COMMON SUB-EXPRESSION IDENTIFICATION STRATEGY FOR MQO

Nor Hawaniah Zakaria, AP Dr Shamsul Sahibuddin, AP Dr Harihodin Selamat
Fakulti Sains Komputer & Sistem Maklumat, UTM
hawaniah, shamsul@fsksm.utm.my,
07-5532410

Abstract

In query optimization, a query can be executed with different strategies, known as execution plan. The query optimizer will determine the best execution plan for a single query. However, when there are more than one query to be executed together, the locally optimal strategies for single queries may not be the best choice for obtaining a globally optimal execution cost. This will require for a multiple query optimizer that is able to select an alternative plan for each query in order to obtain an optimal global execution plan for the multiple queries. An optimal global plan can be obtained in two ways, by using an admissible heuristic with the search algorithms or by decreasing the number of search space through reducing the number of alternative plans generated. In generating alternative plans, the number and quality of the alternative plans produced are the major factors that will determine the performance of multiple-query optimization. In this paper, we propose the sharing opportunities for identifying the common sub-expressions.

Keywords

Query optimization, multiple query optimization, alternative plan, common tasks, global optimal plan

Introduction

One of the fundamental and important problems in the database field is query optimization. In database applications, a set of queries may be submitted together to the system for evaluation. This set of queries is most likely to have some tasks that are common to every query in the set. These queries can be processed one at a time, however it is inefficient. To process these queries efficiently, the optimizer should try to take benefit of the common tasks existence. Multiple-query optimization is beneficial especially when there are many common tasks between the queries. We can reduce the cost of executing these common tasks by executing these tasks only once, avoiding redundant disk accesses, and thus a considerable decrease in the execution time of the queries. This thus becomes the goal of multiple-query optimization, i.e. to reduce the execution time of multiple queries with common tasks.

The multiple-query optimization (MQO) problem has been studied in the database literature since 1980s. In general, the MQO problem can be divided in two phases [1]. The first phase is the process of constructing the search space for the optimizer, which will produce a set of alternative plans for each query in the query set, while the second phase is the process of selecting the most optimal plan from the alternative plans of each query in the query set. In the first phase, there are two main processes involved. The first process is to identify common tasks among for the queries in the query set. Next, a set of alternative plans for each query in the query set is generated. These plans will become the basis of the second phase. The result of the second phase is a single global execution plan for each query in the queries set. Figure 1 reflects the phases involved in MQO problem.

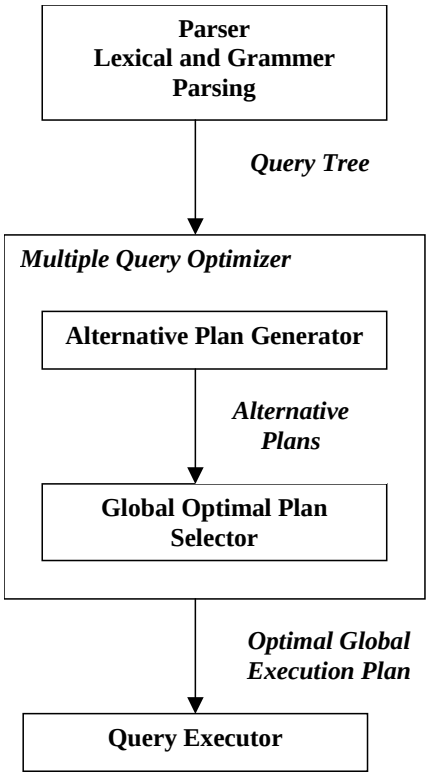

Figure 1. Phases in MQO

This paper is outlining our approach in solving the multiple-query optimization problem, and our focus is at the first phase, the alternative plan generator. In multiple-query optimization, the alternative plans generated will be the content of the search space of the multiple-query optimizer. The number of alternative plans generated will have an effect to the performance of the optimizer. *In this paper we will describe our approach in identifying the common tasks of the alternative plans (the notions of sharing.)*

Research Background

In query optimization, a query can be executed with different strategies, known as execution plan. The query optimizer will determine the best execution plan for a single query. However, when there are more than one query to be executed together, the locally optimal strategies for single queries may not be the best choice for obtaining a globally optimal execution cost. This will require for a multiple query optimizer that is able to select an alternative plan for each query in order to obtain an optimal global execution plan for the multiple queries.

In determining the best optimal solution for multiple-query optimization, several approaches have been proposed in determining the best optimal solution (execution plan) for multiple queries. One approach, as described by Finkelstein [2], is based on the idea of building the multiple query optimizer on top of the current single query optimizers. In this approach, a single query optimizer generates one optimal plan for each query. A plan merger, another component in the system, will next examine all the plans and merges them to generate a global execution plan. This global plan is derived from the shared temporary results of the common parts of the queries. This approach is however may not guarantee the optimal global cost because it may miss some other plans (which are not necessary optimal for each query) that contain more common tasks with other queries.

Another approach in determining the optimal solution tries to solve the problem mentioned above. The approach [1] generates several alternative plans for each query instead of generating only one optimal plan. It detects and uses all common tasks found within the queries.

An optimal global plan can be obtained in two ways, by using an admissible heuristic with the search algorithms or by decreasing the number of search space through reducing the number of alternative plans generated. To generate alternative plans, the relational algebra transformation operations are used, which modify the execution plan for a query without affecting the final output of the plan. A large query may produce many alternative plans. In order to reduce the size of the search space for the alternative plans, queries must be analyzed carefully while generating the alternative plans through maximizing the common tasks of queries. Cosar [3] developed two alternative plan generator methods by considering two and three queries at a time. The first method is through pair-wising the initial plans of the queries (two queries at a time) with common tasks, thus producing new plans for the queries. The second method is by matching each query plan with every other query plan and thus obtaining new plans and the process continues until no more plans can be generated for the queries. Both of

the proposed methods whilst managed to generate alternative plans they also created a large search space, which will increase the cost of MQO.

Alternative Plan Generator

The above Figure 1 depicts the main phases of MQO problem. As mentioned above, the alternative plans generated will be the input to the global optimal plan selector. In generating alternative plans, the number and quality of the alternative plans produced are the major factors that will determine the performance of multiple query optimization [4]. In order for the plans to be generated efficiently, it involves with two major issues, first is the identification of common sub-expressions that will contribute to the query rewriting of the original query and second, the number of alternative plans produced, which will contribute to the size of search space for the *global optimal plan selector*. For the first issue, which is the subject of discussion of this paper, we are proposing our strategy to identify the common sub-expression. The next section outlines our proposed sharing opportunities for consideration. Since the number and the quality of alternative plans generated have a substantial effect on the cost of the MQO, it is important to be able to estimate the cost of the sharing among the queries. This is the next step in our approach. Among factors that can be considered in estimating the sharing cost are the common relations, joins and conditions [4]. Figure 2 reflects our approach in generating the alternative plans (i.e. the alternative plans generator phase).

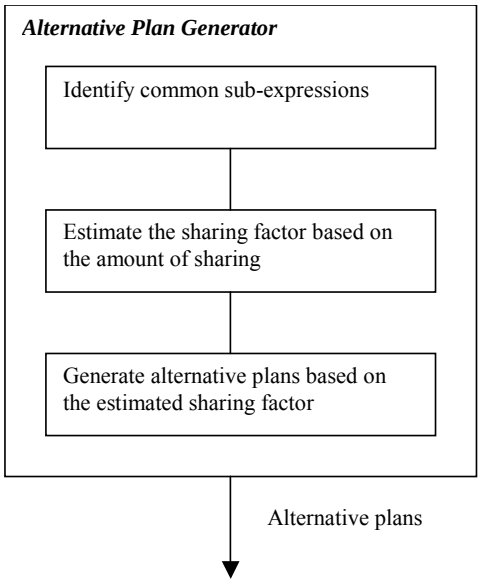

Figure 2. Approach in Alternative Plan Generator

Identifying Common Sub-expressions Strategy

Identifying common sub-expressions (CSE) among queries is important in multiple-query optimization problem. In this paper, we introduce our strategy in identifying the common operators and our focus is currently on the most important and complex operator, the two-way join operator.

Consider two queries, Q1 and Q2. Assume there are two two-way joins, *twj1* and *twj2*, where *twj1* is a join of relations R1 and S1, while *twj2* is a join of relations R2 and S2, both joins are over predicate p1 and p2, respectively.

$$twj1 : R1 \bowtie_{p1} S1$$

$$twj2 : R2 \bowtie_{p2} S2$$

Sharing opportunities considered are:

a) R1 = R2, S1 = S2

For equivalent condition: p1 is identical to p2 (P1 ≡ P2), then *twj1* is

the CSE. In rewriting the query, *twj2* will be deleted, and *twj1* is shared by Q1 and Q2.

For implication condition: p2 implies p1 (P2 → P1), (an example : p1 is r.a >= s.a, p2 is r.a = s.a) then *twj1* is the CSE. A *Select* operator should be added on *twj1*'s result in order to get *twj2*'s result.

For intersection condition: P1 ∧ P2, a join operator with predicate p1 v p2 is the common operator. In rewriting, two *Select* operator is added on top to further restrict the CSE's result.

b) R1 = R2, S1 ⊂ S2 (i.e. S1 = σ_t (S2) or S1 = σ_{t1} (S1) ∧ S2 = σ_{t2} (S) ∧ t1 → t2)

For equivalent condition: P1 ≡ P2, *twj1* is the CSE. In rewriting Q2, predicate t is moved up.

For implication condition: P2 →P1, similar to equivalent condition above.

For intersection condition: P1 ∧ P2, the CSE and rewriting is shown in the Figure 3.

c) S1 = S2, R1 ⊂ R2

The sharing condition is analogous to (b) above.

d) R1 = R2, S1 ∩ S2 ≠ ϕ (i.e. S1 = σ_{t1} (S), S2 = σ_{t2} (S), t1 ∧ t2 ≠ false)

For equivalent condition, P1 ≡ P2, the original two joins will be combined and then divided into 3 joins. One of the join is the CSE, see figure 4.

Similar goes to implication and intersection condition.

The above sharing strategies are applied to the next described algorithm in searching for the CSE and rewriting the queries trees. (To simplify explanation, we are considering only for two queries)

Algorithm: Searching for CSE and Query Rewriting

Input: Operator trees for each query.
Output: Operator forest after rewriting

1. Traverse the operator trees in post-order (post-order traversal).
2. Compare between operator nodes of every operator trees.
2.1 Find the same type nodes.
2.2 Get the predicate over the operator nodes (i.e. p1 and p2)
2.3 If p1 and p2 empty,
2.3.1 Get the input relations for the operator nodes.
2.3.2 If the two operator nodes have common relations, use strategies described above to rewrite the query trees.
2.4 If p1 and p2 is not empty, traverse p1 in inorder way.
2.4.1 For each node n in p1
2.4.1.1 Traverse p2 in inorder way. Get node x in p2.
2.4.1.2 If node x and node n access different relations or different attributes, then backtrack p2 to grandfather node of x.
2.4.1.3 Else compare the father nodes and the right-brother nodes of node x and node n. Rewrite the query trees using the strategies described above.
2.4.1.4 Continue until traversing is over.
2.5 Continue loop 2 until post-order traversal for q1 and q2 is completed.

The efficiency (theoretic) of the CSE search for the above algorithm can be calculated based on the number of comparisons between nodes when traversing the operator trees (step 2), the number of comparisons between the predicates (step 2.4.1-2.4.1.3) and the *average association degree* (AAD) between queries[5]. The AAD is given by

$$AAD = \frac{\sum_{i=1}^{k} i * m_i}{r * m}$$

where r is number of tables, m is the number of queries, m_i is the number of i-table queries

The number of comparison between nodes (CBN) is

$$CBN = n * (n * \sum_{i=1}^{m-1} (m - i))$$

where n is the average number of nodes in a tree, m is the number of queries.

The number of comparison between predicates (CBP) for m queries is

$$CBP = \sum_{i=1}^{m} (i - 1) * p * 3 * AAD$$

where p is the average number of predicate for each query.

Therefore, the total number of comparison (TNC) is the sum of comparison between nodes and comparison between predicates.

$$TNC = CBN + CBP$$

$$= n * (n * \sum_{i=1}^{m-1} (m - i)) + \sum_{i=1}^{m} (i - 1) * p * 3 * AAD$$

$$= \tfrac{1}{2} m (3*p*AAD + n^2)$$

Conclusion

In this paper, we proposed our approach in generating alternative plans. We have discussed briefly on the approach, which is shown in Figure 2. Included in our discussion is our strategy in identifying the common sub-expressions by producing the sharing opportunities, the algorithm to search for common sub-expressions and the theoretical analysis for the efficiency of the search algorithm. Our future work in the research is to experiment this algorithm with the sharing opportunities.

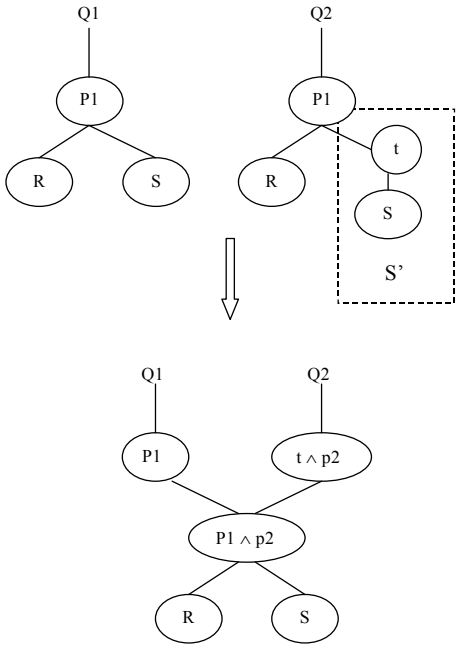

Figure 3. Sharing Opportunity Case (b)

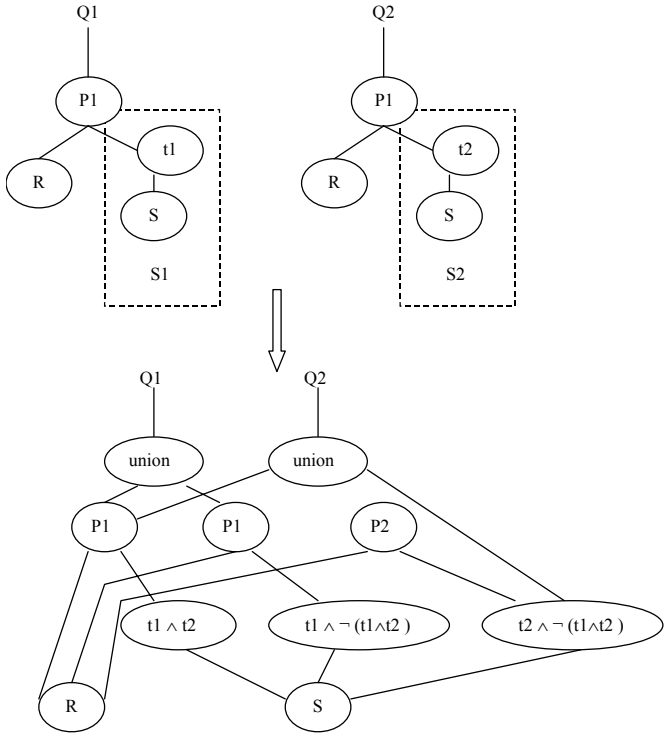

Figure 4. Sharing Opportunity Case (d)

Bibliography:

1. Sellis, T.K., *Multiple-Query Optimization.* ACM Transaction on Database Systems, 1988. **13**(1): p. 23-52.

2. Finkelstein, S. *Common Subexpression Analysis in Database Applications*. in *ACM-SIGMOD International Conference on the Management of Data*. 1982. Orlando, Florida: ACM.

3. Cosar, A., *Multiple Query Optimization, Phd Thesis*, in *Department of Computer Science, University of Minnesota*. 1996: Minneapolis.

4. Polat, F., A. Cosar, and R. Alhajj, *Semantic Information-based Alternative Plan Generation For Multiple Query Optimization.* Information Sciences, 2001. **137**(1-4): p. 103-133.

5. Chen, H., S. Zhou, and S. Wang. *Multiple Query Optimization in PBASE/3*. in *Fourth International Conference on High-Performance Computing in Asia-Pasific Region*. 2000: IEEE.

6. Hong, C., Z. Sheng, W. Shan. *Multiple Query Optimization in PBASE/3. 4th International Conference on High-Performance Computing in Asia-Pasific Region.* 2000. Volume 2.

7. Alsabbagh, J.R., V.V. Raghavan. *A Framework for Multiple-Query Optimization. International Workshop on Research Issues in Data Engineering – Transaction and Query Processing*. 1992. p.157-162.

8. Alsabbahgh, J. R., V.V. Raghavan. *Analysis of Common Sub-expressions Explotation Models in Multiple-Query Processing. 10th International Conference on Data Engineering.* 1994. p. 488-497.

9. Alsabbahgh, J. R., V.V. Raghavan . *A Model for Multiple-Query Processing based Upon Strong Factoring. International Conference on Information Technology: Coding and Computing.* 2004.