Query Optimization in Heterogeneous Distributed Databases

Akanksha Gupta¹, Prof. Vijay B. Aggarwal²
¹MCA-III Yr, Jagan Institute of Management Studies, GGSIP University, Delhi.
²Professor, Jagan Institute of Management Studies, GGSIP University, Delhi.
¹akankshamh@gmail.com , ²vijay.aggarwal@gmail.com

ABSTRACT
Heterogeneous Distributed database management systems (DDBMS) are amongst the most important and successful software developments in this decade. Query Processing is much more difficult in distributed environment than in centralized environment because a large number of parameters affect the performance of distributed queries like data translation, relations may be fragmented and replicated and considering many sites to access, query response time may be high. Database users may need not only their own data, but also data in other databases to solve a specific problem which may be distributed at various sites. The various factors which affect the query optimization are:

1) I/O Access: It involves accessing the physical data stored on disk.
2) CPU Processing time: It is associated with the transmission of data among nodes in distributed database systems.
3) Communication Cost: It is associated with the processing overhead of managing distributed transaction.

In Heterogeneous Distributed Databases, Communication cost is dominant factor. Various algorithms for query optimization were already devised which attempts to reduce the quantity of data transferred. The distributed query optimization has several problems related to cost model, large set of queries, optimization cost and optimization interval.

Few of the algorithms already devised are INGRES Algorithm, R* Algorithm, SDD-1 Algorithm. INGRES Algorithm uses fragment and replicate query processing strategy. Its main goal is to achieve high degree of parallelism by partitioning one relation among the processing sites and replicating all other needed relations. Exact optimization of query is not possible as accurate database statistics is not available. The term “Query Optimization” will be used to refer to strategies intended to improve the efficiency of query evaluation procedure. The goal of this paper is to study various components of Query Optimization and various algorithm already devised and its advantages and disadvantages and also to give some suggestions how query optimization take place as it is not possible to devise any algorithm which consider all the factors required for Query Optimization.


INTRODUCTION
In case of Query Optimization in Heterogeneous Distributed databases, there are different types of database management system present on different sites as some sites may have Oracle, some sites may have db2 and some other may have Sybase. Therefore there is a problem of integrating all the databases into one common interface and doing querying optimization of heterogeneous data sources. There is a need for translator or mediator which can provide uniform query interface and deals with problem of heterogeneity. There should be some common language which can convert all the queries into a single format. For a given query, translator generates various execution plans and all these plans differ widely in execution cost(in terms of execution time, communication cost, I/O access cost). From various execution plans an optimization plan having least cost is chosen. Components of query optimization includes components search space, search strategy, Distributed cost model.

But in heterogeneous databases, there is much more difficulty due to lack of underlying databases and accurate database statistics available. Another way of achieving interoperability in multi-database system is through the support of common data model along with single global query language on top of different types of existing systems. The global schema of a multi-database system is the result of schema integration of the schemas exported from the underlying databases i.e. local databases. A global query language is used by users of the multi-database system to specify query against global query. Each site has a local autonomy to determine and can take into account various factors such as resource consumption, I/O access cost, CPU processing cost and other administrative issues. This paper deals with how query optimization take place on various distributed systems locally and then result is combined to give global query optimization in case of heterogeneous database systems. In this paper we consider that there is a translator which converts query into common interface language and then query is made available at various sites and then local optimization of distributed query at various sites takes place and then finally outputs are combined from various sites to give final output.

In this paper we mainly concentrate how local optimization of query is done and how local optimization can be improved. This paper consists of components of query optimization which includes Search space, Search strategy, and Distributed cost model. Search space includes various plans which make use of joins, semi-joins, linear trees, bushy trees etc. Search strategy includes basically three types of Dynamic programming:
Greedy approach and Random strategies. We will briefly explain these Components. This paper also describes various advantages and disadvantages of already designed algorithms like distributed Ingres, SDD-1, R* algorithms and considers one of the algorithm which is called genetic algorithm based on chromosomes design, genetic operations, fitness function and theory of natural selection. We also suggest some measures how to deal with query optimization in heterogeneous distributed databases.

1. DESIGN OF QUERY OPTIMIZATION PLAN

<table>
<thead>
<tr>
<th>Control Site</th>
<th>Local Sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Decomposition</td>
<td>Local Optimization</td>
</tr>
<tr>
<td>Global Schema</td>
<td>Local Schema</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Control Site</th>
<th>Local Sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Localization</td>
<td>Optimized Fragment Query with Communication Operations</td>
</tr>
<tr>
<td>Fragment Schema</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Control Site</th>
<th>Local Sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global Optimization</td>
<td>Optimized Local Queries</td>
</tr>
<tr>
<td>Statistics on Fragments</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Control Site</th>
<th>Local Sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global Relations</td>
<td>Local Relations</td>
</tr>
</tbody>
</table>

There are four steps in local query optimization as follows:

1) Query decomposition
2) Data localization
3) Global optimization
4) Local optimization

Step 1) and step 2) need to be done for local databases that are fragmented as well as for fragmented distributed databases.

Step 3) and step 4) is to find the best permutation of ordering of operations in order to minimize cost and optimize performance. The calculation to find optimal or minimal optimal solution is too difficult. Therefore goal is to find near-optimal or near-minimal solution or to avoid bad execution plans. To find optimal solution various factors considered are 1) I/O Access cost 2) CPU processing time 3) Communication costs. Therefore to determine these factors we need accurate fragment statistics.

2. COMPONENTS OF QUERY OPTIMIZATION:

   Search space, Search strategy and Distributed cost Model which are described as follows:

2.1. SEARCH SPACE

Search space consists of all possible plans of given complex query i.e. various relations or trees generated after applying different operations like join, semi-join etc on relations present at different sites required for execution. For e.g. total number of alternative join trees that can be produced for N relations is O(N!) which is very high i.e. there are large number of execution plans possible. If the number of relations increases the search space increases exponentially. E.g. if number of relations are 10 than search space is O (10!) which is really a very big number thus it is required to reduce the number of alternative plans. Therefore to reduce the number of alternative plans possible for search space various assumptions are made which are as follows:

Heuristic 1: Selection and Projection are performed first while accessing a base relation.

Heuristic 2: Avoid Cartesian product as Cartesian product includes large number of tuples thus Cartesian product must be avoided wherever possible.

Heuristic 3: If only linear trees are used then Search space is O(2^n).

Search space includes the plans that have low cost. Search space considers Ordering of joins, Semi-joins, Generalize join sequencing i.e Bushy or Linear trees, Group-by & Join, Merging views to determine various alternative plans.

2.1.1 GENERALIZE JOIN SEQUENCING

In this case we consider how join can be applied on various relations as there are two types of trees: Linear trees and Bushy trees.

**Linear Trees:** These are the trees in which at every node a new base relation is introduced.

**Bushy Trees:** These are the trees in which base relations are introduced at the bottom row.

[Image: (a) linear join tree (b) bushy join tree]

Suppose there are four relations A,B,C,D than Search space for linear tree is (Join(Join(Join(A,B),C),D)) and the Search space for Bushy tree is (Join(Join(A,B),Join(C,D))).

2.1.2 JOIN ORDERING:

During join processing, we ignore local processing time for selection and projection. We only consider join queries of relations on different servers. In case of joins whole relation is transferred instead of tuple from one site to another. Suppose there are three relations Emp, Assg, Proj than according to join ordering we estimate the cardinality of each Emp * Assg, Assg * Proj, Emp * Proj. Then calculate lowest cost ordering. Suppose Emp*Assg combined with Proj has lowest cost then...
this order is followed. We mainly use Cartesian product to determine the size.

\[
\text{R} \quad \text{S}
\]

\[
\text{if } \text{size}(R) < \text{size}(S)\]

\[
\text{if } \text{size}(R) > \text{size}(S)
\]

2.1.3 SEMI-JOINS
JOINS can be replaced by combination of semi-joins to optimize communication cost. Considering the same relations as above and performing semi-joins. Empno is send from Emp to Assg and then semi-join is performed on that server. Then send a proj-no column of Empno*Assg semi-join to location of project. Perform a semi-join there. Send the resulting empno column of Emp * Assg semi-join to the locations of Emp & Assg tables asking for corresponding rows to be sent to the location of the proj table than do a three way join of reduced number of rows of Assg and Emp with Proj table. Semi-joins are less costly as it is cheaper to send just one column or to send just needed rows of a table instead of sending full table. It works well only when there is a subset of matches. These are useful only in case of network.[5]

2.1.4 GROUP-BY AND JOINS
In most of the queries select, project, join precedes group-by clause. However it is also possible that group-by precedes select, project and join query. These transformations are applicable to queries with SELECT DISTINCT since the latter work when the views are more complex than single SPJ queries. By means of above techniques we can determine search space and reduce the alternative plans available. Next we will consider search strategy.[5]

2.2 SEARCH STRATEGY:
This explores the search space and various alternative plans from search space are evaluated and than optimal plan is selected. Various strategies to determine the best execution plan are as follows:
1) Dynamic programming
2) Greedy algorithm
3) Randomized strategy

2.2.1 DYNAMIC PROGRAMMING
Various steps of Dynamic programming are as under
1) Characterize the structure of Optimal solution i.e. find the structure of optimal plan from the various alternative plans.
2) Recursively define the value of optimal solution i.e. eliminate non-optimal plans to find optimal solution.
3) Compute the value of optimal solution in bottom up manner.
4) Find the optimal solution from the computed information.
It is deterministic in nature. If there is any plan which is found to be non-optimal that plan is ignored. It produces optimal left–deep processing tree. [10]
The big disadvantage of this technique is that it has exponential running time as this programming involved all possible feasible joins. (Feasible joins means all possible joins). To overcome this problem we have iterative dynamic programming.

2.2.2 ITERATIVE DYNAMIC PROGRAMMING
This programming is used to prune sub-plans earlier so that total number of cost estimates required is much less. The main idea behind this algorithm is to heuristically choose and fix a sub-plan for a portion of query before the optimization is fully finished. Algorithm is as follows:
1) Enumerate all feasible k-way joins i.e. all feasible joins that contains less than or equal to k base tables. K (<n) is a parameter to the algorithm.
2) Find costs for all these joins by communicating the data sources using a single round of communication.
3) Choose on sub-plan (and the corres. K-way join) out of all the sub-plans for these k-way joins using an evaluation function and throw away all the other sub-plans.
4) If not finished yet, repeat the optimization procedure using intermediate relation and rest of the relations not part of it.
In our study, we use a simple evaluation function that chooses the sub-plan with lower cost. This is a better approach than Dynamic Programming.

2.2.3 GREEDY APPROACH
It is a top-down approach. It may not find globally optimal solution but ensure locally optimal solution at each step. Various elements of greedy approach are:
1) We have to determine optimal substructure in the problem i.e. we have to prove that if the problem has optimal solution than its sub-problem will also have optimal solution.
2) Develop a recursive solution to the problem.
3) We have to prove that at any stage of recursion one of the optimal choice is the greedy choice and it is always safe to select the greedy strategy.
4) We have to show that at-least one of the sub-problem using greedy strategy is empty so we have to solve only one sub-problem rather than two as in case of Dynamic programming.
5) Finally greedy strategy is implemented through iterative algorithm.

Greedy algorithm builds only one plan depth first thus it first considers linear trees.

2.2 RANDOMIZED STRATEGY
It starts with greedy algorithm and tries to improve by visiting its neighbours. Neighbours are found by switching two steps in the plan. For more than a few relations, Randomized does better than deterministic.

2.3 DISTRIBUTED COST MODEL
It deals with total time and response time. Total time is the sum of all times i.e
\[ T(CPU) + T(I/O) + T(MSG) + T(TR) \]
where:
- \( T(CPU) \) is the time taken by processor,
- \( T(I/O) \) is the time taken in I/O Access,
- \( T(MSG) \) is the time taken by message from initiation to end and
- \( T(TR) \) is the time taken in transmission of messages.

Response Time of query is the time involved during parallel local processing and communications.
\[ T(CPU)*seq#(INST) + T(I/O)*seq#(I/O) + T(MSG)*seq#(MSGS) + T(TR)*seq#(BYTES) \]

Distributed cost model also involves DATABASE STATISTICS which is explained below.

2.3.1 DATABASE STATISTICS
Database statistics is useful in determining total cost incurred during transmission of messages. It is based on formulas and cardinalities of relations. Generally relations having less number of tuples are send from one site to another. If site A has less number of tuples than relation B then relation A is send to site B and further processing takes place.

Considering all the above mentioned components of query optimization various algorithms are designed which are not up to the mark and problem is not fully solved yet.

3. VARIOUS DISTRIBUTED ALGORITHM:

3.1 DISTRIBUTED INGRES ALGORITHM:
a) It mainly works using Dynamic programming algorithm i.e. it follows bottom-up approach considering all the plans first and then eliminating non-optimal plans.
b) As in case of Dynamic Programming it breaks query into number of smaller queries recursively.

c) Queries are broken on the basis of common variable among relations and than pieces of query are executed turn by turn. It mainly works using Linear join tree strategy.[1] The main disadvantage of this algorithm is that it is based on limited search strategy. This is not useful as the number of relation increases more than three.

3.2 R* ALGORITHM
a) It uses static optimization algorithm based on an exhaustive search of solution space. It uses database statistics to determine the total cost involved.
b) Under this strategy all the candidate trees are given particular cost and lowest cost tree is retained.
c) The total number of trees possible is determined using dynamic programming under which those paths which are not optimal are not considered and also which includes Cartesian product is not considered. [1]
The disadvantage of this algorithm is that it is costly and does not deal with fragments.

3.3 SDD-1
a) This algorithm is derived from hill-climbing algorithm. Under this feasible solution is determined first and than further refinements of feasible solution is done.
b) Refinements are done until optimal solution is obtained
c) This algorithm is mainly based on total cost. It is based on greedy algorithm as it starts with initial feasible solution and then improvements are made iteratively. [1]
The main thing is strategies with high initial costs are eliminated and end with best overall cost.

4. GENETIC ALGORITHM
Proposed by Holland which is further revised by Goldberg describes “Evolutionary Query Optimization for heterogeneous databases” i.e. how at each step new query is evolved from previous query.

As according to Holland and Goldberg this algorithm Genetic Algorithm based optimization is independent of search space and depends upon natural genetics theory which includes chromosome design, GA operators and fitness function. [2]

4.1 CHROMOSOME DESIGN
Under this design join ordering is considered as combination of strings. e.g. (join(join(join(R1,R2)R4)R5)) can be written as 1245. This will convert the join ordering as Travelling Salesman Problem (TSP). This string represents join order from one relation to next relation of query. This will make the processing faster and it is also easy to understand and evaluate.

4.2 GA OPERATIONS
According to this one chromosome i.e. relation is combined with other chromosome i.e. another relation and than resulting chromosome having hybrid characteristics of both the chromosome is obtained having features of both the chromosome which are favourable and best chromosome
survive. It means best relation is transferred at the next generation and this will ensure selection of best query plan.

4.3 FITNESS FUNCTION
It is one of the very important steps in this algorithm. This will lead us to best solution. Under this function one cost estimator is described where Random(x) returns value between 0 and x.

Fitness-Random  $(R*S)+(R+S)$; if $(R+S)>=(R*S)$
Random  $(R+S)+(R*S)$; else

4.4 ALGORITHM DESIGN
Under this value for fitness function is calculated for each parent chromosome and lower the value of chromosome best is the chromosome and it has the more chances of generating the better offspring in next generation. Thus fitness function value is calculated repeatedly until the best chromosome is generated

This algorithm is better than other algorithms which uses dynamic programming as dynamic programming uses exponential running time. Thus this algorithm is superior to DP in terms of access plan as well as in terms of overall running time.

5. SUGGESTIONS
As there is no algorithm which can optimize query in best possible manner in all conditions. Always there are some problems. There should be two types of algorithm INBUILD-ALGORITHM and USER DEFINED ALGORITHM. In-build Algorithm is one which is already defined within the database system and whenever query is executed that particular algorithm is executed during query optimization irrespective of kind of query .User-defined algorithms are those which are defined by the programmer with the help of coding. User defined algorithms consider various factors while coding for query-optimization. Various factors while programming kept in mind are as follows:

1) Size of the tables as this is the most important factor in deciding what the data to be transmitted between sites is.
2) Location of fragments as in case of distributed query relations are fragmented and stored at different sites thus data needs to be retrieved from various sites.
3) Indexing is required or not during data access and the type of index.
4) Logic required solving the complex requests i.e. actual coding required
5) Type of join to be applied i.e. merge scan, nested loop or hybrid join etc

Thus programming is done to get optimal solution by considering all the factors as some types of query processing needs procedural logic like looping, if and else statement and other similar logic. As complexity of query is increasing it is better to design a program using coding which will optimize query. Various other factors kept in mind while programming are as follows:

a) It is possible that any site is not available due to any failure like software or hardware failure.
b) All the data relations may not be available at all the sites.
c) Communication highly depends upon network traffic. There may be chances of high traffic at some point of time and no traffic at other point of time. When there is no traffic response time may be faster. Network may be modified by introducing new nodes and then determining new optimal path. Thus user defined programming is better approach as it leads to develop efficient query processing plans.

CONCLUSION
Though it is not possible to design an algorithm which considers all the factors required during query optimization. As queries involved are very complex thus any single algorithm cannot resolve all the queries in a best possible manner. One algorithm may be suitable for particular query but not suitable for another query. As we have seen various algorithm described above some using dynamic strategy ,some using greedy approach but all these strategies does not produce best optimal plan in every condition. Under some conditions Dynamic programming should be followed and sometimes greedy approach may be beneficial. Type of algorithm used depend upon number of factors like complexity of query, number of relations involved, data to be communicating between sites etc. Thus better to have user-defined algorithm which is dynamic in nature which is designed depending upon type of query to be optimized. For each query some programming logic should be applied to get optimal solution.

FUTURE SCOPE
This Paper will be helpful in making query processing more reliable and we can store user defined algorithm at various sites as stored procedures which will be very useful as this will reduced amount of data to be fetched at various sites.

REFERENCES
[7] Chaudhuri, S.Shim K “Query optimization including Group by-join”
[10] Decoupled query optimization for federated database systems by Amol Deshpande, Joseph M.Hellerstein

Copy Right © INDIACom – 2007
[11] "Distributed query optimization" by Craig S. Mullins