Selection of Materialized Views: A Cost-Based Approach

Xavier Baril and Zohra Bellahsène

LIRMM - UMR 5506
CNRS/Université Montpellier II
161 Rue Ada
F-34392 Montpellier Cedex 5
{baril,bella}@lirmm.fr

Abstract. Recently, multi-query optimization techniques have been considered as beneficial in view selection setting. The main interest of such techniques relies in detecting common sub expressions between the different queries of workload. This feature can be exploited for sharing updates and space storage. However, due to the reuse a query change may entail an important reorganization of the multi query graph. In this paper, we present an approach that is based on multi-query optimization for view selection and that attempts to reduce the drawbacks resulting from these techniques. Finally, we present a performance study using workloads consisting of queries over the schema of the TPC-H benchmark. This study shows that our view selection provides significant benefits over the other approaches.

1 Introduction

The problem is to select a set of views to materialize that optimizes both the view maintenance and query time given some constraints. The ideal selection strategy should provide high query performance and low view maintenance cost. However, these two costs are in conflict, the issue is to find a method that ensures a balance between maintenance and query processing cost.

There are many motivations for investigating the view selection problem. At first, materialized views are increasingly being supported by commercial database systems and are used to speed up query response time. Therefore, the problem of choosing an appropriate set of views to materialize in the database is crucial in order to improve query processing cost. Another application of the view selection issue is selecting views to materialize in data warehousing systems to answer decision support queries. Furthermore, new applications of the problem of view selection arise namely in data placement in distributed databases and in peer to peer computing.

Most of the previous related approaches are more theoretical studies than pragmatic approaches and considered that the cost model is one parameter among others. In this paper, we present a pragmatic approach that is based

J. Eder and M. Missikoff (Eds.): CAiSE 2003, LNCS 2681, pp. 665–680, 2003.

on a cost model and experiments performed on a real data base system. Moreover, the processing cost of query is estimated according to the operation type. For instance, the estimated cost of a join is not the same as the cost of a selection. In related work [12], the operation cost of the query view is estimated independently of their type. We have performed experiments to quantify the effect of some parameters (query and update frequencies, number of queries, etc.) on view selection strategy. For this purpose we have compared our approach, which make use of more precise cost model than the one used in MVPP [12]. Indeed, the MVPP approach doesn't take into account the type of operations to estimate their cost.

1.1 Contribution

We make use of multi-query optimization techniques in order to detect overlapping between queries of workload. Our approach provides the following features:

- we have designed a polynomial time algorithm that select views for materialization,
- we exploit common sub expressions (with reuse factor of a view),
- the view selection is decided under space constraint,
- our approach has been implemented.

Furthermore, we present a performance study using workloads of queries over the schema of the TPC-H benchmark [11]. This study shows that our view selection provides significant benefits over the other approaches.

1.2 Outline

The rest of this paper is organized as follows. In Section 2, it is presented the problem of view selection and our formalism for graphically representing the queries of workload. Section 3 provides an overview of our approach and gives the algorithm of selecting views to be materialized. Section 4 contains the sample example. The cost model that is used in our view selection approach is described in Section 5. In Section 6, is provided the performance evaluations. Finally, Section 7 presents related work and Section 8 contains concluding remarks and future work.

2 Preliminaries

We consider Selection-Projection-Join (SPJ) views that may involve aggregation and group by clause as well. A view is a derived relation defined by a query in terms of source relations and/or other views. It is said to be materialized when its extent is computed and persistently stored. Otherwise, it is said to be virtual.

The problem addressed in this paper is similar to that of deciding which views to materialize in data warehousing [1, 2, 5, 4, 6, 10, 12]. The general problem

of view selection can be formulated as follows. Given a set of source relations $R = \{R_1, \ldots, R_n\}$, a set of queries $Q = \{Q_1, \ldots, Q_k\}$, the problem is to find a set of views to materialize $M = \{V_1, \ldots, V_m\}$ under a storage space constraint, which have the best balance between view maintenance cost and query processing cost.

2.1 The Multi View Materialization Graph

In this subsection, we present the framework for representing views to materialize in order to exhibit common sub-expressions. The task of a view selection module, which is based on multi-query optimization, is (i) to recognize possibilities of shared views and (ii) to apply a strategy for selecting views to materialize. The first task involves setting up the search space by identifying common sub-expressions This task is of importance as in the multi-query optimization. But it is orthogonal to the view selection process itself.

The Multi View Materialization Graph (MVMG) is similar to the AND-OR DAG representation of queries in multi query optimization [8]. The MVMG is a bipartite Directed Acyclic Graph (DAG) composed of two types of nodes: AND-nodes and OR nodes. Each AND-node represents an algebraic expression (Select-Project-Join) with possible aggregate function. An OR node represents a set of logical expression that are equivalent (i.e., that yield the same result). The AND-nodes have only OR-nodes as children and OR-nodes have only AND-nodes as children. In fact, the MVMG represents AND-OR DAGs of several queries in a single DAG. The leaf nodes of the MVMG are equivalence nodes representing the base relations. In general, for each base relation, there is one leaf node except in case of a selfjoin. Equivalence nodes in MVMG correspond to the views that are candidate to the view selection.

We consider the equivalent query graphs of each query and provide the expression DAG derived from these graphs. Then, all the resulting graphs are merged into one Multiple View Materialization Graph (MVMG) where the common sub-expressions are represented once. We borrow the rule provided in [8] for identifying common sub-expressions. For example, equivalent nodes obtained after applying join associativity are replaced by one single equivalence node.

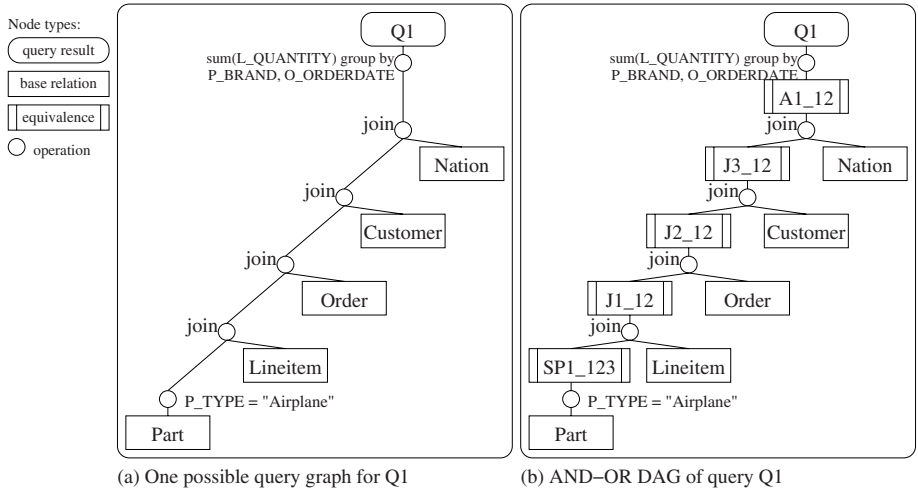
```
Part(partkey, name, brand, type, size, retailprice)
Supplier(suppkey, name, address, nationkey, phone, acctbal)
PartSupp(partkey, suppkey, availqty, supplycost)
Customer(custkey, name, address, nationkey, phone, acctbal)
Orders(orderkey, custkey, orderstatus, totalprice, orderdate)
Lineitem(orderkey, partkey, suppkey, linenumber, quantity)
Nation(nationkey, name, regionkey, regionkey, comment)
Region(regionkey, name, comment)
```

Fig. 1. *Tables of TPC-H benchmark*

Example The queries used in this paper are defined over a simplified version of the TPC-H schema [11] described in Figure 1. Let us consider the query Q_1, which finds the number of orders of Airbus planes ordered by different nations :

```
Select   N.name, P.brand, O.orderdate, Sum(L.quantity)
From     Part P, Customer C, Orders O, Nation N, Lineitem L
Where    P.type = 'airplane'and AND P.brand = 'Airbus' AND
         P.partkey = L.partkey AND L.orderkey = O.orderkey AND
         O.custkey = C.custkey AND C.nationkey = N.nationkey
Group by N.name P.brand, O.orderdate;
```

The AND-OR DAG representation of the query graph of query Q_1 depicted Figure 2(a) is shown in Figure 2(b). Circles represent AND nodes, i.e. operations and boxes represent OR nodes, i.e. equivalence nodes. In the transition from Figure 2(a) to Figure 2(b), new nodes are created to represent the equivalence nodes, corresponding to operation results (e.g. node labelled $A1_{12}$ represents the result of the aggregation operation).



(a) One possible query graph for Q1 (b) AND–OR DAG of query Q1

Fig. 2. *Query graph and its AND-OR representation*

2.2 Notion of Level in the MVMG

Our view selection algorithm is based on the notion of level in the query tree. For this purpose, each view (equivalence node) of the query tree is associated to a level, which is defined as follows:

- $level(root) = 1$ with $root$ the view representing the query result,
- $level(view) = level(parent(view)) + 1$ with $parent$ a function which gives the parent of a view in the query tree.

3 Our View Selection Method

In our previous work on the view selection problem [2], only leaf nodes of the MVMG were materialized. We significantly improved our view selection strategy.

In this section, we present our strategy for selecting a set of views to be materialized. The optimal solution is the one which computes the set of equivalence nodes (i.e., views) of the MVMG such that the sum of cost of processing all the queries and maintaining all the views is minimal. However, the search space for the optimal solution is very large since it entails a great number of comparisons between all possible subsets of this set of vertices. If n is the number of nodes of a MVMG, then the number of combinations of nodes is 2^n. For this matter, we consider the sum cost of processing cost and view maintenance per query. Thus the first argument of our solution is reducing the complexity of the view selection algorithm, which selects views to materialize in the MVMG. The second argument for considering the views to materialize query by query relies in preserving the data independence whenever adding a query to the view configuration or removing one from it. Indeed, the side effect on existing queries is reduced since the view selection is applied query per query. In the related work since the view selection strategy consists in considering all equivalence nodes of the multi view query graph, the impact of adding or removing a query may lead to an important reorganization.

More precisely, we make use of the following heuristics:

- Searching the views to materialize per level and per query. This heuristic is based on the observation that the maintenance cost decreases from the root level down to the leaf level of query tree. This is true in the context when the operations of the query tree are not executed in pipeline way.
- Selecting the level, which provides the minimal sum of query processing and view maintenance cost.
- Taking the reuse of views into account. This feature allows to reduce the view maintenance cost and space storage.
- A pre-selection of beneficial views is performed as follows. A view is considered as beneficial if and only if its materialization reduces significantly the query processing cost and without increasing significantly the view maintenance. For instance, if the materialization of a view v reduces the query cost from 2 minutes to 10 sec, and increases the view maintenance cost from 1 sec to 9 sec, then view v should be materialized. View benefit is formally defined locally to a query (see 3.2) and on the entire MVMG (see 3.3).

3.1 Algorithm for View Selection

In the classical approach[1], which consists in fully materializing the view, the materialization level corresponds to the root level of the query tree. In this

[1] We refer to the approach which does not use the multi-query optimization feature by considering views separately.

approach, the query processing cost is low and the view maintenance cost may be high. The opposite solution is leaving the view virtual. Consequently, the query processing cost is high and the view maintenance cost is null. The aim of our approach is to provide a solution based on the balance between query processing and maintenance cost. For this purpose, our idea is finding an intermediary level for each query tree optimizing the sum of the query processing and the view maintenance cost.

The algorithm 1 computes, for each query, the materialization level according to the query frequency and the data update frequency of each involved relation. The complexity of the algorithm is in $O(n \times k)$ where n is the number of queries and k is the average number of levels in a query. The treatment is done in two main phases. The first one carry out a pre-selection of beneficial views. The second phase computes the total cost (query plus maintenance) for each level of the query graph and selects the one which has the minimal sum of query processing and view maintenance cost.

Due to space limitation, we don't indicate types of the variables used in the algorithms. However, we use the following syntactic convention : variables in capital letter denote sets.

```
   Data   : mvmg, q
   Result : M // set of views to materialize for the considered query

   // set of pre-selected views;
   P = ∅;
   // pre-selection on local benefit;
   for each v of AllChildren(q) do
1  |  b = LocalBenefit(q, v);
   |  if b > 0 then
   |  |  P = P ∪ {v}
   |  end
   end
   // get the level having the minimum total cost;
   mlc = ∞;
   for each L of AllViewsOfLevel(q) do
2  |  PL = L ∩ P;
3  |  lc = TotalCost(q, PL);
   |  if lc < mlc then
   |  |  M = PL;
   |  |  mlc = lc;
   |  end
   end
```

Algorithm 1: Selection of a query materialization level

For each view, the pre-selection step is performed using the following formula (line 1):

$$LocalBenefit(query, view) =$$
$$QueryCost(query, \emptyset) \times frequency(query)$$
$$-(QueryCost(query, \{view\}) \times frequency(query) + \frac{MaintenanceCost(view)}{reuse(view)})$$

During the selection phase, the set of pre-selected views for the current level is computed, and stored in the variable PL (line 2): it is the intersection of the views belonging to the level (L) and the pre-selected views (P). Next, the total cost for pre-selected views of this level is computed in variable lc (line 3). The total cost is given by the following formula:

$$TotalCost(query, M) =$$
$$QueryCost(query, M) \times frequency(query) + \sum_{v \in M} \frac{MaintenanceCost(v)}{Reuse(v)}$$

Finally, the level having the minimum total cost is selected.

3.2 General Algorithm

As explained above, the algorithm 1 provided the set of candidate views for materialization. We now present the second part of the view selection method performed by the algorithm 2. It is aimed to find among the candidates views to materialize, those optimizing the global benefit under the space constraint.

The general algorithm takes into account a constraint for the storage space of the materialized views (variable s). Thus, the problem can be resumed as follows. Let us consider:

- Let C be the set of candidate views for materialization,
- Let s be the available space storage,
- Let $s(v)$ be the function giving the storage space of a view,
- Let $GlobalBenefit(v)$ be the function giving the global benefit of a view.

The first step of the general algorithm is to select the set of candidate views for materialization. For this purpose, we use the level selection algorithm for each query, and the candidates views for materialization in C.

Next, the candidate views are filtered according to their global benefit, and views having a negative benefit are removed from the set of candidate views. The formula used for the global benefit (line 2) is:

$$GlobalBenefit(view) =$$
$$QueryCost(\emptyset) - (QueryCost(\{view\}) + MaintenanceCost(view))$$

The problem is to find a set of views to materialize M, such that $\sum_{v \in M} s(v) < s$ and $\sum_{v \in M} GlobalBenefit(v)$ is maximum. This problem is similar to those of the knapsack therefore it may be solved using the dynamic programming

```
   Data   : mvmg, s // space constraint
   Result : M // set of views to materialize
   // set of candidate views for materialization;
   C = ∅;
   // union of candidate views for each query;
   for each q of mvmg do
1  |  C = C ∪ LevelSelection(mvmg, q)
   end
   ;
   // pre-selection on global benefit;
   for each v of C do
2  |  if GlobalBenefit(v) < 0 then
   |  |  C = C − {v}
   |  end
   end
   // view selection under space constraint according to global benefit;
3  M = knapsack(C, s, s(), GlobalBenefit());
```

Algorithm 2: The general algorithm for selecting views to materialize

paradigm. For this purpose, we make use of a knapsack function (line 3). This function takes as input all the parameters described above (including the set of candidate views, the space constraint, the function given the size of a view and the function given the global benefit of a view) and returns a set of views satisfying the problem. Note that the knapsack algorithm is polynomial in $O(n \times s)$, where n is the number of candidate views for materialization ($n = |C|$) and s the available storage space. Computing benefits and costs of query and maintenance can be done in polynomial time before running the selection algorithm : it needs a traversal of the views of each queries of the MVMG. The level selection algorithm is also polynomial in the number of level of the considered query. Then, the general algorithm is polynomial because it uses only polynomial subroutines.

4 Sample Example

Let us consider four other queries in addition to Q1 presented previously in Section 2. Query Q2 finds the number of Airbus planes bought by the United States for the last 6 years.

```
Select   P.brand, O.orderdate, Sum(L.quantity)
         From Part P, Customer C, Orders O, Nation N, Lineitem L
Where    N.name = 'USA' and P.type = 'Airplane'
         and P.partkey = L.partkey
         and C.nationkey= N.nationkey
         and C.custkey = O.custkey and O.orderkey = L.orderkey
         and O.o_orderdate > 1996 and P.brand = 'Airbus'
Group By P.brand, O.orderdate;
```

Query Q3 lists the name, the brand, the price, the number and the quantity of orders of every brand of planes.

```
Select   P.name, P.brand, P.retailprice,Sum(L.quantity), Count(*) As C
From     Part P, Lineitem L
Where    p.type = 'airplane' and P.partkey = L.partkey
Group By P.name, P.brand, P.retailprice
```

Query Q4 finds the minimal and the maximal supply cost for each country and each product having the brand name 'Renault'. The associated query is as follows:

```
Select   P.partkey, N.nationkey, N.name,Min(PS.supplycost), Max(PS.supplycost)
From     Part P, Supplier S, Nation N, PartSupp PS
Where    P.brand = 'Renault'
         and P.partkey = S.partkey
         and P.partkey = PS.partkey
         and PS.suppkey = S. suppkey
         and S.nationkey = N.nationkey
Group by P.partkey, N.nationkey, N.name
```

Q5 lists the supplycosts and all the identifiers of each product having as brand name 'Peugeot' and supplied in the USA. The associated query is as follows:

```
Select   P.partkey, S.supplycost
From     Supplier S, Part P, Nation N, PartSupp PS
Where    P.P_brand = 'Peugeot'
         and N.N_name  =  'USA'
         and P.partkey = PS.partkey
         and S.nationkey = N.nationkey
         and PS.suppkey = S.suppkey
Group by P.partkey, S.supplycost
```

Figure 3 illustrates the Multi View Materialization Graph of the five queries described above. The equivalence nodes are labeled OPi_{jk} where OP is the operation type, i is a counter and jk is the list of queries sharing the node. Operation type could be A for an aggregation, J for a join and SP for a selection-projection. For example, $J1_{123}$ denotes the first join shared by queries Q_1, Q_2 and Q_3.

5 The Cost Model

5.1 Estimated Cost of the Operations

We have been inspired by the formula given in [3] for estimating the query cost of the operations: join, selection and projection. These costs are estimated according to the size of the involved relations. The formulas used for cost operations estimation are simple but sufficient : it is proven by the performances of our approach, presented in the next section.

- Estimated cost of unary operations.
  - $Cost(op) = rows$, where op is an aggregation operation,
  - $Cost(op) = rows$, where op is a selection operation,
  - $Cost(op) = rows * log(rows)$, where op is a projection.

  Where rows is the number of tuples of the operand.
- Estimated cost of join
  $Cost(op) = \alpha \times lrows \times rrows \times \beta \times (lrows + rrows)$, with α and β are constant, and we assume that α is relatively small. Where $lrows$ and $rrows$ are respectively the number of rows of the left and right operands.

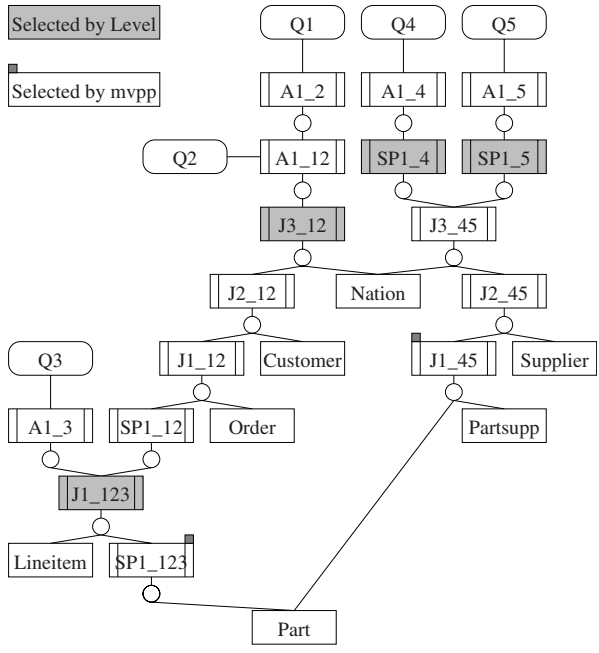


Fig. 3. *Example of MVMG Graph with five queries*

5.2 Maintenance Cost

Our view selection strategy assumes incremental maintenance. We consider two kinds of maintenance operation: add and delete. Let AddCost(view,relation) be the cost of maintaining view when a row is added to the relation and DeleteCost(view,relation) be the cost of maintaining the view when a row is deleted from the relation. To each relation is associated the frequencies of add and delete operations. Then, the maintenance cost for a view, is defined as the sum of add and delete cost for each relation, multiplied by the corresponding frequency:

$$MaintenanceCost(view) = \sum_{r \in R} AddCost(view, r) \times addF(r) + DeleteCost(view, r) \times deleteF(r)$$

Where $addF(r)$ is the frequency of adding tuples to relation r, and $deleteF(r)$ is the frequency of deleting tuples from relation r.

Reuse factor Although our method relies in selecting views per query (i.e. local optimization), it takes the reuse of views into account by merging all the queries of the workload in the same MVMG. For this purpose, we defined a reuse factor as the sum of queries frequencies using a view.

$$Reuse(view) = \sum_{q \in Q_{view}} frequency(q)$$

Where Q_{view} is the set of queries using $view$.

5.3 Query Processing Cost

In the case where there are no materialized views, the query cost is the sum of execution costs of all the operations belonging to the query graph. When there are materialized views, it is not necessary to execute operations which are descendant nodes of the views in the query graph. For that purpose, we define the query cost with to parameters: the considered query and the set of materialized views. The function $OpsToExec(query, M)$ returns the set of operations which are the necessary to execute to compute the result of query, given M a set of materialized views.

$$QueryCost(query, M) = \sum_{op \in OpsToExec(query, M)} Cost(op)$$

Each query is associated to a frequency giving its importance. The global query cost takes into account query frequencies. Then, the global query cost (given a set of materialized views) is the sum of each query cost multiplied by its query frequency.

$$QueryCost(M) = \sum_{q \in Q} QueryCost(q, M) * frequency(q)$$

where Q is the set of queries of the MVMG and $frequency$ a function which gives the frequency of a query.

6 Performance Study

We implemented the two algorithms presented in the section 3 for finding views for materialization. The goal of the experiments was to quantify the benefits of our view selection method. To achieve this purpose, we implemented the TPC-H database at scale of 0.01 (i.e., 10MB total size) on MySQL. We chose as workload consisting of five queries on the database schema of the TPC-H benchmark, which are presented in section 4. The experiment consists of applying the view selection method for finding the views for materialization. Once these views was known, we measured the real cost executing the queries of workload and the view maintenance cost of the materialized views.

6.1 Experiment Process

The experiments were performed on a MySQL server (version 3.23.42) through JDBC interface. The machine is a single processor (300 Mhz UltraSparc) Ultra 2, with 640 MB memory, running Solaris 5.7. Each cost has been measured three times and is expressed in millisecond (ms).

We consider several view selection strategies. The first one is our approach, called "level" approach. The "MVPP" approach [12] : we implemented the algorithm in order to achieve a comparison with our approach. In MVPP, the view maintenance is assumed to be performed by re-computing the views, according to the suggestion done in [12]. Then we consider two basic strategies : the "all" materialized approach which materializes the result of each query (this is the classical approach which does not use the multi-query optimization techniques) and the "virtual" approach which does not materialize views and always recompute queries.

We considered different scales of frequencies for access and update. Concerning update frequencies we try to extend TPC requirements which are not very accurate.

The "real" query cost is defined as the sum of the cost of executing each query of the workload on the MVMG, using materialized views as often as possible. For this purpose, we measured the execution time of each operations of the MVMG in MySQL. The query cost is the sum of all the operations used to compute the result of a query. This strategy is aimed to avoid to use the MySQL query optimizer which would modify the results. To improve query time, namely the join operation, we created indexes on the attributes involved in join conditions.

The "real" maintenance cost is defined as the sum of the cost of maintaining all materialized views. This cost includes the evaluation of cost of computing the new tuples to add (or to remove) from the view plus the cost of writing them. At first, we measured the execution time for adding or deleting a row from each view. Then, we estimated the number of rows to add to the view. Finally, the cost of writing the new added tuples is obtained by multiplying this number by the time to add a row. This cost is computed as the sum of the related operations costs, which is pondered proportionally to the number of added rows (or deleted tuples).

6.2 Experiment Results

Figure 4 presents the performance resulting from evaluating the global cost of the workload involving 1, 3, 5 and 10 queries to test scalability of the selection methods according to the number of queries. The update frequencies and access frequencies are at scale 1. This graphic shows that our approach provides the lowest global cost. The gap between our approach and the other rises according to the number of queries.

The graphic depicted in Figure 5 shows the global cost of the workload involving 10 queries while varying the access frequency. The update frequency is at scale 1. We can see, our approach outperforms the MVPP approach and the "all" materialized approach. The reason is MVPP tends to materialize views near the leaf level in the multi query graph. Therefore, the query processing is high. Note that for the MVPP strategy, the global cost decreases when the access frequency increases from 4 to 8. This is because the MVPP algorithm

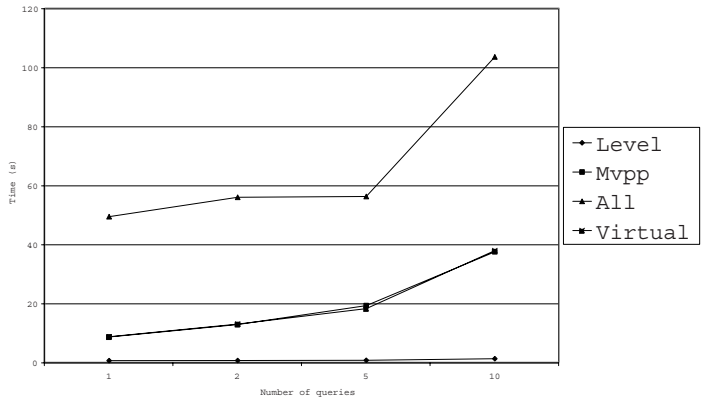


Fig. 4. *Evaluating global cost while varying the number of queries*

detects that the materialization of some views is beneficial with the increase of the access frequency.

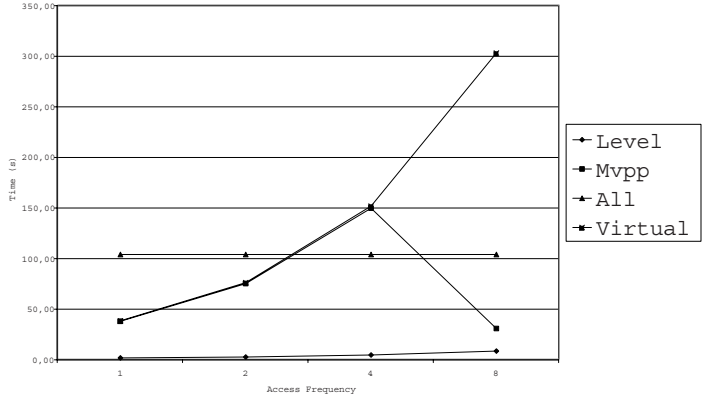


Fig. 5. *Evolving the access frequency*

Figure 6 presents the behavior of the different view selection methods while varying the update frequency for a workload involving 10 queries. The access frequency is at scale 1. We note that our approcah outperforms the others. The approach named "all" is not represented in this figure because its results are too bad and due to space limitation we don't put them on the histogram.

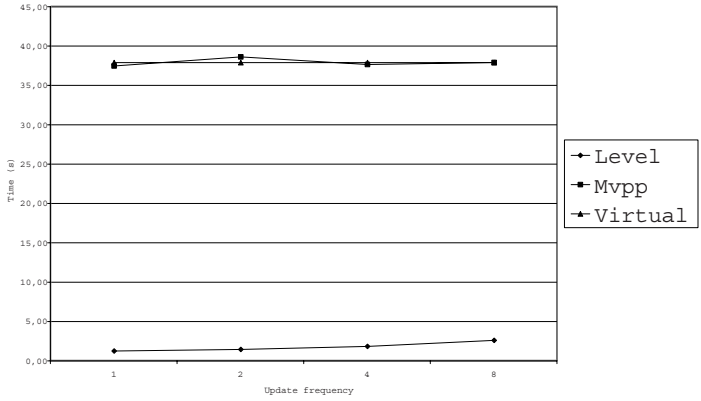


Fig. 6. *Evolving the update frequency*

6.3 Discussion

We note from the above experiments that our approach provides the better performances for global cost in all cases. Furthermore, our method supports scalability when the number of views is large (i.e., greater than 3). We have made experiments with various access and maintenance frequencies and our approach has always outperform the others.

We note that the MVPP approach tends to materialize views near the leaf level in the multi-query graph. In our opinion, there are two reasons to explain this behavior. The first one is that maintenance cost is sur-evaluated with MVPP, because they don't assume incremental maintenance. The second reason is that their cost model doesn't take into account the type of relational operations. So join operations which are very expensive are not in favor for materialization. For this reasons, the results provided by the MVPP approach often tend to be the same as the virtual approach.

Finally, we learnt from this study that the approaches using multi-query optimization techniques provide better performance than the classical approach named here "all" materialized approach.

7 Related Work

One of the key ideas of our approach is to detect common sub-expressions. This strategy has been applied in a significant number of papers in the context of multi-query optimization and also in view selection setting. The problem of the view selection and finding common sub-expressions are quite different: in the context of multi-query optimization, the problem is recognizing possibilities of shared computation whereas in our approach the problem is to find common

views (corresponding to intermediary results) which will be materialized. However, the sharing feature is one parameter among several others. For instance, the strategy of choosing views to be materialized in our approach takes both the view maintenance.

Most of the previous related approaches on view selection are theoretical studies rather than pragmatic approaches and considered that the cost model is one parameter among others. The work reported in [7] is an exception since it was based on a cost model and has been implemented. However, this work focused on view maintenance problem without considering the query processing cost. Moreover, common subexpressions have been exploited for improving view maintenance cost [7].

In [10], the dynamic data warehouse design is modeled as search space problem. Rules for pruning the search space have been proposed. The first rule relies on favoring the query rewriting that uses views already materialized. The second one modifies the previous rule to favor common subexpression. However, their view selection algorithm is still in exponential time. Besides, neither implementation and/or evaluation of their method have been performed.

Our Multi View Materialization Graph is close to the Multi View Processing Plan that has been described in [12]. However, the evaluation cost of the query is not estimated according to the operation type. It seems that the estimated cost of a join is estimated as the same as the cost of a selection. "The cost of a query is the number of rows present in the table used to construct Q" [12]. DynaMat [6] is a system aimed to unify the view selection and the view maintenance problems. The principle of this system is monitoring constantly the incoming queries and materializing the set of views given the space constraint. During the update only the most beneficial subset of materialized views are refreshed within a given maintenance window. However, this approach is efficient when queries are ad hoc especially in OLAP applications. More recently, a formal study of the view selection problem focusing on its complexity has been done in [3]. It shows notably that the cost model is a parameter of importance in the view selection setting.

8 Conclusion

In this paper, we have presented a pragmatic approach of the view selection problem that combines global with local optimization. Due to the reuse a query change may entail an important reorganization of the multi query graph. Thus the data independence of the views may be compromised. In our approach, adding or removing queries can be done without great reorganization as we have shown it in this paper. While in the related work since the materialization strategy consists in considering all nodes of the multi query graph, the impact of adding or removing a query is more important. This is the counterpart of sharing and because the view selection method consider all queries together.

The view selection algorithms that we have presented in this paper have been implemented in java interfaced with a MySQL sever. We have performed several experiments and comparison with the approach reported in [12]. The experiment results have shown that our approach provides significant benefits over the all considered approaches. Furthermore, we learnt from this study that the approaches using multi-query optimization techniques provide better performance than the classical approach named here "all" materialized approach. We are planning to investigate the view selection problem in the context of a P2P Database architecture.

References

1. S. Agrawal, S. Chaudhury, and V. Narasayya. Automated Selection of Materialized Views and Indexes for SQL Databases. In *Proceedings of the 26th International Conference on Very Large Databases, VLDB'2000*, Cairo, Egypt, 2000.
2. Z. Bellahsène and P. Marot. Materializing a Set of Views: Dynamic Strategies and Performance Evaluation. In *Proceedings of International Database Engineering and Applications Symposium*, Yokohoma, Japan, September 2000. IEEE publishing.
3. R. Chirkova, A. Halevy, and D. Suciu. A formal perspective on the view selection problem. In *Proceedings of the 27th International Conference on Very Large Databases, VLDB'2001*, Roma, Italy, September 2001.
4. H. Gupta. Selection of Views to Materialize in a Data Warehouse. In *Proceedings of the International Conference on Database Theory*, Delphi, Greece, January 1997.
5. H. Gupta and I. Mumick. Selection of Views to Materialize Under a Maintenance-Time Constraint. In *Proceedings of the International Conference on Database Theory*, Jerusalem, Israel, January 1999.
6. Y. Kotidis and N. Roussopoulos. DynaMat: A Dynamic View Management System for Data Warehouses. In *Proceedings of the ACM SIGMOD Conference*, Philadelphia, USA, 1999.
7. H. Mistry, P. Roy, and K. Ramamritham. Materialized View Selection and Maintenance Using Multi-Query Optimization. In *Proceeding of the International Conference on Management of Data SIGMOD*, USA, 2001.
8. P. Roy, S Seshadri, S. Sudarshan, and B. Siddhesh. Efficient and Extensible Algorithms for Multiquery Optimization. In *Proceeding of the International Conference on Management of Data SIGMOD*, San Diego, USA, 2000.
9. D. Theodoratos and T. Sellis. Data warehouse configuration. In *In Proceedings of the 23rd International Conference on Very Large Data Bases, VLDB'1997*, 1997.
10. D. Theodoratos and T. Sellis. Incremental Design. *Journal of Intelligent Information Systems*, 15:7–27, 2000.
11. TPC-R Benchmark Standard Specification 2.01, January 1999. *http://www.tpc.org.*
12. J. Yang, K. Karlapalem, and Q. Li. Algorithm for Materialized View Design in data Warehousing Environment. In *Proceedings of the 23rd International Conference on Very Large Data Bases, VLDB'1997*, pages 136–145, Athens, Greece, 1997.