
5        for i= 1 to Q
6                PSi= PSi−1+Pi−1
7                // main part
8        for j = 1 to C // cost values
9                for i= 1 to P { // plans
10                       // query number of plan i
11                       q = pqi
12                       for k = PSq−1 to PSq− 1 // consider plans belonging to previous query only
13                       //query optimization of query number k
14                               While |Gk| > 1 do   //stop when no more relations to join
15                                       MinCost = ∞       //use minimum cost
16                                       //examine all unjoined connected pairs
17                                       for x,y in Gk connected by an edge do
18                                               //try both join order
19                                               Join=mincost(plan[x]×plan[y],plan[y]×plan[x])
20                                               if JoinCost<MinCost then//remember minimum cost join
21                                                       Next Join=Join
22                                                       r=min(x,y)
23                                                       s=max(x,y)
24                                                       MinCost=JoinCost
25                                               endif
26                                       endfor
```

```
27                  //has that sizelimit been exceed?
28                  if |relation[r] U relation[s]| >T_L then
29                      //call DP on larger subgraph
30                      if |relation[r] > relation[s]| then
31                          t = r
32                      else
33                          t = s
34                      endif
35                      //defer join and move to step 14 for subgraph for plan
36                      plan[t] = go to step 14(relation[t])
37                      relation[t] = {(relation[t])}//treat relation[t] as compound
38                                                          element
39                  else
40                      plan[r] = NextJoin      //update plan associated with r
41                      relation[r] = relation[r] U relation[s]  //collapse s into r
42                      relation[s] = 0
43                  endif
44              endwhile
45              relation[1]                         //last relation for subgraph
46              for m = max(j −Ci , 0) to j // consider candidates for previously obtained
                                                            cost values
47                  if S[k,m] _= {} then
48                      for each s_c in S[k,m] // consider all the candidates in the
                                                            entry
49                          if cost(s_c∪ {p_i}) = j then
50                          {
51                              S[i, j] = S[i, j] ∪ {s_c∪ {p_i}
52                          }
53                              if i≥PS_Q then return s_c
54 End Algorithm
```

Onthe other hand to prevent more than one plan for thesame query from appearing in the candidate plan set,only the rows corresponding to the plans of previousquery must be included in the search space. As a result,the set of candidates at column *j* and row *i*is determinedby using the candidates at rows from PS_{q-1} toPS_q-1 where$PS_q \leq i < PS_{q+1}$ andcolumns from $j - C_i$to j. All possiblecandidates obtained at each entry must be kept forfurther iterations. It is possible to obtain a new candidateif there is a candidate in the searched area that can be extended by the currentplan. Notice that the existing candidate can be extended by a new plan if the totalcost of the union of the plans is equal to the currentcost value (or column number) [1]. For internal loop the procedure used produces a canonical Query Graph Model (QGM) herein denominated the "join graph G".

- Join graph G : Query Graph Model (QGM)
- T_L: Enumeration threshold, which is the input of predetermined limit for this process. Enumeration threshold T_L represents the maximum number of relations in any subgraph G_L referred to the dynamic programming (DP) optimization process and operates for DP search space used in optimizing graph G.
- Relations[x]:the base relation or relation subgraph corresponding to relation node x.
- Plan[x]: the query execution plan selected by DP for node x.

The process is initialized with relation[x] ={x} and plan[x] =ACCESS(x)

The mincost is first set as high as possible. In inner loop of internal loop first, connected node pair (relation[r],relation[s]) is tested for execution cost in both the directions. The optimal two-way join plan for two relations from the search space having two plans differing only in join order. The cost of optimal join order for connected node pair (relation[r],relation[s]) is then tested against mincost and, if the cost is not less than mincost, the procedure returns to select another connected node pair for evaluation. If the new two-way join plan has an execution cost that is less than the mincost saved from the previous optimal two-way plan, then reset some parameters to save the two-way join plan as the new "next join" (the new optimal two-way join plan) and tests for more untested connected node pairs.

If more connected node pairs await testing, then selects another such connected pair (arbitrarily) and returns to evaluate the next pair. If no untested connected pairs remain to be evaluated, then test the two-way join complexity by adding the node joinder numbers for the connected node pair (relation[r],relation[s]) found to have the lowest cost of all such pairsin graph G. This sum is compared to the enumeration threshold T_L,if this sum is less than T_L then merges the connected node pair (relation[r],relation[s]) into a single node r having a new node joinder number, i.e. sum of node joinder number of connected node, and node relation[s] is eliminated from the join query graph G. The procedure returns to re-examine every one of the connected node pairs in the join query graph modified by the merger of nodes r and s.

When a candidate isgenerated for a plan that belongs to the last query, thealgorithmstops and returns that candidate as a solution. The verification of whether the obtainedcandidate set is a solution or not is trivial. If the planbelongs to the last query, and a candidate is found,then, that means this candidate contains exactly oneplan for each query, thus it is a solution [7].

13. Conclusion

In Homogeneous Distributed Database the query optimization has boon proposed with dynamic programming approach. Although deterministic dynamic programming algorithmproduces optimal left-

deep processing trees, it has the bigdisadvantage of having an exponential running time. Geneticand randomized algorithms on the other hand do not generallyproduce an optimal access plan. But in exchange they aresuperior to dynamic programming in terms of running time.However,a dynamic programming approach give us efficient solution for Query optimizationin homogeneous distributed database system.We use "JOIN OPERATION" to estimating the cost in an intermediate stage of execution. If a new JOIN OPERATION estimates the lesser cost than we use that NextJoin and corresponding minimum cost. Hence this process used iteratively for multiple queries in homogeneous distributed database system.

14. References

[1] I.H. Toroslu, A. Cosar, Dynamic programming solution for multiplequery optimization problem, in: Information Processing Letters 92 (2004) 149–155

[2] Reza Ghaemi, Amin MilaniFard, Hamid Tabatabaee, and Mahdi Sadeghizadeh, Evolutionary Query Optimization for Heterogeneous Distributed Database Systems, in:World Academy of Science, Engineering and Technology 43 2008

[3] J. Callan, "Distributed information retrieval". In Advances inInformation Retrieval, W. B. Croft, Ed. Kluwer Academic Publishers,2000, pp. 127–150.

[4] Li, Victor O. K. "Query processing in distributed data bases", MIT. Lab.for Information and Decision Systems Series/Report no.: LIDS-P ; 1107,1981

[5]Sukhoon Kang*, Songchun Moon, Global query management in heterogeneous distributed database systems, in: Volume 38, Issues 1–5, Pages 1-861 (September 1993)Proceedings Euromicro 93 Open System Design: Hardware, Software and ApplicationsBarcelona6–9 September 1993

[6]Dilşat ABDULLAH, Query Optimization in Distributed Databases1302108, in: Middle East Technical UniversityDecember 2003

[7] Eugene Jon Shekita, Honesty Cheng Young, Iterative dynamic programming system for query optimization with bounded complexity, in: United States Patent: 5,671,403 September 23,1997

[8]Yannis E. Ioannidis, Query Optimization, in: University of WisconsinMadison, WI 53706

[9]QIANG ZHU, PER-AKE LARSON,Solving Local Cost Estimation Problem for Global
Query Optimization in Multidatabase Systems, in: 1998 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

[10] B.M. MonjurulAlom, FransHenskens and Michael Hannaford, Query Processing and Optimization in Distributed Database Systems, in: IJCSNS International Journal of Computer Science and Network Security, VOL.9 No.9, September 2009

[11] C. S. Mullins, "Distributed Query Optimization," 1996.

[12] Deepak Sukheja ,Umesh Kumar Singh, A Novel Approach of Query Optimization for Distributed Database Systems, in: IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 4, No 1, July 2011 ISSN (Online): 1694-0814

[13] Z. G. Ives, D. Florescu, M. Friedman, A. Y. Levy, and D. S.Weld.An adaptive query execution system for data integration.In*SIGMOD*, 1999.

[14] A. Tomasic, R. Amouroux, P. Bonnet, O. Kapitskaia, H. Naacke, and L. Raschid.The distributed information search component (DISCO) and the world wide web. In IEEE 1998, Volume: 10 Issue: 5 pp. 808-823.

[15] Kristina Zelenay,"Query Optimization", ETH Zürich, SeminarAlgorithmenfürDatenbanksysteme, June 2005

[16] Yannis E. Ioannidis and Youngkyung Cha Kang, "RandomizedAlgorithms for Optimizing Large Join Queries"

[17] Michael Steinbrunn, Guido Moerkotte, Alfons Kemper, "Heuristic andRandomized Optimization for the Join Ordering Problem", The VLDBJournal - The International Journal on Very Large Data Bases, Volume 6, Issue 3 (August 1997), Pages: 191-208, ISSN:1066-8888

[18] M. Tamer Özsu, Patrick Valduriez, "Principles of Distributed DatabaseSystems, Second Edition", Prentice Hall, ISBN 0-13-659707-6, 1999

[19] Stefano Ceri, Giuseppe Pelagatti, "Distributed Databases: Principles andSystems", Mcgraw-Hill, ISBN-10: 0070108293, ISBN-13: 978-0070108295, 1984