View maintenance policy using data aging in data warehousing

Hussien H. Aly, Yousry Taha and Mohamed A. Mohamed
Dept. of Computers and Automatic Control, Faculty of Eng., Alexandria University, Alexandria, Egypt

Materialized views and view maintenance in Data Warehouses (DW) are becoming increasingly important in practice. Selecting the most convenient time to carry on view maintenance is a very crucial issue. In this paper we propose a new view maintenance policy that allows the DW user to define the data freshness requirements. The aging of the data in the DW is controlled using SQL-like user-defined Data Aging Constraints (DACs) that are associated with views in the DW. As soon as a DAC is violated, the view maintenance process takes place to improve view freshness. Given the distributed nature of DW, we propose a distributed evaluation scheme for the DACs that divides the burden of tracking the data aging between the sources and the DW. DACs are decomposed into propagation rules, each of which depends on a single source so that they can be tested locally. A new proposed component, called DW-Agent, running on the source, will be in charge of tracking the changes in the source. The agent collects the changes from the monitor, evaluates the propagation rule, and notifies the DW manager only if the rule is violated. The performance of the new method is evaluated using simulation. The proposed policy indicates a good performance in terms of view maintenance cost, communication cost, with high probability of reading up-to date data by the DW users.

في السنوات الأخيرة زادت أهمية تطبيقات مستودعات البيانات التي يتم بناؤها بغرض تحليل البيانات والحصول على معلومات تحقق أهداف أي مؤسسة، وتعتبر المحتويات المادية جزء أساسي في أي مستودع بيانات، ويتم بناء هذه المحتويات المادية على بيانات مصادر البيانات المختلفة وبهذا فإن هذه المحتويات المادية هي تكرار لبيانات هذه المصادر، واختيار التوقيت المناسب لتحديث وصيانة هذه المحتويات يؤثر على أداء مستودع البيانات بشكل كبير وملحوظ. وتوجد العديد من السياسات التقليدية لاختيار توقيت صيانة هذه المحتويات المادة مثل: الصيانة الفورية والتي تتم عقب أي تعديل في بيانات المصدر و الصيانة الأجلة والتي تتم عند كل استعلام في مستودع البيانات والصيانة الدورية والتي تتم بعد مرور زمن معيّن. وفي كل من السياسات السابقة تكون هناك موازنة بين أداء نظام مستودع البيانات وبين مقدار حداثة البيانات ومواكبتها للتعديلات التي تتم في مصادر البيانات. في هذا البحث نقوم باقتراح سياسة جديدة لصيانة المحتويات المادية، هذه السياسة تعتمد على فرض قيود على تقادم البيانات المخزنة في مستودعات البيانات بحيث أن تقادم هذه البيانات لا يتعدى حد معين يتم تحديده من قبل مستخدمي مستودع البيانات وعند تعدي هذا الحد يقوم النظام بعمل صيانة لتحديث بيانات المحتويات المادية لاستعادة حداثتها. ومفهوم تقادم البيانات الذي يتم تقديمه في هذا البحث لا يعتمد على مبدأ التقادم الزمني والذي فيه تتقادم البيانات بمرور الزمن لأنه في أغلب الأحيان إذا لم يتم تغيير في هذه البيانات فإنها تظل حديثة ولا تعتبر متقادمة، وعليه فإن تقادم البيانات في المحتويات المادية مرتبط بالتعديلات التي تتم على مصادر البيانات التي تم بناء المحتويات المادية عليها. ونقوم في هذا البحث بعرض السياسة المقترحة ووضع إطار العمل بها مع شرح كيفية عملها وكيفية التعبير عن قيود تقادم البيانات بطريقة بسيطة تستخدم لغة الاستعلام الهيكلية (SQL). هذا إلى جانب اقتراح نموذج لتنفيذ السياسة المقترحة لضمان كفاءة متابعة قيود تقادم البيانات، وتعتمد طريقة التنفيذ على تقسيم قيد تقادم البيانات إلى عدة قيود أصغر كل منها يحتوي على شرط خاص بمصدر بيانات معين ويتم توزيع هذه القيود الأصغر على مصادر البيانات المختلفة مع وجود نائب لمستودع البيانات في كل مصدر بحيث يقوم بمراقبة هذه القيود في كل مصدر ولا يقوم بإرسال التعديلات التي تتم في المصدر إلا إذا خالفت التعديلات هذا القيد ففي هذه الحالة يقوم نائب مستودع البيانات الموجود في المصدر بإرسال هذه التعديلات إلى مستودع البيانات ليقوم بعملية التحديث. والسياسة الجديدة تتيح لمستخدمي مستودع البيانات معرفة مستوى حداثة البيانات التي يتم التعامل معها إلى جانب التحكم في هذه الحداثة عن طريق فرض قيود تقادم البيانات على المحتويات المادية الموجودة في مستودع البيانات. وقد أثبتت تجارب المحاكاة كفاءة السياسة الجديدة بمقارنتها مع السياسات الموجودة حاليا من حيث الحفاظ على حداثة البيانات واستهلاك موارد مستودع البيانات في عملية صيانة المحتويات المادية وتقليل تكلفة الاتصال بين مصادر البيانات ومستودع البيانات.

Keywords: Data warehouse, Data aging, View maintenance

1. Introduction

There are three common policies, which deal with view maintenance timing [1,2]:

1- *Immediate maintenance*: where the view is maintained immediately upon an update to one of its base tables.

2- *Deferred (on-demand) maintenance*: where the view is maintained at the time of view query.
3- *Periodical maintenance*: where the view is maintained periodically, e.g., every an hour, once a day or once a week.

In the immediate and deferred policies, a DW query read up-to-date data. However, the response time and the communication cost are usually unacceptable. In the periodical scheme, queries may read data that is not up-to-date with base tables during the periods between successive refreshments. The level of data freshness is not well defined between successive refreshments. However, this policy is the most common used in current commercial data warehousing environment because of simplicity and the view maintenance can be done in times of low workload [3].

If we use the notion of change-based age instead of time-based age, some critical OLAP and DSS application may be useless unless when using the periodical scheme. This is because there is no guarantee of how much the data in the DW differs from the actual operational data in the sources. For example, assume that some OLAP are required or a decision should be taken as soon as the total sales in the regional offices exceed $100,000. If the view maintenance periodically is done every week, it may be too late decision. Also, if sales transactions during the week are small, there is no need to maintain the view. On the other hand, maintaining the view every day is considered costly.

In this paper, we consider the problem of deciding when to do the view maintenance. We consider allowing the user to specify how much difference between the data in the DW and the operational data is allowed before a view maintenance operation is carried out.

The rest of the paper is organized as follows: Section 2 presents the data aging constraint framework. Section 3 introduces the proposed view maintenance policy. Section 4 presents the propagation rules derivation method. Section 5 presents the simulation results of the experiments conducted to compare the proposed policy to the existing ones. Section 6 concludes the paper.

2. Data aging constraint framework

Time-based data freshness is discussed in [4]. In our research, we adopt the notion of change-based age instead of time-based age, where the age of the derived data increases only due to base data updates [5]. A data aging constraint (DAC) is a predicate that can be defined for each data warehouse view in terms of the view and the underlying sources data values. This predicate can be described as an SQL-like expression as follows:

CREATE DAC ON <*DW Object w*>
REFRESH WHEN EXISTS<*SQL SELECTquery*>

Where *w* is any materialized view defined in the data warehouse. The REFRESH clause queries are of SPJ (select, projection, and join) class that may contain aggregates.

The constraint is *violated* if the query is evaluated to a non-empty set. In this case, it denotes stale data. Materialized view is considered valid as long as this aging constraint is not violated by the updates issued against base data.

Example: Suppose that there are two relations **WRS** and **ERS** that belong to different data sources S_1 and S_2 respectively. **WRS** represents the sales transactions at the western region, and **ERS** represents the sales transactions at the eastern region. Suppose that both of them have the same structure:

S_1.WRS (part_no, quantity, sales_value)
S_2.ERS (part_no, quantity, sales_value)

Suppose the following view is defined over these base relations and materialized at the head office DW.

```
CREATE VIEW Total_Part_Sales (part_no,
part_sales_value) AS
 SELECT WRS.part_no, SUM(WRS.sales_value
+ ERS.sales_value) AS part_sales_value
 FROM WRS, ERS
 WHERE WRS.part_no = ERS.part_no
 GROUP BY WRS.part_no
```

This view represents the total sales achieved for common parts at the western

and eastern regions. Suppose that this view is represented at the DW as w. The DW users want to refresh w every time the total current sales for common parts at western and eastern regions diverge from total sales recorded at w by $10,000. This can be expressed using the following DAC.

```
CREATE DAC ON w
REFRESH
WHEN EXISTS (SELECT abs (W.total –
Source.total) FROM (SELECT Sum
(part_total) AS total FROM
        (SELECT WRS.part_no,
        um(WRS.sales_value +
        ERS.sales_value) AS part_total
        FROM WRS, ERS WHERE
        WRS.part_no = ERS.Part_no GROUP
        BY WRS.part_no)) Source,
      (SELECT Sum (part_sales_value) AS
      total FROM Total_Part_Sales) W
HAVING abs (W.total – Source.total) > 10000)
```

This constraint is violated if there are tuples resulting from the execution of the query at the REFRESH clause indicating that total new sales, not reflected at DW, exceeds $10,000.

3. Data aging view maintenance policy

To track the data aging constraints, the query results of the DAC REFRESH clause should be checked against every change at all involved sources. Moreover, the computation of the DAC query implies the computation of a distributed query, which is usually costly. To minimize the overhead of DAC checking, we divide the burden of tracking the DAC between the DW and the data sources. The idea is to break down the DAC query that involves multiple data sources into smaller queries that involves only one data source. These smaller queries are called *propagation rules* (PR) and their semantics is very similar to the DAC. Each PR is checked locally at each data source by a *DW agent*. As long as the PR query does not result in any tuples, the PR is still valid and there is no need to propagate source changes to the DW.

3.1. Data warehouse architecture

It is assumed that the DW uses monitors at the sources as a way to detect source changes [2, 6, 7]. These monitors are supposed to detect only relevant changes to the DW materialized views. Fig. 1 depicts the major components of a data warehouse that uses the data aging constraint policy. The main relevant components are the view specifier, warehouse agent and warehouse manager. We omit other components in the DW architecture that are irrelevant to the aging constraint policy for the sake of simplicity, such as wrappers, query processors, and view managers.

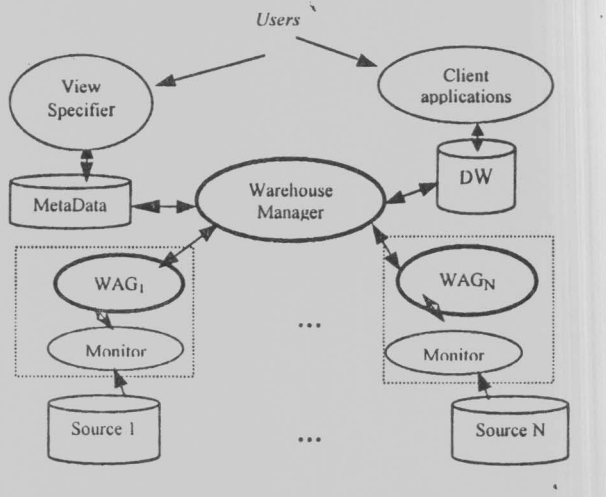

Fig. 1. The proposed data warehouse system architecture.

View specifier is a front-end tool for the DW user or the DW administrator. Through the view specifier, the user can define in a declarative way the desired data aging constraint to be imposed on any DW object. The user may enter the DAC with the syntax mentioned before or through a user-friendly interface like a DAC builder that formulates the entered DAC into a query as described before.

Warehouse manager (WHM) is the main component of the DW architecture. Some literature refers to it as the *integrator*. This name was originated from its responsibility for integrating source updates into the DW materialized views during the view maintenance operation. However, we prefer to call it the warehouse manager since its

responsibilities exceed view maintenance actions.

The tasks of the WHM can be summarized as follows:

1. Computing the materialized views from the underlying sources. For the first time, this computation starts from scratch since there is no prior available information can be utilized.
2. Deriving the propagation rules from the aging constraint imposed on any DW object. The WHM forwards each propagation rule to the corresponding data source.
3. Receiving status reports sent by the warehouse agents about the propagation rules violation at the data sources.
4. Carrying on view maintenance for the defined DW objects.

Warehouse agent (WAG) is a new component added to the typical data warehouse architecture in order to help tracking aging constraints. The warehouse agent is added at the source side beside the source monitor. The main objective of the warehouse agent is to lessen the communication traffic between the data sources and the data warehouse during aging constraint tracking. The agent uses the propagation rule assigned to it to conduct a local test on the new source changes. As long as the propagation rule condition is not violated, the test is successful and there is no need to propagate this change to the warehouse. As soon as the propagation rule condition is violated, the test fails and source changes are forwarded to the warehouse.

3.2. DW management and agents relationship

As data warehouse users define new DW objects, they impose data aging constraints on them. The WHM receives these constraints, analyze them, and generate the corresponding PRs. Each PR is forwarded to the agent in corresponding data source to be tracked locally. This section outlines the scenario of tracking aging. Actually, the scheme is working in a distributed environment where the actions are divided between the data warehouse and the agents in the data sources. Therefore, it is convenient to describe such simultaneous actions by separating the perspective of each side and discussing it individually. The following two perspectives present the aging constraints tracking from the point of view of the DW and the data source.

The data warehouse perspective The warehouse manager (WHM) represents the data warehouse side in this scheme. The following steps describe the actions taken by the WHM to track any aging constraint using the proposed policy.

- First, the WHM receives the view definition and its associated aging constraint from the metadata store. The WHM uses derivation method, presented in the next section, to derive the propagation rules from the aging constraint.
- The WHM queries data sources and computes the materialized view v from scratch for the first time only since there is no prior available information.
- The WHM sends each propagation rule to the corresponding data source agent and marks the view v as a valid and up-to-date view.
- The WHM waits until one of the agents claims that its propagation rule has been violated. At this point, the WHM sends a FLUSH signal for other agents involved in the view definition to send their buffered modifications.
- The WHM may decide to carry on view maintenance for the DW object v upon the first propagation rule violation or may defer it. After maintaining the view, v restores its data freshness. Then it is marked as valid and up-to-date.
- The same cycle is repeated.

The data source perspective. The warehouse agent (WAG) represents the data source side in this perspective.

- First, the WAG receives the PR definition from the WHM.
- As the monitor at this source detects new changes, the WAG validates the PR condition against these changes. Since the PR condition is represented by a query, the WAG executes this query over source changes. The PR is said to be valid if its query returns an empty set of tuples. At this time, there is no need to propagate these changes to the data warehouse.

- As soon as the PR query returns a non-empty set, the WAG notifies the WHM with the violation of its PR and forwards all stored changes. The agent restarts buffering new changes
- If the agent receives a FLUSH signal from the WHM, it sends all source changes and restarts the checking process. This case occurs when one or more of the other source agents notify the WHM with their PR violation.

4. Propagation rules derivation method

A propagation rule (PR) is a predicate derived from the data aging constraint (DAC) predicate. The PR is intended to be tested locally at the data source. A predicate is said to be *local* if all the attributes referenced in the predicate are from the base relations residing on the same data source.

Effectively, the DAC query is decomposed to a number of subqueries or PRs equals to the number of data sources involved in the DAC definition. We will use the following notation to define a PR:

PROPAGATION RULE *<PR name>* **ON***<Source>* **FORWARD WHEN EXISTS** *<PR condition expressed as an SQL-Select query>*

The semantics of the above rule is similar to the DAC. The rule condition is represented by a select SQL query that results in a non-empty set if it is violated. If the query does not return any tuples, the PR still holds and there is no need to forward source changes.
To formally describe the derivation method, we need some definitions.

A *Data warehouse object* is any materialized view defined at the DW. The *Generalized projection operator* (denoted as π_A) is used to represent aggregation attributes in the projected attributes [8, 9]. This is an extension to the distinct projection, where the projected attribute set A may include aggregate functions as well as regular attributes. We use GB (A) to denote the set of group-by attributes in A. For example, we can write $\pi_{d, MAX(s)}$ (R) as the query:

SELECT d, MAX (s) FROM R GROUP BY d

In this example, A = {d, MAX (s)} and GB (A) = {d}.

The *Contribution factor* (CF) is a way to describe the share of a data source element in a DW object value. They are used to derive propagation rules for each source from the DAC. Any contribution factor takes a value between [0, 1] and the sum of all contribution factors involved should be 1. So $CF_{x,w}$ denotes the contribution factor of the data source element x on the DW object w.

Contribution factors can be specified on a low level of granularity like the level of relation columns or a higher level like a whole data source relation. In our research, we consider contribution factors on the whole data source relations.

4.1. Aggregate functions types

In [10], aggregate functions are divided into three classes: *distributive*, *algebraic*, and *holistic*. Distributive aggregate functions can be computed by partitioning their input into disjoint sets, aggregating each set individually, and then further aggregating the partial results from each set into the final result. Amongst the distributive aggregate functions found in standard SQL are COUNT, SUM, MIN, and MAX.

Algebraic aggregate functions can be expressed as a scalar function of distributive aggregate functions. Average is an algebraic aggregate function since it can be expressed as SUM/COUNT.

Holistic aggregate functions cannot be computed by dividing into parts. Median is an example of a holistic aggregate function.

A function $f(x_1,\ldots,x_n)$ of n independent variables is *separable* if it can be expressed as the sum of n single-variable functions $f_1(x_1), \ldots, f_n(x_n)$, that is,

$$f(x_1,x_2,\ldots,x_n) = f_1(x_1) + f_2(x_2) + \ldots + f_n(x_n).$$

The same concept can be applied on conjunctive predicates that may appear in DAC. We deal with source relation columns as the variables. The objective of this property is to be able to decompose the predicate appear in the DAC conditions into smaller

predicates that involve only variables from each individual source.

4.2. Assumptions and restrictions

Our proposed scheme for view maintenance using data aging depends on some assumptions and restrictions that must hold to ensure correct operation.

1-Any change occurs at the source is available for the WAG at this source as reported by the monitor.
2-The WAG receives the changes from the monitor with a notification indicating the class of this change (insert, delete, or update).
3-It is the responsibility of the monitor to detect the source relevant modifications.
4- Aggregate functions used in the DAC expression are either distributive or algebraic.
5-The DAC REFRESH clause is conjunctive.
6-The DISTINCT keyword cannot be used in the DAC expression, e.g., SUM (DISTINCT sales_amt) is not permitted.
7-All functions and expressions used in the DAC should be separable. For example, the use of multiplication and division between involved source variables is restricted, e.g., SUM (X*Y) is not allowed.

4.3. Deriving the propagation rule

Throughout the rest of this paper, the following conventions will be used:

- S is the data source (a set of relations) we derive the PR expression for,
- Z, Y are simple relations,
- a, b are simple attribute,
- w is the DW object intended by the DAC,
- $CF_{S,w}$ is the contribution factor of the source S to the values of the object w,
- p is a predicate containing conditions on regular attributes and/or aggregated attributes,
- A is the any set of attributes.

The syntax of A and p are described by the following simple grammar shown in the table 1 below. The following algorithm shows the steps used to derive the PR expression for each involved data source from any DAC definition. This algorithm is computed at compile-time when the user defines the DAC. The algorithm steps refer to the rules appear in table 2.

Table 1.
Selection predicate and projected attributes grammar

1	*P*	=	*Item₁ compare Item₂* |
2			*p₁* AND *p₂*
3	*Item*	=	ColumnName |
4			constant |
5			*Agg-Fn (Item)* |
6			*Fn (Item₁ , Item₂)* |
7			*WarehouseCalculatedItem*
8	*Compare*	=	= | != | < | > | <= | >=
9	*Agg-Fn*	=	Sum | min | max | avg | count
D	*Fn*	=	+ | - | any separable user-defined function
11	*A*	=	*Item₁ , Item₂* |
12			*Item*
B	*WarehouseCalculatedItem*	=	*Agg-Fn (w.ColumnName)*

Table 2
Rules to derive PR expression from DAC expression

Rule	Construct in the DAC expression	Construct in the PR expression
1	π_A (Z)	(i) π_A (S) ,when Z ∈ S (ii) $CF_{S,w}$. π_A(w) ,when Z = w and A on the form of WarehouseCalculatedItem appearing in production (7) in fig. 4-1. (iii) Nil ,otherwise
2	σ_p (Z)	(i) $\sigma_{p'}$ (Z) , when Z ∈ S, where p' like p but multiplying all items on the form mentioned in productions (4) and (7) by $CF_{S,w}$ (ii) Nil, when Z does not belong to S
3	Z × Y Z $\infty_{Z.a=Y.b}$ Y	(i) Z × Y or Z $\infty_{Z.a=Y.b}$ Y , when Z,Y ∈ S (ii) Z, when Z ∈ S and Y ∉ S (iii) Nil, when Z, Y ∉ S
4	Agg-Fn (Item)	If the Item (refer to production 3 in Fig. 4-1) is a column belongs to a removed relation, replace Agg-Fn(Item) with NIL. Otherwise, if Item is a column in a relation that belongs to S, put Agg-Fn(Item) as it is in the PR expression.

Algorithm 4.1 *Generate propagation rules*
Input: *(Data Aging Constraint (DAC) definition, S data source to derive its PR)*
Result: *(The propagation rule for the data source S)*
Method:
1. Initialization
- Construct the parse tree of the DAC expression (in its relational form) mentioned at the REFRESH clause.
- Let PR parse tree equal to DAC parse tree.

2. Traverse the PR tree using postorder scheme. FOR EACH parent node (operator nodes) in the parse tree REPEAT
3. Apply one of the above rules (for the source S) on the parent node and its children.
4. Remove the nodes according to the rules.
5. If there is a binary operator, e.g., × (Cartesian product), and one of its operands has been removed then remove the binary operator node and reconnect its subtree to the parent of the removed node.

It is important to notice that the PRs derived are not equivalent to the DAC from which they are derived. However, they represent a necessary condition for the DAC. In other words, if the DAC is violated then at least one of its PRs is violated. To prove this, we need to prove that it is impossible to have all derived PRs hold while the DAC at the DW is violated. At the point of time that the DAC at the DW is violated there should be at least one PR at any data source is violated.

Lemma 1

Modified tuples at source S that violate the DAC will also violate the PR for the same source.

Proof

This lemma can be proved through the following reasoning using the rules appear in table 1.
1- The selection predicate for source S appearing in the DAC expression is preserved. This means that any modified tuple accepted by the DAC query will be accepted by the PR query at source S.
2- The equijoin operations, with other sources or with the DW view, are removed from the derived PR expression. This implies that the set of tuples belong to the relation at source S and joined with relations at other sources is subset of the tuples at this relation.
3- The use of contribution factor in PR expression does not reject tuples that are accepted by the DAC expression. Suppose that we have two source elements X and Y belong to different data sources that derive a DW object V. Assume that the contribution factor of X is $CF_{X,V} \in [0,1]$, the contribution factor of Y is $CF_{Y,V} \in [0,1]$ and $CF_{X,V} + CF_{Y,V} = 1$.
If the DAC predicate states that: $X+Y < k$, (where k is any numerical value)
We can deduce that:

PR_X contains a predicate $X < k.CF_{X,V}$ and
PR_Y contains a predicate $Y < k.CF_{Y,V}$.

Theorem 1

If a DAC is violated, then there exist at least one violated PR.

Proof

The objective here to prove that it is impossible to have all PRs valid while the DAC is violated. This theorem can be proved using contradiction.

Assume there is a tuple *t* at source S that is changed by some operation (inserted, modified, or deleted). Assume also *t* violates the DAC expression and does not violate the PR expression at S. This can be achieved if the tuple *t* belongs to the set of tuples selected by the DAC query and does not belong to the set of tuples selected by the PR query at S.
According to lemma 1, any tuple, changed at any source and appears in the results of the DAC query, should belong to the set of tuples selected by the PR query at this source. This implies that the PR is violated at this source which contradicts the assumption of no PRs are violated.

The idea of the above theorem can be captured from the following Venn diagram in fig. 2. The diagram illustrates the relationship between different derived PRs and the DAC. The diagram assumes that there are only two PRs derived from the DAC just for clarity of the diagram.

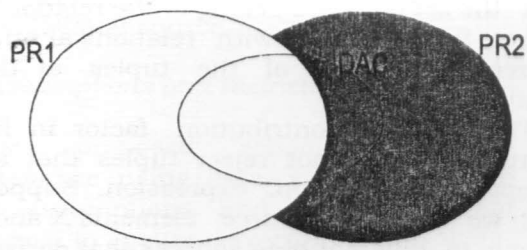

Fig. 2. Venn diagram illustrates the relationship between different database state sets.

As illustrated in the fig. 2, we note that the DAC area must be contained within the union of PR_1 and PR_2 areas. In other words, it is impossible to have a database state that makes all PRs valid while the DAC is violated.

5. Performance study

In this section, the simulation experiments and their results are discussed and analyzed. More details of this study can be found in [11]. The simulation considered the following view maintenance policies:

1- Immediate,
2- Deferred,
3- Periodical with different refresh times (1 minute to 20 minutes), and
4- Data aging constraint (DAC) with different propagation rule (PR) violation probabilities (0.1, 0.5, and 0.9). When we denote DAC(0.1), we mean DAC policy with PR violation probability 0.1. Where the PR violation probability represents the probability of violating the propagation rule (PR) condition against each source update transaction.

For each view maintenance policy, the source update transactions workload and the data warehouse query workload are varied to inspect their effect on the behavior of the different policies. The performance indices are measured for each experiment.

Each simulation experiment runs for approximately 24 hours of simulation time and generates only one data point. We are interested in the following performance indices:

1- Communication cost between the data sources and the data warehouse.
2- View maintenance cost represents the view lock time at the data warehouse.
3- The probability of a warehouse query to read stale data is a measure of data freshness.
4- The warehouse query service time.

5.1. Varying source workload v

5.1.1. View maintenance cost

View maintenance cost is important to measure the efficiency of any view maintenance policy. The importance of the view maintenance cost is originated from that it represents the availability of the data at the data warehouse [12]. During the view maintenance operations, the view is locked and it is not available for querying so that users see a consistent snapshot of the data.

At this experiment, the average view maintenance time is measured for each simulation run. This time is divided by the total simulation time to get the percentage of the time the data warehouse engine spent in maintaining the view. This also represents the average percentage of data unavailability time of the data at the data warehouse. Fig. 3 depicts the results of these experiments.

We can notice that the DAC (0.1) is the policy with the least view maintenance cost at high source workload. This is because it is a low probability to violate the data aging constraint and carrying on the view maintenance operation. So the rate of view maintenance is very low.

On the other hand, the immediate policy shows the highest view maintenance cost because of the very high maintenance rate.

The periodical policy shows a good performance at high source workloads. However, at low source workloads the periodical refresh rate may be greater than source workload. Consequently redundant view maintenance operations will be done.

The DAC policy performance depends on the PR violation probability. On the average DAC (0.5) shows a reasonable performance compared with immediate, deferred, and periodical polices.

5.1.2.Communication cost

The communication cost is represented by the average network delay. This average time is normalized by dividing it by the total

simulation time. By this way, we can know the time percentage the communication channel between the data warehouse and the data source is busy. Fig. 4 depicts the results of these experiments.

The immediate policy shows the highest communication cost. The DAC (0.1) policy shows a low communication cost at the high source workload.

The deferred and the periodical policies show a similar behavior as long as their refresh rate is less than the source update transactions rate.

The DAC policy shows an adaptive behavior to the source workload. As source workload increases, the communication cost increases.

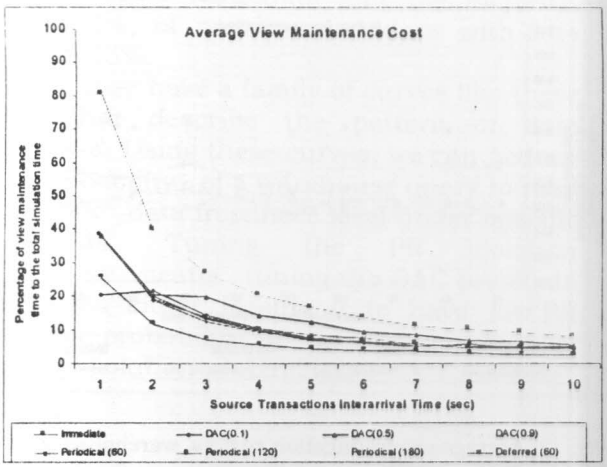

Fig 3. Average view maintenance cost vs. source update transactions workload.

5.1.3. Data warehouse query service time

The data warehouse query service time is a measure that indicates how the view maintenance policy is responsive to the data warehouse workload. Fig. 5 depicts this experiment results.

All view maintenance policies show good service time except the deferred policy. The deferred policy carries on the view maintenance operation when a new data warehouse query is issued and the query answer will be fetched from the materialized view after its maintenance. On the other hand, other policies answer the data warehouse query as soon as it comes from the current materialized view data even it is not fully refreshed and synchronized with the underlying data sources.

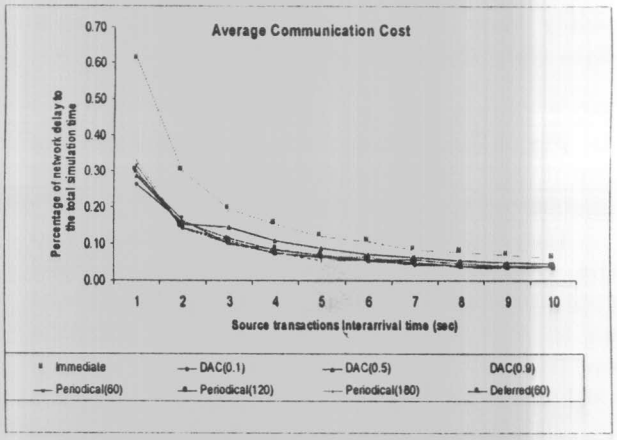

Fig. 4. Average communication cost vs. source update transactions workload.

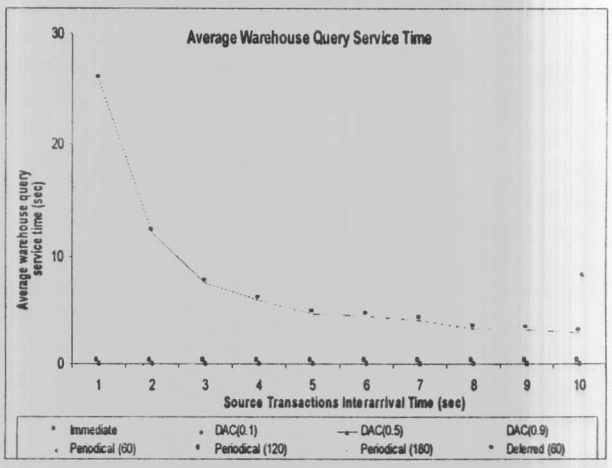

Fig. 5. Average warehouse query service time vs. source update transactions workload.

5.2. Varying warehouse workload

5.2.1. view maintenance cost

The way we measure the view maintenance cost is similar to the one explained before. Fig. 6 depicts the results of these simulation experiments.

All policies show almost the same behavior except the deferred policy. The reason behind the nearly constant cost of

each policy is that the view maintenance cost is highly dependent on the source workload.

The deferred policy depends on the data warehouse workload and is insensitive to the source workload. That is why the deferred policy view maintenance cost decays as the data warehouse workload decreases.

5.2.2. Communication cost

Fig. 7 depicts the results of the simulation experiments responsible for measuring the communication cost.

The results show a similar behavior to the view maintenance cost. Since the rate of communication messages is either constant as in the case of the periodical or depends on the source workload as in the DAC and immediate policies.

The deferred policy communication cost is proportional to the data warehouse workload.

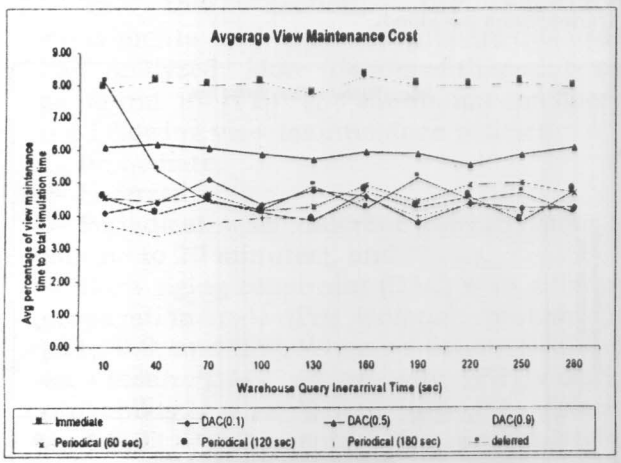

Fig. 6. Average view maintenance cost vs. warehouse query workload.

5.3. Data freshness

The best view maintenance policy in terms of data freshness is the one that keeps up-to-date materialized views most of the time. Measuring the data freshness of any materialized view is not trivial.

Intuitively, the materialized view is considered fresh when it is not different from the underlying data sources.

To facilitate such measurement process, we take the assumption that all source updates affect the source data values uniformly. In other words, each source update transaction affects the data values with the same amount.

One important metric is the count of source update transactions that are not reflected at the materialized view at the time of answering an incoming data warehouse query. These source update transactions are called source-missed transactions.

At the time a new data warehouse query is served, the source update transactions that are stored at the source monitor are counted and considered misses. This miss count is a quantitative measure of the data freshness read by the data warehouse query. As small misses occur, the data warehouse query reads fresher data.

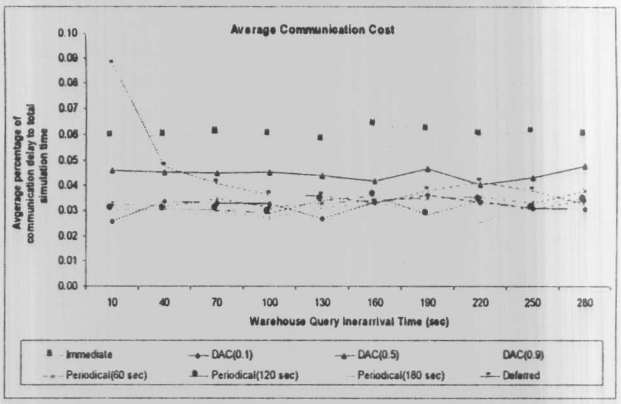

Fig. 7. Average communication cost vs. warehouse query workload.

In these experiments, the average count of misses is calculated for each data warehouse query. A histogram is built to classify the data warehouse queries according to their miss count. Buckets of 20 misses are created and the number of data warehouse queries in each bucket is calculated.

To unify the results, the count of data warehouse queries in each bucket is divided by the total number of queries occurs during the whole simulation time.

Similarly, the miss buckets will be divided by the total number of source transactions occurs during the simulation experiments.
The data age of data read by a query = (count of source transaction misses) / (total source transactions)

By this histogram, we can get a family of curves describing the probability density function of the data freshness. Each curve is created for specific data warehouse and source workloads.

Fig. 8 depicts the probability density function and Fig. 9 depicts the cumulative density function (CDF) of the data freshness. It can be noticed that as the curve is skewed to the left this means a high probability for data freshness. As the curve goes to the right, the data freshness becomes poorer.

It is obvious that the immediate and the deferred policies show a high probability of reading fresh data. The DAC shows a good behavior also. For the DAC (0.5) under the given data source and warehouse configuration, about 45% of warehouse queries read fresh data and the majority, about 54%, of queries read data with data age of 0.23%.

We may have a family of curves like these ones that describe the pattern of data freshness. Using these curves, we can deduce the probability of a warehouse query to read a specific data freshness level under specific workloads. Tuning the PR violation probability means tuning the DAC predicate conditions either loosing it to have low PR violation probability or tightening it to have high PR violation probability.

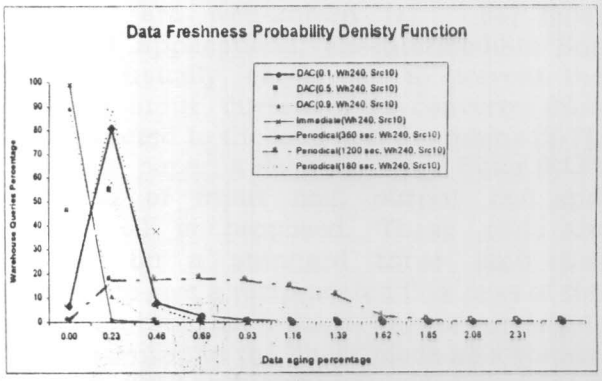

Fig. 8. Probability density function of the data freshness (warehouse workload interarrival time = 240 sec, source workload interarrival time = 10 sec).

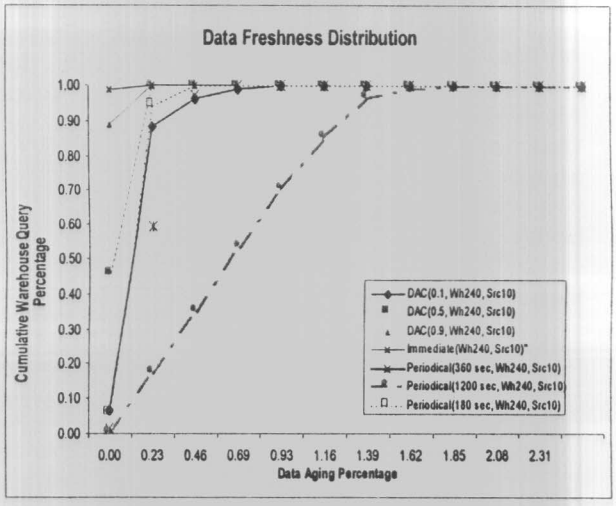

Fig. 9. cumulative density function of the data freshness (warehouse workload interarrival time = 240 sec, source workload interarrival time = 10 sec).

6. Conclusions

Data warehouse applications have specific requirements for data freshness of the stored objects. We have indicated that existing view maintenance policies cannot fulfill user requirements regarding DW object freshness with reasonable cost.

The proposed DAC policy allows the DW users to control the view data aging. An implementation scheme for the DAC policy is proposed to track the aging constraints efficiently using the PR(s) and warehouse agents. This scheme ensures that the DAC tracking process is divided between the source and the data warehouse. A simulation study is conducted and it is shown that the DAC policy cost is intermediate and depends on the user requirement of data freshness. Other policies are extremes in the sense that they deliver either up-to-date data with unacceptable cost or stale data with minimum cost.

There are many issues need further studying. For example, generalize the PR derivation method by relaxing some of the assumptions of the DAC expression to accommodate a wider range of expressions. Also, search for a proper assignment of contribution factors to the sources. This assignment optimizes the performance of the proposed scheme.

Investigating the problem of keeping mutual view consistency at the data warehouse is also a problem for future research. For example, we may have a view that is defined on the same base tables like another view. We should consider keeping the two views consistent so that there is no lag between them.

References

[1] L.S. Colby, A. Kawaguchi, D.F. Lieuwen, I.S. Mumick, and K.A. Ross, *"Supporting Multiple View Maintenance Policies,"* ACM SIGMOD (1997).

[2] Y. Taha, A. Helal, and K. Ahmed, *"Data Warehousing: Usage, Architecture, and Research Issues,"* ISMM Microcomputer Applications Journal, 16 (2) (1997).

[3] R. G. Bello, K. Dias, A. Dowring, J. Feenan, J. Finnerty, W.D. Norcott, H. Sun, A. Witkowski, and M. Ziauddin, *"Materialized Views in Oracle,"* proceedings of the 24th VLDB conference, New York, USA (1998).

[4] J. Cho and H. Garcia-Molina, *"Synchronizing a Database to Improve Freshness,"* Proceedings of 2000 ACM International Conference on Management of Data (SIGMOD), May (2000).

[5] B. Adelberg, B. Kao, and H. Garcia-Molina, *"Database Support for Efficiently Maintaining Derived Data,"* proceedings of the international conference on Extending Database Technology (EDBT) (1996).

[6] J. Hammar, H. Garcia-Molina, J. Widom, W. Labio, and Y. Zhuge, *"The Stanford Data Warehousing Project,"* IEEE Data Engineering Bulletin, June (1995).

[7] J. Widom, *"Research Problems in Data Warehousing,"* Proceedings of the ACM CIKM, November (1995).

[8] Dallan Quass, *"Maintenance Expressions for Views with Aggregation,"* Stanford Research Report (1996).

[9] A. Gupta, V. Harinarayan, and D. Quass, *"Generalized Projections: A Powerful Approach to Aggregation,"* In Proceedings of the 21st International Conference on Very Large Databases, Zurich, Switzerland, September 11-15 (1995).

[10] D. Agrawal, A. El Abaddi, A. Singh, and T. Yarek, *"Efficient View Maintenance at Data Warehouses,"* SIGMOD RECORD (1997).

[11] M.A. Mohamed, *"View Maintenance Policy Using Data Aging in Data Warehouse"*, M.Sc. Thesis, Alexanria University, September (2001).

[12] W. J. Labio, J. Yang, Y. Cui, H. Garcia-Molina, and J. Widom, *"Performance Issues in Incremental Warehouse Maintenance,"* Technical Report, Stanford University (1999).

Received September 29, 2001
Accepted November 29, 2001