International Journal of Database Theory and Application
Vol.9, No.6 (2016), pp.219-226
http://dx.doi.org/10.14257/ijdta.2016.9.6.22

Multi Query Optimization Algorithm Using Semantic and Heuristic Approaches

L. J. Muhammad[1], Yahaya Bala Zakariyau[2], Abdullahi Garba Ali[3] Ibrahim A. Mohammed[3]

[1,2]Mathematics & Computer Science Department, Federal University, Kashere Computer Science Department, Bayero University, Kano[3] Computer Science Department, Yobe State University[4]
lawan.jibril@fukashere.edu.ng[1], baladada57@yahoo.com[2] jgewel@yahoo.com[3] ibrahimsallau@gmail.com[4]

Abstract

Multi Query Optimization is one of the most important tasks in Relational Database Management System (RBMS) and it becomes common due to high usage of online decision support management systems in every industry nowadays. In multi query optimization, queries are optimized and executed in batches. However, there are many algorithms use to detect and unified common sub-expressions among multiple queries and unified them so that the more encompassing sub- expression is executed and the other sub-expressions are derived from. In this work, multi-query optimization algorithm using heuristics and semantic approaches was proposed and encoded on SQL Server version 10.0.1600 and three queries were used for the experiment between the proposed algorithm and most recent basic Multi Query Optimization Algorithm (Volcano RU). The result of experiment showed that, Proposed Algorithm gave the best plans compared Volcano RU Algorithm, across all three queries and was best for all queries in terms of execution time and CPU time.

Keywords*: Multi Query Optimization, Semantic, Heuristic, Systematic, inter-query shareability, common sub-expressions*

1. Introduction

In multi-query optimization, queries are not optimizing one by one but rather optimizing and executing in batches. Complex queries are becoming common, with the growing use of decision making support systems and other analytical support systems. [3] The query optimizer therefore is responsible for finding the best execution strategy so that fewer resources are used to retrieve data [4]. However, there are three main approaches to query optimization which include the following

1.1 Systematic Query Optimization

In systematic query optimization, the system estimates the cost of every plan and then chooses the best one [1]. The best cost plan is not always universal since it depends on the constraints put on data. For example, joining on a primary key may be done more easily than joining on a foreign key since primary keys are always unique and therefore after getting a joining partner, there is no other key expected. The system therefore breaks out of the loop and hence does not scan the whole table. The costs considered in systematic query optimization include access cost to secondary storage, storage cost, computation cost for intermediate relations and communication costs [5]. The importance put on these costs depend on the type of database.

ISSN: 2005-4270 IJDTA
Copyright ⓒ 2016 SERSC

1.2 Heuristic Query Optimization

In the heuristic approach, the operator ordering is used in a manner that economizes the resource usage but conserving the form and content of the query output [2]. The principle aim is to:

(i) Set the size of the intermediate relations to the minimum and increase the rate at which the intermediate relation size tend towards the final relation so as to optimize memory [2].

(ii) Minimize on the amount of processing that has to be done on the data without affecting the output [3].

1.3 Semantic Query Optimization

This is a combination of Heuristic and Systematic optimization. The constraints specified in the database schema can be used to modify the procedures of the heuristic rules making the optimal plan selection highly creative [7]. This leads to heuristic rules that are locally valid though cannot be taken as rules of the thumb.

2. Related Works

Most of the multi query optimization algorithms do not take into consideration possibilities of sub-expressions that may be common and save more in overall costs except in reuse based optimization sharing possibilities are explored [6]. These algorithms are Basic Volcano, Volcano SH , and Volcano RU use the same principal though with different philosophies. These algorithms use DAG to represent the search space. In some cases however, search space is represented as an AND-OR DAG. [5]. 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 [3]. 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. These algorithms include the following:-

2.1 Basic Volcano Algorithm

The Basic Volcano Algorithm 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. 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 [2], 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. Below is the pseudocode of the algorithm.

```
PROCEDURE: Volcano(eq)
Input: Root node of Expanded DAG
Output: Optimized plan
Read (op ∈ child (eq))
Read (InpEq ∈ child (op))
For (every non-calculated op ∈ child (eq)
        For (every inpEq ∈ child (op))
                Volcano(inpEq)
                  If inpEq∈ leaf node
```

 Copyright ⓒ 2016 SERSC

```
                    cost(inpEq)= 0
                        If  (eq ∉ M )
                    cost(op) = cost of executing (op) +ΣCost(inpEq)
                    cost(eq) = min{cost(op)|op ∈ children(eq)}
                else
                 If (eq ∈ M )
                        cost(eq) = min{cost(eq),reusecost(eq)}
                mark op as calculated
endfor
        endfor
                endif
                endif
                endif
```

2.2 Volcano SH Algorithm

The Volcano-SH is an extension of the Basic Volcano algorithm in that 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 [2]. This is done by considering a scenario of materialization and reuse against re computation. . Below is the pseudocode of the algorithm.

Procedure VOLCANO-SH
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
Consolidated Volcano best plan P for virtual root of DAG
M = { }
Perform prepass on P to introduce subsumption derivations
Let C_{root} = COMPUTEMATSET(root)
Set $C_{root} = C_{root} \sum +_{d \in M}(cost(d) + matcost(d))$
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 o_e be the child of e in P
For each input equivalence node e_i of o_e
Let C_i= COMPUTEMATSET(e_i) // returns computation cost of e_i
If e_i is materialized, let C_i = reusecost(e_i)
Compute cost(e) = cost of operation $o_e + \sum_i C_i$
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)

Copyright ⓒ 2016 SERSC

2.3 Volcano-RU Algorithm

The Volcano-RU exploits sharing well beyond the optimal plans of the individual queries but it aims at reusing and sharing sub-expressions which are not necessarily in the individual query optimal plans [5]. 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 [3]. The Volcano- SH is then applied to further detect and exploit more sharing opportunities. In such a case, a query is able to share sub-expressions within itself and materializable plans are all identified [6]. Below is the pseudocode of the algorithm.

```
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 = Φ // 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
```

3. Result and Discussion

3.1 Multi Query Optimization Algorithms Using Heuristic Approach

In this proposed improved algorithm, two query plans are compared for each pair of node so to establish whether or not they are sharable. If they are sharable, the appropriate entry in the multi query sharing matrix will incremented. For example, for a batch of n queries Q_1, Q_2,…… Q_n. For any query Q_x, a set of nodes in the Volcano optimal plan as a set of volcano best plans and Let S_x be the set of equivalence nodes in the plan of Q_x. Therefore, the algorithm checks nodes pairwise and establishes whether sharing can be possible. If it is possible for any queries Q_x and Q_y , the query popularities and M[x,y] are incremented and it continues until the all they are exhausted. Considering a general query sharing matrix M below:

$$\begin{Bmatrix} M1_1 & M1_2 \ldots\ldots\ldots & M1_n \\ M2_1 & M2_2 \ldots\ldots\ldots & M2_n \\ Mn_n & Mn_n \ldots\ldots\ldots & Mn_n \end{Bmatrix}$$

The entry M [x, y]= mxy is an integer that shows how many sub-expressions (equivalence nodes in AND-OR DAGs) that are sharable between queries Q_x and Q_y . This is the same as the value of M [y, x]. The sum up of the entries in a column (or a row), will result the total number of instances in which nodes in the plan have sharable

 Copyright ⓒ 2016 SERSC

partners. This is called query popularity. All nodes that have partners are marked so that the optimizer can identify which nodes necessitate checking other plans while searching for common sub-expressions and which nodes do not. This helps in eliminating null searches hence a more efficient strategy. After identifying the extent of sharing among the queries, we can be able to tell which pair or pairs of queries have nothing in common.

When start searching for common sub-expressions, it starts with one query and search for the sharable sub-expressions with other queries in the DAG. The Query plan, which is at the center of the searching process, is called the focal node. Since the plans are already traversed at the searching stage, the nodes will be marked (those with sharable sub-expressions elsewhere) and those node that will not be marked. It is only on marked nodes that the search for other plans for sharable nodes will be done. It should be noted that a node being marked does not mean that its partners will be searched for in all plans. Since the query sharing matrix has the summary of what query shares with what, only those with non-zero entries in the sharing matrix are searched for a column/ row representing the extent of sharing between the focal plan and such a plan are checked. The algorithm therefore searches only in optimistic cases. Below is the pseudocode of the algorithm.

```
Procedure Heuristics (shareability Search Algorithm)
Input: Expanded DAG on queries Q1, . . .
Output: Set of plans in decreasing order of their popularity
for x =1; x<=n; i++
  for y =1; y<=n; j++
    Order of Each Query = 0
      repeat
           NextOrder = 0;
           Sx = Set of nodes in Query x
           k = First node in SX
           Sy = Set of nodes in Query y
           z = First node in Sy
           while(Sx still has nodes)
             while(Sy still has nodes)
                    if(k = = z)
                    nextOrder = 1
                       if(ORDER = 0)
                          break out of the two while loops
                       else
                             increment M[x,y] and M[y,x] and mark z and k
                    endif
                  endif
                endwhile
              endwhile
        ORDER = next(ORDER)
      until(nextOrder= 0 or orders are exhausted)
  endfor
endfor
```

3.2 Multi Query Optimization Algorithm Using Semantic Approach

This algorithm identifies the plan with the highest popularity and establishing its details (node costs and frequency). The common sub-expressions are searched for in the rest of the queries following the stable marriage preference list of the focal plan. If according to

Copyright ⓒ 2016 SERSC

the sharing matrix M, a certain plan is not having any node to share with the focal plan, it does not search it. When a focal plan is identified, the costs and number of times a node is used in the plan, count are established. If a node is marked, then it means that it has sharable partners in other plans. The sharable nodes are searched for in the plans that share at least a node with the focal plan. Each time a sharable node is found, the count is incremented by one and it's assured to be of the same cost as the node in the focal plans. The test for materialization used in the Volcano SH is then applied using the new estimated cost and the node count in the pseudo rooted plan. In this algorithm, nodes were chosen to materialize from the high order nodes downwards. This is used in such a way that if say two nodes are sharable and two have to be removed, this reduces the sample space of subsequent searches and saves the cost of computing already catered for nodes.

```
Procedure Proposed Multi Query Optimization Algorithm Using Semantic Approach
Input: Set of plans in decreasing order of their popularity (Proposed Multi Query
Optimization Algorithm Using Heuristic Approach)
Output: Set of best plans
S = Set of plans in decreasing order of their popularity
  X = first plan in S
  repeat
     Calculate the cost of each node
     Calculate the numuses of each node
     S* = subset of S which share atleast a node with x
     XOrder = highest order of x
          repeat
             n = node in XOrder
             e = equivalent node
             if (e is marked)
                  repeat
                     traverse S* for e of the same order with and sharable with e
                     increment numuses(e) whenever a sharable node is met
                  until(S* is exhausted)
                  if(sharable condition(e))
                       select the node to be materialized and add it to the
                       materializable node
                       remove the rest of the node
                       update the DAG for the materialized node to cater for the removed
                       node's parents
                       unmark the chosen node
                 endif
            endif
        until(nodes in XOrder are exhausted)
        X = next(X)
  until(plans on non-zero popularity are exhausted)
```

4. Performance Study

To study the benefits of multi-query optimization algorithm on a real database, both the proposed improved and Volcano RU algorithms were encoded on Microsoft SQL Server version 10.0.1600.22 running on Windows 7 64 bits operating system, with AMD A4 3320M APU With Radeon ™ HD Graphic Processor, 2.00 GHz and 4GB of RAM and three queries were executed to test their effects. Thus, we measured the total elapsed time for executing all these three queries and the results are shown in Figure 1.1, Figure 1.2

 Copyright ⓒ 2016 SERSC

and Figure 1.3. The results indicate that the proposed multi-query optimization gives significant time improvements on a real system compared to Volcano RU Algorithm.

Query One: The time RU Volcano Algorithm takes before to execute query one on database is 4810ms while the proposed algorithm takes 141ms. On other hand, the CPU time for RU Volcano Algorithm was 31ms while for the proposed algorithm was 21ms. Therefore, the proposed Algorithm is more efficient and effective than Volcano RU Algorithm in terms of execution and CPU time with respect to query one. The results are shown in Figure 1.1

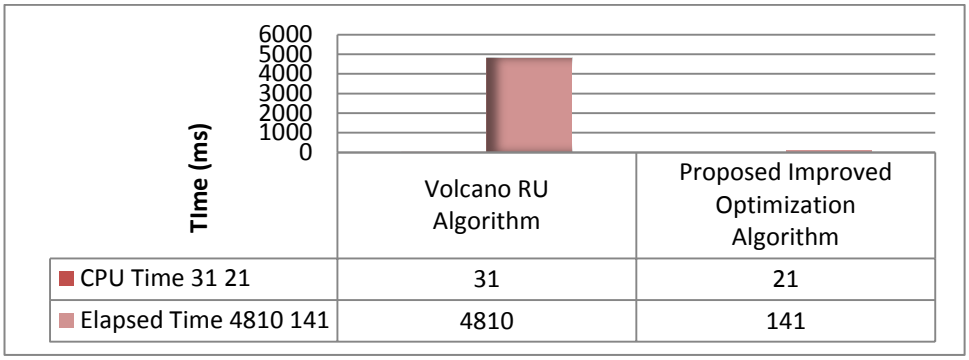

Figure 1.1 Performance Study Result of Query One

Query Two: The time RU Volcano Algorithm takes before to execute query two on database is 3790ms while the proposed algorithm takes 117ms. On other hand, the CPU time for both RU Volcano Algorithm and the proposed algorithm was 0ms respectively. Therefore the Proposed Optimization Algorithm is more efficient and effective than Volcano RU Algorithm in terms of execution time only. The results are shown in Figure 1.2

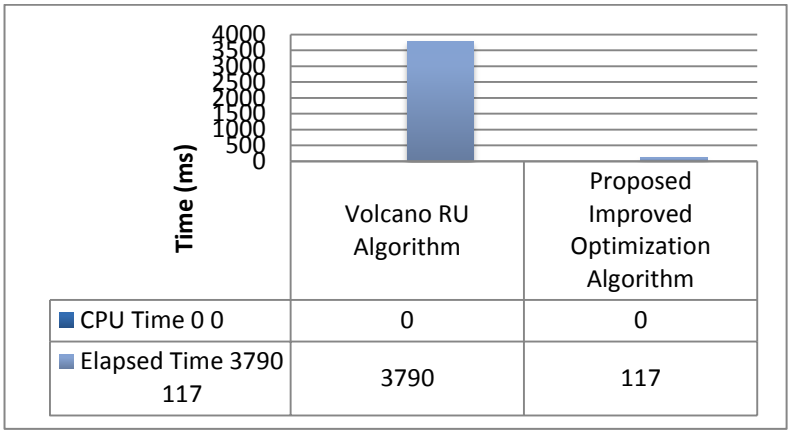

Figure 1.2 Performance Study Result of Query Two

Query Three: The time RU Volcano Algorithm takes before to execute query three on database is 14ms while the proposed algorithm takes 3ms. On other hand, the CPU time for both RU Volcano and Proposed Improved Optimization Algorithms was 0 ms. Therefore, the Proposed Algorithm is more efficient and effective than Volcano RU Algorithm in terms of execution time only with respect to the query. The results are shown in Figure 1.3

Copyright ⓒ 2016 SERSC

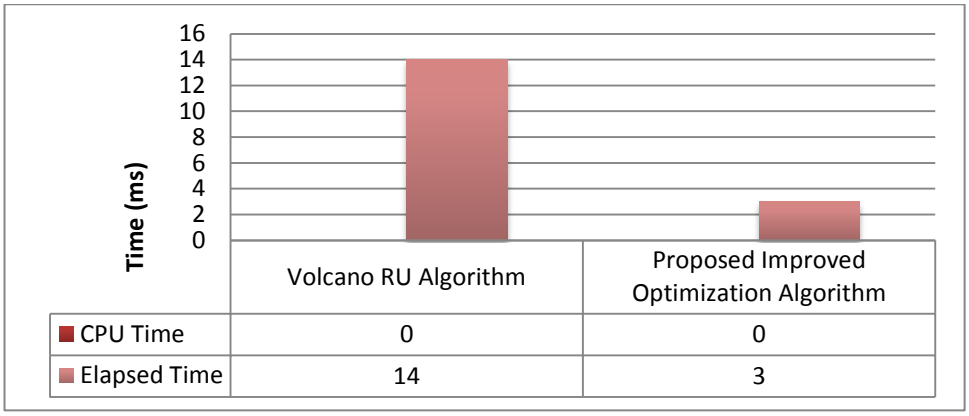

Chart 4.4 Performance Study Result of Query Three

5. Conclusion and Future Study

In this work multi query optimization algorithm using heuristic and semantic was proposed and its benefits were demonstrated on a real database system. The performance of the algorithm, using queries based on the TPC-D benchmark, demonstrates that multi-query optimization is practical and gives significant benefits at a reasonable cost. The algorithm uniformly gave the best plans, across all the queries and Volcano-RU, which is cheaper, may be appropriate for inexpensive queries.

However, for future study, the proposed algorithm can be further improved by incorporating Cache. Thus In a traditional database engine, every query is processed independently but in decision support applications, queries often overlap in the data that they access and in the manner in which they utilize the data, i.e., there are common expressions between queries. A natural way to improve performance is to allocate a limited-size area on the disk to be used as a cache for results computed by previous queries. The contents of the cache may be utilized to speed up the execution of subsequent queries. Therefore, there is need to incorporate cache to store final and/or intermediate results of queries in a cache.

References

[1] Dalvi, J. R., Raghavan, V. V., Brian B. (2001). Analysis of Common Subexpression Exploitation Models in Multiple Query Processing, in Proc. 10th Int. Conf. on Data Engineering, IEEE Press, pp. 82-97

[2] Graefe, G. & McKenna, W. J. (1991). Extensibility and Search efficiency in the Volcano Optimiser generator. Technical report CU-CS-91-563. University of Colorado.,

[3] Muhammad, L. J., Yahaya Bala Zakariyau and Abdullahi Garba Ali. (2015). Efficient Multi-Query Optimization Algorithm. International Journal of Database Theory and Application Vol.8, No.6, pp.133-138. Science & Engineering Research ISSN: 2005-4270

[4] Kyuseok, S., Sellis, T. K and Nau, D. (1994). Improvements on a Heuristic algorithm for Multiple-query Optimization. Technical report, University of Maryland, Department of Computer Science.

[5] Muhammad, L. J., Yahaya Bala Zakariyau, Abdullahi Garba Ali and Abba Garba. (2014). On Multi-Query Optimization Algorithms Problem. International Journal of Database Theory and Application Vol.7, No.6), pp.13-20. Science & Engineering Research ISSN: 2005-4270

[6] Park, P. and Segar, A. (1988). Using common sub-expressions to optimise multiple queries. Proceedings of the IEEE International Conference on Data Engineering.

[7] Roy, P. Seshadri,S. Sudarshan,S. and Bhobe, S (1998). Practical Algolithms for multi query Optimisation. Technical report, Indian institute of Technology, Bombay.

[8] Pandao, S. D. and Isalkar, A. D. (2012). Multi Query Optimization Using Heuristic Approach, International Journal of Computer Science and Network (IJCSN) Volume 1, Issue 4

 Copyright ⓒ 2016 SERSC