International Journal of Database Theory and Application
Vol.7, No.6 (2014), pp.13-20
http://dx.doi.org/10.14257/ijdta.2014.7.6.02

On Multi Query Optimization Algorithms Problem

Muhammad, L. J.[1], Yahaya Bala Zakariyau[2], Abdullahi Garba Ali[3] and Abba Garba

[1, 2]Dept. of Maths & Comp. Sceince, Federal University, Kashere
[3] Computer Science Department, Bayero University, Kano
Dept. of Comp. Science & Info Systems, Kampala International University

lawan.jibril@fukashere.edu.ng; baladada57@yahoo.com; jgewel@yahoo.com; abbaggumel@yahoo.com

Abstract

Without multi query optimization, Relational Database Management System for online and analytical decision support systems would have been inefficient and hence unpractical. It is an expensive process because it relies at a great extent on evaluating the different plans (access paths) and choosing an optimal one among them. In Multi Query Optimization, queries are executed in batches and there were many different algorithms acted in such way that, in case some queries have a common sub-expression such a sub- expression is executed once and the output shared.

We studied the basic multi query optimization algorithms including Basic Volcano, Volcano-SH and Volcano RU, identified their strengths and weaknesses and recommend strategies for developing new improved multi query optimization algorithm so as to reduce weaknesses and integrate strengths of the different basic multi query algorithms into one efficient algorithm.

Keywords*: Multi-Query, Optimization, Algorithm, inter-query shareability, Basic Volcano, Volcano-SH, Volcano RU, common sub-expressions*

1. Introduction

In multi-query optimization, a query is not optimized one by one; instead, the queries are optimized and hence executed in batches. [5, 13 and 8] carried out research on multi- query optimization using basically exhaustive algorithms. These algorithms traverse a good number of the different options, look for the minimum among the different query plans per query and then output the set of optimal plans. Though optimized together, plans of queries whether final or intermediate do not intervene or interfere in the generation of other plans. In fact, after individual query optimization, the queries compete for computer resources at execution [1] advocates for cooperation and do not entirely optimize the individual queries. They aim at getting the cheapest way of retrieving all the data so that all the query requests are serviced. However, many researches are still going on multi query optimization algorithms of how to improve them. Therefore, more better and efficient algorithm can be only developed if the weaknesses and strengths of the existing algorithms were identified.

2. Multi-Query Optimization (MQO)

In MQO, queries are executed in batches. Some of the MQO techniques act in such a way that in case some queries have a common sub-expression such a sub- expression is executed once and the output shared. In some cases, the sharing does not necessarily take place on

ISSN: 2005-4270 IJDTA
Copyright ⓒ 2014 SERSC

individual optimal plans. According [10] but instead sub-optimal plans are used, decision may as well have to be taken whether the common sub- expressions should be pipelined or materialized. S`ome multi-query optimization techniques basically aim at having parallel optimization of many queries [5]. These queries pass through the different optimization steps together and as an output, which is a set of optimal plans for each query generated [10] criticized this approach on a basis that further cooperation can be made between the queries that make up the batch. If a certain sub-expression is common, then the computer should execute it once and share out the results. This is a guiding principle to the Basic Volcano algorithm proposed by [4] and the Volcano-SH and Volcano RU optimizer algorithms proposed by [10] further put the sharing of the sub-expressions to a great importance that even if the sharing takes place on a non-optimal plan of the query, as long as the total resource requirement is optimal, it is acceptable.

A multi-query optimizer is responsible for recognizing the possibilities of shared computations and modifying the optimizer search strategy to explicitly account for shared computations so as to find a globally optimal plan [9]. This sharing however may not be necessarily optimal since:-

i. The cost of queries may be too high such that the sum of the independent optimal plans will be still the global optimal;

ii. The shared plan may have a lower resource requirement than the non shared ones but when the resources taken to achieve the plan take more resources than the trade off hence an efficient query in an inefficient system [2];

iii. The sharable components may be too few and the ratio of sharable to total components is too low. The search for sharable components may produce very little sharable components that the saving to be far less than the searching cost.

The inclusion of sub-optimal plans increases the sample space hence a more efficient search technique is required. Sharing in multi-query optimization highly hinges on the availability of the sharable sub-expressions [2]. The elimination of some plan types, like in the way it is done in IBM System R optimizer may not be useful since the eliminated types may have sharable components. The elimination may instead depend on the cost of the sub-expressions in the plan itself since excessively expensive expressions, even if shared, are very unlikely to create global optimality. Elimination of some expression is done so as to reduce the search sample space.

3. Multi Query Optimization Algorithms

All the multi query algorithms do not take into consideration possibilities of sub-expressions that may be common and save more in overall. These algorithms are Basic Volcano, Volcano SH, and Volcano RU algorithms. The algorithms use DAG to represent the search space. In some cases however, search space is represented as an AND-OR DAG.

 Copyright ⓒ 2014 SERSC

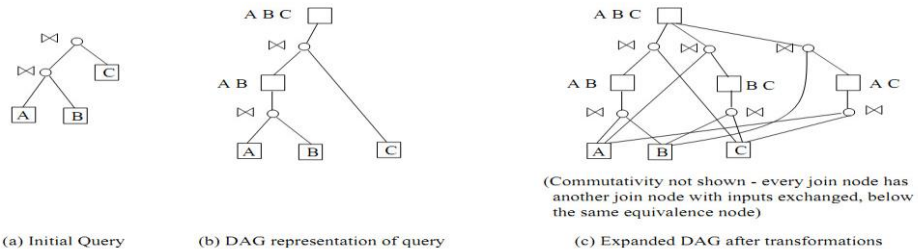

Figure 1. Query Representation: Tree, DAG and Extended DAG

An AND-OR DAG is a DAG whose nodes are divided into two: the AND nodes and the OR nodes. AND nodes have only OR nodes as their children and OR nodes have only AND nodes as their children. The AND node in an AND-OR DAG has algebraic operation like select (σ), project (π), etc. They are therefore referred to as operational nodes. The OR node of an AND-OR DAG represents a logical expressions that generates the same result set as when a child operational node is applied on its children/ child. OR nodes are referred to as equivalence nodes. The expanded DAG is used as a representation for modern optimizers because they are easily extensible.

3.1 Basic Volcano Algorithm

Basic Volcano Algorithm was proposed by [4] as a reaction to the previously proposed Exodus Optimizer. It uses DAG as a representation of the query plans. It has a problem of extensibility since AND-OR DAGs are easier to extend than the DAGs [11].

The Basic Volcano algorithm materializes all nodes that appear more than once. This brings in a problem that not all nodes that appear more than once cause savings when materialized. As observed in [9], for some nodes, it is cheaper to recompute than to materialize and reuse them. This is because materialization involves writing and reading to disk which is costly. This algorithm determines the cost of the nodes by using a depth first traversal of the DAG. The cost of operational and equivalence nodes are given by

$$\text{Cost}(o) = \text{cost of executing}(o) + \sum_{ei \in \text{children}(o)} \mathbf{cost}(e_i)$$

and the cost of an equivalence node is given by

$$\text{cost}(e) = \min(\text{cost}(oi) | o \in \text{children}(e)$$

If the equivalence node has no children, then cost (e) = 0. In case a certain node has to be materialized, then the equation for cost (o) is adjusted to incorporate materialization. For a materialized equivalence node, the minimum between the cost of reusing the node and the cost of recomputing the node is used. The equation therefore becomes

$$\text{Cost}(o) = \text{cost of executing}(o) + \sum_{ei \in \text{children}(o)} \mathbf{cost}(e_i)$$

where $C(ei) = cost(ei)$ if $ei! \in M$, and $= min(cost(ei), reusecost(ei))$ if $ei \in M$.

The Basic Volcano lays a foundation for cost- effective reuse but:-

i. It does not establish the cost effectiveness of the candidate node to materialize before choosing it for materialization [5].

Copyright ⓒ 2014 SERSC

ii. Its search is exhaustive therefore the optimizer incurs a high search cost which bring a negative impact on the overall cost effectiveness of the query processor [5].

iii. The Basic Volcano algorithm therefore incurs a lot of cost in searching for the sharable sub expressions which may render it inefficient especially for large (and therefore) complex queries [1].

3.2 Volcano SH Algorithm

The Volcano-SH is an extension of the Basic Volcano algorithm. It uses the Basic Volcano optimal plans as an input. The volcano SH computes the cost of each node and decides whether or not it is cost effective to materialize it. This is done by considering a scenario of materialization and reuse against re-computation. If for example we have an equivalence node e with the following characteristics:

Number of times it is to be used = numuses (e)

Cost of computing the node = cost (e)

Cost of materializing the node = matcost (e) and

Cost of reusing the node =reusecost (e).

A decision has to be made whether to materialize and reuse the node or to recompute the node whenever it is needed. If all the nodes are computed from the database, the cost incurred would be

$$\text{cost}(e) \times \text{numuses}(e)$$

and if the node was computed once, then materialized so that for subsequent times it is just reused, the cost incurred would be

$$\text{cost}(e) + \text{matcost}(e) + \text{reusecost}(e) \times (\text{numuses}(e) - 1)$$

Materialization is cost effective if

$$\text{cost}(e)+\text{matcost}(e)+\text{reusecost}(e)\times(\text{numuses}(e)-1) < \text{cost}(e)\times\text{numuses}(e)$$

or more simply

$$\text{reusecost}(e) + \frac{\text{matcost}(e)}{(\text{numuses}(e) - 1)} < \text{cost}(e)$$

Volcano SH algorithms selects whether or not to materialize depending on the cost effectiveness of the scheme. The volcano-SH traverses the DAG from the leaves towards the root. Since the cost of a node is computed from the children (leaves), the cost of a node can be accurately established as the algorithm traverses the DAG. The number of times a node is used however depends on the materialization status of the parents. Since a node is reached before the parents are reached, it cannot be easily established. [11] uses an under estimate numuses−(e) which was obtained by counting the number of parents of a node. The condition for materialization is therefore modified to
or more simply

 Copyright ⓒ 2014 SERSC

$$\text{reusecost}(e) + \frac{\text{matcost}(e)}{(\text{numuses}-(e) - 1)} < \text{cost}(e)$$

the advantage of Volcano SH is that, it eliminates blind materialization hence saving more resources [9]. It however has some shortcomings that need to be addressed are as follows:-

i. Its estimate of the frequency a node appears puts no consideration of other plans in the pseudo rooted DAG yet the inter-query sharing makes a lot of savings on resources [1].

ii. Its estimate is inaccurate. In fact in some cases the under estimate is higher than the actual value which may lead to wrong decisions [5].

iii. It does not attempt to exploit the inter-query extents of similarities. This makes it unable to decide on the optimal order in which the queries should be processed [6].

iv. It does not trim already catered for nodes hence the search works on a fixed sample space leading to non-worthwhile search efforts [6].

3.3 Volcano-RU Algorithm

The Volcano-RU exploits sharing well beyond the optimal plans of the individual queries. Though volcano SH algorithm considers sharing, it does it on only individually optimal plans therefore some sharable components which are in sub-optimal plans are left out. Including sub-optimal states however implies that the sample space of the nodes has to increase. The search algorithm must be able to put it into consideration so that the searching cost is still below the extra savings made. The volcano RU algorithm aims at reusing and sharing sub-expressions which are not necessarily in the individual query optimal plans [9].

Volcano-RU is sequential, considering possibilities of reusing expressions of previously optimized queries in subsequent queries. For a set of queries in the same pseudo root, after optimizing Q_i, the nodes in the plans of Q_i are identified. Since at that moment the algorithm has no idea of the structure of the subsequent queries, it checks whether it would be optimal if a certain node was materialized for reuse one extra time. While optimizing the next query, costs saving expressions are considered to be present. The Volcano- SH is then applied to further detect and exploit more sharing opportunities [9]. In such a case, a query is able to share sub-expressions within itself and materializable plans are all identified.

Volcano-RU depends on the order in which queries are optimized. It can be done in a certain sequence, then in reverse order and the cheaper alternative is chosen. Considering more orders have a probability of getting a cheaper order but it increases the optimization time.

Unlike the volcano-SH, the volcano-RU does not take in the Basic Volcano outputs and neither does it attempt to establish the number of times a node is used. It optimizes one query at ago and any node, (whether it is on the optimal plan or not) that would cause savings if reused once is chosen for materialization [6]. The subsequent queries are optimized putting into consideration the fact that some nodes are already materialized. Its strength lies in exploiting shareability beyond the Basic Volcano optimal plans. Given its approach, the order of optimization is of paramount importance since the node to be materialized depends on which query has been optimized so far [9]. It however has the following weaknesses:-

i. It does not go into details of establishing the exact optimal order of optimization. Roy *et al*. (2001) proposed that after optimizing in a specific order, we optimize in the reverse order

Copyright ⓒ 2014 SERSC

and the cheaper option is chosen. However, since the first order was arbitrary, a more optimal order is very likely to exist. Attempting to randomly choosing other orders while searching for the optimal order leads to time wasting [5].

ii. It also has a problem of excessive materialization since not all nodes that would cause saving if reused twice actually exist more than once. The excessive materialization leads to further costs incurred at materialization hence a more costly query processor [12].

iii. In a DAG, some nodes appear more than twice and cause savings yet they would not make the savings if they appeared twice. Such nodes are left out since the criteria only consider those which would cause savings when reused once. The materialization therefore may be insufficient [8].

4. Future Work

Multi-query optimization takes place on many complex queries with many relations, however comparing sub-expressions among those queries to find common sub-expressions exhaustively leads to too many comparisons hence high comparison cost and time. However, the future work should address the extent of sharing without necessarily traversing all the nodes. If the queries have no node in common, then there is no need to exploit the similarity because it is not there. Therefore, the new multi query optimization algorithm should be developed with these functions.

5. Conclusion

In this work the existing basic multi query optimization algorithms were studied, how they order queries for optimization, how they optimize and how they exploit the geometry of query plans representation (Trees, DAGs and AND-OR DAGs) to make the scheme more cost effective. The strengths and weaknesses of these algorithms were identified. However, new algorithm that will provide an alternative approach to these existing algorithms need to be developed, implemented and encoded into the existing optimizer. The algorithm should address the identified weaknesses of the existing algorithm and integrate strengths of those algorithms into more efficient algorithm.

References

[1] J.R. Alsabbagh and V.V. Raghavan, "Analysis of Common Subexpression Exploitation Models in Multiple Query Processing", Proc. 10th Int. Conf. on Data Engineering, IEEE Press, **(1994)**, pp. 488-497.
[2] R. Choenni and A. Siebes, "Query Optimization to Support Data Mining", Proc. DEXA '97 8th Int. Workshop on Database and Expert Systems Applications, IEEE Press, **(1997)**, pp. 658-663.
[3] A. Cosar, E. Lim, and S. Jaideep, "Multiple Queries Optimization with depth-first branch and bond and dynamic query ordering", International Conference on Information and Knowledge Management, **(2001)**.
[4] G. Graefe and W. J. McKenna, "Extensibility and Search efficiency in the Volcano Optimiser generator", Technical report CU-CS-91-563. University of Colorado., **(1991)**.
[5] N. John, "Query Optimization in Relational Databases", Msc Dissertation., Makerere University, Uganda, **(2004)**.
[6] S. Kyuseok, T. K Sellis and D. Nau, "Improvements on a Heuristic algorithm for Multiple-query Optimization", Technical report, University of Maryland, Department of Computer Science, **(1994)**.
[7] N. V. Nilesh, K.S. Sumit, P. Roy and S. Sudarshan, "Pipelining in multi-query optimization", Research paper, Indian Institute of Technology, **(2001)**.
[8] P. Park and A. Segar, "Using common sub-expressions to optimise multiple queries", Proceedings of the IEEE International Conference on Data Engineering, **(1988).**
[9] S. D. Pandao and A. D. Isalkar, "Multi Query Optimization Using Heuristic Approach", International Journal of Computer Science and Network (IJCSN), vol. 1, no. 4, **(2012)**.

 Copyright ⓒ 2014 SERSC

[10] P Roy, S. Seshadri, Sudarshan, S. and S. Bhobe, "Efficient and Extensible algorithms for Multi query optimization", Research Paper, SIGMOD International Conference on management of data, **(2001)**.
[11] P. Roy, S. Seshadri, S. Sudarshan and S. Bhobe, "Practical Algolithms for multi query Optimisation", Technical report, Indian institute of Tech- nology, Bombay, **(1998)**.
[12] L. Sallis, "A new Heuristic for optimizing large queries", Department of Computer Science and Engineering, University of Texas at Arlington, **(1998)**.
[13] T. K. Sellis and S. Gosh, "On Multi-query Optimization Problem", IEEE Transactions on Knowledge and Data Engineering, **(1990)**, pp. 262-266.
[14] A. Swami, "Optimization of Large Join Queries: Combining Heuristics and Combinatorial Approach", Proc. of the 1989 ACM Int. Conf. on Management of Data, ACM Press, **(1989)**, pp. 367-376.
[15] T. Urhan and M. J. Franklin, "Dynamic Pipelining Scheduling for improving interactive query performance", proceedings of the 27th Very Large Databases Conferencem, **(2001)**; Roma - Italy.
[16] J. L. Whitten and L. D. Bentley and K. C. Dittman, "Database System", (5^{th} ed.) (international edition), McGraw Higher Education (A Division of the McGraw-Hill Companies), **(2001)**.
[17] J.D. Ullman, "Principles of Database and Knowledge-Base Systems", The New Technologies, Computer Science Press, vol.2, **(1989)**.
[18] S.B. Yao, "Approximating Block Accesses in Database Organizations, in Comm", ACM, vol. 32, no. 5, Press, **(1977)**, pp. 260-261.

Copyright ⓒ 2014 SERSC

 Copyright ⓒ 2014 SERSC