Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®
The Journal of Systems and Software 68 (2003) 103–119
The Journal of Systems and Software
www.elsevier.com/locate/jss

Usability-based caching of query results in OLAP systems

Chang-Sup Park [a,*,1], Myoung Ho Kim [b], Yoon-Joon Lee [b]

[a] *Service Development Laboratory, Korea Telecom Corporation, 17 Woomyun-Dong, Suhcho-Gu, Seoul 137-792, South Korea*
[b] *Department of EECS, Korea Advanced Institute of Science and Technology, 373-1 Kusung-Dong, Yusung-Gu, Taejon, South Korea*

Received 4 August 2001; received in revised form 10 February 2002; accepted 20 March 2002

Abstract

In this paper we propose a new cache management scheme for online analytical processing (OLAP) systems based on the usability of query results in rewriting and processing other queries. For effective admission and replacement of OLAP query results, we consider the benefit of query results not only for recently issued queries but for the expected future queries of a current query. We exploit semantic relationships between successive queries in an OLAP session, which are derived from the interactive and navigational nature of OLAP query workloads, in order to classify and predict subsequent future queries. We present a method for estimating the usability of query results for the representative future queries using a probability model for them. Experimental evaluation shows that our caching scheme using the past and future usability of query results can reduce the cost of processing OLAP query workloads effectively only with a small cache size and outperforms the previous caching strategies for OLAP systems.

1. Introduction

Queries used in online analytical processing (OLAP) systems perform complex multi-dimensional aggregation over a huge amount of data in data warehouses (DWs). They require fast response time for interactive execution. A commonly used approach to process the expensive OLAP queries efficiently is to exploit materialized views (MVs), i.e., the results of pre-selected or previously issued queries which are stored in DWs. [2] This approach requires methods for selecting appropriate views to materialize in a limited space in DWs and rewriting given queries using the MVs to process them efficiently. Static view selection techniques deliver a solution optimized for a pre-defined set of queries, hence we should provide appropriate input queries and periodically reselect MVs to cope with the changing query workloads in OLAP systems. Query result caching approaches store the results of the queries issued by users dynamically, replacing some previous query results in the cache with a new one. In order to process online ad hoc queries efficiently, an effective cache management scheme is required that can maintain the query results *useful* for the future query workload. Most of the proposed caching schemes, however, select the query results for caching based on their recomputation cost and the statistics of past queries. Moreover, they cannot handle general types of OLAP queries having selection conditions in processing queries using MVs and caching their results.

In this paper, we propose a new cache management scheme for OLAP systems based on the usability of a query result which means the profit that we can obtain by using the query result in processing other queries. In the scheme, we use the usability of query results to measure their value for caching. Due to the computation dependencies among OLAP queries derived from the hierarchical classification information in DW schema and the semantically overlapping

* Corresponding author. Tel.: +82-2-526-6584; fax: +82-2-526-6792.
E-mail addresses: cspark0@kt.co.kr (C.-S. Park), mhkim@dbserver.kaist.ac.kr (M.H. Kim), yjlee@dbserver.kaist.ac.kr (Y.-J. Lee).
[1] This work was done while the author was with KAIST.
[2] In this paper we use the terms materialized views and stored query results interchangeably.

0164-1212/$ - see front matter © 2002 Elsevier Inc. All rights reserved.
doi:10.1016/S0164-1212(02)00142-5

selection predicates of the queries, there are in general various ways of evaluating a query using a different set of MVs, which greatly differ in execution cost. Thus the usability of a materialized view is affected by other MVs in the cache and determined by the employed query rewriting method.

For more effective management of OLAP query results, we consider the usability of the query results not only for recently issued queries but for the predicted future queries of a current query. Query workloads in OLAP applications have been known to have an explorative and navigational nature (Codd et al., 1993; Sapia, 1999). In an OLAP session, users often create a subsequent query based on the previous query result, performing an OLAP operation such as a drill-down, roll-up, and slice&dice. We classify the possible subsequent future queries of the current one considering the semantic relationships between successive OLAP queries. For each query result in the cache, we define a set of representative future queries which can be rewritten using it, presenting a probability model for them. We compute the goodness of a query result based on the profit that we can obtain when we process its representative future queries using the query result.

In this paper, we utilize the semantic rewriting method for OLAP queries proposed in our previous work (Park et al., 2002) to evaluate the past and future usability of query results. However, the notion of usability and the proposed caching schemes are not dependent on a specific query rewriting method. We show by experiments that our cache management scheme using the usability of query results for future queries as well as the past ones can reduce the cost of processing OLAP queries more effectively than the previous query result caching algorithms for DWs and OLAP systems in various query workloads.

The rest of the paper is organized as follows. We first discuss the related work and present basic terminology in Section 2. In Section 3, we propose a cache management scheme using the usability of query results for past queries. In Section 4, we suggest a method for predicting subsequent queries in an OLAP session and describe a caching scheme based on the future usability of query results. We show our experimental studies in Section 5 and draw conclusions in Section 6.

2. Background

2.1. Related work

There have been two approaches for selecting views to materialize in DWs, i.e., static view selection and dynamic caching of query results. Static view selection methods proposed some kinds of heuristic search techniques to select a set of views that optimize evaluation cost of the pre-defined queries under a given storage space limitation or view maintenance cost constraint (Baralis et al., 1997; Gupta, 1997; Gupta and Mumick, 1999; Harinarayan et al., 1996; Theodoratos and Sellis, 1997). In these techniques, however, the MVs are selected with respect to a set of pre-defined queries, hence they cannot be effectively used in processing the real query workloads which are likely to change and become much different from the given queries.

Recently, query result caching approaches for DWs and OLAP systems have been proposed which select query results to be stored in the cache dynamically (Scheuermann et al., 1996; Shim et al., 1999; Deshpande et al., 1998; Kortidis and Roussopoulos, 1999; Albrecht et al., 1999; Roy et al., 2000; Kalnis and Papadias, 2001). These approaches presented some goodness measures to choose previous query results to replace from the cache in order to adapt the cache contents to the changing workload. The DW cache manager proposed by Scheuermann et al. (1996) uses the reference rate and computation cost of the stored query results to select replacement victims. It does not consider inter-dependencies among OLAP queries and only uses the result of an exactly matched query in the cache in processing queries, hence it cannot evaluate OLAP queries efficiently. Shim et al. (1999) proposed a method for processing canonical OLAP queries using other MVs and caching the results based on the re-computation cost. However, they only considered computation dependencies on the result of a single cube query which is a simple aggregation query with no selection condition. The dynamic view management technique proposed by Kortidis and Roussopoulos (1999) uses the cost of re-computing MVs using other MVs in the DW as a goodness measure for the MVs, but like the method suggested by Shim et al. (1999), it can evaluate queries only from one father materialized view in the cache. Furthermore, the authors focused on a limited class of queries whose selection region can only have a full range or a single-valued range on each dimension. The replacement strategy for multi-dimensional objects used by Albrecht et al. (1999) also takes dependencies between cached objects into consideration and uses the re-computation cost of the objects as a parameter in their profit measure.

Roy et al. (2000) proposed an automatic query result caching system which is closely coupled with a Volcano-like cost-based query optimizer. They cache intermediate results of queries generated during query evaluation as well as final query results. They proposed an incremental algorithm that selects query results to be stored in the cache, based

on the benefit of caching a particular query result to the recent query workload. However, their algorithm is too expensive due to its scheme to reselect intermediate or final results to be stored in the cache from scratch, potentially large number of nodes in the AND-OR DAG of candidate intermediate or final results for caching, and the overheads in computing the benefits of candidate nodes repetitively, which makes the algorithm inappropriate and inefficient for online cache management. Kalnis and Papadias (2001) proposed an architecture for caching query results in a network of multiple distributed OLAP cache servers. They proposed three caching policies suitable for different configurations of the network. However, their caching algorithm can be thought of as a specialization of the algorithm suggested by Roy et al. (2000), which only caches the final query results. Moreover, they only support aggregation queries with no selection condition which involve a whole view in the data cube lattice and the queries are calculated from the cached result of one of their descendent views. That is a simple and unpractical workload in OLAP applications.

In this paper, we do not consider some well-known cache replacement schemes such as LRU and LFU that are often used for page-based caching in relational database systems and operating systems since we believe that they are not appropriate for caching query results in DWs and OLAP systems. Query result caching is different from the page-based caching in many ways: query results have different sizes, execution costs of queries are different from each other, and there exist computational dependencies among query results.

To answer queries efficiently using the stored query results, a query rewriting technique that can make effective use of the MVs is required. Several rewriting strategies for conjunctive queries and aggregation queries have been studied in the literature (Chaudhuri et al., 1995; Levy et al., 1995; Srivastava et al., 1996; Zaharioudakis et al., 2000; Goldstein and Larson, 2001). In the previous studies (Park et al., 2001a, 2002), we exploited semantic information in DWs and the characteristics of OLAP queries, and proposed a new method for rewriting OLAP queries using multiple MVs together to yield a general type of rewriting which can be executed efficiently. Our rewriting method uses a greedy heuristic algorithm to select a set of MVs to be used in rewriting a given query. At each selection of a materialized view, the algorithm selects the MV that gives the maximum profit for the remaining selection region of the query and generates a query over the MV with a selection predicate determined by the overlap of selection regions. The results of all the sub-queries are integrated into a final result using either a union or a union-all followed by a group-by operator.

2.2. Basic terminology

DWs for the relational OLAP systems typically have a star schema consisting of a fact table and a set of d dimension tables. The fact table has foreign keys to all dimension tables whose values map a tuple in the fact table to a point in the d-dimensional domain space formed by the dimension tables. We suppose that we can obtain a dimension hierarchy from each dimension table, which is defined by consecutive functional dependencies among sets of dimension attributes, called dimension levels, in the dimension table. The Cartesian product of all dimension hierarchies generates a partially ordered set $S = (\mathrm{DH}, \leqslant)$ defined by a class DH of the ordered sets of dimension levels from all different hierarchies and a partial ordering relation $\leqslant$ among the elements in DH derived from the functional dependencies in all hierarchies. The poset S can be represented as a lattice called the *Dimension Hierarchies* (*DH*) *lattice*. The set of attributes in a node in the lattice can be used as a criterion for selecting or grouping the detailed data in the fact table.

The OLAP queries we consider in this paper are un-nested single-block aggregate queries over the fact table and dimension tables. They select the detailed data with a selection predicate and then aggregate them by grouping attributes. The selection and grouping attributes are dimension attributes and can be specified by a node in the DH lattice, called the *Selection Granularity* (*SG*) and *Aggregation Granularity* (*AG*) of the queries, respectively. The selection predicate expressed in a disjunctive canonical form can be geometrically represented as a set of hyper-rectangles in the d-dimensional domain space, called the *selection region* (R) of the queries. The queries may contain a set of aggregate functions AGG on the measure attributes and a HAVING condition HAV on the grouping attributes or aggregate functions. Using these components of the queries, a canonical form of the OLAP queries in our paper is defined as

$$q(\mathrm{SG}, R, \mathrm{AG}, \mathrm{AGG}, \mathrm{HAV})$$

$\mathrm{SG}(q)$, $R(q)$, $\mathrm{AG}(q)$, $\mathrm{AGG}(q)$, and $\mathrm{HAV}(q)$ denote the elements in the canonical form of the query q. For simplicity in notation, we also use the notation $\mathrm{SG}(q)$ and $\mathrm{AG}(q)$ to denote the union of the dimension levels in the SG and AG of q. $\mathrm{AG}(q,i)$ and $\mathrm{SG}(q,i)$ $(1 \leqslant i \leqslant d)$ denote the dimension levels from the dimension hierarchy of the dimension d_i contained in $\mathrm{AG}(q)$ and $\mathrm{SG}(q)$, respectively, and $R(q,i)$ is the set of selection ranges on the dimension d_i in $R(q)$.

We assume that valid queries in OLAP systems select the detailed data on the SG that is not finer than their AG, satisfying $\mathrm{SG}(q) \geqslant \mathrm{AG}(q)$. For the sake of simplicity and usefulness in query rewriting, we only consider the MVs that

store the results of canonical form OLAP queries with no HAVING condition. Then we can prove that if a materialized view mv satisfies the following condition for a given query q, it can be used in rewriting q (Park et al., 2002): [3]

$$\mathrm{AG}(\mathrm{mv}) \leqslant \mathrm{AG}(q), \quad R(\mathrm{mv}) \cap^* R(q) \neq \phi \quad \text{and} \quad \mathrm{AGG}(\mathrm{mv}) \supseteq \mathrm{AGG}(q) \tag{1}$$

We call such materialized view the candidate materialized view for rewriting the query.

In rewriting a given query, a new query is generated over each MV or the fact table chosen for rewriting irrespective of the employed query rewriting method. The query performs selection and/or aggregation over the selected MV or the fact table. We call the selection region of the query the *query region* for the involved MV or the fact table.

3. A caching scheme based on past usability

In this section, we propose a new cache management policy termed the *Lowest-Usability-First for Past queries* (*LUF-Past*) which is based on the usability of query results for the recently executed queries. The past usability reflects how effectively the query results can be used for the current query workload that is represented by the k different queries most recently executed.

Let MV be the set of MVs currently in the cache and Q_k be the set of different k queries that have been executed most recently. The replacement policy selects MVs in the reverse order of the total profits gained by each materialized view in processing the queries in Q_k. This policy utilizes the temporal locality in the OLAP query workload and keeps the MVs in the cache that maximize the total profit in evaluating the recently issued queries.

Given a materialized view mv in MV that is usable in computing a query q, the profit of mv for q means the reduction in query execution cost that we can achieve by processing q using mv. The profit can be estimated by the difference between the execution costs of two queries, one over the fact table ft and the other over mv, having the same selection region. Specifically, given a query region $\mathrm{QR}(\mathrm{mv}, q)$ for mv, the profit of mv in rewriting q using MV is obtained as

$$\begin{aligned}\mathrm{profit}(\mathrm{mv}, q, \mathrm{MV}) &= \mathrm{cost}(\mathrm{ft}, R(q)) - (\mathrm{cost}(\mathrm{mv}, \mathrm{QR}(\mathrm{mv}, q)) + \mathrm{cost}(\mathrm{ft}, R(q) -^* \mathrm{QR}(\mathrm{mv}, q))) \\ &\cong \mathrm{cost}(\mathrm{ft}, \mathrm{QR}(\mathrm{mv}, q)) - \mathrm{cost}(\mathrm{mv}, \mathrm{QR}(\mathrm{mv}, q))\end{aligned}$$

The execution cost of a query over mv or ft with a query region QR generated in query rewriting can be estimated using an extended linear cost model (Park et al., 2002). We assume that special access methods such as the bitmap join index (O'Neil and Graefe, 1995) are available for fast access to the fact table while MVs, i.e., the stored query results have no such index or clustering on their attributes, which is a typical configuration in real-world DW systems. We also suppose that the values of dimension attributes are uniformly distributed. Thus we have

$$\mathrm{cost}(\mathrm{mv}, \mathrm{QR}) = N_{\mathrm{mv}} \quad \text{and} \quad \mathrm{cost}(\mathrm{ft}, \mathrm{QR}) = \min\left\{ n_{\mathrm{ft}} \cdot \frac{\mathrm{size}(\mathrm{QR})}{\mathrm{size}(R(\mathrm{ft}))}, N_{\mathrm{ft}} \right\}$$

where N_{mv} and N_{ft} denote the number of pages in mv and ft respectively, n_{ft} is the number of tuples in the fact table, and $\mathrm{size}(R)$ denotes the size of the area of the selection region R.

The past usability of mv is defined as the weighted sum of the rewriting profits of mv for the queries in Q_k, i.e.,

$$\text{past-usability}(\mathrm{mv}, Q_k, \mathrm{MV}) = \sum_{q_i \in Q_k} \mathrm{profit}(\mathrm{mv}, q_i, \mathrm{MV}) \cdot \frac{w(q_i)}{\sum_{q_j \in Q_k} w(q_j)}$$

We apply an exponential decay function to the weight of the queries in Q_k in order to decrease their significance to the past usability of MVs as new queries are executed and inserted into Q_k. Let t be the elapsed time represented by the number of queries performed since the last execution of a query q_i in Q_k and T be a constant that controls the decay rate. Then, the weight of q_i is computed by

$$w_t(q_i) = w_{t-1}(q_i) \cdot \mathrm{e}^{-\frac{t}{T}} \quad \text{and} \quad w_0(q_i) = 1$$

The MV replacement and admission algorithm using the LUF-Past scheme is shown in Fig. 1. The past usability normalized for the size of a materialized view is calculated for each materialized view in the cache and used as a measure for selecting MVs to replace when there is not enough space for storing the result of a current query. In the algorithm, we first suppose that the result of the current query q_c is stored in the cache and select victims from the cache

[3] The region intersection operation $\cap^*$ between the selection regions of two queries is defined as the set of hyper-rectangles which are the result of intersection between two hyper-rectangles contained in two selection regions. The region difference operation $-^*$ can also be defined in a similar way (Park et al., 2001a).

Procedure: LUF-Past Cache Management Algorithm

Input: the current query q_c, the result mv_c of q_c, and the set Q_k of k most recently executed queries

begin

MV := the set of MVs that are currently in the cache;

V := ∅;

if $size(mv_c)$ > CACHE_SIZE **then**

return; // do not admit mv_c into the cache

end if;

MV := MV ∪ $\{mv_c\}$;

while $size$(MV) > CACHE_SIZE **do**

Let $utility(mv_j) = \dfrac{past - usability(mv_j, Q_k, MV)}{size(mv_j)}$ for all mv_j in MV;

Select mv_i in MV that has the minimum utility value, i.e.,

$$utility(mv_i) = \min_{mv_j \in MV} utility(mv_j);$$

if mv_i is equal to mv_c **then**

return; // do not admit mv_c into the cache

end if;

MV := MV – $\{mv_i\}$;

V := V ∪ $\{mv_i\}$;

end while;

Evict the materialized views in V from the cache;

Insert mv_c into the cache;

end

Fig. 1. The LUF-Past cache management algorithm.

iteratively until the total size of the remaining MVs becomes no greater than the cache size. Whenever a materialized view is replaced from the cache, past usabilities of the other MVs should be re-computed by rewriting the queries in Q_k using the MVs. During these repetitive selections, if the stored result of the current query q_c is chosen as the next victim for replacement, we do not admit the result of q_c and keep all the previous MVs in the cache.

For efficient implementation of the above algorithm, we use the *rewriting profit graph*, which is a weighted bipartite graph $G = (\mathrm{MV} \cup Q_k, E, f)$, where $E = \{e_{ij} | e_{ij} = (\mathrm{mv}_i, q_j), \mathrm{mv}_i \in \mathrm{MV}, q_j \in Q_k,$ and mv_i is a candidate materialized view for $q_j\}$ and $f : E \to \Re$ such that $f(e_{ij}) = \mathrm{profit}(\mathrm{mv}_i, q_j, \mathrm{MV})$. This graph represents the usage relation between the set of MVs and the set of k recently executed queries, including the profits of the MVs in rewriting the queries. Fig. 2 shows an example of the rewriting profit graph. In the graph, there exists an edge e_{ij} between a node for mv_i in MV and a node for q_j in Q_k for which mv_i is one of the candidate MVs. The weight function $f(e_{ij})$ on the edge e_{ij} gives the rewriting profit of the materialized view mv_i for the query q_j. Using this graph, we can reduce the overheads of performing query rewriting using remaining MVs and computing usability of the MVs repetitively in the algorithm. The cache manager maintains and updates the rewriting profit graph while caching query results.

4. A caching scheme based on future usability

In this section, we propose a cache management scheme based on the usability of query results for future queries. We first classify the subsequent queries of a currently executed query into four different categories using relationships

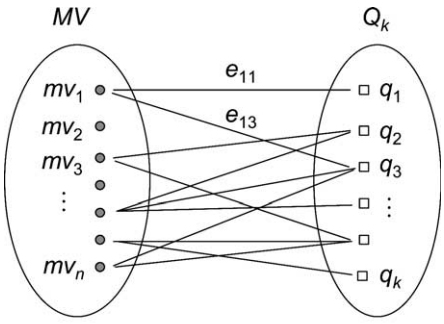

Fig. 2. An example rewriting profit graph.

between two consecutive queries in an OLAP session. We take into account the benefit of a materialized view in processing possible subsequent queries of the current query, called the *future usability* of the materialized view. We present another cache management policy for the results of OLAP queries, termed the *Lowest-Usability-First for Future queries* (*LUF-Future*), which exploits the future usability of MVs. It predicts subsequent queries that can be answered using each MV in the cache and calculates the profits of the MV in processing the queries. Since it is not practical to consider all the possible subsequent queries of the current one, we select a set of representative queries that can be processed using the MV, based on query types, aggregation granularities, and query regions for the MV, and then compute the profits using them. We also consider the probabilities of the representative queries to estimate the future usability of MVs.

4.1. Classification of the subsequent queries

We assume that the aggregation granularities of two consecutive queries in the same OLAP session are ordered by the computation dependency. According to the order between them, the subsequent query can be a drill-down or roll-up query. A drill-down (roll-up) subsequent query has an aggregation granularity finer (coarser) than that of the preceding one. We also consider relationships between their selection granularities and selection regions to classify the subsequent OLAP queries.

Definition 1. Let q' be a subsequent query of the query q.

(1) q' is a non-selective drill-down (ND) query from q if
$$\mathrm{AG}(q') < \mathrm{AG}(q), \quad \mathrm{SG}(q') = \mathrm{SG}(q) \quad \text{and} \quad R(q') = R(q).$$

(2) q' is a selective drill-down (SD) query from q if
$$\mathrm{AG}(q') < \mathrm{AG}(q), \quad \mathrm{SG}(q') \leqslant \mathrm{SG}(q), \quad R(q') \subset R(q),$$
$$\mathrm{SG}(q', i) < \mathrm{SG}(q, i) \rightarrow R(q', i) \subset R(q, i) \quad \text{and} \quad R(q', i) \subset R(q, i) \rightarrow \mathrm{SG}(q', i) = \mathrm{AG}(q, i).$$

(3) q' is a non-selective roll-up (NR) query from q if
$$\mathrm{AG}(q') > \mathrm{AG}(q), \quad \mathrm{SG}(q') = \mathrm{SG}(q) \quad \text{and} \quad R(q') = R(q).$$

(4) q' is a selective roll-up (SR) query from q if
$$\mathrm{AG}(q') > \mathrm{AG}(q), \quad \mathrm{SG}(q') \geqslant \mathrm{SG}(q), \quad R(q') \supset R(q),$$
$$\mathrm{SG}(q', i) > \mathrm{SG}(q, i) \rightarrow R(q', i) \supset R(q, i) \quad \text{and} \quad R(q', i) \supset R(q, i) \rightarrow \mathrm{AG}(q', i) = \mathrm{SG}(q, i).$$

Non-selective drill-down (roll-up) queries decrease (increase) the AG of the preceding query in the DH lattice without changing its SG and selection region. Selective drill-down queries restrict the range of values of some group-by attributes in the result of the preceding query q and then decrease its AG. Hence, they have a smaller selection region and a finer AG than q. The SG also decreases if users reduce the selection region of q on a dimension level in the AG of

q which is lower than the level of the same dimension in the SG of q. Therefore, the set of all possible selection granularities of the selective drill-down queries is $\mathrm{SG_{SD}}(q) = \{n(n_1, n_2, \ldots, n_d) | n_i \in \{\mathrm{SG}(q,i), \mathrm{AG}(q,i)\}\}$.

Selective roll-up queries, on the other hand, expand the range of values of some aggregation attributes in the result of the preceding query q and increase the AG. Hence, they have a larger selection region and a coarser AG than q. During navigation through the DH lattice in an OLAP session, users often issue a selective roll-up query to return to the state in which they have executed a selective drill-down query. We assume that if a selective roll-up query q' expands the selection region of the preceding query q on a level of a dimension higher than that of the same dimension in the SG of q, the dimension level of the same dimension in the AG of q' should be equal to the one in the SG of q. The set of all possible AGs of the selective roll-up queries is defined as

$$\mathrm{AG_{SR}}(q) = \{n(n_1, n_2, \ldots, n_d) | \mathrm{AG}(q,i) \leqslant n_i \leqslant \mathrm{SG}(q,i)(1 \leqslant i \leqslant d) \text{ and there exists } j \text{ s.t. } n_j = \mathrm{SG}(q,j)\}$$

We also denote the sets of all possible AGs of a drill-down query and a non-selective roll-up query by $\mathrm{AG_{DD}}(q) = \{n | n < \mathrm{AG}(q)\}$ and $\mathrm{AG_{NR}}(q) = \{n | \mathrm{AG}(q) < n \leqslant \mathrm{SG}(q)\}$, respectively.

Example 1. Consider a DW with a star schema shown in Fig. 6 and assume that the dimensions in a node in the DH lattice are in order of Part, Customer, Supplier, and Time. Suppose that a user issues an OLAP query q_1 in a canonical form where $\mathrm{SG}(q_1) = (\mathsf{none}, \mathsf{nation}, \mathsf{none}, \mathsf{year})$, $Rq_1 = \{((-\infty, +\infty), [\text{'Korea'}, \text{'Korea'}], (-\infty, +\infty), [1991, 2000])\}$, $\mathrm{AG}(q_1) = (\mathsf{brand}, \mathsf{city}, \mathsf{none}, \mathsf{month})$, $\mathrm{AGG}(q_1) = \{\mathsf{SUM(price)}\}$, and $\mathrm{HAV}(q_1) = \mathsf{null}$. The query asks for the total price of the parts ordered from Korea between 1991 and 2000 aggregated by the brand of the parts, the city that the customers are located in, and the month when the orders were placed.

On the result of q_1 the user can generate a non-selective drill-down query q_2 that performs drill-down along the Customer dimension to get the total price of the same parts aggregated by each brand, customer, and month. It has the aggregation granularity $\mathrm{AG}(q_2) = (\mathsf{brand}, \mathsf{name}, \mathsf{none}, \mathsf{month})$ which is finer than that of q_1 while the SG, R, AGG, and HAV of q_2 are the same as those of q_1. If the user rolls up the result of q_2 by one level along the Part dimension, the next query q_3 will be a non-selective roll-up query from q_2 which computes the total price of the same parts aggregated by each group of the parts, customer, and month. We have $\mathrm{AG}(q_3) = (\mathsf{group}, \mathsf{name}, \mathsf{none}, \mathsf{month})$ and the other components are the same as those of q_2.

Now suppose that the user wants to focus on the parts ordered only from July to December in year 2000 and compute their total price aggregated by each group, customer, and day of order. He will generate a selective drill-down query q_4 from q_3 which reduces the selection region of q_3 over the Time dimension and drills down by one level along the same dimension. We have $\mathrm{SG}(q_4) = (\mathsf{none}, \mathsf{nation}, \mathsf{none}, \mathsf{month})$, $R(q_4) = \{((-\infty, +\infty), [\text{'Korea'}, \text{'Korea'}], (-\infty, +\infty), [2000/7, 2000/12])\}$, and $\mathrm{AG}(q_4) = (\mathsf{group}, \mathsf{name}, \mathsf{none}, \mathsf{day})$. Finally, if the user intends to return to the previous result of analysis, he will issue the query q_3 again, which will perform a selective roll-up from q_4.

The other types of queries popular in OLAP applications include slice&dice and pivot queries (Chaudhuri and Dayal, 1997). The slice&dice queries slice the result of the preceding query with a slicing value on a particular dimension and then aggregate the data in the hyper-plane by some levels of the other dimensions. They can be answered by performing a local operation over the previous query results in the front-end OLAP tools or can be regarded as the selective drill-down queries in Definition 1. The pivot queries can also be processed by a local operation in the front-end tools.

4.2. Aggregation granularity of the subsequent queries

To calculate the profit of a materialized view in the cache, we only have to consider the subsequent queries that can be rewritten using the MV. We find the set of possible AGs that such queries can have. We first identify three sets of nodes in the DH lattice for the AGs of different types of subsequent queries.

Observation 1

(1) Given a query q and a materialized view mv satisfying $\mathrm{AG}(\mathrm{mv}) < \mathrm{AG}(q)$, let $\mathrm{AG_{DD}}(q, \mathrm{mv}) = \{n | \mathrm{AG}(\mathrm{mv}) \leqslant n < \mathrm{AG}(q)\}$. If q' is a drill-down query from q which satisfies $\mathrm{AG}(q') \geqslant \mathrm{AG}(\mathrm{mv})$, there exist a node n in $\mathrm{AG_{DD}}(q, \mathrm{mv})$ such that $n = \mathrm{AG}(q')$, and for all n in $\mathrm{AG_{DD}}(q, \mathrm{mv})$, there exists a drill-down query q' from q which satisfies $\mathrm{AG}(q') = n$.

(2) Given a query q and a materialized view mv satisfying $\mathrm{AG}(\mathrm{mv}) \leqslant \mathrm{SG}(q)$, let $\mathrm{AG_{NR}}(q, \mathrm{mv}) = \{n | \mathrm{LUB}(\mathrm{AG}(\mathrm{mv}), \mathrm{AG}(q)) \leqslant n \leqslant \mathrm{SG}(q), n \neq \mathrm{AG}(q)\}$. If q' is a non-selective roll-up query from q which satisfies $\mathrm{AG}(q') \geqslant \mathrm{AG}(\mathrm{mv})$, there exist a node n in $\mathrm{AG_{NR}}(q, \mathrm{mv})$ such that $n = \mathrm{AG}(q')$, and for all n in $\mathrm{AG_{NR}}(q, \mathrm{mv})$, there exists a non-selective roll-up query q' from q which satisfies $\mathrm{AG}(q') = n$.

(3) Given a query q and a materialized view mv satisfying $\mathrm{AG}(\mathrm{mv}) \leqslant \mathrm{SG}(q)$, let $\{\mathrm{AG}_{\mathrm{SR}}(q,\mathrm{mv},i) = \{n(n_1, n_2, \ldots, n_d) | n \in \mathrm{AG}_{\mathrm{NR}}(q,\mathrm{mv}), n_i = \mathrm{SG}(q,i)\}$ for a dimension d_i and $\mathrm{AG}_{\mathrm{SR}}(q,\mathrm{mv}) = \cup_{1 \leqslant i \leqslant d} \mathrm{AG}_{\mathrm{SR}}(q,\mathrm{mv},i)$. If q' is a selective roll-up query from q which satisfies $\mathrm{AG}(q') \geqslant \mathrm{AG}(\mathrm{mv})$, there exist a node n in $\mathrm{AG}_{\mathrm{SR}}(q,\mathrm{mv})$ such that $n = \mathrm{AG}(q')$, and for all n in $\mathrm{AG}_{\mathrm{SR}}(q,\mathrm{mv})$, there exists a selective roll-up query q' from q which satisfies $\mathrm{AG}(q') = n$.

The above observation, which can be proven by Definition 1, implies that $\mathrm{AG}_{\mathrm{DD}}(q,\mathrm{mv})$, $\mathrm{AG}_{\mathrm{NR}}(q,\mathrm{mv})$, and $\mathrm{AG}_{\mathrm{SR}}(q,\mathrm{mv})$ contain all the granularities that three types of subsequent queries rewritable using mv can have as their AGs while they exclude any other granularities. Therefore, we only consider the granularities in those sets to find the AGs of the subsequent queries that may have any effect on the future usability of mv. Fig. 3 shows $\mathrm{AG}_{\mathrm{DD}}(q,\mathrm{mv})$, $\mathrm{AG}_{\mathrm{NR}}(q,\mathrm{mv})$, and $\mathrm{AG}_{\mathrm{SR}}(q,\mathrm{mv})$ for the example queries and MVs satisfying $\mathrm{AG}(\mathrm{mv}) < \mathrm{AG}(q)$ or $\mathrm{AG}(\mathrm{mv}) <> \mathrm{AG}(q)$.

4.3. Selection granularity and selection region of the subsequent queries

The selection granularity and selection region of a non-selective drill-down or roll-up query are identical to those of the preceding query. As for a selective drill-down or roll-up query, however, there are potentially too many different SGs and selection regions that the query can have. We cannot consider all the SGs and selection regions, and it is also not fair to choose SGs and selection regions for representative subsequent queries randomly. We note that the cost of processing a query using an MV is dependent on the size of the query region for the MV as given in the cost model in Section 3. Therefore, we estimate the size of the overlap between selection regions of a subsequent query and an MV using the selection regions of the preceding query and the MV. To reduce the number of representative subsequent queries, we consider the selective queries that restrict or expand the selection region only along a single dimension.

4.3.1. Selective drill-down queries

Let q' be a selective drill-down query from a query q reducing the selection region of q on a dimension d_i. By Definition 1, q' has the selection granularity

$$\mathrm{SG}_{\mathrm{SD}}(q,i) = n(n_1, n_2, \ldots, n_d) \quad \text{where } n_i = \mathrm{AG}(g,i) \text{ and } n_j = \mathrm{SG}(q,j) \quad \text{for all } j \neq i$$

and its selection region is contained in that of q. Thus a materialized mv can be used in rewriting q' only if $R(q) \cap^* R(\mathrm{mv}) \neq \phi$.

Suppose that q' can be rewritten using mv. In order to calculate the expected size of the query region for mv, we first compute the expected length of the range reduced on the dimension d_i in the query region for mv (see Fig. 4(a)). Since the restrictive selection is performed on the aggregation result of the preceding query, the reduced range $R(q',i)$ of q' in the dimension d_i is on the dimension level contained in the aggregation granularity of the preceding query, i.e., $\mathrm{AG}(q,i)$. We assume that the end points of $R(q',i)$ are independent and uniformly distributed over the selection range $R(q,i)$ of q in the same dimension. We let $R(q,i) = (s_1 e_1)$, $R(q,i) \cap R(\mathrm{mv},i) = (s_2 e_2)$, $l_1 = e_1 - s_1$ and $l_2 = e_2 - s_2$ and denote the length of the range on the dimension d_i in the query region of q' for mv by

$$L_{\mathrm{SD}}(q,\mathrm{mv},i) = \mathrm{size}(R(q',i) \cap R(\mathrm{mv},i))$$

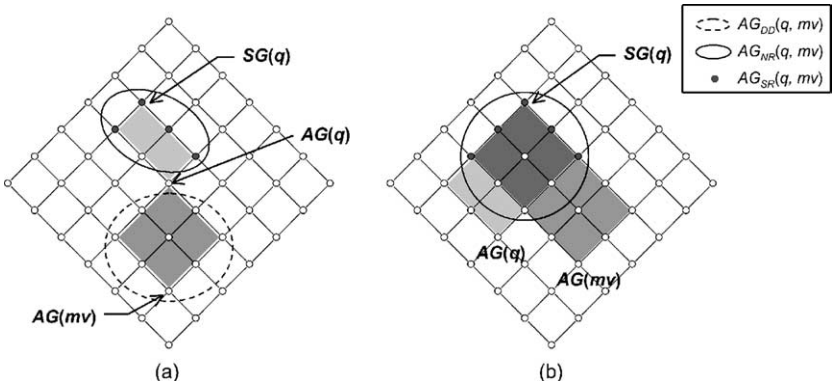

Fig. 3. The AGs of the subsequent queries which can be rewritten using mv: (a) $\mathrm{AG}(\mathrm{mv}) < \mathrm{AG}(q)$, (b) $\mathrm{AG}(\mathrm{mv}) <> \mathrm{AG}(q)$.

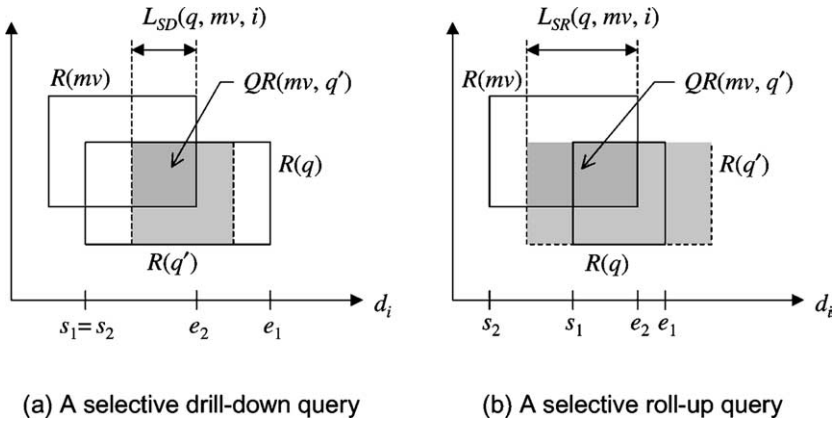

Fig. 4. The query region of a selective subsequent query rewritable using an MV.

Then its expected value can be obtained by

$$\mathrm{EL}_{\mathrm{SD}}(q,\mathrm{mv},i) = E[L_{\mathrm{SD}}(q,\mathrm{mv},i)] = \frac{l_2^2}{3l_1^2}(l_2 + 3(s_2 - s_1) + 3(e_1 - e_2) + 6(s_2 - s_1)(e_1 - e_2))$$

as derived by Park et al. (2001b). Consequently, the expected size of the query region of q' for mv which is reduced on the dimension d_i can be computed as

$$\mathrm{EA}_{\mathrm{SD}}(q,\mathrm{mv},i) = \mathrm{EL}_{\mathrm{SD}}(q,\mathrm{mv},i) \cdot \prod_{j \neq i} \mathrm{length}(R(q,j) \cap R(\mathrm{mv},j))$$

where $\mathrm{length}(R(q,j) \cap R(\mathrm{mv},j))$ is the length of the range on the dimension d_j in $R(q) \cap^* R(\mathrm{mv})$.

For specifying a representative subsequent query of q in Section 4.4, we assume that q' has a selection region denoted by $R_{\mathrm{SD}}(q,\mathrm{mv},i)$ satisfying

$$L_{\mathrm{SD}}(q,\mathrm{mv},i) = \mathrm{EL}_{\mathrm{SD}}(q,\mathrm{mv},i)$$

4.3.2. Selective roll-up queries

Let q' be a subsequent selective roll-up query from a query q expanding the selection region of q on a dimension d_i. The selection region of q' contains that of q as defined in Definition 1. In general, there are too many possible selection granularities that q' can have, and we assume for simplicity that q has the same selection granularity as q.

Like the selective drill-down queries, we first calculate the expected length of the range expanded on the dimension d_i in the query region of q' for mv (see Fig. 4(b)). We assume that the end points of the selection range $R(q',i)$ of q' in the dimension d_i are jointly uniformly distributed over the range *outside* the selection range $R(q,i)$ of q in the same dimension. We suppose that $R(q,i) = (s_1,e_1)$, $R(\mathrm{mv},i) = (s_2,e_2)$, $l_1 = e_1 - s_1$, $l_2 = e_2 - s_2$, and l be the domain size of the dimension d_i. If we denote the length of the range on the dimension d_i in the query region of q' for mv by

$$L_{\mathrm{SR}}(q,\mathrm{mv},i) = \mathrm{size}(R(q',i) \cap R(\mathrm{mv},i))$$

then its expected value can be obtained according to the relative locations of $R(q,i)$ and $R(\mathrm{mv},i)$ as follows:

$$\mathrm{EL}_{\mathrm{SR}}(q,\mathrm{mv},i) = E[L_{\mathrm{SR}}(q,\mathrm{mv},i)]$$

$$= \begin{cases} \dfrac{1}{2s_1}(e_2^2 - s_2^2), & \text{if } s_2 \leqslant e_2 < s_1 \leqslant e_1 \\ e_2 - \dfrac{s_1}{2} - \dfrac{s_2^2}{2s_1}, & \text{if } s_2 < s_1 \leqslant e_2 \leqslant e_1 \\ \dfrac{(ac+bc+bd)(l_1+l_2)+bc(a+d)+2adl_2}{2(a+b)(c+d)}, & \\ \quad \text{where } a = s_2, b = s_1 - s_2, c = e_2 - e_1, \text{ and } d = l - e_2 & \text{if } s_2 < s_1 \leqslant e_1 < e_2 \\ e_2 - s_2, & \text{if } s_1 \leqslant s_2 \leqslant e_2 \leqslant e_1 \\ \dfrac{1}{l-e_1}\left(e_2 l - \dfrac{1}{2}e_1^2 - \dfrac{1}{2}e_2^2\right) - s_2, & \text{if } s_1 \leqslant s_2 \leqslant e_1 < e_2 \\ \dfrac{e_2 - s_2}{l - e_1}\left(l - \dfrac{1}{2}(s_2 + e_2)\right), & \text{if } s_1 \leqslant e_1 < s_2 \leqslant e_2 \end{cases}$$

The detailed development is shown by Park et al. (2001b). Consequently, the expected size of the query region of q' for mv that is expanded on the dimension d_i can be computed as

$$\mathrm{EA}_{\mathrm{SR}}(q,\mathrm{mv},i) = \mathrm{EL}_{\mathrm{SR}}(q,\mathrm{mv},i)\cdot\prod_{j\neq i}\mathrm{length}(R(q,j)\cap R(\mathrm{mv},j))$$

For specifying a representative subsequent query of q in Section 4.4, we assume that q' has a selection region denoted by $R_{\mathrm{SR}}(q,\mathrm{mv},i)$ satisfying

$$L_{\mathrm{SR}}(q,\mathrm{mv},i) = \mathrm{EL}_{\mathrm{SR}}(q,\mathrm{mv},i)$$

4.4. The representative subsequent queries

For each materialized view in the cache, the set of representative subsequent queries of the current query is defined using the AGs, SGs, and query regions described in the previous sections.

Definition 2. The sets of the representative subsequent queries with different query types are defined as follows:

$$\mathrm{ND}(q,\mathrm{mv}) = \begin{cases} \{q'(\mathrm{SG},R,\mathrm{AG},\mathrm{AGG},\mathrm{HAV})|\mathrm{SG}=\mathrm{SG}(q), & \\ R=R(q),\mathrm{AG}\in\mathrm{AG}_{\mathrm{DD}}(q,\mathrm{mv}), & \\ \mathrm{AGG}=\mathrm{AGG}(q),\mathrm{HAV}=\mathrm{HAV}(q)\}, & \text{if } R(q)\cap^{*}R(\mathrm{mv})\neq\phi \quad\text{and}\quad \mathrm{AGG}(q)\subseteq\mathrm{AGG}(\mathrm{mv}) \\ \phi, & \text{otherwise} \end{cases}$$

$$\mathrm{NR}(q,\mathrm{mv}) = \begin{cases} \{q'(\mathrm{SG},R,\mathrm{AG},\mathrm{AGG},\mathrm{HAV})|\mathrm{SG}=\mathrm{SG}(q), & \\ R=R(q),\mathrm{AG}\in\mathrm{AG}_{\mathrm{NR}}(q,\mathrm{mv}), & \\ \mathrm{AGG}=\mathrm{AGG}(q), & \\ \mathrm{HAV}=\mathrm{HAV}(q)\}, & \text{if } R(q)\cap^{*}R(\mathrm{mv})\neq\phi \quad\text{and}\quad \mathrm{AGG}(q)\subseteq\mathrm{AGG}(\mathrm{mv}) \\ \phi, & \text{otherwise} \end{cases}$$

$$\mathrm{SD}(q,\mathrm{mv}) = \begin{cases} \{q'(\mathrm{SG},R,\mathrm{AG},\mathrm{AGG},\mathrm{HAV})|\mathrm{SG}=\mathrm{SG}_{\mathrm{SD}}(q,i), & \\ R=R_{\mathrm{SD}}(q,\mathrm{mv},i), & \\ \mathrm{AG}\in\mathrm{AG}_{\mathrm{DD}}(q,\mathrm{mv}), & \\ \mathrm{AGG}=\mathrm{AGG}(q), & \\ \mathrm{HAV}=\mathrm{HAV}(q)(1\leqslant i\leqslant d)\}, & \text{if } R(q)\cap^{*}R(\mathrm{mv})\neq\phi,\mathrm{EA}_{\mathrm{SD}}(q,\mathrm{mv},i)>0 \\ & \quad\text{and}\quad \mathrm{AGG}(q)\subseteq\mathrm{AGG}(\mathrm{mv}) \\ \phi, & \text{otherwise} \end{cases}$$

$$\mathrm{SR}(q,\mathrm{mv}) = \begin{cases} \{q'(\mathrm{SG},R,\mathrm{AG},\mathrm{AGG},\mathrm{HAV})|\mathrm{SG}=\mathrm{SG}(q), & \\ R=R_{\mathrm{SR}}(q,\mathrm{mv},i),\mathrm{AG}\in\mathrm{AG}_{\mathrm{SR}}(q,\mathrm{mv},i), & \\ \mathrm{AGG}=\mathrm{AGG}(q),\mathrm{HAV}=\mathrm{HAV}(q)(1\leqslant i\leqslant d)\}, & \text{if } \mathrm{EA}_{\mathrm{SR}}(q,\mathrm{mv},i)>0 \quad\text{and}\quad \mathrm{AGG}(q)\subseteq\mathrm{AGG}(\mathrm{mv}) \\ \phi, & \text{otherwise} \end{cases}$$

The set of the representative subsequent queries of q for mv is defined as the union of the above sets and denoted by $\mathrm{SQ}(q,\mathrm{mv})$. We note that mv is usable for processing all the queries in $\mathrm{SQ}(q,\mathrm{mv})$.

Theorem 1. *Given a query q and a materialized view mv, if a subsequent query q′ from q is contained in SQ(q, mv), q′ can be rewritten using mv.*

Proof. By Observation 1, every node n in either $\mathrm{AG}_{\mathrm{DD}}(q,\mathrm{mv})$, $\mathrm{AG}_{\mathrm{NR}}(q,\mathrm{mv})$, or $\mathrm{AG}_{\mathrm{SR}}(q,\mathrm{mv})$ satisfies $\mathrm{AG}(\mathrm{mv})\leqslant n$. Thus $\mathrm{AG}(\mathrm{mv})\leqslant\mathrm{AG}(q')$ for all $q'\in SQ(q,\mathrm{mv})$. For every query q' in $ND(q,\mathrm{mv})$ or $\mathrm{NR}(q,\mathrm{mv})$, $R(q')\cap^{*}R(\mathrm{mv})\neq\phi$ since $R(q')=R(q)$ and $R(q)\cap^{*}R(\mathrm{mv})\neq\phi$. For every query q' in $\mathrm{SD}(q,\mathrm{mv})$ or $\mathrm{SR}(q,\mathrm{mv})$, $R(q')\cap^{*}R(\mathrm{mv})\neq\phi$ because $\mathrm{EA}_{\mathrm{SD}}(q,\mathrm{mv},i)>0$ and $\mathrm{EA}_{\mathrm{SR}}(q,\mathrm{mv},i)>0$. Finally, for all query $q'\in SQ(q,\mathrm{mv}),\mathrm{AGG}(q')\subseteq\mathrm{AGG}(\mathrm{mv})$ since $\mathrm{AGG}(q')=\mathrm{AGG}(q)$ and $\mathrm{AGG}(q)\subseteq\mathrm{AGG}(\mathrm{mv})$. As a result, the condition (1) in Section 2.2 holds for all queries in $SQ(q,\mathrm{mv})$, and they can be rewritten using mv. □

4.5. Computing future usability

The future usability of a materialized view means the expected profit we can get when we rewrite and process the representative subsequent queries of the current query using the materialized view. Given a representative subsequent query q' for the current query q, let $\text{profit}(\text{mv}, q')$ be the profit of mv for q' and $\Pr(q')$ be the probability of occurrence of q' as the subsequent query of q. Then, the future usability of mv for q is defined as follows:

$$\text{future-usability}(\text{mv}, q) = \sum_{q' \in \text{SQ}(q,\text{mv})} \text{profit}(\text{mv}, q') \cdot \Pr(q')$$

As described in Section 4.3, a selective drill-down and roll-up query in $\text{SQ}(q, \text{mv})$ is defined by the expected size of the query region for the MV rather than by a particular selection region of the query. Hence, we redefine the profit measure presented in Section 3 as the function of mv and the size σ of the query region of q' for mv as follows:

$$\text{profit}'(\text{mv}, \sigma) = \min\left\{ n_{\text{ft}} \cdot \frac{\sigma}{\text{size}(R(\text{ft}))}, N_{\text{ft}} \right\} - N_{\text{mv}}$$

where $\sigma = \text{size}(R(q') \cap^* R(\text{mv}))$.

We assume that the dimension on which the selection region is reduced or expanded in subsequent selective queries is chosen uniformly among all dimensions. The probability of a subsequent query is dependent on its query type and aggregation granularity and it can be estimated by investigating and analyzing the query sessions executed in the past in OLAP systems. We use a simple probability model for subsequent queries in this section. Considering OLAP applications that usually perform drill-down and roll-up operations along dimension hierarchies step by step, we suppose that the probability of occurrence of the AG of the subsequent query is closely related with its distance from the AG of the current query in the DH lattice, which implies the drill-down depth or roll-up height of the subsequent query. We define the distance D of the AG of a subsequent query q' from that of the preceding query q as

$$D = |\text{AG}(q'), \text{AG}(q)| = \sum_{1 \leqslant i \leqslant d} |\text{AG}(q', i), \text{AG}(q, i)|$$

We assume that the distance follows a power law distribution with the exponent 1, i.e., $\Pr(D = d) \sim d^{-1}$ and consider the conditional probability of the distance between two AGs given a type of the subsequent query. For example, if the subsequent query is known to be a drill-down query, the probability can be obtained by

$$\Pr(|\text{AG}(q'), \text{AG}(q)| = d | q' \text{ is a drill-down query from } q) = \begin{cases} \dfrac{d^{-1}}{\displaystyle\sum_{1 \leqslant d_i \leqslant |L, \text{AG}(q)|} d_i^{-1}}, & \text{if } 1 \leqslant d \leqslant |L, \text{AG}(q)| \\ 0, & \text{otherwise} \end{cases}$$

where L is the finest granularity in the DH lattice. We also assume that the nodes for $\text{AG}(q')$ in the same distance from $\text{AG}(q)$ are uniformly distributed. For example, if we denote the number of nodes in $\text{AG}_{\text{DD}}(q)$ whose distance from $\text{AG}(q)$ is d by $N_{\text{DD}}(q, d)$, we can compute the conditional probability of a node n being selected as the AG of a drill-down query from q given that the distance between the AGs of two queries is d as follows:

$$\begin{aligned} &\Pr(\text{AG}(q') = n | |\text{AG}(q'), \text{AG}(q)| = d, q' \text{ is a drill-down query from } q) \\ &\quad = \begin{cases} \dfrac{1}{N_{\text{DD}}(q, d)}, & \text{if } n \in \text{AG}_{\text{DD}}(q) \quad \text{and} \quad 1 \leqslant d \leqslant |L, \text{AG}(q)| \\ 0, & \text{otherwise} \end{cases} \end{aligned}$$

Consequently, we have

$$\begin{aligned} &\Pr(\text{AG}(q') = n | q' \text{ is a drill-down query from } q) \\ &\quad = \Pr(\text{AG}(q') = n | |\text{AG}(q'), \text{AG}(q)|) \\ &\quad = d, q' \text{ is a drill-down query from } q) \cdot \Pr(|\text{AG}(q'), \text{AG}(q)| = d | q' \text{ is a drill-down query from } q) \\ &\quad = \begin{cases} \dfrac{1}{N_{\text{DD}}(q, d)} \cdot \dfrac{d^{-1}}{\displaystyle\sum_{1 \leqslant d_i \leqslant |L, \text{AG}(q)|} d_i^{-1}}, & \text{if } n \in \text{AG}_{\text{DD}}(q) \quad \text{and} \quad 1 \leqslant d \leqslant |L, \text{AG}(q)| \\ 0, & \text{otherwise} \end{cases} \end{aligned}$$

Probabilities for the AGs of the non-selective and selective roll-up queries can be computed in a similar way. That is,

$$\Pr(\mathrm{AG}(q') = n | q' \text{ is a non-selective roll-up query from } q) = \begin{cases} \dfrac{1}{N_{\mathrm{NR}}(q,d)} \cdot \dfrac{d^{-1}}{\sum_{1 \leqslant d_i \leqslant |\mathrm{AG}(q),\mathrm{SG}(q)|} d_i^{-1}}, & \text{if } n \in \mathrm{AG}_{\mathrm{NR}}(q) \quad \text{and} \quad 1 \leqslant d \leqslant |\mathrm{AG}(q),\mathrm{SG}(q)| \\ 0, & \text{otherwise} \end{cases}$$

$$\Pr(\mathrm{AG}(q') = n | q' \text{ is a selective roll-up query from } q) = \begin{cases} \dfrac{1}{N_{\mathrm{SR}}(q,d)} \cdot \dfrac{d^{-1}}{\sum_{m \leqslant d_i \leqslant |\mathrm{AG}(q),\mathrm{SG}(q)|} d_i^{-1}}, & \text{if } n \in \mathrm{SG}_{\mathrm{SR}}(q) \quad \text{and} \quad m \leqslant d \leqslant |\mathrm{AG}(q),\mathrm{SG}(q)| \\ 0, & \text{otherwise} \end{cases}$$

where $N_{\mathrm{NR}}(q,d)$ and $N_{\mathrm{SR}}(q,d)$ denote the number of nodes in $\mathrm{AG}_{\mathrm{NR}}(q)$ and $\mathrm{AG}_{\mathrm{SR}}(q)$ respectively, whose distance from $\mathrm{AG}(q)$ is d, and $m = \min_{n \in \mathrm{AG}_{\mathrm{SR}}(q)} |\mathrm{AG}(q), n|$.

The future usability can be computed using the redefined profit function and the assumed probability distribution of the subsequent queries as follows:

$$\begin{aligned}
&\text{future-usability}(\mathrm{mv}, q) \\
&= \sum_{q' \in SQ(q,\mathrm{mv})} \mathrm{profit}(\mathrm{mv}, q') \cdot \Pr(q') \\
&= \mathrm{profit}'(\mathrm{mv}, \mathrm{size}(R(q) \cap^* R(\mathrm{mv}))) \cdot \Pr(q' \text{is a ND query}) \cdot \sum_{n \in \mathrm{AG}_{\mathrm{DD}}(q,\mathrm{mv})} \frac{1}{N_{\mathrm{DD}}(q, |n, \mathrm{AG}(q)|)} \cdot \frac{|n, \mathrm{AG}(q)|^{-1}}{\sum_{1 \leqslant d_i \leqslant |L,\mathrm{AG}(q)|} d_i^{-1}} \\
&\quad + \mathrm{profit}'(\mathrm{mv}, \mathrm{size}(R(q) \cap^* R(\mathrm{mv}))) \cdot \Pr(q' \text{ is a NR query}) \cdot \sum_{n \in \mathrm{AG}_{\mathrm{NR}}(q,\mathrm{mv})} \frac{1}{N_{NR}(q, |n, \mathrm{AG}(q)|)} \cdot \frac{|n, \mathrm{AG}(q)|^{-1}}{\sum_{1 \leqslant d_i \leqslant |\mathrm{AG}(q),\mathrm{SG}(q)|} d_i^{-1}} \\
&\quad + \sum_{1 \leqslant i \leqslant d} (\mathrm{profit}'(\mathrm{mv}, \mathrm{EA}_{\mathrm{SD}}(q, \mathrm{mv}, i))) \cdot \Pr(q' \text{ is a SD query}) \cdot \sum_{n \in \mathrm{AG}_{\mathrm{DD}}(q,\mathrm{mv})} \frac{1}{N_{\mathrm{DD}}(q, |n, \mathrm{AG}(q)|)} \cdot \frac{|n, \mathrm{AG}(q)|^{-1}}{\sum_{1 \leqslant d_i \leqslant |L,\mathrm{AG}(q)|} d_i^{-1}} \\
&\quad + \sum_{1 \leqslant i \leqslant d} \left(\mathrm{profit}'(\mathrm{mv}, \mathrm{EA}_{\mathrm{SR}}(q, \mathrm{mv}, i)) \cdot \Pr(q' \text{ is a SR query}) \cdot \sum_{n \in \mathrm{AG}_{\mathrm{SR}}(q,\mathrm{mv},i)} \frac{1}{N_{\mathrm{SR}}(q, |n, \mathrm{AG}(q)|)} \cdot \frac{|n, \mathrm{AG}(q)|^{-1}}{\sum_{m \leqslant d_i \leqslant |\mathrm{AG}(q),\mathrm{SG}(q)|} d_i^{-1}} \right)
\end{aligned}$$

The detailed derivation is shown by Park et al. (2001b).

4.6. An algorithm based on future usability

The outline of a cache management algorithm based on the LUF-Future policy is given in Fig. 5. In the algorithm, future usability normalized for the size of a materialized view is computed for all MVs in the cache and used as a goodness measure for selecting MVs to replace with the result of a current query. At each selection of an MV, if the normalized future usability of the current query result is lower than the sum of those of the query results previously selected as replacement victims, we do not admit the current query result into the cache and maintain all the previous MVs.

5. Performance evaluation

In this section, we present the results of experiments that we have conducted to evaluate the performance of our caching schemes and compare them with the previous caching algorithms for DWs and OLAP systems.

5.1. Experimental setup and query workloads

We implemented a cache system employing the proposed caching schemes and conducted experiments on a SUN workstation with an UltraSPARC-II 450 MHz processor and 1 GB main memory, running Solaris 2.7. In the

Procedure: LUF-Future Cache Management Algorithm

Input: the current query q_c and the result mv_c of q_c

begin

MV := the set of materialized views currently in the cache;

V := ∅;

sum_of_utilities := 0;

if $size(mv_c)$ > CACHE_SIZE **then**

return; // do not admit mv_c into the cache

end if;

for each mv_i in MV **do**

Compute the utility of mv_i, $utility(mv) = \dfrac{future - usability(mv_i, q_c)}{size(mv_i)}$;

end for;

Compute the utility of mv_c;

while $size$(MV) + $size(mv_c)$ > CACHE_SIZE **do**

Select mv_i in MV that has the minimum utility value, i.e.,

$$utility(mv_i) = \min_{mv_j \in MV} utility(mv_j);$$

if sum_of_utilities + $utility(mv_i) \geq utility(mv_c)$ **then**

return; // do not admit mv_c into the cache

end if;

MV := MV − $\{mv_i\}$;

V := V ∪ $\{mv_i\}$;

sum_of_utilities := sum_of_utilities + $utility(mv_i, q_c)$;

end while;

Evict the MVs in V from the cache;

Insert mv_c into the cache;

end

Fig. 5. The LUF-Future cache management algorithm.

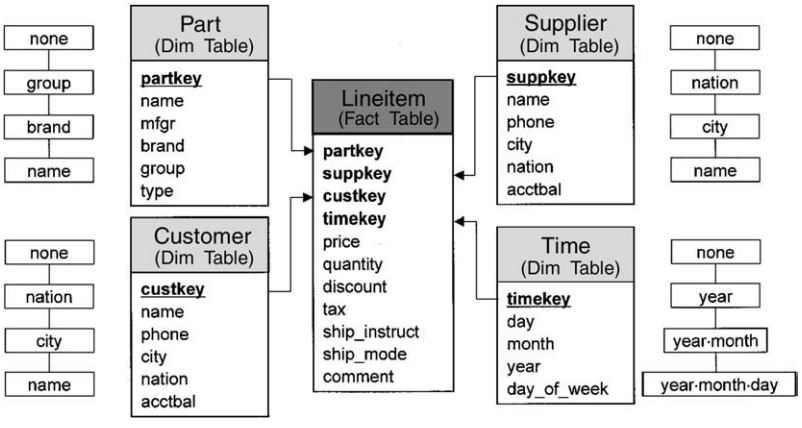

Fig. 6. An experimental star schema from TPC-H benchmark.

experiments we used a sub-set of the TPC-H schema included in the TPC-H benchmark (TPC, 1999) to design a star schema consisting of one fact table Lineitem and 4 dimension tables as shown in Fig. 6. The fact table has 64 M tuples and its total size is about 8.9 GB. The values of a foreign key attribute referring to a dimension table are uniformly distributed over the domain of the attribute. Each dimension table has a dimension hierarchy as given in the figure. The fan-outs along the dimension hierarchies are 10 for all dimensions except the Time dimension. In this configuration the size of the full data cube, i.e., the total size of all views in the DH lattice becomes about 184 GB. We carried out experiments with the cache size ranging from 1% to 20% of the data cube size. The size of a disk page was set to 8 KB.

We tested the caching schemes under three different query workloads for OLAP systems, each of which has 3000 OLAP queries in canonical forms.

- Random queries: This workload consists of randomly generated OLAP queries. The SGs and AGs of the queries are uniformly distributed over all granularities in the DH lattice and the selection regions are randomly chosen on the SG of the queries under the constraint that 50% of the selection ranges should be points. All the queries compute SUM of the measure attribute price in the fact table and have no HAVING condition.
- Single-user session queries: This workload consists of a sequence of OLAP sessions which have a start query and several consecutive subsequent queries. The start query is generated like the queries in the random workload. The subsequent query is one of four types of drill-down and roll-up queries in Definition 1, which have the same probability, and their aggregation granularities and selection regions have the probabilities described in Section 4. The length of an OLAP session follows a normal distribution with a mean of 10 queries and a standard deviation of 2 queries.
- Multi-user session queries: In this workload we simulate an environment where multiple users issue consecutive queries in different OLAP sessions independently and concurrently. The queries from different users are executed in an interleaved fashion. We performed experiments for 5, 10, and 15 concurrent users issuing queries in an OLAP system.

5.2. Compared algorithms and performance metric

We implemented the LUF-Past and LUF-Future schemes, as well as their combination called the *Lowest-Usability-First* (*LUF*) that exploits the usability for recent and future queries together. The outline of the LUF caching algorithm is shown by Park et al. (2001b). In the experiments, we have compared our caching schemes with the following previous ones.

- The least normalized cost replacement and admission (LNC-RA) scheme (Scheuermann et al., 1996) uses a goodness measure defined as

$$\text{goodness}_{\text{LNC}}(v) = \frac{\text{ref\_rate}(v) \cdot \text{cost}_{\text{LNC}}(v)}{\text{size}(v)}$$

  where $\text{ref\_rate}(v)$ is the average rate of reference to a query v, $\text{cost}_{\text{LNC}}(v)$ is the cost of evaluating v from the detailed data in the fact table, and $\text{size}(v)$ is the size of the result of v.
- The smaller penalty first (SPF) scheme (Kortidis and Roussopoulos, 1999) selects replacement victims using a measure

$$\text{goodness}_{\text{SPF}}(v) = \frac{\text{freq}(v) \cdot \text{cost}_{\text{SPF}}(v)}{\text{size}(v)}$$

  where $\text{freq}(v)$ is the frequency of access to the result of a query v and $\text{cost}_{\text{SPF}}(v)$ is the cost of re-computing v from the result of its father query, i.e., the MV in the cache which can calculate the result of v most efficiently.
- The lowest benefit first (LBF) policy (Kalnis and Papadias, 2001) uses the goodness of a query result defined by the normalized benefit of caching it for processing data cube queries, i.e.,

$$\text{goodness}_{\text{LBF}}(v) = \frac{\sum_{u \in L} \text{freq}(u) \cdot (\text{cost}_{\text{LBF}}(u, M) - \text{cost}_{\text{LBF}}(u, M \cup \{v\}))}{\text{size}(v)}$$

  where L is the data cube lattice, M is the set of query results in the cache, and $\text{cost}_{\text{LBF}}(u, M)$ is the minimum cost of calculating the query u using a materialized view in M.

As was pointed out in Section 2, all the above caching schemes have some limitations in the kind of queries they can deal with and in the way to process them using MVs in the cache. In order to make fair comparison with our caching schemes, we enhanced them by adopting the query rewriting method we had proposed (Park et al., 2002). In addition,

the measure of the LBF scheme was modified to use the set of k most recently executed queries instead of L like our caching algorithm in Fig. 1.

We employed the *cost saving ratio* as a metric for evaluating the effectiveness of different caching schemes, which is defined as

$$\mathrm{CSR} = \frac{\sum_i (\mathrm{cost}(\mathrm{ft}, R(q_i)) - \sum_{mv_j \in S(q_i)} \mathrm{cost}(mv_j, \mathrm{QR}(mv_j, q_i)))}{\sum_i \mathrm{cost}(\mathrm{ft}, R(q_i))}$$

where $S(q_i)$ is the set of MVs (including the fact table) which are used in rewriting and processing the query q_i. This measure is similar to DCSR used by Kortidis and Roussopoulos (1999) as well as by Kalnis and Papadias (2001), but it considers the amount of cost saved in processing queries using the general query rewriting method.

5.3. Experimental results

In the experiments, we used first 10% of the queries in the test workloads to warm up the cache and measured the cost saving ratio of the caching schemes for the other queries. In implementing the LUF-Past scheme, we have managed 10 most recent queries in Q_k and used a parameter $T = 2$ in the decay function for the weight of the queries.

Fig. 7 shows the result of the experiment for the random query workload. The result of full materialization indicates the performance of processing queries given all queries for the nodes in the DH lattice materialized statically. It provides the upper bound on performance of caching schemes. LUF and LUF-Future show similar results and outperform the other schemes. Their CSRs increase drastically and reach a steady state with the cache size of only 3% of the data cube size. LBF shows better performance than LUF-Past for the most of considered cache sizes. Though the CSRs of all the caching schemes are improved as the cache size increases, they remain relatively low for the random query workload.

Fig. 8 shows the performance of the caching schemes for the single-user OLAP workload. LUF and LUF-Future have very similar results and perform the best among the compared algorithms. Their CSRs reach about 88.7% and 89.6% of that of the full materialization respectively with the cache size of only 4% of the data cube size. It is due to close prediction of future queries and exploiting the future usability to manage query results for the sequence of session queries.

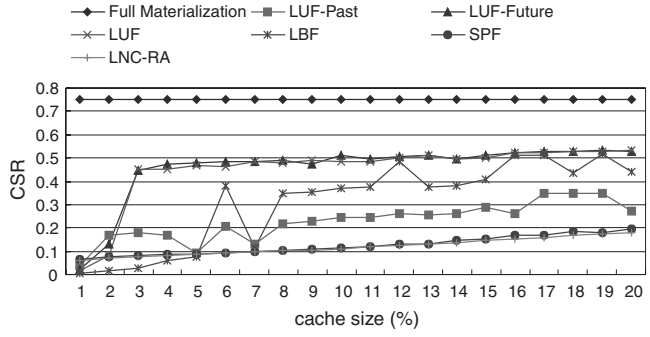

Fig. 7. Performance of the caching schemes for the random query workload.

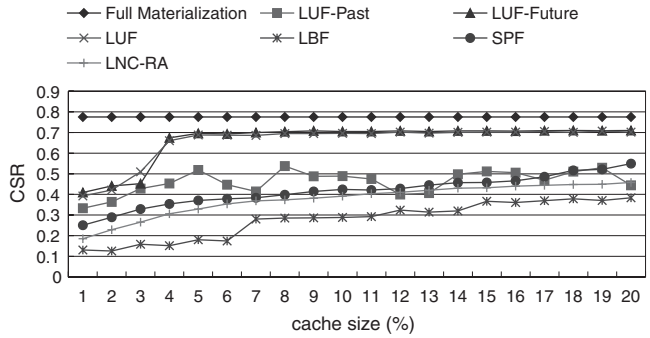

Fig. 8. Performance of the caching schemes for the single-user session query workload.

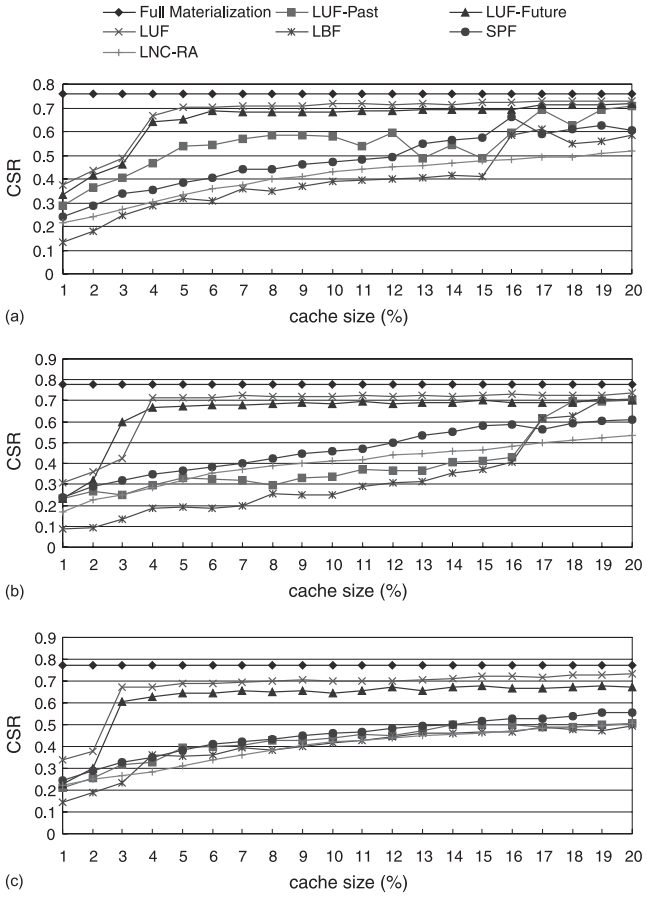

Fig. 9. Performance of the caching schemes for the multi-user session query workloads: (a) 5 users, (b) 10 users and (c) 15 users.

Fig. 9 shows the results for the multi-user workloads with 5, 10, and 15 users. In all cases, the LUF and LUF-Future schemes perform most effectively even with small cache size. For the workloads of 5, 10, and 15 users, the LUF scheme achieves high cost saving ratios about 91.9%, 91.7%, and 89.1% of that of the full materialization respectively with the cache size of 5% of the data cube size. LUF performs slightly better than LUF-Future, which may be due to interference among concurrently executing OLAP sessions from different users and the replacement policy of LUF that considers usability for both past and future queries together. We observe that SPF and LNC-RA show similar performance characteristics regardless of the number of concurrent users. This is due to the fact that they have similar goodness measures based on the reference rate and access frequency, respectively, and that both do not consider semantic dependencies in the sequence of queries in OLAP sessions. The results also show that LBF performs poorly for all the session query workloads, which is probably because of its coarse granularity of caching that must be a query result representing a node in the DH lattice.

6. Conclusions

In this paper we have proposed a new cache management policy lowest-usability-first for OLAP queries based on the usability of query results in processing other queries. This scheme uses the profit of the query results for the expected future queries of the current query as well as for the recently performed queries as a goodness measure for deciding what to replace from the cache. We have considered the interactive and navigational nature of OLAP query workloads and exploited semantic relationships between successive queries in an OLAP session to classify the possible subsequent future queries and predict a set of representative ones using a probability model for them. We have presented a method for estimating the usability of a query result for the representative future queries. Our experimental results indicate that our cache management scheme using the past and future usability of the query results can reduce the cost of processing OLAP queries effectively. For various OLAP query workloads tested in the experiments, the LUF scheme consistently outperforms the previous query result caching strategies for DWs and OLAP systems with small cache space.

References

Albrecht, J., Bauer, A., Deyerling, O., Gunzel, H., Hummer, W., Lehner, W., Schlesinger, L., 1999. Management of multidimensional aggregates for efficient online analytical processing. In: Proceedings of International Database Engineering and Applications Symposium, Montreal, Canada, pp. 156–164.

Baralis, E., Paraboschi, S., Teniente, E., 1997. Materialized view selection in a multidimensional database. In: Proceedings of the 23rd International Conference on Very Large Data Bases, Athens, Greece, pp. 318–329.

Chaudhuri, S., Dayal, U., 1997. An overview of data warehousing and OLAP technology. SIGMOD Record 26 (1), 65–74.

Chaudhuri, S., Krishnamurthy, R., Potamianos, S., Shim, K., 1995. Optimizing queries with materialized views. In: Proceedings of the 11th IEEE International Conference on Data Engineering, Taipei, pp. 190–200.

Codd, E.F., Codd, S.B., Salley, C.T., 1993. Providing OLAP to user-analysts: An IT mandate, White Paper, Hyperion Solutions Corporation, CA.

Deshpande, P.M., Ramasamy, K., Shukla, A., Naughton, J.F., 1998. Caching multidimensional queries using chunks. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Seattle, Washington, pp. 259–270.

Goldstein, J., Larson, P.A., 2001. Optimizing queries using materialized views: A practical, scalable solution. In: Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, Santa Barbara, CA, pp. 331–342.

Gupta, H., 1997. Selection of views to materialize in a data warehouse. In: Proceedings of the International Conference on Database Theory, Delphi, Greece, pp. 98–112.

Gupta, H., Mumick, I.S., 1999. Selection of views to materialize under a maintenance cost constraint. In: Proceedings of the International Conference on Database Theory, Israel, pp. 453–470.

Harinarayan, V., Rajaraman, A., Ullman, J.D., 1996. Implementing data cube efficiently. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Montreal, Canada, pp. 205–216.

Kalnis, P., Papadias, D., 2001. Proxy-server architectures for OLAP. In: Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, Santa Barbara, CA, pp. 367–378.

Kortidis, Y., Roussopoulos, N., 1999. DynaMat: A dynamic view management system for data warehouses. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Philadelphia, pp. 371–382.

Levy, A.Y., Mendelzon, A.O., Sagiv, Y., Srivastava, D., 1995. Answering queries using views. In: Proceedings of the ACM Symposium on Principles of Database Systems, San Jose, CA, pp. 95–104.

O'Neil, P., Graefe, G., 1995. Multi-table joins through bitmapped join indices. SIGMOD Record 24 (3), 8–11.

Park, C.-S., Kim, M.H., Lee, Y.-J., 2001a. Rewriting OLAP queries using materialized views and dimension hierarchies in data warehouses. In: Proceedings of the 17th IEEE International Conference on Data Engineering, Heidelberg, Germany, pp. 515–523.

Park, C.-S., Kim, M.H., Lee, Y.-J., 2001b. Usability-based caching of query results in OLAP systems. Technical Report. CS-TR-2001-166, Department of EECS, KAIST, Korea.

Park, C.-S., Kim, M.H., Lee, Y.-J., 2002. Finding an efficient rewriting of OLAP queries using materialized views in data warehouses. Decision Support Systems 32 (4), 379–399.

Roy, P., Ramamritham, K., Seshadri, S., Shenoy, P., Sudarshan, S., 2000. Don't trash your intermediate results, Cache 'em. Technical Report. Deparment of CSE, IIT-Bombay.

Sapia, C., 1999. On modeling and predicting query behavior in OLAP systems. In: Proceedings of the International Workshop on Design and Management of Data Warehouse, Germany.

Scheuermann, P., Shim, J., Vingralek, R., 1996. WATCHMAN: A data warehouse intelligent cache manager. In: Proceedings of the 22nd International Conference on Very Large Data Bases, Bombay, India, pp. 51–62.

Shim, J., Scheuermann, P., Vingralek, R., 1999. Dynamic caching of query results for decision support systems. In: Proceedings of the 11th International Conference on Scientific and Statistical Database Management, Cleveland, OH, pp. 254–263.

Srivastava, D., Dar, S., Jagadish, H.V., Levy, A.Y., 1996. Answering queries with aggregation using views. In: Proceedings of the 22nd International Conference on Very Large Data Bases, Bombay, India, pp. 318–329.

Theodoratos, D., Sellis, T., 1997. Data warehouse configuration. In: Proceedings of the 23rd International Conference on Very Large Data Bases, Athens, Greece, pp. 318–329.

Transaction Processing Performance Council, 1999. TPC Benchmark™ H Standard Specification, Revision 1.3.0.

Zaharioudakis, M., Cochrane, R., Lapis, G., Pirahesh, H., Urata, M., 2000. Answering complex SQL queries using automatic summary tables. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, TX, pp. 105–116.

Chang-Sup Park received his B.S. and M.S. degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST), Teajon, Korea, in 1995 and 1997, respectively, and his Ph.D. degree in Electrical Engineering and Computer Science from KAIST, Taejon, Korea, in 2002. He currently works for Service Development Laboratory of Korean Telecom. His research interests include OLAP, data warehouses, multi-dimensional indexing, and semi-structured data management. He is a member of the ACM.

Myoung Ho Kim received his B.S. and M.S. degree in Computer Engineering from Seoul National University, Seoul, Korea, in 1982 and 1984, respectively, and his Ph.D. degree in Computer Science from Michigan State University, East Lansing, MI, in 1989. In 1989 he joined the faculty of the Department of Computer Science at KAIST, Taejon, Korea, where currently he is a professor. His research interests include OLAP, data mining, information retrieval, mobile computing and distributed processing. He is a member of the ACM and IEEE computer Society.

Yoon-Joon Lee received his B.S. degree in Computer Science and Statistics from Seoul National University, Seoul, Korea, in 1977, his M.S. degree in Computer Science from Korea Advanced Institute of Science and Technology (KAIST), Taejon, Korea, in 1979, and his Ph.D. degree in Computer Science from INPG-ENSIMAG, France, in 1983. In 1984, he joined the faculty of the Department of Computer Science at KAIST, Taejon, Korea, where currently he is a professor. His research interests include data warehouses, OLAP, WWW, multimedia information systems, and database systems. He is a member of the ACM and the IEEE Computer Society.