Obsolescent Materialized Views in Query Processing of Enterprise Information Systems

Avigdor Gal
Rutgers University
Email: avigal@rci.rutgers.edu

Abstract

In recent years, query processing has become more complex as data sources are frequently replicated and data are periodically processed and embedded within several data sources simultaneously. These trends have necessitated the optimization of techniques for query processing in order to exploit these new alternatives. Accordingly, this paper introduces an improved query optimization technique, which is capable of assessing query plans that use both current and obsolescent data. In particular, we provide a cost model by which the trade-offs of using obsolescent materialized views can be evaluated and we also discuss the method's applicability to contemporary query optimization techniques.

1 Introduction

Contemporary enterprise information systems consist of various types of data sources (e.g., conventional databases, extranets, and semi-structured data sources) whose data are periodically replicated, processed, and materialized in other data sources simultaneously. This phenomenon is commonly identified with *data warehouses*, which provide easy and optimized access to multiple, distributed, heterogeneous databases by separating the operational system from the retrieval system. It is also a common phenomenon in Web-based information systems, where extranets provide a restricted access to external data, which are periodically propagated and materialized within the enterprise's data sources.

The inclusion of periodically updated materialized views made query processing a complex task. Nevertheless, replicated data and materialized views offer new opportunities for query optimization, as was suggested in [4]. In this paper, we provide such an optimization technique in which a query optimizer is equipped with the capability to provide an efficient query plan by using current as well as obsolescent data. We use conceptual modeling techniques in introducing an information model for structuring an efficient query plan as well as a cost model that allows the optimizer to account for both traditional and new trade-offs. Lastly, we demonstrate the model's applicability to distributed query processing as well as to materialized views update plans in data warehouse.

Our work builds on the existing research in query optimization. Traditional query optimizers use a cost model that is based on the generation cost incurred when evaluating alternative query plans. Query optimizers in a distributed environment consider an additional cost which evaluates the cost of transmitting data over a network. We further extend existing cost models to include a third parameter, which we term the *obsolescence cost*. This parameter signifies the trade-off between the instant use of a materialized view (in terms of generation cost and transmission cost) and the usefulness of such data, as derived from its periodic update. In this work, the obsolescence cost is utilized in developing new methods by which the cost of query processing in the presence of periodically updated materialized views can be reduced.

1.1 Related work and contribution

This work is motivated by the need to ameliorate query performance by taking advantage of the structure of contemporary enterprise information systems. The parameters of cost models for query processing have long been studied and are well understood within the realm of conventional databases, as well as distributed databases [3]. Yet, not until now were such results applied to the emerging environment that consists of periodically updated materialized views.

The problem of supporting queries using materialized views has been studied intensively, e.g., [4] and [1].

$el; EL; \|el\|$	a data source basic element; a set of elements; estimated element size
$q; clause_q; G = (V, E); leaf(G); P(G);$ $q_{ext}; q^t_{ext};$ $q_{ext}(G)$	a query; a query's clause; a query's plan; the set of leaf nodes; the set of paths; a query's extension; a query's extension at time t; a query's extension using query plan G
$mv; clause_{mv}; MV; mv_{ext};$ $\tilde{mv}_{ext}; mv_{ext}(G)$	a materialized view; a view's clause; a set of views; a materialized view's extension; a materialized view's current extension; a materialized view's extension using G
$g(q, G) = \{g^c, g^n\};$ $t(q, G) = \{t^c, t^n\};$ $c^n(q, G); b(q, G); PC(q, G)$	generation cost, commulative & non-commulative; transmission cost, commulative & non-commulative; non-commulative cost; obsolescence cost; processing cost

Table 1: List of Symbols

In contrast to our study, current research does not consider the obsolescence factor in materialized views update. For example, in [1], the relationships between a materialized view I and the "true" query result $\mathcal{V}(D)$,[1] taken from a database D, can be either $I = \mathcal{V}(D)$ or $I \subseteq \mathcal{V}(D)$. The latter relationship represents a situation where the materialized view stores only a partial subset of the query result. However, taking the obsolescence factor into account, it is also possible that $I \supset \mathcal{V}(D)$, whenever tuples are deleted from $\mathcal{V}(D)$ and I is periodically updated. Moreover, modifications to the base data may result in both $I \not\subseteq \mathcal{V}(D)$ and $I \not\supseteq \mathcal{V}(D)$.

Recent research in data warehouses involves self maintained materialized views [11]. Two significant underlying assumptions differ this area of research from the research in this paper. First, the materialized views are refreshed before querying [13] and therefore are never obsolescent. In contrast, we take a more general approach towards materialized views where updates are not necessarily coordinated with retrieval requests. This approach is common whenever access to external data sources (e.g., extranets) entails a high recomputation cost. Second, self-maintained materialized views assume the continuous update of materialized views in the absence of the base data (as is assumed in the majority of the works alluded to above). Consequently, there is no other alternative but to use the view while ensuring (or at least estimating) the completeness of the query result. Conversely, our approach assumes the availability of both the base data and the (possibly obsolescence) materialized view, which gives rise to the cost model provided in Section 3 and the need to evaluate alternative query plans that consist of materialized views. It is worth noting that if the base data is available for computation, the cost of replicating the base data at the data warehouse (which is roughly the solution for self-maintenance) may be too high to be considered, and thus the cost model may become handy in evaluating the alternative, namely using obsolescent data.

Some research was devoted to querying a partially replicated distributed database (e.g., [10]). While replicated data bear a similar problem to that introduced in this paper, they are assumed to be current while materialized views are not necessarily so. Therefore, the obsolescence cost introduced in this work does not apply to replicated data.

The main contribution of this paper is in introducing a novel temporal dimension (the obsolescence factor) to the design of query plans in the presence of periodically updated materialized views.

To illustrate our approach, we introduce a simplified case-study of a distributed database that consists of information regarding documents. The global description of the database consists of two relations, as follows:

- DESCRIPTION(**bookNo**, title, year, publisher, location) supplies information about books; and
- AUTHORS(**bookNo**, **author**, biography) stores information about those authors whose books are recorded in the DESCRIPTION table.

The rest of the paper is organized as follows. Section 2 provides an information model for identifying tradeoffs in query plans. The cost model is given in Section 3, followed by a short discussion of the applicability of the approach and some directions for future research in Section 4.

2 The information model

In this section we introduce the information base for the cost model that follows in Section 3. The information model is an abstraction of information gathering process in enterprise information systems. We use conceptual modeling techniques to enhance abstractions from the area of federated database and provide a unified model of information gathering. The resulting model is specified in relational database terminology to benefit from the existing research into relational query optimization. Nevertheless, the information model is generic and can accommodate the needs of a wide range of data sources.

We start by introducing the concept of a query processor (Section 2.1), to be followed by a discussion on materialized views (Section 2.2), including the notions of obsolescent and current materialized views. Finally, Section 2.3 discusses query plans.

[1] The "truth" of a query result depends on the underlying base data and is formally defined in Section 2.2.

2.1 Query processor

Conceptually speaking, a *query processor* is a computerized tool that utilizes a user interface in order to obtain a query request q from a user and returns a set of instances q_{ext} that fulfill the query constraints by accessing one or more data sources. A query processor can be a conventional DBMS query processor, a World-Wide-Web extranet that serves as a host language for a database application, etc. A query processor communicates with the data source by using a query language. We use SQL as a common query language to provide access to distributed heterogeneous information sources.

A query processor utilizes an *information base* $\mathcal{IB}$ of the data sources it queries. The information base consists of information regarding the structure of the data sources, resolutions of heterogeneous issues, and inter-relationships (e.g., data replication, view materialization, etc.) among the data sources. The information base is stored in either a database or a repository, updated semi-automatically using techniques for self-populating ontologies (e.g., [5]).

Each data source is identified by a unique name. Its type can be one of the types the query processor supports, e.g., RDB, HTML, and ASCII file. Its structure is defined by a set of elements $el_1, el_2, ..., el_n$ (e.g., relations), using a DDL that utilizes translators to/from the local data definition languages. The definition of the type and the structure of a data source are not elaborated in this paper, yet suffice it to say that they provide the query processor with sufficient information for accessing the data source, whether it is via an API or using a direct access to a file. As an example, consider the NADatabase (North-American Database) data source with a relation NADescription. The relation consists of a horizontal fragment of the relation Description with a fragmentation constraint that can be expressed as the following SQL clause (annotated as q_1): SELECT bookNo, title, year, publisher FROM Description WHERE location= "North America" (in relational algebra: $\pi_{\text{bookNo, title, year, publisher}}(\sigma_{\text{location="North America"}}(\text{Description}))$). The EuropeDatabase data source is a horizontal fragment of the relation Description which consists of books located in Europe. The EuropeDatabase data source also consists of the Authors relation.

2.2 Materialized views

In what follows we use a strict definition of views, based on the definition of a *derived data item* [9]: "a data item that appears to exist in its declared form, but it is actually derived from related data in the database." A materialized view is a materialized form of a derived data item. Thus, for any given materialized view mv, exist a computation method $clause_{mv}$ and a set of accessible data D over which mv ranges, such that $mv_{ext} = clause_{mv}(D)$. Note that other works (e.g., [13]) have taken a more relaxed approach towards materialized views, where the base data may not be available for recomputation.

A materialized view mv uses a set of *base data* elements (e.g., relations) and is associated with statistics regarding its update distribution, workload (retrieval) distribution, and expected data volume, as well as the time of its last update. The information regarding the view's last update and an information quality-loss function (see Section 4) jointly provide the novel trade-off, as introduced in this paper, in query optimization. The update frequency and the workload estimations can be assumed to be homogeneous over time.[2] The information regarding the query clause ($clause_{mv}$) that defines the view is also available in the information base. As an example, one may consider the TorontoDatabase as a data source with the TitleAuthorQuery materialized view (annotated as q_2):

SELECT DISTINCT author, title FROM Authors, NADescription, EuropeDescription WHERE
Authors.bookNo=NADescription.bookNo OR
Authors.bookNo=EuropeDescription.bookNo

$((\pi_{\text{author, title}}(\pi_{\text{author, bookNo}}(\text{Authors}) \bowtie (\pi_{\text{bookNo, title}}(\text{NADescription})))) \cup$
$(\pi_{\text{author, title}}(\pi_{\text{author, bookNo}}(\text{Authors}) \bowtie (\pi_{\text{bookNo, title}}(\text{EuropeDescription})))))$

Let the pair $(\mathcal{T};\leq)$ be a discrete temporal domain, isomorphic to the natural numbers, where $\mathcal{T}$ is a nonempty set ($\mathcal{T} \cong \mathbf{N}$) and $\leq$ is a total order on $\mathcal{T}$. $t \in \mathcal{T}$ is a nondecomposable unit of time, whose granularity is application-dependent. $q^t_{ext} : \mathcal{T} \rightarrow P(q_{ext})$ is a time varying function, where P is the power set of q_{ext}. For simplicity's sake, the time varying notation is dropped whenever possible. Should the unique identification of time become necessary, the distinction will be made explicit, by using q^t_{ext} instead of q_{ext}.

Let mv be a materialized view, and let $u_{bd}(mv) = \langle u^1_{bd}(mv), u^2_{bd}(mv), ..., u^n_{bd}(mv)\rangle$ be a sequence of update times to the base data of mv ($\forall 1 \leq i \leq n, u^i_{bd}(mv) \in \mathcal{T}$), and $u_{mv} = \langle u^1_{mv}, u^2_{mv}, ..., u^m_{mv}\rangle$ be the update times of mv ($\forall 1 \leq i \leq m, u^i_{mv} \in \mathcal{T}$). Obviously, for each u^i_{mv} exists $u^j_{bd}(mv)$ such that $u^j_{bd}(mv) \leq u^i_{mv}$, since a materialized view is not updated as long as there are no updates to relevant base data. We term a materialized view mv *obsolescent* at time $t \in \mathcal{T}$ iff

$$\max_{i=1}^{n}\{u^i_{bd}(mv) | u^i_{bd}(mv) \leq t\} \geq \max_{i=1}^{m}\{u^i_{mv} | u^i_{mv} \leq t\}$$

mv is *current* at time $t \in \mathcal{T}$ iff

$$\max_{i=1}^{n}\{u^i_{bd}(mv) | u^i_{bd}(mv) \leq t\} < \max_{i=1}^{m}\{u^i_{mv} | u^i_{mv} \leq t\}$$

Figure 1 illustrates the obsolescent and current times of a materialized view with three updates to base data and

[2] We are currently investigating the more realistic case of a non homogeneous stochastic process for updates.

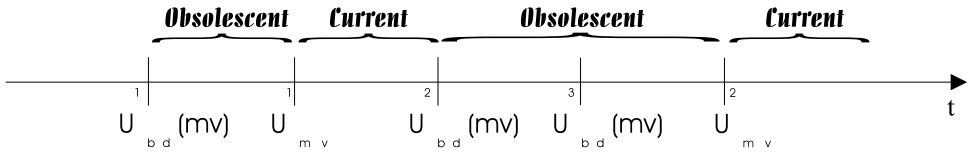

Figure 1: Obsolescent and current times of mv

two updates to the materialized view. It is worth noting that the obsolescence of a materialized view is fully defined by the update sequences of the base data and the materialized view. Also, a materialized view that describes past information that is aggregated on a periodic basis can still be current, as long as the base data have not changed retroactively since the last update of the materialized view.

In what follows it becomes necessary to differentiate between the extension of mv at time t and the result of performing $clause_{mv}$ had it been performed at time t. The former will be annotated as mv^t_{ext}, similar to a query extension, while the latter will be annotated as $\tilde{m}v^t_{ext}$.

Let a_{mv} be an access time to a materialized view mv. The elapse time from the most recent update to mv until a_{mv} (annotated as Δa_{mv}) is computed as follows: $\Delta a_{mv} = \min_{u^i_{mv} \leq a_{mv}}(a_{mv} - u^i_{mv})$.

2.3 Query plans

Let q be a query with a clause $clause_q$ and let q_{ext} be the extension of q. A *query plan* G of a query q provides a sequence of activities to generate q_{ext}. In a relational database model, a query plan is typically represented as a graph $G = (V, E)$ (either an operation tree or an operation lattice) where each vertex is either a base relation, an intermediate relation, or the final result of a query, and an edge $\langle v_i, v_j \rangle$ connects v_i with v_j iff v_i is utilized in generating v_j. A *subexpression sq* of q is an intermediate result during the process of query evaluation [8]. Therefore, q is computed using subexpressions $sq_1, sq_2, ..., sq_n$ with query plans $G_1, G_2, ..., G_n$, respectively, which in turn are computed using additional subexpressions. For example, Figure 2 provides a graphical representation of a query plan of q_2.

Let G be a query plan with a single root node v_r and k leaf nodes $leaf(G) = \{v_{l1}, v_{l2}, ..., v_{lk}\}$. $P(G) = \{p_1, p_2, ..., p_n\}$ is the set of all paths $\langle v_{(1)}, v_{(2)}, ..., v_{(m)} \rangle$ induced by G such that $v_{(1)} = v_r$ and $v_{(m)} \in leaf(G)$. As a notation convention we use the symbols $\in$ and $\subseteq$ to indicate that a node or a path is part of G, respectively. At times when a unique reference to a specific query plan of a query q becomes necessary, it will be made explicit by using the notation $q_{ext}(G)$.

Let EL and MV be the set of all elements and materialized views that are available in an information base $\mathcal{IB}$, respectively, and let q be a query. The computation of q can be based solely on base data elements. Alternatively, q can utilize materialized views in a query plan. G is an *unfolded query plan* [4] iff for each $v \in leaf(G) \Rightarrow v \in EL$. Otherwise, G is a *folded query plan*. Traditional query processing systems are capable of unfolding a folded query plan by using materialized views' definitions, and therefore for each given query q whose domain is within $\mathcal{IB}$, there exists an unfolded query plan G^*. As will be discussed in the ensuing sections, a query result based on obsolescent data may still serve the user. Hence, we can define an alternative query plan that uses a materialized view as one of its data sources and evaluate its usefulness by using the cost model devised in Section 3.

According to conventional interpretation [7], two query plans G and G' are considered alternatives to one another if they are equivalent, i.e., if they produce the same answer for any given database extension

$$(\forall t \in \mathcal{T}, q^t_{ext}(G) = q^t_{ext}(G'))$$

The following definition refines the above statement to adopt it to query processing in the presence of periodically updated materialized views.

Definition 1 Query plans' equivalence: *Let G and G' be two query plans of a query q. G and G' are* equivalent *iff $\forall t \in \mathcal{T}$:*

1. $(\exists v \in leaf(G) \cup leaf(G') | v = mv \in MV \wedge mv^t_{ext} \neq \tilde{m}v^t_{ext}) \vee$

2(a). $((\forall v \in leaf(G) \cup leaf(G'), v \in EL \vee v = mv \in MV \wedge mv^t_{ext} = \tilde{m}v^t_{ext}) \wedge$

2(b). $(q^t_{ext}(G) = q^t_{ext}(G'))$

The parentheses of the compound equivalence condition set a precedence of the $\vee$ connector at the end of Condition 1 over the $\wedge$ connector at the end of Condition 2(a). Definition 1 generalizes the conventional definition of equivalent query plans to include alternative query plans that do not produce the same answer for all database extensions due to obsolescent data (identified as $mv^t_{ext} \neq \tilde{m}v^t_{ext}$ in Condition 1). Therefore, the equivalence of query plans is determined based on extension equivalence of current materialized views only (identified as $mv^t_{ext} = \tilde{m}v^t_{ext}$ in condition 2(a) of Definition 1). It is noteworthy that in the absence of periodically updated materialized views, Condition 1 and

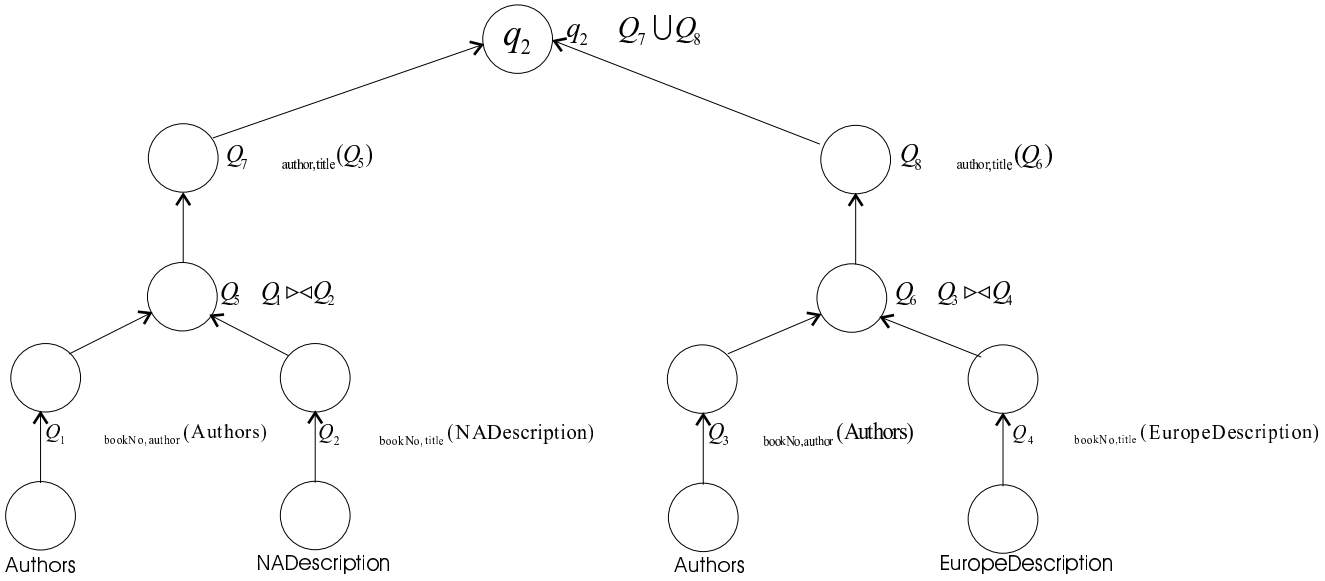

Figure 2: A query plan of query q_2

Condition 2(a) become redundant and the equivalence condition is reduced to its conventional interpretation. It is also worth noting that the condition should hold for **all** $t \in \mathcal{T}$, and therefore G and G' should have the same extension for any time point where the materialized view is current. The decision of using an extension that deviates from the "true" extension is motivated by a possibly high overhead involved in recomputation of the extension. This is in particular the case whenever extranets data are involved, and the cost of accessing an extranet is included in the query processing cost.

3 The cost model

Having introduced the information model we now define the cost model for evaluating the cost of query execution in the presence of periodically updated materialized views. In what follows, we identify the three parameters that affect the cost of a query, namely the generation cost, the transmission cost, and the obsolescence cost. The first two parameters were previously discussed in the literature (e.g., [3]) and are surveyed here (with slight modifications) for completeness sake.

As a running example, consider the following query (termed q_3), posed by a user at the TorontoDatabase data source:

SELECT DISTINCT author FROM Authors (σ_{author}(Authors)). An optimized query plan for processing q_3 using the base data (annotated as G_1) involves accessing the EuropeDatabase data source, projecting a temporary relation with a single attribute, namely author, from the Authors relation and transmitting it to the TorontoDatabase data source (Figure 3(a)). An alternative query plan (annotated as G_2) uses the materialized view TitleAuthor-Query (q_2), as follows. Since the Authors relation represents the set of all authors whose books are available in the union of the two Description relations, the attribute author is projected from q_2 to produce the required response. As shall be demonstrated shortly, a trade-off exists between these two plans, where a transmission cost in G_1 serves as a trade-off to a local computation of possible obsolescent data in G_2.

3.1 The generation cost

The generation cost is the cost of manipulating existing data to generate new data in terms of delays (e.g., seconds), monetary values (e.g., dollars) or size (e.g., bytes) [3]. In the relational database domain, this cost includes the costs of selecting, projecting, joining, and grouping. At the file and memory management level, this cost is based on the evaluation of the number of fetched blocks, the actual main memory size, etc. In addition, whenever an external information service provider is involved (e.g., extranet), a fee may be charged, either a flat fee (per query) or a fee based on the size of the query result.

The generation cost consists of a cumulative component (g^c) and a non-cumulative component (g^n). Whereas the cumulative component reflects the cost of the overall generation cost for executing a query request, usually in terms of monetary values or size, the non-cumulative component reflects the delay time, i.e., the time that is required for query completion. Let q be a query with a query plan $G = (V, E)$. A generation cost $g_{v_i} = \{g^c_{v_i}, g^n_{v_i}\}$ is associated with each vertex $v_i \in V$. For example, for each leaf node $v \in leaf(G)$, $g_v = \{0, 0\}$. Also, if either the base data for computing the query

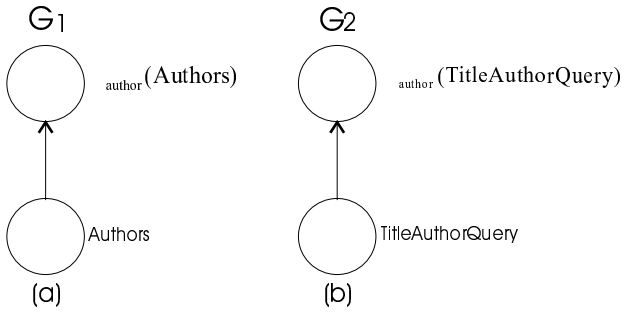

Figure 3: Query plans for q_3

is not available or if $clause_q$ is not available (which is the case if the query is computed by an external source), then $g_v = \{\infty, \infty\}$. The cumulative generation cost of q sums all the cumulative costs of generating data, hence $g^c(q,G) = \sum_{v_i \in V}(g^c_{v_i})$. Based on the nodes' costs the non-cumulative communication cost of a path for each path $p = \langle v_{(1)}, v_{(2)}, ..., v_{(m)} \rangle \subseteq P(G)$ is defined to be $g^n_p = \sum_{i=1}^{m}(g^n_{v_{(i)}})$.

3.2 The transmission cost

The transmission cost involves the cost of transferring data over a network. A transmission cost also consists of a cumulative component (t^c) and a non-cumulative component (t^n). The cumulative component is the cost of accessing a network (e.g., the cost of a long distance call or the delay resulting from transferring data over the network) and is typically a function of the data volume being transferred. The non-cumulative component encapsulates the gain in performance from using parallel channels to transfer data and is less common in current cost models. Let q be a query with a query plan $G = (V, E)$. A transmission cost $t_{\langle v_i,v_j \rangle} = \{t^c_{\langle v_i,v_j \rangle}, t^n_{\langle v_i,v_j \rangle}\}$ is associated with each edge $\langle v_i, v_j \rangle \in E$. If $\langle v_i, v_j \rangle$ requires no communication whatsoever, as is the case in a centralized database system, then $t^c_{\langle v_i,v_j \rangle} = t^n_{\langle v_i,v_j \rangle} = 0$. Otherwise, the costs $t^c_{\langle v_i,v_j \rangle}$ and $t^n_{\langle v_i,v_j \rangle}$ represent the cumulative and non-cumulative costs of accessing the network, respectively. The cumulative cost of a query plan G of q sums all the cumulative costs of accessing the network, hence $t^c(q,G) = \sum_{\langle v_i,v_j \rangle \in E}(t^c_{\langle v_i,v_j \rangle}) + t^c_{\langle root,dest \rangle}$, where $t^c_{\langle root,dest \rangle}$ represents the cost of the final transmission of data from the root node to the user's terminal. Based on the edges' costs, the non-cumulative transmission cost of a path for each path $p = \langle v_{(1)}, v_{(2)}, ..., v_{(m)} \rangle \subseteq P(G)$ is defined as follows: $t^n_p = \sum_{i=1}^{m-1}(t^n_{\langle v_{(i)},v_{(i+1)} \rangle}) + t^n_{\langle root,dest \rangle}$.

The non-cumulative transmission cost and the non-cumulative generation cost are inter-related, as their combined impact identifies a critical path within the query plan whose cost serves as the non-cumulative cost of the query. Thus, the non-cumulative cost of a query plan G for a query q (denoted $c^n(q,G)$) is computed as $c^n(q,G) = \max_{p \in P(G)}(t^n_p + g^n_p)$.

3.3 The obsolescence cost

The *obsolescence cost* $b(q,G)$ of a query q using a query plan G represents the penalty of basing a query result on obsolescent materialized views. This cost corresponds to the measure of deviation of the query result using G from a query result using an equivalent unfolded query plan G^*. Unlike the generation cost and the transmission cost, the obsolescence cost does not have a cumulative component. To manifest this point, consider the example in Figure 3(b) above. TitleAuthorQuery is a materialized view that may carry an obsolescence cost. π_{author}(TitleAuthorQuery) may also carry an obsolescence cost due to the utilization of an obsolescent materialized view. Yet, this cost is not additive to the one of TitleAuthorQuery, but rather a derivative of it.

An obsolescence cost b_{v_i} is associated with each node in the graph, where $b_{v_i} = 0$ if the subtree rooted at v_i consists of no materialized views and $b_{v_i} \geq 0$ otherwise. The deviation of a query result using G from a query result using some equivalent unfolded query plan G^* is computed as follows. Let q be a query with a clause $clause_q$. Let $q^t_{ext}(G^*)$ be the result of performing $clause_q$ using an unfolded query plan G^* at time t. Let tuple $tup \in q^t_{ext}(G^*)$ be a tuple in $q^t_{ext}(G^*)$ with a key $key(tup)$ (in the absence of a declared primary key, the internal tuple-id can be used as a key). A comparison between $q^t_{ext}(G)$ and $q^t_{ext}(G^*)$ yields one of the following four scenarios:

No change: $\exists tup' \in q^t_{ext}(G) | tup = tup'$.

Tuple modification: $\exists tup' \in q^t_{ext}(G) |$ $key(tup) = key(tup') \wedge tup \neq tup'$. Let $\Delta(tup, tup')$ be the number of different bytes between tup and tup', and 0 if either $tup = tup'$ or $key(tup) \neq key(tup')$.

Tuple insertion: $\forall tup' \in q^t_{ext}(G), key(tup) \neq key(tup')$. Let $\Delta(tup)$ be 0 is exists $tup' \in q^t_{ext}(G^*)$ such that $key(tup) = key(tup')$ and the size of tup in bytes, otherwise.

Tuple deletion: A tuple $tup' \in q^t_{ext}(G)$ does not have any counterpart in $q^t_{ext}(G^*)$. This scenario is a result of a tuple deletion that occurred after the materialized views in G were last updated. Let $\Delta(tup')$ be 0 if exists $tup \in q^t_{ext}(G^*)$ such that $key(tup) = key(tup')$ and the size of tup' in bytes, otherwise.

The obsolescence cost is computed as follows:

$$b(q,G) = K \cdot (\sum_{tup \in q^t_{ext}(G^*) \cup q^t_{ext}(G)} \Delta(tup) +$$

$$\sum_{\langle tup, tup' \rangle | tup \in q^t_{ext}(G^*) \wedge tup' \in q^t_{ext}(G)} \Delta(tup, tup'))$$

where K represents the significance of deviation.

3.4 The overall query processing cost

The overall query processing cost for a query q strikes a balance between the generation and transmission costs on the one hand, and the obsolescence cost on the other hand. The use of a materialized view may reduce the generation cost as it provides a readily available data. Also, it may reduce the cost of the transmission cost whenever the raw data must be transmitted from a distant site while the materialized view is available locally. However, materialized views may reduce the overall reliability of the query result, as their information may be based on obsolescent data. Therefore, the cost of a query q using a query plan G is $PC(q,G) = \delta_1(t^c(q,G) + g^c(q,G)) + \delta_2 c^n(q,G) + \delta_3 b(q,G)$, such that $PC(q,G)$ is a convex combination of the respective costs (i.e., the weights $\{\delta_i\}$ $(i = 1,2,3)$ are non-negative and sum to unity). $\{\delta_i\}$ $(i = 1,2,3)$ represents the relative weights of the components of the cost model. It is worth noting that $PC(q,G)$ also captures the benefits of using parallel mechanism as a vehicle to minimize the response cost. It is also noteworthy that all costs should either be measured using the same units (e.g., bytes) or must be converted to a unified scale.

Tuning the weight of the obsolescence cost depends on the user's preferences. A user who is interested in a prompt response would set a higher weight to the generation and transmission cost, while a user who requires an accurate query result would set an emphasis on the obsolescence cost. The user can easily specify her preference with respect to this trade-off, either on a single query basis or as part of general preferences. Traditional trade-offs (e.g., the selection of a site for processing intermediate results) are not user dependent, and therefore are handled transparently by the optimizer.

Some of the parameters of the cost model may be accurately obtained in the process of computation, e.g., the size of relations, while others can only be estimated, e.g., network traffic. However, there is an overhead involved in obtaining accurate values of parameters. Therefore, the computation of the cost model is commonly done locally, based on estimates available at the site where the query is posed.

4 Applicability and future research

This paper has introduced a temporal dimension to the optimization of queries in the presence of periodically updated materialized views. This temporal dimension provides an additional criterion for the evaluation of alternative query plans and enables a refined approach

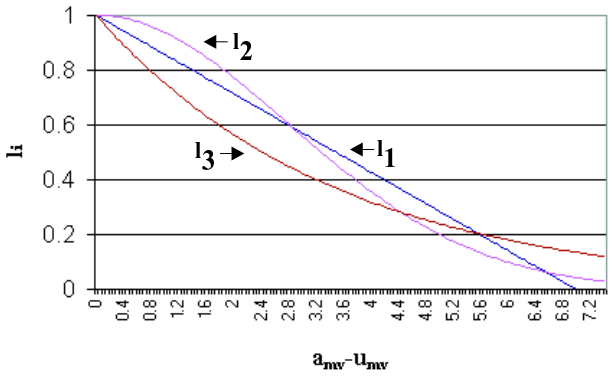

Figure 4: Examples of quality-loss functions

towards the usefulness of materialized views. In particular, we have introduced a cost model that lends itself well to the contemporary enterprise information systems where data sources are frequently replicated and their data are periodically processed and embedded within other data sources. By using this model, the cost of using materialized views as an alternative to computing the query using base data can be evaluated.

Computing the obsolescent cost by using a comparison of the end result of $q^t_{ext}(G)$ and $q^t_{ext}(G^*)$ yields an accurate result. However, once $q^t_{ext}(G^*)$ is computed and made available for comparison, there is no need for using $q^t_{ext}(G)$. Therefore, this method is too expensive to be of any practical use in this framework. An alternative method involves an estimation of base data modifications and serves our purpose better. For example, Figure 4 provides a visual representation of three quality-lost functions, representing the user's preferences over time. Statistics regarding the update distribution of the base data can be utilized in providing quick estimations of the information quality deterioration. This information, in turn, can be utilized in computing the deterioration of predecessor nodes in the query plan by using local mechanisms and therefore it adds a minimal overhead to the query optimization process.

The main contribution of this paper is in the introduction of extended trade-offs in query processing and in the coercion of these trade-offs into a three-way cost model. Previous developments in the area have not identified the obsolescence cost as a component of a cost model for query processing, nor was the impact of embedding an obsolescent materialized view as part of a query plan evaluated prior to this paper. Both issues have been addressed herein.

The approach provided in this paper has a promising impact on existing technologies. First, it can improve the reliability of distributed query processing. For example, in [2] a method for scrambling query plans with unexpected delays was introduced. This method

is based on a single query plan in which an unexpected delay results in branching to a different subtree of the plan. This method can be extended to a set of alternative query plans, where materialized views and base data serve as alternative sources of information for the query processing. Also, the use of this approach can assist in devising an update plan for interrelated materialized views in a data warehouse by using a bottom-up traversal on a joined query plan [6]. A data warehouses' joined query plan unveils materialized views' inter-relationships, where materialized views may use other materialized views as subexpressions. These inter-relationships allow a more efficient update process for data warehouses, where the update timing of materialized views is coordinated to enable the use of common subexpressions.

Several interesting problems remain to be researched. First, the embedment of materialized views as part of a query plan was discussed under a "whole or nothing" approach, where materialized views are either utilized as a whole or not at all [4]. An extension to this approach allows a partial embedment of materialized views, which requires methods for reconstructing information from materialized views. Another future research involves a stochastic approach to improve the estimation of information deterioration. In addition to the mathematical formulation, the impact of such approach on the required statistics and estimations that need to be collected by the query processor has yet to be considered. Finally, additional development is necessary to enable researchers to find the most appropriate method for minimizing the involvement of users in deciding on their own preferences, possibly through the use of profiling (e.g., [12]). This research would involve collaboration between Computer Science, Information Studies, and HCI, and it is currently being considered under the auspices of the Rutgers Distributed Laboratory for Digital Libraries (http://www.scils.rutgers.edu/RDLDL/).

Acknowledgment

I would like to thank Jonathan Eckstein and Bracha Shapira for their constructive remarks. The work was supported, in part, by the Rutgers Undergraduate Research Fellow Program, Rutgers Faculty of Management.

References

[1] S. Abiteboul and O.M. Duschka. Complexity of answering queries using materialized views. In *Proceedings of the 1998 ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Sys. (PODS)*, 1998.

[2] L. Amsaleg, M.J. Franklin, A. Tomasic, and T. Urhan. Scrambling query plans to cope with unexpected delays. In *Proceedings of the Conference on Parallel and Distributed Information Systems (PDIS)*, Miami Beach, Florida, December 1996.

[3] P. Bodorik and J.S. Riordon. Distributed query processing optimization objectives. In *Proceedings of the Fourth International Conference on Data Engineering*, pages 320–329, Los Angeles, CA, February 1988. IEEE Computer Society.

[4] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *Proceedings of the 11th International Conference on Data Engineering*, pages 190–200, Taipei, Taiwan, 1995.

[5] A. Gal. Semantic interoperability in information services: Experiencing with CoopWARE. *SIGMOD Record*, 28(1):68–75, 1999.

[6] P.A.V. Hall. Optimization of single expressions in a relational data base systems. *IBM Journal of Research and Development*, pages 244–257, May 1975.

[7] ISO. Database language SQL. Technical Report, Document ISO/IEC9075:1992, 1992.

[8] M. Jarke. Common subexpression isolation in multiple query optimization. In W. Kim, D. Reiner, and D. Batory, editors, *Query Optimization in Database Systems*. Springer-Verlag, New-York, 1984.

[9] S. Koenig and R. Paige. A transformational framework for the automatic control of derived data. In *Proceedings of the 7th Conference on Very Large Databases, Morgan Kaufman pubs. (Los Altos CA), Zaniolo and Delobel(eds)*, pages 306–318, September 1981.

[10] R. Mukkamala, S.C. Bruell, and R.K. Shultz. A heuristic algorithm for determining a near-optimal set of nodes to access in a partially replicated distributed database system. In *Proceedings of the Fourth International Conference on Data Engineering*, pages 330–336, Los Angeles, CA, February 1988.

[11] D. Quass, A. Gupta, I.S. Mumick, and J. Widom. Making views self-maintainable for data warehousing. In *Proceedings of the 1996 International conference on Parallel and Distributed Information Systems*, pages 158–169, December 1996.

[12] B.D. Seth. A leraning approach to personalized information filtering. Master's thesis, Electrical Engineering and Computer Science Dept. MIT, Cambridge Mass., 1994.

[13] J. Yang and J. Widom. Maintaining temporal views over non-temporal information sources for data warehousing. In *Proceedings of the 1998 International Conference on Extending Database Technology*, pages 389–403, March 1998.