IJREAT International Journal of Research in Engineering & Advanced Technology, Volume 2, Issue 2, Apr-May, 2014
ISSN: 2320 – 8791 (Impact Factor: 1.479)
www.ijreat.org

Optimization of Join Queries on Distributed Relations Using Semi-Joins

Suresh Sapa[1], K. P. Supreethi[2]

[1, 2]JNTUCEH, Hyderabad, India

Abstract

The processing and optimizing a join query in distributed database system using sequence of semi-joins has been done by using semi-join based algorithm. The main objective of this kind of algorithm is to predict the best execution plan for a join query retrieving data from two or more remote computers.To accomplish this, the two major components are query rewriter and query optimizer, but we focus on query optimizer part, particularly, on cost based query optimization. The algorithm which has been used to optimize a join query is SDD-1 algorithm, which uses semi-joins concept extensively. The objective function of SDD-1 is to minimize total communication time.

Keywords: *Distributed database, semi-joins, query optimization, distributed relations, semi-join based algorithm, optimization techniques, query processing.*

1. Introduction

1.1 Distributed Database System

Distributed database is a collection of multiple, logically interrelated databases over a computer network. Simply, it is a collection of data which belong logically to the same system but are spread over a network of interconnected systems. A distributed database management system is then defined as the software system that permits the management of the DDBS and makes the transparent to the users. The two important terms in these definitions are "logically interrelated" and "distributed over a computer network". [1, 2]The sample distributed database environment looks like:

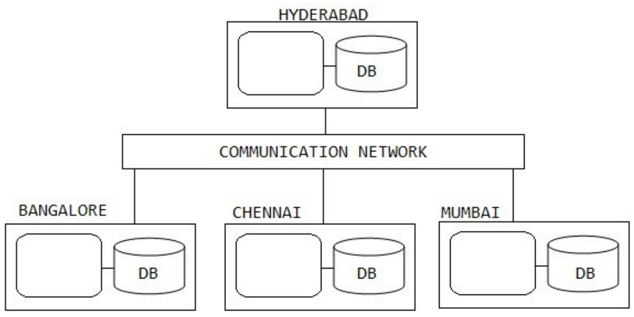

Figure 1 : Distributed database environment

1.2 Query Processing

Query Processing is the process of translating a high-level query (i.e., in relational calculus) into an equivalent lower-level query (i.e., in relational algebra). *Query Processor* helps to simplify and facilitate the access to the database.

The input to the query processor must be a relation calculus query.

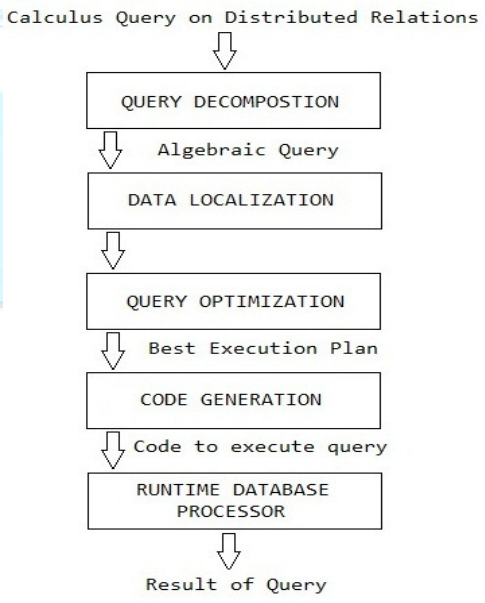

Figure 2 : Query Processing

Query decomposition is the initial step where scanning, parsing and validation of an input query are done. It decomposes distributed calculus query into an algebraic query on distributed relations. The main role of *data localization* is to localize the query's data using data distribution information. We see information about Query optimization in section 1.3. In the next step, code would be generated for the selected execution strategy; this code is the executed in either compiled or interpreted mode to produce the query result [5, 6].

1.3 Query Optimization

Query optimization refers to the process of producing a query execution plan (QEP) which represents an execution strategy for the query. The selected plan minimizes an objective cost function. The cost function can be expressed with respect either the total time or response time.
A *Query Optimizer*, the software module that performs query optimization and is usually seen as three components: a search space, a cost model, and a search strategy [1].

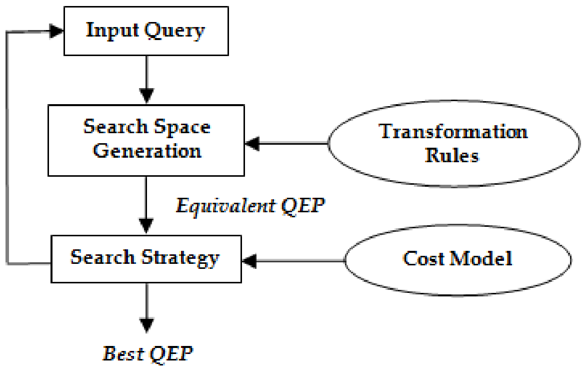

Figure 3 : Query Optimization Process

A. The *search space* is a set of alternative execution plans to represent the input query. These plans are equivalent, in the sense that they yield the same result but they differ on execution order of operations and the way these operations are implemented, and therefore on performance.
B. The *search strategy* explores the search space and selects the best plan, using the cost model.
C. An *optimizer's cost model* includes cost functions to predict the cost of operators, statistics and base data and formulas to evaluate the sizes of intermediate results. The cost model must have good knowledge about the distributed execution environment.

2. Related Work

Many research projects and a few commercial systems have implemented and evolved the concept of distributed query optimization.

Mariposa [4] is a distributed database research system, which proposed the use of an *economic paradigm*. The main idea behind the economic paradigm is to integrate the underlying data sources into a computational economy that captures the autonomous nature of various sites in the federation. A significant goal of Mariposa was to demonstrate the global efficiency of this economic paradigm. In terms of distributed load balancing, the "global efficiency" is closely related to the reason why we need to consider run-time condition. However, the paradigm is built on the assumption that each site has total local autonomy to determine the cost to be reported for an option, and can take into account factors such as resource consumptions and hard-ware conditions. There are a few controversies over this assumption: (1) the fully decoupled costing process without a global coordinator / mediator cannot ensure quality of query answering; (2) the requirements for data sources that want to join in the system will be high.

Several approaches have been proposed to enumerate over the search space of equivalent plans for a given user query. These include randomized solutions such as Iterative Improvement (II) and Simulated Annealing (SA) [23] and heuristic-based methods such as the minimum selectivity heuristic [19]. This heuristic is described further in section 2.3.2. Query re-writing [25] and query simplification [12] techniques have also been recently explored where the query graph is restricted in an attempt to reduce the complexity of the optimization.

Iterative Dynamic Programming (IDP) was proposed in [9] in 2000 to overcome the space complexity problem of DP. However, DP and IDP do not consider the query structure or join graph, which consequently leads them to consider cross products. Cross products are costly and result from joining relations together that do not have a join condition present. The DPccp algorithm proposed in [21] in 2006 uses the join graph of a query to perform dynamic programming without considering cross products. DPccp still has the same worst case running time as DP (in the case where a clique query is being optimized) but in practise it produces considerable savings in time when optimizing chain and cycle queries.

Garlic at IBM [7] is the first research project to exploit the full power of a standard relational database (DB2). The wrapper architecture and cross-source query optimization of Garlic are now fundamental components of IBM‟s federated database offerings [9]. Later, Garlic developed a complicated framework for cost-based query optimization across sources [8]. However, all cost factors and cost formulas used in the framework are within the context of traditional query optimizer. Features of distributed system such as hardware conditions and run-time conditions of data sources were not studied in Garlic.

3. Preliminary

3.1 Proposed Method

One important observation in query optimization over distributed database system is that run-time conditions (namely available buffer size, CPU utilization in machine and network environment) can significantly affect the execution cost of a query plan. It is a challenging problem, because considering run-time conditions of remote sites will bring extra complexity to the optimizer.
This paper proposes *EFFICIENT QUERY OPTIMIZER* while efficiently considering these runtime conditions.

Semi-join based Algorithm
Semi-join operation can be used to decrease the *total time* of join queries. The semi-join acts as a size reducer for a relation much as a selection does. It is said to be a better approach if the semi-join acts as a sufficient reducer, i.e., if only a few tuples of a relation participates in the join.

Let us consider an example of a program to compute a join EMP $\bowtie$ ASG $\bowtie$ PROJ is EMP' $\bowtie$ ASG' $\bowtie$ PROJ.
Where EMP' = EMP $\ltimes$ ASG
ASG'= ASG $\ltimes$ PROJ.
We can even reduce the size of an operand relation by using more than one semi-join. It can be derived as

EMP''=EMP $\ltimes$ (ASG $\ltimes$ PROJ).

If *size*(ASG $\ltimes$ PROJ) <= *size*(ASG) then the result could be *size*(EMP'') <= *size*(EMP'). In this way, EMP can be reduced by the sequence of semi-joins. Such a sequence of semi-joins is called as *'semi-join program'* for EMP.

Similarly for PROJ relation could be reduced by the semi-join program,

PROJ''= PROJ $\ltimes$ (ASG $\ltimes$ EMP).

Not all relations involved in a query need to be reduced; ignore relations that are not involved in final joins.
There exist many *semi-join programs* for a particular given relation. But there is only one optimal semi-join program, called the *Full Reducer* [12]. A simple method is to evaluate the size reduction of all possible semi-join programs and to select the *best one*.
The problems with this enumerative approach are:-

1. *Cyclic Queries* are the queries which have cycles in their join graph for which full reducers cannot be found.
2. *Tree Queries* for which full reducers exists, but the problem of finding them is NP-hard.

The solutions for 1, is to transform the cyclic graph in a join graph into an equivalent acyclic graph by removing one arc of the graph and adding appropriate predicates to the other arcs.

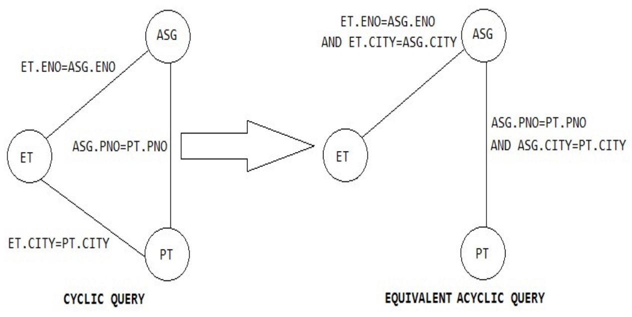

Figure 4 : Transformation of Cyclic Query

In this graph, the removed predicate is preserved by *transitivity* [13].

3.2 Assumptions and Other Restrictions

Below is the list of current assumptions and restrictions. These assumptions make our problem feasible and hold in almost all real systems.

- ✓ The physical database design of each data source would be known.
- ✓ The Statistical information about the data should be known before query execution.
- ✓ Queries are answered based on up-to-date knowledge.
- ✓ The optimizer's cost model must have good knowledge about distributed execution environment.
- ✓ Semi-join based algorithm is studied in this paper; when estimate resources consumed in a data source by a query, our proposed algorithm can be used to

optimize a query on distributed relations that costs less.

4. Experimental Setup and Evaluation

Let us consider two phases; first phase deals with distributed database environment and retrieves data from different sites. Second phase deals with optimizing the query in terms of time, that is, either total time or response time.

Phase I:

Initially, I have considered five systems which can communicate through LAN among themselves and the data which consists of *crime statistics* of each US state from 1960-2005. The data what I considered is downloaded from [14] in csv file format, later I stored into database using a JDBC application.
Instead of all US states, I have considered only five states data (Arizona, Texas, New york, Florida, California) and distributed among five sites (systems) in such a manner that each site consists of one particular state data, as shown in figure 5.

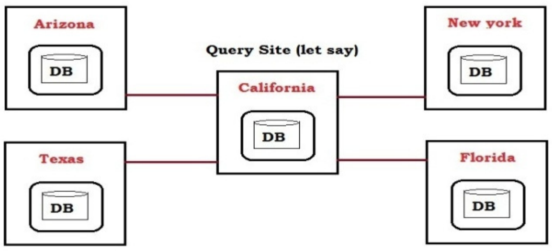

Figure 5 : Distributed Database Environment

Now, distributed database environment is developed in such a fashion that, from any site (system), we can retrieve information from other four sites. It can be achieved through Socket programming where it gets given query as input through input stream and sends result of query as output through output stream. The distributed data information is captured in a file. By using this, our application knows from which site to retrieve data as per user request.
The block diagram of this application is,

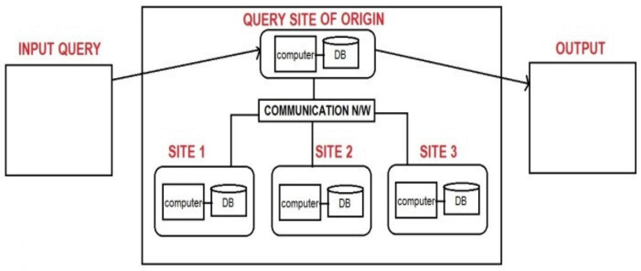

Figure 6 : Block Diagram

Whenever user sends a query, they might think that data are stored at single site and required information is retrieving from this current site only, not from other remote sites, but it is false. This is what we call *transparency.*
A *query input screen* is just like a SQL command prompt, but the difference is sql prompt retrieves data from current site where our *query screen* can retrieve data from current site as well as remote sites based on query we given.
A query has to be given at input screen and the query is evaluated at query site of origin, retrieves information from required sites and displays it to the user.

Phase II:

To optimize any query, we need to go through an optimization technique. Here, I have considered SDD-1 algorithm to perform optimization of query.
Let us see how optimization technique works with this algorithm.

SDD-1 Algorithm:

SDD-1 is derived from hill climbing algorithm. *Hill-climbing algorithm* is the first distributed query processing algorithm [15]. It has been substantially improved in SDD-1 in a number of ways. The improved version makes extensive use of semi-joins. The objective function is expressed in terms of total communication time. Finally, this algorithm uses statistics on database, called *database profiles*, where each profile is associated with a relation.
The main step of the algorithm consists of determining and ordering beneficial semi-joins, that is semi-joins whose cost is less than their benefit. The cost of a semi-join is that of transferring the semi-join attribute A.

$$\boxed{\text{Cost}(R \ltimes_A S) = T_{MSG} + T_{TR} * \text{size}(\Pi_A(S))}$$

While its benefit is the cost of transferring irrelevant tuples of R which is avoid by semijoins.

$$\boxed{\text{Benefit}(R \ltimes_A S) = (1-SF_{SJ}(S.A))*T_{TR}*\text{size}(\Pi A\,(S))}$$

Algorithm:-

Input: QG: query graph with n relations; statistics for each relation.

Output: ES: execution strategy

```
begin
  ES ← local-operations (QG).
  Modify stats to reflect the effect of local    processing.
  BS ← null {set of beneficial semi-join}
   for each semi-join SJ in QG do
        if cost(SJ) < benefit(SJ) then
              BS ← BS U SJ.
        end-if.
  end-for
 while BS != null do {selection of beneficial semi-join}
  begin
 Sj ← most_beneficial(BS) {SJ: semijoin with max(benefit-cost)}
 BS ← BS - SJ    {remove SJ from BS}
 ES ← Es + SJ    {append SJ to execution strategy}
 Modify stats to reflect the effect of incorporating SJ
 BS ← BS – non beneficial semi-joins
 BS ← BS U new beneficial semi-joins
end-while
{assembly site selection}
AS(ES) ← select site i such that i stores the largest amount of data after all local operations
ES ← ES U transfers of intermediate relations to AS(ES)
{post optimization}
for each relation $R_i$ at AS(ES) do
  for each semijoin SJ of $R_i$ by $R_j$ do
        if cost(ES) > cost(ES-SJ) then
              ES ← ES – SJ
        end-if
   end-for
 end-for
end.
```

This algorithm proceeds in four phases: initialization, selection of the beneficial semi-joins, assembly site selection, and post optimization. The output of the algorithm is a global strategy for executing the query.

The *initialization phase* generates a set of beneficial semi-joins, BS={SJ_1, SJ_2, SJ_3, … , SJ_k} and an execution strategy that includes only local processing. The next phase selects the *beneficial semi-joins* form BS by iteratively choosing the most beneficial semi-join, SJ_i, and modifying the database statistics and BS accordingly. The modification affects the statistics of relation R involved in SJi and the remaining semi-joins in BS that use relation R. The iterative phase terminates when all semi-joins in BS have been appended to the execution strategy. The order in which semi-joins are appended to ES will be the execution order of the semi-joins. The next phase *selects the assembly site* by evaluating, for each candidate site, the cost of transferring to it all the required data and taking the one with the least cost. Finally, a *post optimization phase* permits the removal from the execution strategy of those semi-joins that affect only relations stored at the assembly site. This phase is necessary because the assembly site is chosen after all the semi-joins have been ordered. The *SDD-1 optimizer* is based on the assumption that relations can be transmitted to another site. This is true for all relations except those stored at the assembly site, which is selected after beneficial semi-joins are considered. Therefore, some semi-joins may incorrectly be considered beneficial. It is the role of post optimization to remove them from the execution strategy.
By using this algorithm, we can optimize join queries.

5. Conclusion

In this paper, we have presented the basic concepts of distributed query optimization and the semi-join based algorithm i.e., SDD-1 Algorithm is considered as best technique to optimize a query on distributed relations. Important inputs to the query optimization problem are the database statistics and the formulas used to estimate the size of intermediate results.
SDD-1 Algorithm computes joins with semi-joins. Semi-joins can act as powerful size reducers only when a join has a good selectivity. Semi-joins implemented by hash bit arrays [10] are shown in [11] to be very beneficial.
Like its predecessor hill-climbing algorithm, the SDD-1 algorithm selects locally optimal strategies. Therefore, it ignores the higher-cost semi-joins which could result in increasing the benefits and decreasing the costs of other semi-joins. Finally, this query optimization technique works very well.

References

[1] “Principles of Distributed database systems” - M.Tamer OZSU and Patrick Valduriez . Chapther 7, 8,9, pp 188-271.
[2] “Distributed databases principles and systems”- Stefano ceri and pelagatti.
[3] Oszu, M. T. and Valduriez P., “Distributed and Parallel Database Systems,” in Trucker A. (Ed), The Computer

Science and Engineering Handbook, CRC press, pp. 1093-1111, 1997.

[4] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, Carl Staelin, A. Yu. Mariposa: "A wide-area Distributed Database System". The VLDB Journal, 1996.

[5] Elmasri R. and Navathe S. B., "Fundamentals of Database Systems", Reading, MA, Addison-Wesley, 2000.

[6] Ioannidis Y. E., "Query Optimization," in Trucker A. (Ed), The Computer Science and Engineering Handbook, CRC press, pp. 1038-1054, 1996.

[7] IBM Research, The Garlic Project, http://www.almaden.ibm.com/cs/garlic/.

[8] M. T. Roth, F. Ozcan, L. Hass. "Cost Models Do Matter", Providing Cost Information for Diverse Data Sources in a Federated System. VLDB 1999.

[9] L. M. Haas, E. T. Lin, M. A. Roth," Data Integration through Database Federation", IBM System Journal, VOL. 41, No 4, 2002

[10] P. Valduriez. "Semi-Join Algorithms for Distributed Database Machines". In J.-J. Schneider (ed.,), Distributed Data Bases, Amsterdam: North-Holland, 1982, pp 27-37.

[11] L.F. Mackert and G. Lohman. "R* Optimizer Validation and Performance Evaluation for Local Queries". In proc.ACM SIGMOD Int. conf. on Management of Data, May 1986, pp 84-95.

[12] D.M. Chiu and Y.C Ho. "A Methodology for Interpreting Tree Queries into Optimal Semi-join Expressions". In proc. ACM SIGMOD Int, conf. on Management of Data, May 1980, pp 169-178.

[13] Y. Kambayashi, M.Yoshikawa and S.Yajima. "Query Processing for Distributed Databases using Generalized Semi-joins". In proc. ACM SIGMOD Int. Conf. on Management of Data, June 1982, pp 151-160.

[14] http://hci.stanford.edu/jheer/workshop/data/

[15] E. Wong. "Retrieving Dispersed Data from SDD-1". In Proc. 2nd Berkeley Workshop on Distributed Data Management and Computer Networksm *1977, pp* 217-235.