Research Paper Engineering

An Evolutionary Approach to Materialized View in Data Warehousing

* Sanket S. Patel ** Mr. Deepak Dembla

* Research student, Arya Institute of Engineering & Technology, Jaipur

** Associate Professor, Arya Institute of Engineering & Technology, Jaipur

ABSTRACT

A data warehouse (DW) contains multiple views accussed by queries. One of the most important decisions in designing a DW is selecting views to materialize for the purpose of efficiently supporting decision making. The search space for possible materialalized views is exponentially large. Therefore heuristics have been used to search for a near optimal solution. In this paper, we explore the use of an evolutionary algorithm for materialized view selection based on multiple global processing plans for queries. We apply a hybrid evolutionary algorithm to solve three related problems. The first is to optimize queries. The second is to choose the best global processing plan from multiple global processing plans. The third is to select materialized views from a given global processing plan. Our experiment shows that the hybrid evolutionary algorithm de- livers better performance than either the evolutionary algorithm or heuristics used alone in terms of the minimal query and maintenance cost and the evaluation cost to obtain the minimal cost.

Keywords : Data mining, data warehousing, evolutionary algorithms, materialized view selection.

I. INTRODUCTION

DATA warehousing is an approach to the integration of data from multiple, possibly very large, distributed, heterogeneous databases and other information sources. A data warehouse (DW) is a repository of integrated information available for querying and analysis. To avoid accessing the original data sources and increase the efficiency of the queries posed to a DW, some intermediate results in the query processing are stored in the DW. These intermediate results stored in a DW are called materialized views. On a sufficiently abstract level, a DW can be seen as a set of materialized views over the data extracted from the distributed heterogeneous databases. There are many research issues related to DWs [1], among which materialized view selection is one of the most challenging ones. On one hand, materialized views speed up query processing. On the other hand, they have to be refreshed when changes occur to the data sources. Therefore, there are two costs involved in materialized view selection: the query processing cost and the materialized view maintenance cost. The question we are interested in is: what views should be materialized in order to make the sum of the query performance and view maintenance cost minimal?

The materialized view selection involves a difficult trade-off between query performance and maintenance cost.

- Materializing all the views in a DW can achieve the best performance but at the highest cost of view maintenance.
- Leaving all the views virtual will have the lowest view maintenance cost but the poorest query performance. The word “virtual” here means that no intermediate result will be saved in the DW.
- We can have some views materialized (e.g., have those shared views materialized), and leave others virtual. In this way we may achieve an optimal (or near optimal) balance between the performance gain and maintenance cost. Unfortunately the materialized view selection design problem has been proven to be NP-hard [2]. Heuristics have to be used in practice to find a near optimal solution to this problem.

The problem considered in this paper can be described as follows. Based on a set of frequently asked DW queries, select a set of views to materialize so that the total query and maintenance cost is minimized. Our problem is related to three different is- sues:

1) query optimization;
2) multiple query optimization;
3) materialized view selection.

The existing algorithms for solving one or more of the above optimization problems can be classified into four categories according to [3].

Deterministic algorithms usually construct or search a solution in a deterministic manner either by applying heuristics or by exhaustive search.

Randomized algorithms pursue a completely different approach. First, a set of moves are defined. These moves constitute edges between different solutions in the solution space. Two solutions are connected by an edge if and only if they can be transformed into one another by exactly one move. Each of the algorithms performs a random walk along the edges according to certain rules, terminating as soon as no more applicable ones exist or a time limit is exceeded. The best solution encountered so far will be the result.

Evolutionary algorithms use a randomized search strategy similar to biological evolution in their search for good solutions. Although an evolutionary algorithm resembles

Terminates as soon as there is no further improvement over a period or after a predetermined number of generations. The fittest individual found is the solution.

Hybrid algorithms combine deterministic and randomized algorithms in various ways, e.g., solutions obtained by deterministic algorithms can be used as starting points for ran-

domized algorithms or as initial population members for evolutionary algorithms; a deterministic algorithm can be applied to the best solution found by an evolutionary algorithm, etc.

In [4], the technique used was to reduce the solution space by considering only the relevant elements of the multidimensional lattice. Unfortunately, potential good solutions may be lost in the reduction process. In [2] and [5], the goal was to select an appropriate set of views that minimizes the total query response time and/or the cost of maintaining materialized views, given a limited amount of resources such as materialized time, storage space, or total view maintenance time. A greedy heuristic algorithm was used. The performance of the algorithm is highly problem dependent because the greedy nature of the algorithm makes it susceptible to poor local minima.

In [6], a framework and algorithms were described for an-analyzing the issues in materialized view selection in order to achieve the best combination of good query performance and low view maintenance cost. The 0–1 integer programming technique was used to obtain the optimal global processing plan and then a heuristic algorithm was employed to select the materialized views based on this global processing plan. It is worth noting that the optimal global processing plan found in such a way may not lead to the best set of materialized views. It is possible that another near optimal global processing plan may lead to a better set of materialized views. The two optimization problems should not be separated.

This paper adopts a holistic approach to materialized view selection and considers local processing plans, global processing plans, and materialized view selection in an integrated framework and algorithm [7]. The relationships among the three can be explored and exploited by our algorithms. Hence algorithms proposed in this paper are more likely to find better solutions than other methods, [2], [4], [5], [6], [8],[9].

Although evolutionary algorithms have been applied to query optimization in recent years [3], because of its robustness and strong global search ability, few attempts have been made to make use of evolutionary algorithm's power in solving more complex problems, such as materialized view selection. In this paper, we propose several hybrid evolutionary and heuristic algorithms for optimizing global processing plans and materialized view selection. The hybrid algorithms combine evolutionary algorithm's power in global search with heuristic's ability in fine-grained local search to find a good set of materialized views. Our experimental results show that the hybrid algorithms performed better than the existing algorithms. They also performed better than either evolutionary algorithms or heuristic algorithms alone.

In this paper, the data model is based on selection-projection-join (SPJ) model rather than the multidimensional model.

The rest of this paper is organized as follows. Section II

explains and formulates the problem of materialized view

Selection based on global processing plans. It also describes the cost model considered in this paper.

II. MATERIALIZED VIEW SELECTION

Materialized view selection consists of three optimization problems, i.e., query optimization, multiple query optimization, and materialized view selection. It should be pointed out that a set of locally optimized queries may not be optimal anymore if multiple queries are considered together. Similarly, an optimal set of multiple queries does not guarantee the optimal selection of materialized views because a different set may lead to better materialized views. It is important to consider all three problems together in materialized view selection.

A. Query Optimization

A lot of research has been done on this topic. In query optimization, join operation is one of the most expensive operations. For simplicity we only consider join operation in this paper. That is, query optimization will be regarded as join order optimization here.

Assume that a database is given a set of relations R_1, R_2, R_m. A local processing plan is defined as a query graph, in which all relations are leaf nodes and all operations (e.g., join, projection, and selection) are specified as its inner nodes. Since we only consider join operation, a local processing plan for a query can be regarded as a binary join tree that consists of all relations as its leaves and join operations as its inner nodes. The edges are labeled with the join predicate and join selectivity. The join predicate maps tuples from the Cartesian product of the adjacent nodes to {false, true} depending on whether the tuple is to be included in the result or not. The join selectivity is the ratio between the included and total number of tuples.

The search space for query optimization is the set of all possible local processing plans. A point in the search space is one particular plan. Every point of the search space has a cost associated with it. Since there are lots of methods, such as nested loop, sort-merge, and hash loop, to perform a join operation, there exist several cost functions with respect to the processing tree. For example, the left trees in Fig. 1 denote nested loop join method. For nested loop join with no indices available, each tuple of the outer relation must be checked against every tuple of the inner relation, so the cost

. In the solution space, the left-deep processing trees have been

$$(R_1 \bowtie \quad ||R_1||R_2||_*$$

Of special interest to researchers. The left-deep tree is a tree where inner relation of each join is a base relation.

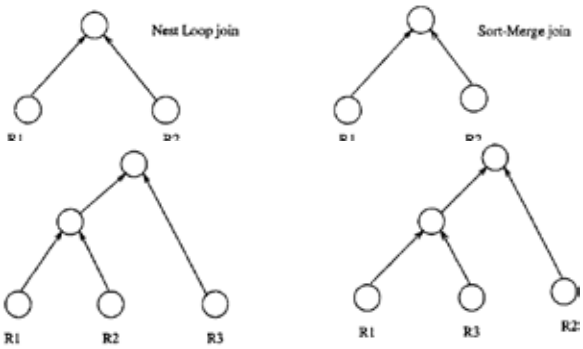

Fig. 1. Examples of join trees using nested loop join (left trees) and sort-merge join (right trees) operations.

TABLE I
POSSIBLE NESTING ORDERS FOR JOIN OPERATIONS [3]

Relations(n)	Processing Trees	solutions(Trees*n!)
1	1	1
2	1	2
3	2	12
4	5	120
5	14	1,680
6	42	30,240
7	132	665,280
8	429	17,297,280
9	1,430	518,918,400
10	4,862	17,643,225,600
11	16,796	670,442,572,800
12	58,786	28,158,588,057,600

we focus on left-deep trees. For relations, Table I [3] illustrates how big the solution space is. In fact, it has been shown that query optimization is NP-hard.

In Fig. 1, the costs of two join trees in (1) may be different Due to different join methods. The costs of two join trees in (2) are different as well although the structure is the same, because the join orders of relations are different. The left tree in (2) is. Assume that the sizes for R1, R2, and R3 are 20, 30, and 40, respectively. The cost of the left tree in (2) is derived as follows. First calculate the cost of $(R_1 \bowtie R_2)$ as $\|R_1\| * \|R_2\| = 20 * 30 = 600$ $(R_1 \bowtie R_2)$ results in 20 tuples, then the cost of $((R_1 \bowtie R_2) \bowtie R_3)$ will be $20 * 40 = 800$. Hence the total cost is $600 + 800 = 1400$. However, the cost of the right tree in (2) will be different. The cost of $(R_1 \bowtie R_3)$ is $\|R_1\| * \|R_3\| = 20 * 40 = 800$ $(R_1 \bowtie R_3)$ results in 10 tuples, then the cost of $((R_1 \bowtie R_3)$ will be $10 * 30 = 300$, i.e., a total cost of $800 + 300 =$.

Given a set of processing plans for a query, the goal of query optimization is to find a processing plan with the lowest query processing cost. There has been some work on applying evolutionary algorithm to query optimization [3]. In this paper, query optimization is only part of a large problem—materialized view selection.

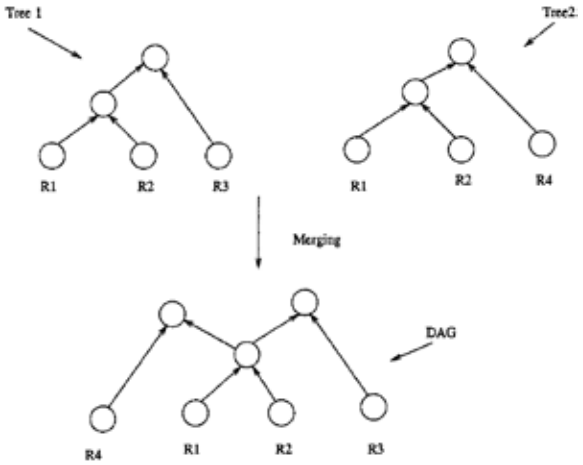

Fig. 2. Global processing plan should be a directed acyclic graph (DAG).

B. Multiple Query Optimization

A DW is a repository of integrated information available for querying analysis. One issue we have to deal with is multiple query processing. In a systematic look at the problem has been presented.

Assume that a set of queries Q={Q1,Q2,.....Qn} are given. For every query Q_i, there exists at least one processing plan, called a local processing plan. A global/multiple processing plan for Q corresponds to a global plan that provides a way to compute the results for queries. A global/multiple processing plan can be constructed by choosing one plan for each query and then merging them together. A locally optimal plan is referred to as the cost plan for processing a query Q_i individually. This corresponds to query optimization. The globally optimal plan is referred to as the global processing plan by merging the common parts of individual local plans.

The multiple query optimizations (MQO) problem can be formulated as follows. Given sets of local processing plans P_1, P_2, P_n, with $P_i\ p_{ik_i}\}$ being the set of possible plans for query $Q_i, \leq \leq$, k_i is the number of local processing plans for Q_i. Find a global/multiple processing plan by selecting one plan from each P_i such that the cost (query cost) of the global/multiple processing plan is minimized.

In general, the union of locally optimal plans does not necessarily form a globally optimal plan. Hence, we cannot find the globally optimal plan by simply combining locally optimal plans. A heuristic algorithm is often needed in searching for a globally optimal plan. In a heuristic search algorithm is proposed, which only examines a fraction of all possible global processing plans. Some potentially good plans may be lost. By using the evolutionary approach, our algorithms are capable of carrying out global search and looking into all possible combinations of individual plans.

When combining multiple query processing plans, i.e., multiple join trees, the produced global processing plan should be a directed acyclic graph (DAG) not a tree. This is shown in Fig 2.

C. Materialized View Selection

In DW, selected information is extracted in advance and stored in a repository. A DW can therefore be seen as a set of materialized views defined over the sources. The problem we are dealing with now is how to select the views to be materialized so that the cost of query processing and view maintenance for all the nodes in a global processing plan is minimized.

An easy approach would be to use exhaustive search to find the optimal set of materialized views on the set of queries. However, this approach is impractical if the search space is big. It has been shown that materialized view selection is NP-hard [2]. Heuristic algorithms have to be used to trim the search space in order to get the results quickly [2], [4], [5]. However, the performance of a heuristic algorithm depends heavily on the quality of heuristics which may be difficult and/or costly to obtain in practice. Heuristic algorithms also get stuck easily in a local optimum. Compared with heuristic algorithms, evolutionary algorithms have many advantages, such as searching from a population of points using probabilistic transition rules. In order to avoid an exhaustive search in the whole solution space and obtain a better solution than that obtained by heuristic methods, we propose a new evolutionary approach to materialized view selection.

D. Cost Model of Materialized View Selection

1) Motivating Example: Our example is taken from a DW application which analyzes trends in sales and supply [6]. The relations and the attributes of the schema for this application are the following.

Item(I_id, I_name, I_price) Part(P_id, P_name, I-id)

Supplier(S_id, S_name,P_id, City, Cost, Preference)

Sales(I_id, Month, Year, Amount)

There are five queries, as follows.

Q1: Select P_id, min(cost), max(cost) From Part, Supplier

Where Part.P_id Supplier.P_id

And P_name in {"spark_plug," "gas_kit"} Group by P_id

Q2: Select I_id,

sum(amount number min_cost) From Item, Sales, Part

Where I_name in {"MARUTI," "NISSAN," "TOYOTA"}

And year 1996

And Item.I_id Sales.I_id And Item.I_id Part.I_id And Part.P_id

(Select P_id, min(cost) as min_cost

From Supplier

Group by P_id) Group by I_id

Q3: Select P_id, month sum(amount) From Item, Sales, Part

Where I_name in {" MARUTI," "NISSAN," "TOYOTA"}

And year 1996

And Item.I_id Sales.I_id And Part.I_id Item.I_id Group by P_id, month

Q4: Select I_id, Sum(amount I_price)

From Item, Sales

Where I_name in {" MARUTI," "NISSAN," "TOYOTA"}

And year 1996

And Item.I_id Sales.I_id

Group by I_id

Q5: Select I_id, avg(amount I_price) From Item, Sales

Where I_name in {" MARUTI," "NISSAN," "TOYOTA"}

and year 1996

and Item.I_id Sales.I_id

Group by I_id.

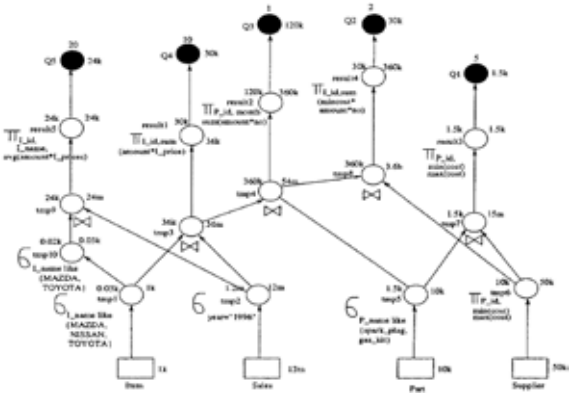

Fig. 3 gives a possible global query processing plan for the five queries listed above, in which the local access plan for individual queries are merged based on the shared operations on common data sets. We call it the multiple view processing plan (MVPP).

The query access frequencies are labeled on the top of each query node. For simplicity, we assume that all the base relations Item, Sales, Part, and Supplier are updated only once for a certain period of time. In Fig. 3, we abbreviate one thousand as "k," one million as "m," and one billion as "b." For example, the cost for obtaining tmp3 by using tmp1 is 36 m.

Now we have to decide which node(s) to materialize so that the total query and view maintenance cost is minimal. It is obvious from this graph that we have several alternatives for choosing the set of materialized views: e.g.,

1) materialize all the application queries;

2) materialize some of the intermediate nodes (e.g., tmp1, tmp3, tmp7, etc.);

3) leave all the nonlife nodes virtual.

The cost for each alternative can be calculated in terms of query processing and view maintenance.

REFERENCES

[1] J. Widom, "Research problems in data warehouse," in Proc. 4th Int. | Conf. Inform. Knowledge Manage. 1995, | [2] H. Gupta and I. S. Mumick, "Selection of views to materialize under a maintenance cost constraint," in Proc. Int. Conf. Database Theory | (ICDT), 1999, pp. 453–470. | [3] M. Steinbrunn, G. Moerkotte, and A. Kemper, "Heuristic and randomized optimization for the join ordering problem," Very Large Data Base | J., vol. 6, no. 3, pp. 191–208, 1997. | [4] E. Baralis, S. Paraboschi, and E. Teniente, "Materialized view selection in a multidimensional database," in Proc. 23rd Int. Conf. Very Large | Data Base (VLDB), 1997, pp. 156–165. | [5] H. Gupta et al., "Index selection for olap," in Proc. Int. Conf. Data Eng. (ICDE), 1997, pp. 208–219. | [6] J. Yang, K. Karlapalem, and Q. Li, "Algorithm for materialized view | design in data warehousing environment," in Proc. 23th Int. Conf. Very | Large Data Bases (VLDB), 1997, pp. 136–145. | [7] H. Gupta. "Selection of Views to Materialize in a Data Warehouse". Proceedings of International Conference on Database Theory, Athens, Greece 1997. | [8] W. J. Labio, D. Quass, and B. Adelberg, "Physical database design for data warehouses," in Proc. Int. Conf. Data Eng. (ICDE), 1997, pp. | 277–288. | [9] K. A. Ross, D. Srivastava, and S. Sudarshan, "Materialized view maintenance and integrity constraint checking: Trading space for time," in | Proc. ACM SIGMOD Int. Conf. Manage. Data, 1996, pp. 447–458.