Volume 4, Issue 11, November 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Improvement of the Performance of a Query Optimization for Distributed System

Ashok Kumar, Shweta Singh
DIT University, Dehradun
Uttarakhand, India

Abstract: ***Query processing is an important concern in the field of distributed databases. The main problem is: if a query can be decomposed into sub queries that require operations at geographically separated databases, determine the sequence and the sites for performing this set of operations such that the operating cost (communication cost and processing cost) for processing this query is minimized. The problem is complicated by the fact that query processing not only depends on the operations of the query, but also on the parameter values associated with the query. Distributed query processing is an important factor in the overall performance of a distributed database system. Query optimization is a difficult task in a distributed client/server environment as data location becomes a major factor. In order to optimize queries accurately, sufficient information must be available to determine which data access techniques are most effective (for example, table and column cardinality, organization information, and index availability). Optimization algorithms have an important impact on the performance of distributed query processing.***

Keywords: Optimization, Processing, Materialized Views, Distributed Database, Cost

I. INTRODUCTION

Distributed Database is a type of database in which the storage devices are not attached to a common processing unit such as the CPU controlled by a distributed database management system, It is a physically dispersed. Due to which the problem of solving the same query is encountered, to avoid it optimization of the query is done. Distributed database System is divided into two types i.e., Isomorphic distributed database management and Heterogeneous Distributed Database Management System. Query Optimization and Query Processing are fundamentals steps for executing the Query. Query Processing means how RDBMS can evaluates and process a query while Query Optimization means how an RDBMS can improve on the performance of a query by re-ordering the operations.

There are some phases involved in distributed Query Processing:-

(a) Local Processing Phase
(b) Reduction Phase
(c) Final Processing or Assembly Phase

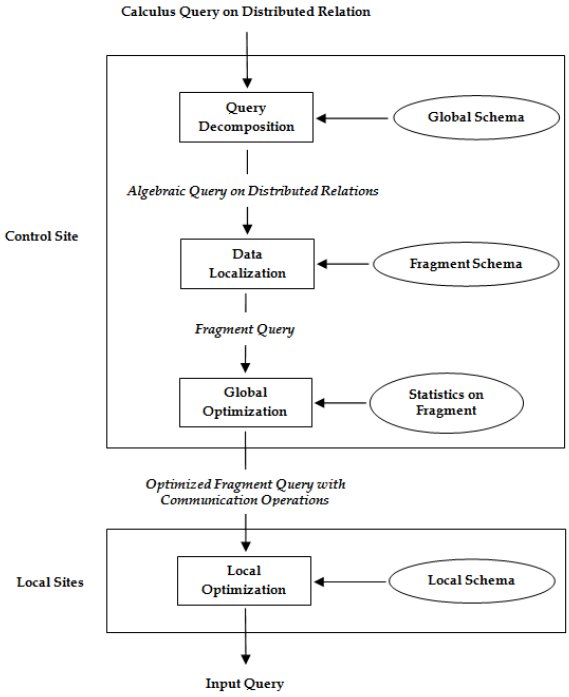

Fig 1. Distributed Query Optimization

The queries on Distributed Databases require efficient processing, which mandate devising of optimal query processing strategies. Query optimization is a difficult task in Distributed Environment because of numerous factors like data allocation, speed of communication channel, indexing, availability of memory, size of the database, storage of intermediate result, pipelining and size of data transmission. Query Optimization in Distributed Database consists of four phases:

(a) Query Optimization
(b) Data Localization
(c) Global Optimization
(d) Local Optimization

The number of possible alternatives query plans increases exponentially with increase in the number of relations required for processing the query. Optimizer is a software module works on three basics component.

(a) Search Space
(b) Search Strategies
(c) Cost Model

II. EFFICIENT AND EXTENSIBLE ALGORITHM

Multi- Query Optimization aims at exploiting common sub-expression to reduce evaluation cost. It uses volcano framework for query optimization i.e. Volcano-SH and Volcano-RU. Volcano algorithm uses hashing scheme to detect repeated derivation and avoid creating duplicate equivalence node due to cyclic derivation. It also detect and handle subsumption.

Volcano-SH algorithm, its plan is choosen taken sharing of parts of earlier queries into account but in Volcano-RH, depends on the order in which queries are considered, it materialize e if

Cost(e) + mat cost(e) + reuse cost(e) x (num uses(e)-1) < num uses(e) x cost(e)

Where equivalence node e

Cost(e)- Computation Cost of node e

Num uses(e)-No. of time node e is used in course of executing of plan.

Matcost(e)- Cost of materializing node e

Reusecost(e)- Cost of reusing the materialized result of e

Procedure VOLCANO-SH(P)

Input: Consolidated Volcano best plan P for virtual root of DAG

Output: Set of nodes to materialize M, and the corresponding best plan P

Global variable: M, the set of nodes chosen to be materialized

M = { }

Perform prepass on P to introduce subsumption derivations

Let Croot = COMPUTEMATSET(root)

Set Croot = Croot +Pd∈ M(cost(d) + matcost(d))

Undo all subsumption derivations on P where the subsumption node is not chosen to be materialized.

return (M,P)

Procedure COMPUTEMATSET(e)

If cost(e) is already memoized, return cost(e)

Let operator oe be the child of e in P

For each input equivalence node ei of oe

Let Ci= COMPUTEMATSET(ei) // returns computation cost of ei

If ei is materialized, let Ci = reusecost(ei)

Compute cost(e) = cost of operation oe + Pi Ci

If (matcost(e)/(numuses−(e) − 1) + reusecost(e) < cost(e))

If e is *not* introduced by a subsumption derivation

add e to M // Decide to materialize e

else if cost(e) + matcost(e) + reusecost(e) ∗ (numuses−(e) − 1) is less than

savings to parents of e due to introducing materialized e

add e to M // Decide to materialize e

Memoize and return cost(e)

Figure 2: The Volcano-SH Algorithm

Volcano-RU Algorithm, the initialization behind Volcano-RU is to optimize them in sequence ,keeping track of what plans have already been chosen for earlier and considering the possibility of reusing parts of the plans.

Procedure VOLCANO-RU

Input: Expanded DAG on queries Q1, . . . ,Qk (including subsumption derivations)

Output: Set of nodes to materialize M, and the corresponding best plan P

N = _ // Set of potentially materialized nodes

For each equivalence node e, Set count[e] = 0

For i = 1 to k
Compute Pi, the best plan for Qi, using Volcano, assuming nodes in N are materialized
For every equivalence node in Pi
set count[e] = count[e] + 1
If (cost(e) + matcost(e) + count[e] ∗ reusecost(e) < (count[e] + 1) ∗ cost(e))
// Worth materializing if used once more
add e to set N
Combine P1, . . . , Pk to get a single DAG-structured plan P
(M,P) = VOLCANO-SH(P) // Volcano-SH makes final materialization decision

Figure 3: The Volcano-RU Algorithm

AND-OR-DAG, It is directed acyclic graph whose node is divided into AND-nodes and OR-nodes, AND-nodes have OR-nodes as children, OR-nodes have AND-nodes as children.

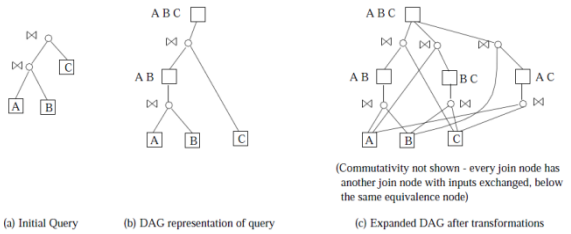

Figure 4: Initial Query and DAG Representations

Greedy Algorithm-The algorithm pick a set of nodes s to be materialized.The main motive behind Greedy Algorithm is Exhaustive Algorithm. To motivate our greedy heuristic, we first describe a simple exhaustive algorithm. The exhaustive algorithm, iterates over each subset S of the set of nodes in the DAG, and chooses the subset Sopt with the minimum value for bestcost(Q, S). Therefore, bestcost(Q, Sopt) is the cost of the globally optimal plan for Q.
Three important points:-

(a) Depends on nodes materialized in the globally optimal plan.
(b) Based on the observation that there any many best cost(Q,S).
(c) Based on monotonicity heuristic.

Monotonicity Heuristic:-
It determines the nodes with smallest value of best cost efficiently. For handling nested queries decorrelation technique is used. In, Decorrelation techniques, queries are transformed into set of queries, with temporary relation being created.

III. OPTIMIZING QUERIES WITH MATERIALIZED VIEWS

Enumeration of possible alternative by the optimizer must be syntax independent and efficient .The presence of materialized views provide the opportunity to fold one or more of the sub expression in the query, thus generating additional alternatives to the unfolded query.

$$L(x,y) \longrightarrow V(x)$$

This is one level Rule, where, L(x,y) is Conjunctive Query and V(x) is Single Literal.
X is projection variable and y is variable in the body of the view definition that do not occurs among projection variable.
It is called one-level rule as, a literal that occurs in the right side of any of the rules does not occurs in any left hand side may have references to only base tables.
Three main steps for optimizing of materialized views:-

(i) The query is translated in the canonical unfolded from as is done in todays's relational system that support views.
(ii) For the givensQuery,Using the one-levels rules,we identify possible ways in which one or more materialized views may be used to generate alternatives formulation of the query.
The above two system ensure syntax independence
(iii) An efficient join enumeration algorithm,that retrofits the system R style join enumeration algorithm is used to ensure that the cost of alternatives formulation are determined and the execution plan with the least code is selected.

A. Safe Substitution:-

Every safe substitution identifies a sub expression in the given query that may be substituted by a materialized view to generate an equivalent query.
More detailed representation of one-level rules that recognizes existence of inequality constraint.

$$L(x,y)\ ,\ I(x) \longrightarrow V(x)$$

I(x) denotes conjunction of inequality constraints that involve only the projection variable x of the rule.
L(x,y) denotes variable y that are not projection variable .

It can also be encoded into double [σ (L), σ (v)]
1st component of doublet is Deletlist which,denotes the subexpression in the query that is replaced due to the safe sustitution.
2nd component of doublet is Addliterals which denotes the literals that replaces deletlist.

B. Traditional Algorithm:-

Query is represented as annotated join tree where the internal node is a join operation and each leaf node is a base table.
Optimal paln for every subexpression Qs of Q is constucted exaclty one and it is stored in the data structre palntable.The complexity of the algorithm is $O(n2^{n-1})$.

C. Extended Algorihtm:-

Optimization is done in presence of equivalent queries(impicity).The execution space over which the optimal paln for the query is being sought is the set of all left-deep trees over the queries that are obtained from Q by safe substitution with respect to R.The optimization problem is to pick an optimal paln from the above execution space with respect to a cost model respect the principle of optimality.The cost of optimization is normalized with respect to the cost of optimizing a single query,as in traditional optimizer. Time complexity is $O(n2^{n-1})$.

IV. COST BASED QUERY OPTIMIZATION

The cost of executing a query includes the following component.

(i) Secondary Storage Access Cost-
Cost for accessing ,reading,searching and writing blocks that resides in secondary storage.
(ii) Storage Cost-
Cost for soring any intermediate file
(iii) Computation Cost-
Cost of performing in-memory operation.
(iv) Memory Usage Cost-
Cost pertaining to the number of memory buffer needed.
(v) Communication Cost-
Cost of communicatnig the query from source to databse and then the query result back to the terminal.

A. Cost Function for Join

(i) Nested Loop Join-
$Cost=B_R+(B_R*B_S)+((JS*r*S)/BFR_J)$
(ii) Single Loop Joins-
(a) For Secondary index
$Cost=B_R+(r*(X_B+SC))+((JS*r*s)/BFR_s)$
(b) For Clustering Index
$Cost=B_r+(r*(X_B+(SC/BRB_b)))+((JS*r*S)/BRBs)$
(c) For Primary Index
$Cost=B_R+(r*(X_B+1))+((JS*r*s)/BRB_J)$
(iii) Merge Sort Joins
$Cost=B_R+Bs+((JS*r*s)/BRB_{J)}$

Where,

B_R = Number of Blocks in Relation R
B_S = Number of Blocks in Relation S
JS = Join Selectivity
BFR_J = Blocking Factor of the 'join' File
X_B = Number of levels of a multi level index for the join attribute 'b' of s
SC = Selection cardinality for the join attribute 'b' of s
BFR_b = Blocking Factor of attribute 'b' of S

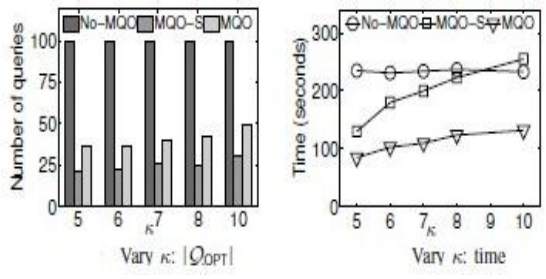

Fig 5.Impact of k no. of speed queries

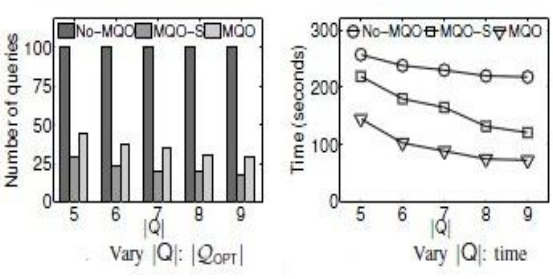

Fig 6. Impact of query size

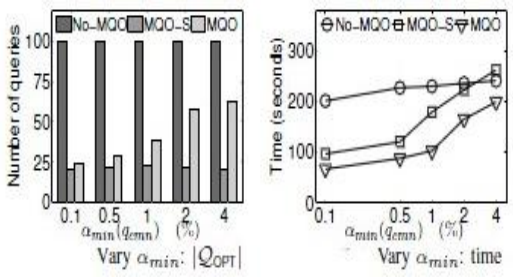

Fig 7. Impact of minimum predicate selection

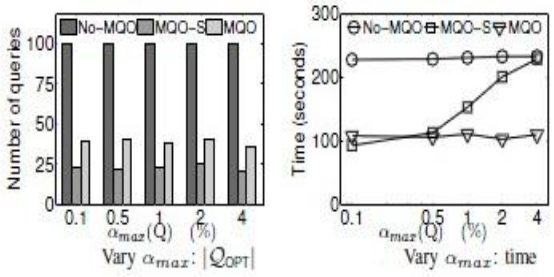

Fig 8 . Impact of maximum selectivity

V. EXPERIMENT STUDY

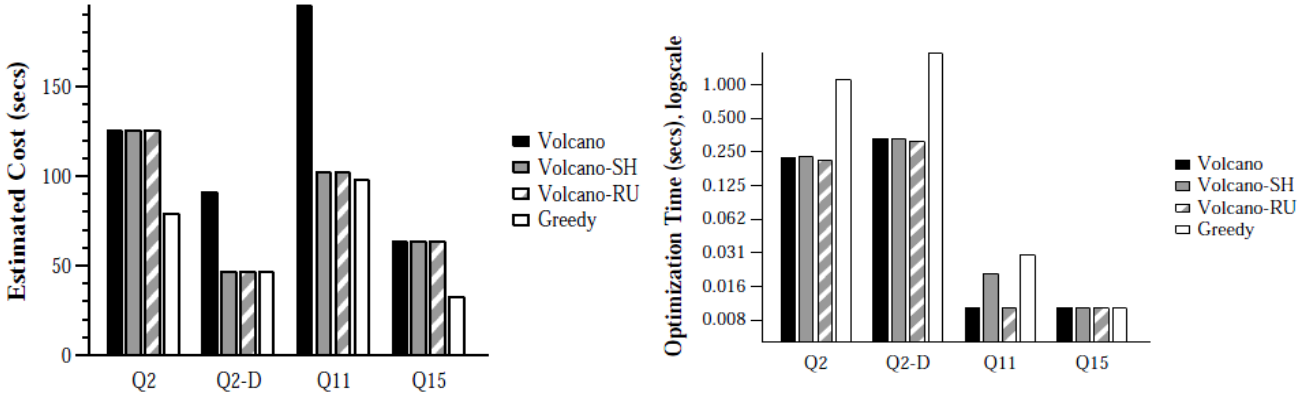

Fig.9 Optimization of stand alone queries

The workload for the first experiment consisted of four queries based on the TPCD benchmark. It used the TPCD database at scale of 1 (i.e., 1 GB total size), with a clustered index on the primary keys for all the base relations.

VI. RELATED WORK

The problem of multi-Query optimization has been well studied in [1][4][6].The main focus is to optimize the multi-query in less cost i.e, in less time of execution. In [8] authors have considered claim that the greedy algorithm can be quite inefficient for selecting views to materialize for cube queries may be due to the lack of an efficient implementation. Another reason is that, for multi-query optimization of normal SQL queries (modeled by our TPC-D based benchmarks) the DAG is "short and fat", whereas DAGs for complicated cube queries tend to be taller.

Semi-connected database query algorithm is designed , which has the data of the intermediate results generated from the implementation of all sub-queries as the decisive factor of network cost, and defines a function to determine the optimization benefits.

VII. CONCLUSION

We studied the problem of multi-Query optimization and analysis many algorithm and techniques which can be used efficiently for the optimization of the queries in which we studied that Runtime optimization plays a better role as compared to the static query optimization and described some best techniques such as Volcano-SH, Volcano-RU and Greedy, for multi-query optimization.

ACKNOWLEDGMENT

It is my immense pleasure to express my deep sense of gratitude and indebtedness to my highly respected guide Ashok Kumar for his guidance during my work who helped me in understanding the optimization of queries. I also extend my thanks to all the staff member of the department who extended their co-operation and help throughout the work and as always to my parents who have been a constant source of inspiration.

REFERENCES

[1] Prasan Roy ,S. Seshadri,S. Sudarshan and Siddhesh Bhobe *Efficient and Extensive algorithm for multi-Query Optimzation.* arXiv:cs/9910021v1 [cs.DB] 25 Oct 1999.

[2] Ms. Preeti Tiwari and Swati V. Chande. *Query Optimization Strategies in Distributed Databse*. Special Issue: Proceedings of 2nd International Conference on Emerging Trends in Engineering and Management, ICETEM 2013.

[3] B.M. Monjurul Alom, Frans Henskens and Michael Hannaford .*Query Processing and Optimization in Distributed Database Systems* .IJCSNS International Journal of Computer Science and Network Security, VOL.9 No.9, September 2009.

[4] Surajit Chaudhuri, Ravi Krishnamurthy, Spyros Potamianos, and Kyuseok Shim. *Optimizing queries with materialized views.* In Intl. Conf. on Data Engineering, Taipei, Taiwan, 1995.

[5] Wangchao Le, Anastasios Kementsietsidis, Songyun Duan and Feifei Li. *Scalable MultiQuery Optimization for SPARQL*. 2012 IEEE 28th International Conference on Data Engineering.

[6] Latifur Khan, Dennis Mcleod and Cyrus Shahabi. *An Adatptive Probe-Based Techniques to Optimize Join Queries in Distributed Internet Database*.This reasrch was supported in part by goft from Informix,lntel NASA/JPL Contract no.961518 and NSf grants EEC-9529152(IMSC ERC) and MRI-9724567.

[7] Fan Yuanyuan and Mi Xifeng. *Distributed Database System Query Optimization Algorithm Research*. 978-1-4244-5539-3/10/$26.00 ©2010 IEEE.

[8] Amit Shukla, Prasad Deshpande, and Jeffrey F. Naughton. *Materialized view selection for multidimensional datasets*. In Intl. Conf. Very Large Databases, New York City, NY, 1998

