An Approach to Specify When Reselecting Views to be Materialized

Henda Ben Ghezala*, Abdelaziz Abdellatif**, Ali Ben Ammar ***

*Ecole Nationale des Sciences de l'Informatique , 2010 Manouba. Tunisie
Henda.bg@cck.rnu.tn
**Faculté des Sciences de Tunis, 2092 Tunis. Tunisie
abdelaziz.abdellatif@fst.rnu.tn
***Institut Supérieur de Documentation, 2010 Manouba. Tunisie
ali.benammar@isd.rnu.tn

Abstract: A data warehouse stores a large volume of data extracted from multiple sources. A set of materialized views is defined over the base tables in order to optimize OLAP (On-Line Analytical Processing) query response time. The selection of materialized views may be static or dynamic. The dynamic selection is continually controlled by a system that calibrates the set of views. The static selection is controlled periodically by the data warehouse administrator who provides the parameter values to the view selection program and defines the selection period. A short period may increase the system workload if there are unnecessary executions of the view selection program. A long period may decrease the query response time. In this paper we propose an algorithm to specify when to select views to be materialized in a static policy. That is when the view selection program should be run. Our main contribution is the use of some tolerance parameters to update and reselect the materialized views. The materialized views will be updated only when it is necessary. The view selection program will be executed either at the end of the selection period, defined by the administrator, or when there is a non tolerated increase of the query execution cost. The aim is to reduce the materialization cost and to guarantee a high query response time. Our experiment results show that, for some values of the tolerance parameters, our approach is more profitable than the static view selection algorithms.

1. Introduction

A data warehouse stores a large volume of data extracted from multiple sources. A set of materialized views is defined over the base tables in order to optimize OLAP (On-Line Analytical Processing) query response time. The selection of these materialized views is based on a cost model that combines, in general, the view maintenance cost and the query execution one. It respects a given limited amount of resources such as materialization time, storage space, or total view maintenance time.

The selection of materialized views may be static [Agrawal et al., 2000][Gupta,1997][Gupta et Mumick, 1999][Gupta et al., 1997][Harinarayan et al., 1996

][Theodoratos et Sellis, 1997] or dynamic [Kim et al., 2003][Kotidis et Roussopoulos, 1999][Theodoratos et Sellis, 1999]. The static selection is controlled periodically by the data warehouse administrator who provides the parameter values to the view selection program. The dynamic selection is controlled by an online system that collects information about workload, user requirements and data updates to select the appropriate views.

In general, a data warehouse is constructed to reach some analysis objectives. Also, it stores old data that are aggregated and rarely updated. So, we think that a dynamic selection is only necessary in some cases like the e-commerce domain and that the static selection is more adequate for a data warehouse environment. In order to accelerate the static selection, some tasks may be automated like the collection of information and the materialized views selection. In this paper we propose an algorithm to specify the moments of materialized view selection in a static policy. That is when the view selection program should be run. The goal is to re-execute the view selection program only when there is an unacceptable decrease of the query response time or when we reach the end of the selection period.

It's known that, in the static policy, the selection period is defined by the data warehouse administrator. A short period may increase the system workload if there are unnecessary executions of the view selection program. This is because there is a small change in the materialization plan (the set of materialized views) or the users are satisfied by the current plan. A long period may decrease the query response time. This is because the current materialization plan has become obsolete and the queries are executed on the base tables. So there is a problem to define the selection period since the materialization plan is selected based on observed values of some parameters.

To resolve this problem, we propose firstly defining a long period and then dividing it into several short periods. As will be explained in section 4, these short periods correspond to the update moments of some materialized views. At the end of each short period, we calculate the query execution cost and then we compare it with that estimated when views are materialized. If there is a high increase along one of the short periods, we re-execute the view selection program else we continue using the current materialization plan until the end of the long period. Our idea is to begin by a long selection period P and then, at predefined moments included in P, checking the performance of the materialization plan. The goal is to reduce the re-materialization cost and to guarantee a good query response time. This is the first contribution of our paper.

The data warehouse maintenance is achieved through two steps. The first step concerns the update of the base tables when there is a change in the data warehouse sources. However the second concerns the transfer of these updates to the materialized views. Several maintenance scenarios and policies are developed [Engström et al., 2003][Engstrom et al.,1999][Teschkle et Ulbrich, 1997][Engström et Lings, 2003][Engström et al., 2002][Chakravarthy et al., 2003] to carry out these two steps. [Engstrom et al.,1999] distinguishes two major maintenance policies: immediate and deferred. We think that the deferred mode is more adequate to a data warehouse for two reasons: i) the data warehouse maintenance is highly complex, which affects the system performance if we realize it immediately; ii) the stored data are used for analysis and so, it is unnecessary to be usually up to date.

In our approach, we use the concept of tolerated update period (TUP) to decide when updating the materialized views. Based on the query text, the data warehouse administrator should specify a tolerated period for the update of the sources. For example, for a query Q extracting the sales per region and per month, the tolerated period is a month. I.e. the update period required to satisfy the users is the month. The role of the TUP, in our approach, is to limit the number of the materialized view updates. That is, we update the materialized views only when it is required by their queries.

The second tolerance parameter, called tolerance fraction and noted θ Є [0,1] is used to check the materialization plan performance. Let PQC be the query execution cost calculated along a short period SPk and EQC be the query execution cost estimated for SPk when views are selected to be materialized. At the end of the short period, if PQC- EQC is greater than θ*EQC then the materialization plan should be changed.

The use of tolerance parameters constitutes the second contribution of this paper. It allows reducing the maintenance and the reselection costs of materialized views. The overall aim is to optimize the query response time without affecting the system workload. How to select views is not the concern of this work. Any selection algorithm can be used in our approach test.

The rest of the paper is organized as follows: section 2 describes the related works. Section 3 gives an overview of the materialized view selection. We present in section 4 our approach and in section 5 some experimental results. Section 6 is the conclusion.

2. Related works

Recently, there has been a lot of interest on how to select views to materialize in a data warehouse and how to update a data warehouse. The view selection task may be static or dynamic. The major part of works belongs to the first class [Agrawal et al., 2000][Gupta,1997][Gupta et Mumick, 1999][Gupta et al., 1997][Harinarayan et al., 1996] [Theodoratos et Sellis, 1997]. The second class [Kim et al., 2003][Kotidis et Roussopoulos, 1999][Theodoratos et Sellis, 1999] concerns particular domains like the e-commerce and is in contrast with the known data warehouse characteristics. The static selection approaches use a fixed time period to reselect the appropriate set of views. However in the dynamic selection the materialization plan is continually controlled to add [Kotidis et Roussopoulos, 1999][Theodoratos et Sellis, 1999] or to change [Kim et al., 2003] views. We are not aware of any work addressing the problem of when modifying the materialization plan in the static selection.

3. Materialized view selection

The data warehouse architecture is represented in figure 1. The data is extracted from different sources, integrated and then stored in the data warehouse. In order to accelerate the data access, a set of query responses are stored. They are called materialized views.

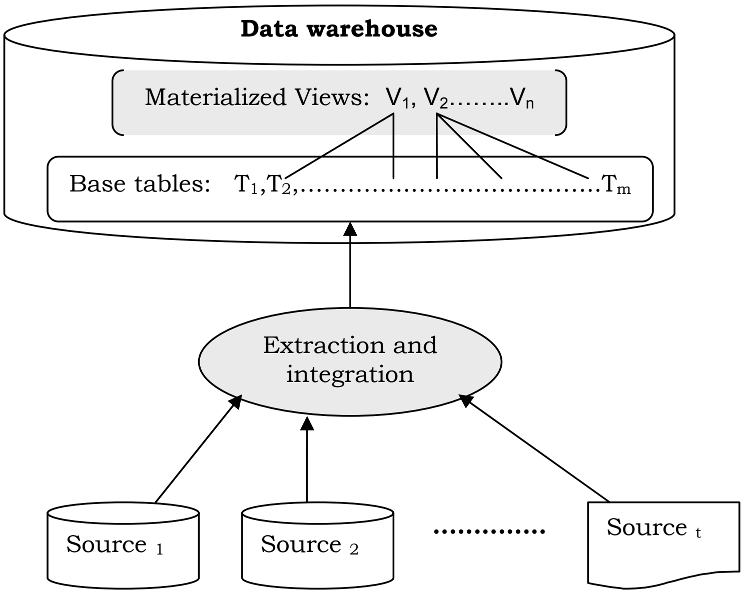
Data warehouse
Materialized Views: V_1, V_2........V_n
Base tables: T_1,T_2,.....................................T_m
Extraction and integration
Source 1
Source 2
....
Source t

FIGURE 1: *Data warehouse architecture*

The view materialization goal is to optimize the query response time. This technique consists in storing complete or partial query results called views. The materialized view management addresses three main tasks: which views to materialize, which update policy to apply and where to store the materialized views.

The first task is called the materialized view selection. It allows identifying a subset of views W from a list of candidate views Q. It's based on a cost model and respects some constraints like space. The subset W is characterized by an optimal materialization cost and by a global size smaller than the reserved space. The cost model includes, in general, the query execution cost and the materialized view update one. The query execution cost depends on some parameters like the access frequency and the data extraction cost. The values of these parameters may be partial that is estimated or complete that is the results of some experiments [Chirkova et al., 2001]. Finally, the selection of the materialization plan may be periodic or dynamic.

In the second task, an update policy determines when and how to update the materialized views. A policy can, for example, be to recompute a view or to do incremental maintenance; and to maintain the view immediately when changes are detected, on-demand when the view is queried; or periodically. The choice of a maintenance policy depends on several parameters such as the system workload, the data size and the required data freshness. The system workload is measured in general by the number and the cost of queries.

The materialized view storage may be costly. So, only the appropriate and the needed views should be materialized. In a distributed architecture, the materialized views may be distributed over several nodes.

4. Specifying when reselecting views to be materialized

4.1 Approach overview

Our goal is to determine when to re-execute the view selection program. Our approach concerns the static materialization where the views are selected periodically. Generally, the selection period is defined by the data warehouse administrator. Along this period he collects some statistics like the access and the update frequencies needed for the view selection program. Since, the views are selected based on these statistics, the choice of the selection period will be a complex decision. Keeping a set of materialized views for long or short time may be not profitable. A short time may increase the cost of view computation; while a long time may increase the query execution cost. Our solution consists in combining both, a long time period and a short one. The data warehouse administrator defines a long period to re-execute the view selection program. This period is divided into several short periods corresponding to the update moments of some materialized views. The short periods are used to check the performance of the materialization plan. If there is a high increase of the total query execution cost along one of the short periods, we should reselect the appropriate set of views else we continue using the current materialization plan until the end of the long period. Two tolerance parameters are used to determine the short periods and to check the query execution cost: the tolerated update period (TUP) and the tolerance fraction. In the rest of this section we will demonstrate how to define and how to use these parameters.

4.2 Specification and use of the tolerated update period

As we have already seen within this paper, that the view update may be immediate or periodic. We have explained, also, that the periodic mode is more adequate in the case of data warehouse. In our approach, we propose to update a materialized view V only when it is required by some queries accessing V. From the query formulation, the administrator determines the TUPs of the query sources. For example, the sources of a query extracting the sales per region and per month must be updated monthly. The TUPs may be calibrated, after, based on the query frequencies or on the user exigencies. That is for the queries which are rarely asked or their result freshness is not important for the users, we can extend theirs TUPs if they are short. The opposite case is also possible also.

Now, let Q1, Q2 and Q3 be three queries accessing a materialized view V ; and let P1, P2, P3 be their corresponding required maintenance periods. So, V should be updated after P1, after P2 and after P3. To simplify the problem, we propose choosing the least period to be the TUP of V. This solution satisfies automatically the two other queries since theirs results will be up to date.

In general, let:

{V1, V2,...,Vm} be the set of materialized views;

{Q1, Q2,…,Qn} be the set of queries and P1,P2,…,Pn be theirs corresponding maintenance periods defined by the administrator;

Ri = {Qj / j=1,…,r and $r \leq n$} be the set of queries accessing Vi;

Si = {Pj / j=1,…,r and $r \leq n$} be the update periods of Ri.

It is obvious that Si concerns automatically Vi. We call the tolerated update period of Vi , the least period of Si and we note TUPi = min {Pj / Pj ϵ Si} . TUPi is usually less than or equal to P, the long period fixed by the administrator to reselect views. It is said tolerance period because it extends the view maintenance period and reduces the number of updates. The short periods of P corresponding to the multiples of TUPi will be used to check the performance of the materialization plan. For example, if we have:

P = 4 months

TUP1, TUP2, TUP3, TUP4, the TUPs of four materialized views V1, V2, V3, V4 are respectively 1, 1.5 ,1 and 2 months

Then the deduced short periods are 1, 1.5, 2 and 3. They correspond respectively to the update of the set of views {V1,V3}, {V2}, {V1,V3,V4} and {V1,V2,V3}. At the end of each short period, we check the query execution cost. If there is a non tolerated increase, we re-execute the view selection program else we update the corresponding views and we continue using the current materialization plan until the end of the next short period or the end of P.

4.3 Specification and use of tolerance fraction

Two kinds of periods are defined above to reselect the appropriate set of views: a long period P and some short periods or sub-periods of P. The idea is to modify the materialization plan only if we detect, at a short period SPk $\subset$ P, a non tolerated increase of the query execution cost or when we reach the end of P. The tolerance fraction is used to measure the tolerated increase of the query execution cost.

In general, the materialization cost includes the update cost of materialized views and the query execution cost. It is used to select the optimal materialization plan. However to check the performance of a materialization plan after a short period SPk, we use only the query execution cost. We eliminate the update cost for two reasons:

- There is no change, before the materialization, in the determined update frequencies that are based on the tolerated periods.
- In general, the view maintenance is carried out at deferred time.

The query execution cost depends on the query frequencies, on the number of news queries and on the execution cost of each query. These parameters which are estimated at the beginning of each P may be changed after the materialization. A high change of these parameters may decrease the query response time.

The following values are needed to check the evolution of the query execution cost:

- The estimated query execution cost by short period:

  $$EQC(SPk) = (SPk/P) * TEQC$$

  Where TEQC = total estimated execution cost of all the queries before the materialization.
- The practical (real) execution cost by short period:

  $$PQC(SPk) = \sum_{i=1,\ldots,n} (fi * PQCi)$$

  Where n = number of all the queries, fi = the query frequency along SPk and PQCi = the real execution cost of the query Qi. fi may be null.
- The tolerance fraction θ ϵ [0,1] specified by the administrator. It is used to determine the tolerated increase of the query execution cost. That is if PQC(SPk) – EQC(SPk) > θ* EQC(SPk) then we must re-execute the view selection program.

The specification of θ depends on some parameters like the view generation cost and the system workload (number and frequencies of queries).

4.4 Problem formulation

Based on the above definitions we can formulate the problem of when reselect the materialization plan as follows:

Given a set of materialized views W={V1, V2,…,Vm} , a set of queries Q={Q1,Q2,…,Qn}, a selection period P, a tolerance fraction θ, searching M={SP1,SP2,…,SPz / SPi <P for i=1,…,z} the set of check sub-periods and then specify the adequate moment from M∪{P} to re-execute the view selection program.

4.5 Algorithm

In our algorithm, described in figure 2, we use a time counter t which is initiated at each re-execution of the algorithm. t, P and the check sub-periods have the same unit of measure (days, weeks, months,…). The algorithm execution begins with searching M the set of short periods which will be used to check the query execution cost. For each value of t there is an action to be carried out:

- For t= P, we reselect the materialized views and we restart the algorithm execution.
- For t∈ M, we verify the query execution cost. If there is a non tolerated increase we reselect the materialized views and we restart the algorithm execution else we increment t and we start a new iteration of the algorithm.
- For t ∉ M and t < P, we increment t and we start a new iteration of the algorithm.

We will not talk about how to select views since it is not the concern of our work and any selection algorithm can be used.

```
1. t=1
2. searching M ={$SP_1$,$SP_2$,…,$SP_z$} the set of sub-periods of P
3. if t= P then go to step 8
4. if t∉ M then go to step 9
5. EQC (t) = (t/P)* TEQC
6. PQC (t) = $\sum_{i=1,\dots,n}$ ($f_i$* $PQC_i$)
7. if PQC(t) –EQC(t) ≤ θ* EQC(t) then go to step 9
8. Re-execute the view selection program. Go to step 1
9. t=t+1 go to step 3
```

FIGURE 2: *An algorithm to specify when reselecting views to be materialized*

4.6 Example

This example demonstrates the algorithm execution. We will use the week as a time measure unit. Suppose we have four queries and three views as follows:

Q1: the sales per region per term (3 months)
Q2: the sales per product by month
Q3: the stored quantity by depot
Q4: the average unit-price per depot per fortnight (2 weeks).
The set of materialized views is W ={V1, V2,V3}
R1 = {Q3, Q4}, the set of queries accessing V1
R2 = {Q4}, the set of queries accessing V2
R3 = {Q1, Q2}, the set of queries accessing V3

4.6.1 Specifying the TUPs

To satisfy the users without increasing the views update cost, the administrator must define the tolerated update periods of the queries sources as follows:

P1 = term. That is the sources of Q1 must be updated each 3 months.
P2 = month. That is the sources of Q2 must be updated monthly.
P3 = fortnight. This is not clear in the text of Q3 but since this later have a common source only with Q4, then its tolerance update period will be automatically 2 weeks.
P4 = fortnight. That is the sources of Q4 should be updated each 2 weeks.
Consequently, the possible TUP values of the views are:
S1 = {P3, P4}, the tolerance update periods corresponding to V1
S2 = {P4}, the tolerance update periods corresponding to V2
S3 = {P1, P2}, the tolerance update periods corresponding to V3
The TUPs of the materialized views are:
V1: TUP1 = min {P3, P4} = fortnight= 2 weeks.
V2: TUP2 = min {P4} =fortnight = 2 weeks.
V3: TUP3 = min {P1, P2} = month = 4 weeks.

4.6.2 Checking the query execution cost

Suppose that:

- The selection period defined by the administrator is P=12 weeks (3 months).
- The estimated execution cost of all the queries TEQC = 150000 units of measure.
- The tolerance fraction θ =0.3

So the short periods SPk of P corresponding to the view update moments and to the check moments are summarized in figure 3. For example, after 4 weeks or 8 weeks we must update V1, V2 and V3.

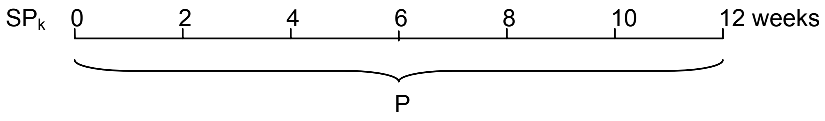

FIGURE 3: *The view update and the check moments*

The estimated (EQC) and the practical (PQC) Query execution costs measured at each check moment are summarized in table 1.

SP_k or t	2	4	6	8	10
EQC(t)	25000	50000	75000	100000	125000
PQC(t)	28000	40000	70000	140000	-

TABLE 1: *The EQC and PQC measured at each check moment SPk.*

The EQC of SP2=4, for example, is obtained as follows:

EQC(SP2) = (4/12)*150000.

After SP4= 8 weeks, we have PQC(SP4) –EQC(SP4) >0.3*EQC(SP4). So, the view selection program should be re-executed to select a new set of materialized views and we restart the algorithm execution.

4.7 Comparison with the dynamic selection

In table 2, we compare the principle of our approach with that of the dynamic selection.

Criterion	*Dynamic selection*	*Our approach*
When checking?	Regular periods	Irregular periods
How specifying the periods of check?	Short	Depends on the tolerated update periods.
Number of checks?	High	Limited
What checking?	The materialization cost (use and update) of a view	The total query execution cost
How modifying the materialization plan?	Partially, by adding or deleting views	Researching a complete new optimal plan.

TABLE 2: *Comparison between our approach principle and that of the dynamic selection.*

5. Experiments

The goal of our experiments was twofold. First, we wanted to study the application of our algorithm for different values of the tolerance fraction θ and to specify the cases when our approach is more efficient. Second, we wanted to compare the query execution costs obtained by two different manners. In the first case, we apply a periodic selection using a known algorithms such as the Greedy Algorithm [Gupta,1997] and the Inverted-Tree Greedy Algorithm [Gupta et Mumick, 1999].In the second case, we apply our approach to specify when re-executing the selection algorithm and we use the same algorithms as the first case to identify the appropriate set of materialized views.

To carry out these experiments, we used a data warehouse, organized in a star schema, with 8 dimensions and a fact table containing more than 2 million tuples. We assumed 20 complex queries to be executed and more than 20 views organized in an AND-OR graph. Our experiments were all run on a Pentium IV 3 GHz running Windows XP with 512 MB RAM.

After each update of some views we calculate the update cost, research the optimal query execution plan and evaluate the query execution cost. Since the update frequency of each view is pre-specified, we only estimated the query frequencies of each sub-period to evaluate the total query execution cost and to apply our algorithm. We don't consider the evaluation cost because in most case it is negligible.

To compare the results of our approach with those of a Greedy Algorithm or of an Inverted-Tree Greedy Algorithm, we used a time period of 6 months when there is an update of some views at the end of each month. So, the evaluation of the query execution cost and the application of the tolerance fraction will be monthly. But, in order to well compare these results we extended our study for 2 periods that is along 12 months.

5.1 Applying the Greedy Algorithm

The Greedy Algorithm [Gupta,1997] allows selecting, periodically, the appropriate set of views that minimizes the total query response time and the cost of maintaining the selected views given a limited space S. In table 3 we present the results of its use and those of its integration in our approach for some values of the tolerance fraction θ.

Value of θ		*Total query execution cost*	*Total intervention cost*	*Number of interventions*	*Benefit*	*Total cost reduction*
0.2	Greedy Algorithm	278175100	125100	1	30700500	11.03%
	Our approach	247026200	573500	4		
0.3	Greedy Algorithm	278175100	125100	1	22084300	7.93%
	Our approach	255863500	352400	3		
0.4	Greedy Algorithm	278175100	125100	1	8392350	3.01%
	Our approach	269679150	228700	2		
0.5	Greedy Algorithm	278175100	125100	1	0	0.00%
	Our approach	278175100	125100	1		

TABLE 3: *Benefits of applying our approach with the Greedy Algorithm.*

In table 3, the number of interventions is at least 1 because if the tolerated cost increase is respected until the end of the first period (P= 6 months), we must research the optimal materialization plan.

The query execution cost measured along each month for θ=0.2, θ=0.3, θ=0.4 are represented respectively in the figure 4, figure 5, figure 6.

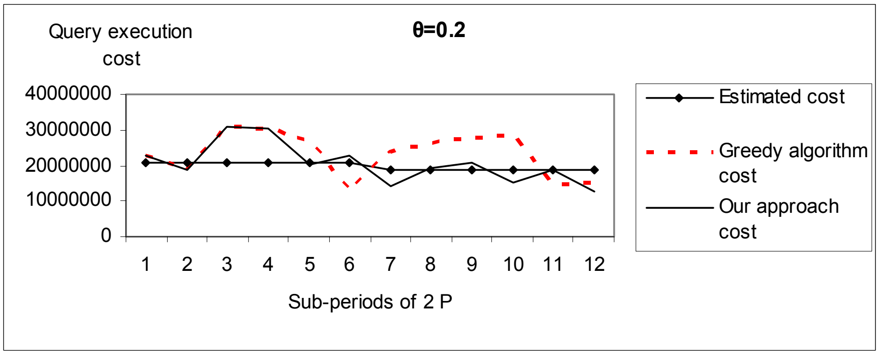

FIGURE 4: *Query execution costs when using the Greedy Algorithm and when θ=0.2*

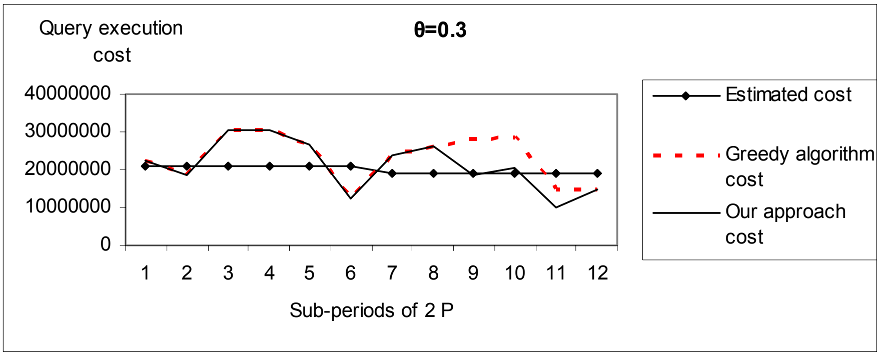

FIGURE 5: *Query execution costs when using the Greedy Algorithm and when θ=0.3*

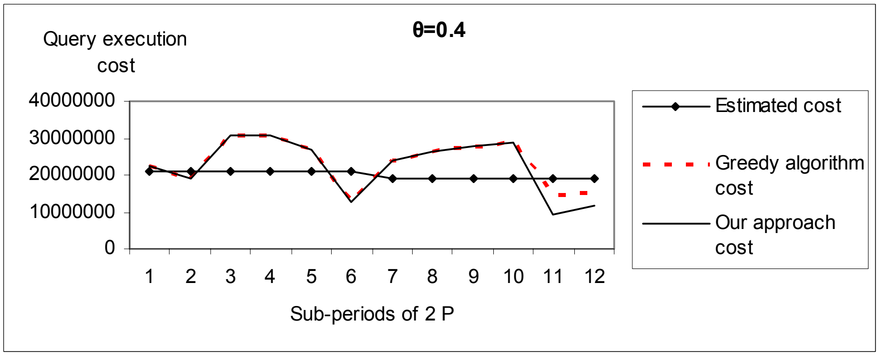

FIGURE 6: *Query execution costs when using the Greedy Algorithm and when θ=0.4*

5.2 Applying the Inverted-Tree Greedy Algorithm

The Inverted-Tree Greedy Algorithm (I.T.G. Algorithm) [Gupta et Mumick, 1999] allows to select, periodically, the appropriate set of views that minimizes the total query response time and the cost of maintaining the selected views given a limited view maintenance time. In table 4 we present the results of its use and those of its integration in our approach for some values of the tolerance fraction θ.

Value of θ		*Total query execution cost*	*Total intervention cost*	*Number of interventions*	*Benefit*	*Total cost reduction*
0.2	I.T.G. Algorithm	242062420	132000	1	19787480	8.17%
	Our approach	222131140	275800	2		
0.3	I.T.G. Algorithm	242062420	132000	1	8303990	3.43%
	Our approach	233674630	215800	2		
0.4	I.T.G. Algorithm	242062420	132000	1	0	0.00%
	Our approach	242062420	132000	1		
0.5	I.T.G. Algorithm	242062420	132000	1	0	0.00%
	Our approach	242062420	132000	1		

TABLE 4: *Benefits of applying our approach with the inverted-tree greedy algorithm.*

The query execution cost measured along each month for θ=0.2, θ=0.3, θ=0.4 are represented respectively in the figure 7, figure 8, figure 9.

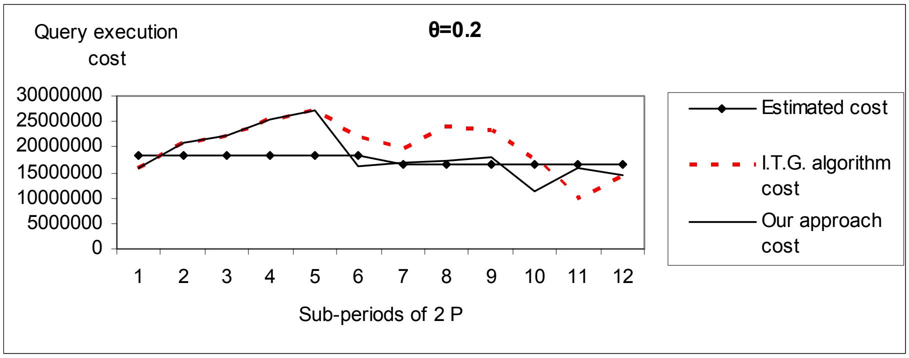

FIGURE 7: *Query execution costs when using the I.T.G Algorithm and when θ=0.2*

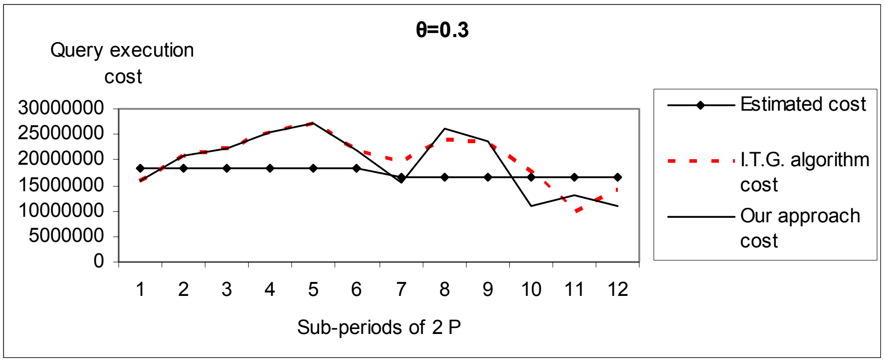

FIGURE 8: *Query execution costs when using the I.T.G Algorithm and when θ=0.3*

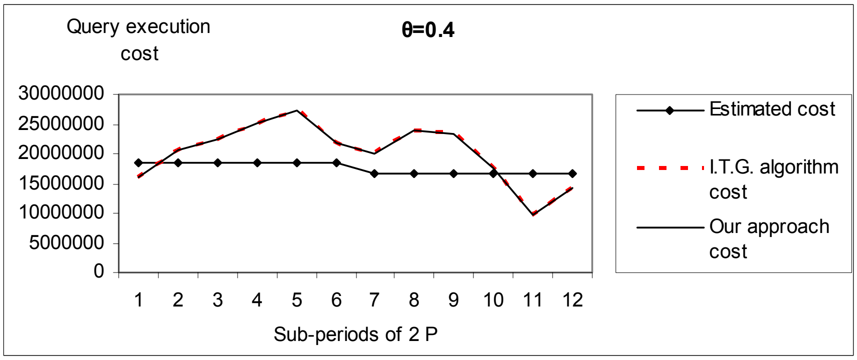

FIGURE 9: *Query execution costs when using the I.T.G Algorithm and when θ=0.4*

5.3 Observations

Based on the above results, we can make the following observations:

- The benefit of applying our algorithm depends, especially, on the used selection algorithm and on the value of the tolerance fraction θ.
- A high benefit is obtained by small values of θ.
- There is no benefit if θ > 0.5
- For θ =0.2, we remark, almost, a stability of the query execution cost.
- Our solution is more adequate for the selection algorithms applying restrictive constraints. That is for the algorithms selecting a small set of materialized views that may be unrapidly usable.

6. Conclusion

In this paper we have proposed an algorithm to specify, in the static policy, when the view selection program should be run. Our solution depends on two tolerance parameters such as the tolerated update period and the tolerance fraction. The aim is to optimize the query response time by reducing the view update cost and by checking, periodically, the performance of the materialization plan. We propose to update the materialized views only when it is needed and to change the materialization plan when there is a non tolerated increase of the query execution cost.

Our experiments results show that, compared to some view selection algorithms which are executed at regular and long periods, our algorithm provide a benefit exceeding 10%. This benefit depends, essentially, on the used view selection program and on the tolerated increase of the query execution cost.

References

[Agrawal et al., 2000] S. Agrawal, S. Chaudhuri, et V. Narasayya. Automated Selection of Materialized Views and Indexes for SQL Databases. In Proc of the 26th International Conference on Very Large Databases, Cairo, Egypt. 2000.

[Chakravarthy et al., 2003] S. Chakravarthy, H. Engström, et B. Lings. Maintenance Policy Selection in Heterogeneous Data Warehouse Environments: A Heuristics-based Approach. International Workshop on Data Warehousing and OLAP (DOLAP), pp 71-78, New Orleans, USA. November 7, 2003.

[Chirkova et al., 2001]R. Chirkova, A. Halevy, et D. Suciu. Formal Perspective on the View Selection Problem. In Proc. VLDB, Roma, Italy, pp. 59-68. 2001.

[Engstrom et al.,1999] H. Engstrom, S. Chakravarthy, et B. Lings. A Rule-Based Adaptive Approach to the Design and Implementation of data Warehouses. In IRMA Workshop, Pittsburgh. May 1999

[Engström et al., 2002] H. Engström, S. Chakravarthy, et B. Lings. A systematic approach to selecting maintenance policies in a data warehouse environment. Advances in Database Technology-EDBT 2002, 8th International Conference on Extending Database Technology, Prague, Czech Republic, March 25-27, Proceedings(C.S. Jensen, K.G. Je.ery, J. Pokorn´y, S. Saltenis, E. Bertino, K. B¨ohm, and M. Jarke, eds.), Lecture Notes in Computer Science, vol. 2287, Springer, 2002, pp. 317 335.

[Engström et al., 2003] H. Engström, S. Chakravarthy, et B. Lings. A heuristic for refresh policy selection in heterogeneous environments. In 19th International Conference on Data Engineering, Bangalore, India, March 5-8, 2003.

[Engström et Lings, 2003] H. Engström et B. Lings. Evaluating Maintenance Policies for Externally Materialised Multi-source Views. In Twentieth British National Conference on Databases (BNCOD), pp. 140-156, Coventry University, Coventry, UK, July 15-17, Proceedings, Springer. 2003.

[Gupta,1997] H. Gupta. Selection of Views to Materialize in a Data Warehouse. ICDT. 1997.

[Gupta et al., 1997] H. Gupta, V. Harinarayan, A. Rajaraman, et J.D.Ullman. Index Selection for OLAP. ICDE. 1997.

[Gupta et Mumick, 1999] H. Gupta et I.S. Mumick. Selection of Views to Materialize Under a Maintenance-Time Constraint. ICDT. 1999.

[Halevy, 2000]A. Y. Halevy. Theory of Answering Queries Using Views. SIGMOD. 2000

[Harinarayan et al., 1996]V. Harinarayan, A. Rajaraman, et J.D. Ullman. Implementing Data Cubes Efficiently. ACM SIGMOD. 1996.

[Kim et al., 2003] H. Kim, T. Lee, S. L, et J. Chu. Automated Data Warehousing for Rule-based CRM Systems. Fourteenth Australasian Database Conference (ADC2003), Adelaide, Australia. 2003

[Kotidis et Roussopoulos, 1999] Y. Kotidis et N. Roussopoulos. DynaMat: A Dynamic View Management System for Data Warehouses. ACM SIGMOD. 1999.

[Srivastava et Rotem, 1988] J. Srivastava et D. Rotem. Analytical Modeling of Materialized View Maintenance. ACM PODS. 1988

[Teschkle et Ulbrich, 1997] M. Teschkle et A. Ulbrich. Using Materialized Views To Speed Up Data Warehousing. Technical Report, IMMD 6, University of Erlangen-Nürnberg, 1997

[Theodoratos et Sellis, 1997] D. Theodoratos et T. Sellis. Data Warehouse Configuration. Proceedings of the 23rd VLDB Conference, Athens, Greece. 1997

[Theodoratos et Sellis, 1999] D. Theodoratos et T. Sellis. Dynamic Data Warehouse Design. DaWaK, 1—10. 1999

Value of θ		*Total query execution cost*	*Total intervention cost*	*Number of interventions*	*Benefit*	*Total cost reduction*
0.2	Greedy Algorithm	278175100	125100	1	30700500	11.03%
	Our approach	247026200	573500	4		
0.3	Greedy Algorithm	278175100	125100	1	22084300	7.93%
	Our approach	255863500	352400	3		
0.4	Greedy Algorithm	278175100	125100	1	8392350	3.01%
	Our approach	269679150	228700	2		
0.5	Greedy Algorithm	278175100	125100	1	0	0.00%
	Our approach	278175100	125100	1		

TABLE 3: *Benefits of applying our approach with the Greedy Algorithm.*

In table 3, the number of interventions is at least 1 because if the tolerated cost increase is respected until the end of the first period (P= 6 months), we must research the optimal materialization plan.

The query execution cost measured along each month for θ=0.2, θ=0.3, θ=0.4 are represented respectively in the figure 4, figure 5, figure 6.

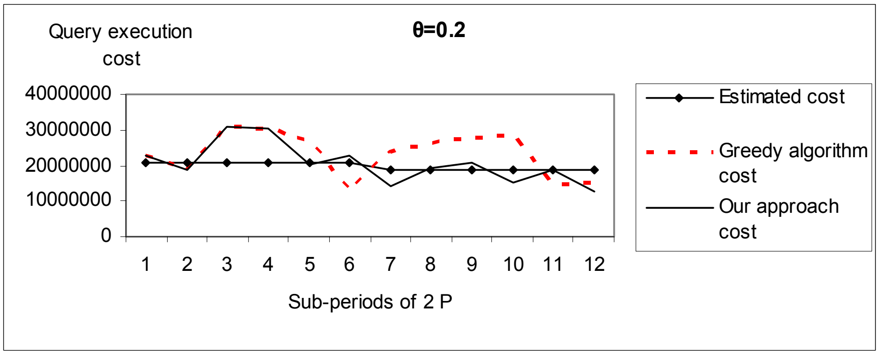

FIGURE 4: *Query execution costs when using the Greedy Algorithm and when θ=0.2*

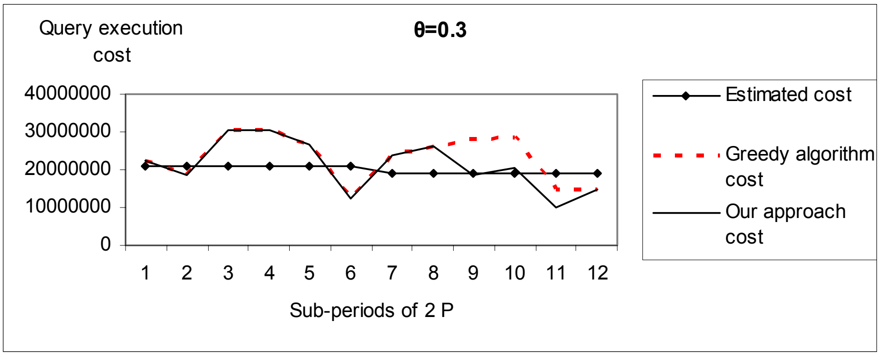

FIGURE 5*: Query execution costs when using the Greedy Algorithm and when θ=0.3*

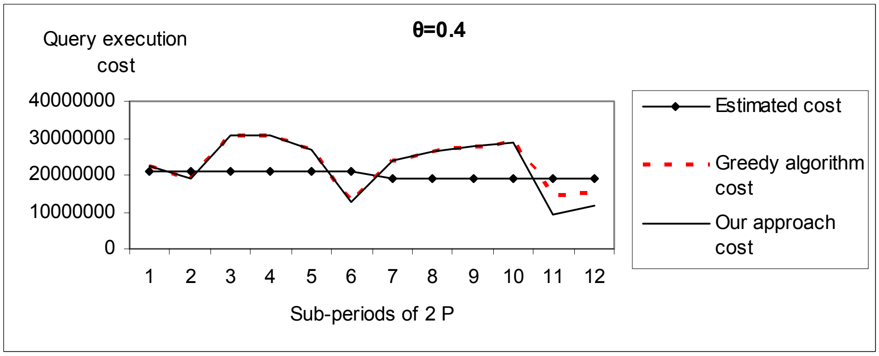

FIGURE 6: *Query execution costs when using the Greedy Algorithm and when θ=0.4*

5.2 Applying the Inverted-Tree Greedy Algorithm

The Inverted-Tree Greedy Algorithm (I.T.G. Algorithm) [Gupta et Mumick, 1999] allows to select, periodically, the appropriate set of views that minimizes the total query response time and the cost of maintaining the selected views given a limited view maintenance time. In table 4 we present the results of its use and those of its integration in our approach for some values of the tolerance fraction θ.

Value of θ		*Total query execution cost*	*Total intervention cost*	*Number of interventions*	*Benefit*	*Total cost reduction*
0.2	I.T.G. Algorithm	242062420	132000	1	19787480	8.17%
	Our approach	222131140	275800	2		
0.3	I.T.G. Algorithm	242062420	132000	1	8303990	3.43%
	Our approach	233674630	215800	2		
0.4	I.T.G. Algorithm	242062420	132000	1	0	0.00%
	Our approach	242062420	132000	1		
0.5	I.T.G. Algorithm	242062420	132000	1	0	0.00%
	Our approach	242062420	132000	1		

TABLE 4: *Benefits of applying our approach with the inverted-tree greedy algorithm.*

The query execution cost measured along each month for θ=0.2, θ=0.3, θ=0.4 are represented respectively in the figure 7, figure 8, figure 9.

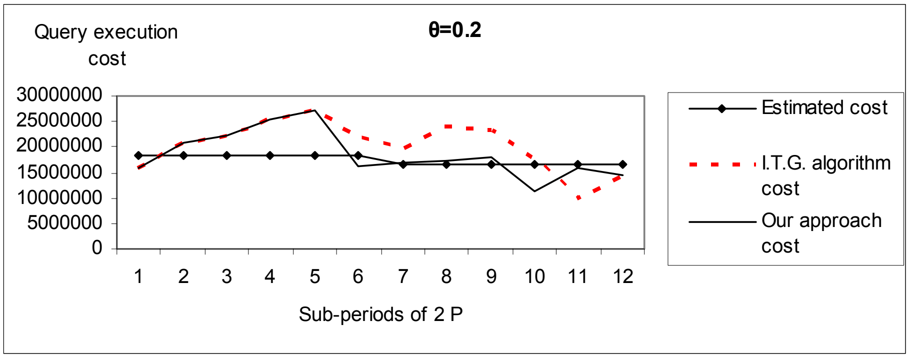

FIGURE 7: *Query execution costs when using the I.T.G Algorithm and when θ=0.2*

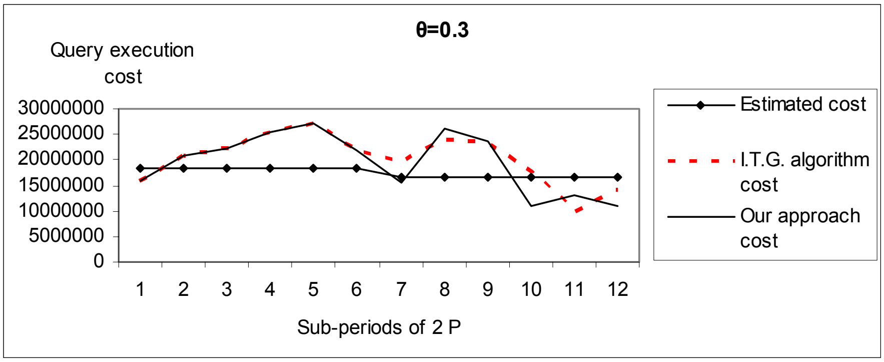

FIGURE 8: *Query execution costs when using the I.T.G Algorithm and when θ=0.3*

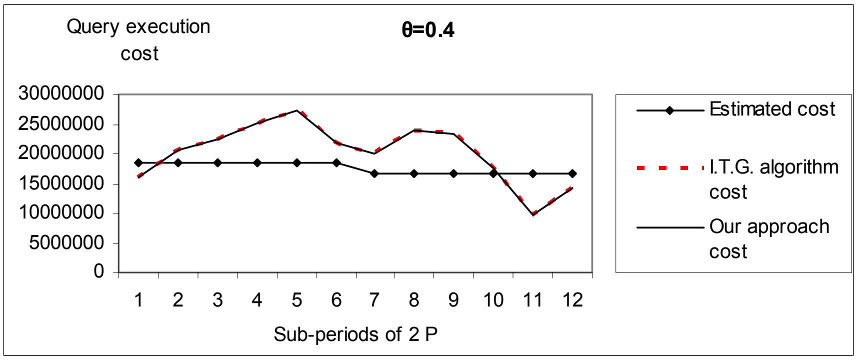

FIGURE 9: *Query execution costs when using the I.T.G Algorithm and when θ=0.4*

5.3 Observations

Based on the above results, we can make the following observations:

- The benefit of applying our algorithm depends, especially, on the used selection algorithm and on the value of the tolerance fraction θ.
- A high benefit is obtained by small values of θ.
- There is no benefit if θ > 0.5
- For θ =0.2, we remark, almost, a stability of the query execution cost.
- Our solution is more adequate for the selection algorithms applying restrictive constraints. That is for the algorithms selecting a small set of materialized views that may be unrapidly usable.

6. Conclusion

In this paper we have proposed an algorithm to specify, in the static policy, when the view selection program should be run. Our solution depends on two tolerance parameters such as the tolerated update period and the tolerance fraction. The aim is to optimize the query response time by reducing the view update cost and by checking, periodically, the performance of the materialization plan. We propose to update the materialized views only when it is needed and to change the materialization plan when there is a non tolerated increase of the query execution cost.

Our experiments results show that, compared to some view selection algorithms which are executed at regular and long periods, our algorithm provide a benefit exceeding 10%. This benefit depends, essentially, on the used view selection program and on the tolerated increase of the query execution cost.

References

[Agrawal et al., 2000] S. Agrawal, S. Chaudhuri, et V. Narasayya. Automated Selection of Materialized Views and Indexes for SQL Databases. In Proc of the 26th International Conference on Very Large Databases, Cairo, Egypt. 2000.

[Chakravarthy et al., 2003] S. Chakravarthy, H. Engström, et B. Lings. Maintenance Policy Selection in Heterogeneous Data Warehouse Environments: A Heuristics-based Approach. International Workshop on Data Warehousing and OLAP (DOLAP), pp 71-78, New Orleans, USA. November 7, 2003.

[Chirkova et al., 2001]R. Chirkova, A. Halevy, et D. Suciu. Formal Perspective on the View Selection Problem. In Proc. VLDB, Roma, Italy, pp. 59-68. 2001.

[Engstrom et al.,1999] H. Engstrom, S. Chakravarthy, et B. Lings. A Rule-Based Adaptive Approach to the Design and Implementation of data Warehouses. In IRMA Workshop, Pittsburgh. May 1999

[Engström et al., 2002] H. Engström, S. Chakravarthy, et B. Lings. A systematic approach to selecting maintenance policies in a data warehouse environment. Advances in Database Technology-EDBT 2002, 8th International Conference on Extending Database Technology, Prague, Czech Republic, March 25-27, Proceedings(C.S. Jensen, K.G. Je.ery, J. Pokorn´y, S. Saltenis, E. Bertino, K. B¨ohm, and M. Jarke, eds.), Lecture Notes in Computer Science, vol. 2287, Springer, 2002, pp. 317 335.

[Engström et al., 2003] H. Engström, S. Chakravarthy, et B. Lings. A heuristic for refresh policy selection in heterogeneous environments. In 19th International Conference on Data Engineering, Bangalore, India, March 5-8, 2003.

[Engström et Lings, 2003] H. Engström et B. Lings. Evaluating Maintenance Policies for Externally Materialised Multi-source Views. In Twentieth British National Conference on Databases (BNCOD), pp. 140-156, Coventry University, Coventry, UK, July 15-17, Proceedings, Springer. 2003.

[Gupta,1997] H. Gupta. Selection of Views to Materialize in a Data Warehouse. ICDT. 1997.

[Gupta et al., 1997] H. Gupta, V. Harinarayan, A. Rajaraman, et J.D.Ullman. Index Selection for OLAP. ICDE. 1997.

[Gupta et Mumick, 1999] H. Gupta et I.S. Mumick. Selection of Views to Materialize Under a Maintenance-Time Constraint. ICDT. 1999.

[Halevy, 2000]A. Y. Halevy. Theory of Answering Queries Using Views. SIGMOD. 2000

[Harinarayan et al., 1996]V. Harinarayan, A. Rajaraman, et J.D. Ullman. Implementing Data Cubes Efficiently. ACM SIGMOD. 1996.

[Kim et al., 2003] H. Kim, T. Lee, S. L, et J. Chu. Automated Data Warehousing for Rule-based CRM Systems. Fourteenth Australasian Database Conference (ADC2003), Adelaide, Australia. 2003

[Kotidis et Roussopoulos, 1999] Y. Kotidis et N. Roussopoulos. DynaMat: A Dynamic View Management System for Data Warehouses. ACM SIGMOD. 1999.

[Srivastava et Rotem, 1988] J. Srivastava et D. Rotem. Analytical Modeling of Materialized View Maintenance. ACM PODS. 1988

[Teschkle et Ulbrich, 1997] M. Teschkle et A. Ulbrich. Using Materialized Views To Speed Up Data Warehousing. Technical Report, IMMD 6, University of Erlangen-Nürnberg, 1997

[Theodoratos et Sellis, 1997] D. Theodoratos et T. Sellis. Data Warehouse Configuration. Proceedings of the 23rd VLDB Conference, Athens, Greece. 1997

[Theodoratos et Sellis, 1999] D. Theodoratos et T. Sellis. Dynamic Data Warehouse Design. DaWaK, 1—10. 1999