Divide and Aggregate: Caching Multidimensional Objects

Wolfgang Lehner, Jens Albrecht, Wolfgang Hümmer
Database Systems (IMMD6)
University of Erlangen-Nuremberg
Martensstr. 3, 91058 Erlangen, Germany
{lehner, jalbrecht, huemmer}@informatik.uni-erlangen.de

Abstract

A common optimization technique in the OLAP application domain is the use of summary aggregate data. For an appropriate support of analysis hot spots, we propose a query based aggregate cache of 'multidimensional objects'. The major drawback of previous query caching methods is that new queries must be completely subsumed by a single cached object in order to utilize it. Our solution is the introduction of 'set-derivability' which allows the combination of several aggregates to derive a single query. The set of cached multidimensional objects is dynamically determined using both users access patterns and semantical knowledge of dimensional structures. Experimental results show that average cost reductions of over 50% are easily reached while spending only 10% of additional storage for summary data.

1 Introduction

Online Analytical Processing (OLAP) offers analysts an efficient interactive access to consistent business data, typically stored in a data warehouse. From the physical database design point of view, the two main obstacles to provide efficient multidimensional access to those data are to choose the right partitioning scheme and the right selection of summary tables. Whereas partitioning is a common concept in physical database design, the generation of summary tables is specific to the OLAP application domain with mostly read-only data access. Due to a limited storage capacity not all possible aggregations may be precomputed. In this paper we present a dynamic caching strategy for multidimensional data incorporating both partitioning schemes and summary tables. Our strategy is easy to implement in relational OLAP servers and provides excellent support of analysis hot spots. Experiments show that average cost reductions of over 50% are easily reached while spending only 10% of additional storage for summary data.

Derivability

To speed up queries, multidimensional objects are cached and reused during runtime. The framework of Multidimensional Objects provide a consistent and formalized view to database queries as well as to materialized summary data. They may be visualized as a partition (restricted to certain selection predicates) of an aggregated data cube (using grouping attributes) specified by a regular star-query.

A necessary prerequisite for reuse is derivability ([Klug82], [Fink82]). Derivability in the context of partitioned aggregates concerns the following two aspects:

- *compatibility of grouping attributes:*
  The precomputed data from which a query may be derived has to have an equal or finer granularity with regard to the corresponding dimensional structures.*
- *compatibility of selection predicates:*
  This condition for derivability states that the precomputed partition of summary data must not be more restrictive than the query which is the subject of derivation.
  Since predicate comparison in general is NP-complete ([SuKN89]), recent work either does not consider partitioning at all ([HaRU96], [BaPT97]), is limited to test for equality ([ShSV99]) or reduced to statically predefined partitions (chunks; [DRSN98]).

Contribution

To overcome these limits and to provide a simple and powerful strategy to select and release precomputed summary data with limited scope, this paper contributes the following:

- *set-derivability*: Our approach is not limited to derive a single query from a single summary data table but computes a nearly optimal construction plan for a

*. To illustrate these dependencies [HaRU96] introduced to notion of an aggregation lattice.

query based on a *set of precomputed summary data partitions* ('patch-working on multidimensional objects'; section 4).
- *use of dimensional structures*: Since we allow partitioning only with respect to classification hierarchies (and not for dimensional attributes) algorithms to compose a multidimensional aggregate of a set of cached objects become feasible.
- *dynamic caching policy*: Since *static precomputation* of *complete data cubes* does not adequately support data analysis hot spots, our approach decides 'on-the-fly' which summary data partitions (multidimensional objects) are worth to keep materialized. Moreover, materialization of the same data cube partition at different aggregation levels is explicitly allowed and the size of a partition is not a system parameter but implicitly determined by the user.
- *easy to integrate*: Since we perform algebraic optimization, our algorithm may be easily integrated into relational OLAP-servers.
- *high average cost reduction*: As our experiments show, we achieve high cost reductions with only very decent storage overhead at almost no cost.

Structure of the Paper

In the next section, we give a comprehensive overview of recent work which is related to selection and use of materialized aggregates in the context of OLAP and data warehousing. The third section formally introduces the one- and multidimensional structures of the considered multidimensional model. Based on the notion of multidimensional objects, section 4 defines the concept of set-derivability using 'patch-working' and outlines the four different influence factors to compute the overall benefit of a single cached multidimensional object. Finally, results of various experiments are presented in section 5. The paper concludes with a summary.

2 Related Work

The general idea of precomputing summary data is rooted in the application area of 'Statistical and Scientific Databases'. Several proposals like [ChMM88] were made but the idea became popular with the emergence of 'Data Warehousing' together with 'Online Analytical Processing' and the resulting need for an efficient and mostly read-only access to aggregates in a multidimensional context ([ChSh95]).

Static versus Adaptive Precomputation and Caching Schemes

A first classification of existing work may be specified according to the flexibility of the scheme with regard to user access behavior. Given a set of relational queries, the algorithm of [YaKL97] computes the global query execution plan and selects the node of the resulting plan with the highest number of references for precomputation. [Gupt97] extends this procedure in the context of AND/OR-graphs by considering alternative local query execution plans. Based on the concept of an aggregation lattice, the algorithm specified in [HaRU96] ([GHRU97] additionally considers existing index structures) statically selects those nodes of the lattice with the best cost saving ratio in relation to the additional storage required by that node. Since this approach shows a complexity of $O(n^3)$ with n as the number of *possible nodes* of an aggregation lattice, the work of [BaPT97] introduces heuristics to reduce this number and make the approach more applicable.

In opposite to these static algorithms, dynamic or adaptive strategies try to incrementally optimize the set of precomputed aggregates. On the relational level, [ShSV99] materializes the results of arbitrary complex queries. Unfortunately, by using hash codes they are able to reuse materialized queries only in the case of an exact match. This is not acceptable in a real OLAP scenario. In the case of an exact match with regard to the group-by attributes, the recent work of [DRSN98] proposes a chunk-based caching strategy. The materialization of chunks (see below) is resolved according to a classical CLOCK scheme, thus directly reflecting the user access behavior. In [KoRo99] a similar approach is presented. Again, a single query can only be answered from a single materialized fragment.

Table-, Query-, Chunk-based Partitioning Schemes

The second classification criteria of existing strategies is based on the unit of precomputation. Basically there are three levels, namely table-, query-, and chunk-based strategies.

Algorithms like [HaRU96], [GHRU97], and [BaPT97] do not support partitioning and are therefore classified as *table-based strategies*. Compatibility of selection predicates has not to be checked since all summary tables are defined on an unrestricted scope (the second condition of derivability is trivial). However, this implies that analysis hot spots like "the current period" are not reflected appropriately, because the algorithms materialize only summary tables containing ***all*** periods.

On the opposite, chunk-based partitioning schemes like [DRSN98] define relatively small aggregate partitions of a fixed size (chunks). Queried chunks are cached and later reused to answer new queries. On the one hand, [DRSN98] reports very high cost saving ratios in their experiments. On the other hand however, the implementation of special bitwise index structures to identify single chunks is extremely expensive. Moreover, they use a special chunk-based file system preventing a seamless integration into relational OLAP servers. At last, the choice of the chunk size must be statically determined and is crucial for the efficiency of the whole system.

The third class of different partitioning schemes is made up by *query-based partitioning schemes* ([SDJL96], [ShSV99], [YaKL97], [GuHQ95]). Single user queries reflect the unit of caching and make the selection of the right partition size obsolete. Recent work reuses results of former queries either by *query containment* or by *exact match* ([SDJL96], [ShSV99]) where the incoming query must be completely derivable from the materialized query. Moreover, it is further argued against query-based partitioning ([DRSN98]) that several materialized queries may address a common part of a data cube. This would result in a redundant storage of that part and decrease the utilization of the additional storage.

3 The Multidimensional Model

The following section introduces the dimensional and multidimensional structures which build the formal representation of the multidimensional model. The description is divided into the presentation of dimensions and multidimensional objects both of which are necessary to define a multidimensional database ([LeAW98]).

3.1 Dimensional Structures

Dimensional structures are defined in the conceptual schema design phase of a database. The semantic information coded into these structures is heavily used for cache replacement.

Definition 1: A *dimension D* of a multidimensional database is a set of attributes $D = \{ PA \} \cup \{ CA_1, ..., CA_p \} \cup \{ DA_1, ..., DA_q \}$ and the following set of functional dependencies between these attributes:

- $\forall i\ (1 \le i \le p)$: $PA \rightarrow CA_i$ and $\forall i\ (1 \le i \le q)$: $PA \rightarrow DA_i$
  The *primary attribute* PA of a dimension functionally determines all other attributes of a single dimension. The instances of the primary attribute are called *dimensional elements*.
- $\forall i\ (1 \le i \le p-1)$: $CA_i \rightarrow CA_{i+1}$
  The *classification attributes* CA_i of a dimension are used to define a multi-level grouping of single dimensional elements and are ordered according to their functional dependencies. The aggregation level or granularity i of a classification attribute CA_i is denoted as $\dot{CA}_i$. By definition, the primary attribute has the aggregation level 0 ($PA = CA_0$).
- $\forall i,j\ i \ne j\ (1 \le i,j \le q)$: $DA_i \not\rightarrow DA_j$
  The dimensional attributes DA_i of a dimension represent all attributes describing properties of the dimensional elements.

Although the concept of dimensional attributes is more complex ([LeAW98], [ABD+99]), dimensional attributes may be seen as additional single-level hierarchies on a dimension, which implies that they are functionally independent of each other†.

Example 1: The set of dimensions of a typical OLAP sample scenario consists of a product, time, and shop dimension. Within the shop dimension for example, the primary attribute corresponds to a unique shop identifier. A regional classification may be defined by the sequence city → state → region → country. The set of dimensional attributes may be defined as { shop-type, purchase-class, selling-area, shop-window-size }. In real OLAP scenarios, a single (usually a product or a customer) dimension may easily have 20 different attributes.

3.2 Multidimensional Structures

The basic data structure in the multidimensional context, the notion of a multidimensional object, is of fundamental importance. With regard to the caching algorithms described in the following section, all incoming queries as well as materialized results of former queries are internally represented as multidimensional objects.

Definition 2: A *multidimensional object* (MO) is a triple $M = (M, S, G)$ consisting of a measure M, an n-ary vector of classes $S = (s_1, ..., s_n)$ denoting the scope, and the granularity specification G.
A *measure* is a tuple M = [N, A, O] where N is a reserved name for the corresponding fact, $A \in \{$ FLOW, STOCK, VALUEperUNIT $\}$ is the *aggregation type*, and $O \in \{$ NONE, SUM, AVG, COUNT $\}$ is the applied aggregation function on that specific fact.
A *granularity specification* is a tuple G = [L, P] where L ('aggregation **L**evel') is an n-ary vector of classification attributes of each dimension

$$(L \in \{CA_0^1, ..., CA_{p_1}^1\} \otimes ... \otimes \{CA_0^n, ..., CA_{p_n}^n\}),$$

and P ('**P**roperties') is a set of dimensional attributes

$$(P \in \{DA_1^1, ..., DA_{q_1}^1\} \cup ... \cup \{DA_1^n, ..., DA_{q_n}^n\}).$$

Example 2: The formal representation of a sales data cube for video products in Germany would be as follows:
M = ([SALES , FLOW, SUM],
(p.group = 'Video', s.country = 'Germany'),
[(p.family, s.region), { p.brand, s.shoptype }])

The aggregation type determines the set of applicable operations on the numeric values to ensure correct summarizability ([LeSh97]). For example, all aggregation opera-

†. Moreover it is worth to mention that this definition does not explicitly consider multiple hierarchies with more than one level. Multiple hierarchies would introduce a partial order on the classification attributes (instead of a total order for the single classification hierarchy). This extension would not affect any of the presented derivability or caching algorithms but would complicate the presentation. Therefore, we only consider a single classification hierarchy.

tions are applicable to sales figures (A = FLOW), whereas the sum operation would not be allowed on figures denoting the price of single articles (A = VALUEperUNIT). The aggregation type STOCK prohibits summation in the temporal dimension. Finally the size of a multidimensional object *M* at the instance level is denoted by |*M* |.

Relations and Operations on Multidimensional Objects

To formally define 'derivability' on multidimensional objects, we need to introduce the '≤'-relation for the granularity specification and the intersection of scopes of multidimensional objects. Moreover, we need intersection and difference operators of multidimensional objects themselves.

Definition 3: A granularity specification G = [L, P] is *finer than or equal* to a granularity specification G' = [L', P'], i.e. G ≤ G', if and only if

- all aggregation levels of L are lower or equal than the aggregation levels of L'
- P is a superset of dimensional attributes with regard to P', i.e. P' ⊆ P

or

- G represents raw data, i.e. L = (PA_1, ..., PA_n).

Definition 4: The *intersection of multidimensional scopes* is defined as the component-wise intersection of the classes in each dimension, i.e. by the homomorphism

$S \cap S' = (s_1, ..., s_n) \cap (s'_1, ..., s'_n)$
$:= ((s_1 \cap s'_1), ..., (s_n \cap s'_n))$

The intersection is empty if two classes s_i and s'_i do not show a parent-child relationship within the corresponding classification hierarchy of dimension i.

It is important to note that the problem of intersecting classes in a classification hierarchy may be solved by checking the simple parent-child relationships. This concept is fundamental for the applicability of the set-derivability mechanism.

Definition 5: The *intersection of multidimensional objects* *M* = (M, S, G) and *M'* = (M',S',G'), i.e. *M* ∩ *M'* is defined as *M* ∩ *M'* = (M, S ∩ S', G) if and only if

- both objects refer to the same measure, i.e. M = M'
- the intersection of the scopes is not empty, i.e. S ∩ S' ≠ ∅
- the granularity specification of *M* is finer than or equal to the granularity specification of *M'*, i.e. G ≤ G'.

As seen in this definition the coarser multidimensional object *M'* is implicitly refined to the granularity level G.

To be able to introduce the concept of set-derivability by substitution ('patch-working'; section 4), we need to define the difference of two multidimensional objects. As illustrated in figure 1 for the two-dimensional case, the difference of two n-dimensional cubes would result in the worst case in a set of 3^n-1 convex residual cubes - assuming exactly 3 classification nodes partitioning each dimension.

It can be shown that intelligent slicing can reduce this number to 2n residual cubes ([ChMM88], proof of proposition 4). Informally, the cube is cut iteratively for each dimension into maximal three slices; two slices are residual cubes and are not considered any longer; the third slice contains the operand and is sliced further in the remaining dimensions. For n dimensions we obtain 2n residual cubes.

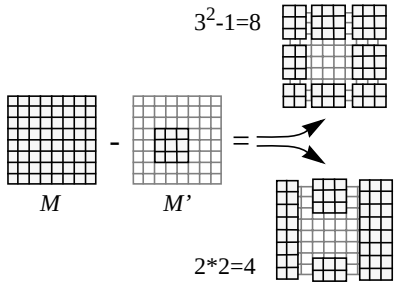


Figure 1: Difference of multidimensional objects

In the general case with dimensions consisting of an arbitrary number of classification nodes for each dimension i (1≤i≤n) the minimum number p_i of partitioning classification nodes including the operand to be subtracted has to be determined. Then the number of residual cubes will be $\sum(p_i - 1)$, 1≤i≤n, because of the same argumentation as above.For more detailed information refer to [AlGL99].

Definition 6: The difference of two multidimensional objects *M* = (M, S, G) and *M'* = (M',S',G'), i.e. *M* – *M'* is defined as a set $\{(M, S_1, G)_1, ..., (M, S_k, G)_k\}$ where k is the number of residual cubes and S_i are the scopes of residual objects obeying the following conditions:

$$S = S' \cup \bigcup_{i=1}^{k} S_i \text{ and } \bigcap_{i=1}^{k} S_i = \emptyset .$$

Derivability of Multidimensional Objects

To reuse the results of former queries, derivability of multidimensional objects as the units of caching is a necessary prerequisite. The following definition of the total derivability will be relaxed in the next section to the partial derivability which is sufficient to define the set-derivability.

Definition 7: A measure M = (N, A, O) *is derivable* from a measure M'= (N', A', O') if and only if

- the measures refer to the same real world entity (N = N' and A = A')
- the operations are compatible, i.e., O = O' or O' = NONE

Note that semi-additive functions like AVG are split into the additive and non-additive counterparts ({SUM, COUNT} and division). The additive operations build a regular subject of the caching mechanism. Thus by materializing multidimensional objects with the applied operations SUM and COUNT, an AVG-query will never be directly but only indirectly cached.

Definition 8: A multidimensional object M = (M, S, G) is *totally derivable* from a multidimensional object M' = (M',S',G'), if and only if

- the measure M in M is derivable from the measure M' in M'
- M' contains M, i.e. S ∩ S' = S
- the granularity specification of M' is finer than M, i.e. G' ≤ G

4 Set-Derivability using 'Patch-Working'

Although derivability is the fundamental concept for the use of precomputed aggregates, in many cases definition 8 it is too restrictive. Consider the example in figure 2. The query object M_Q is not derivable from any of M_1, M_2, or M_3 alone. However, it can be derived from a combination of M_1, M_2 and M_3. The example illustrates the problem of previous query caching methods ([ShSV99], [SDJL96]) where a materialized aggregate can only be accessed if the complete query is derivable from this single cached object. This is because query containment can be checked with a relatively simple algorithm where the determination of the overlapping regions for arbitrary predicates is NP-complete ([SuKN89]). However, in the context of multidimensional objects where the predicates consist of scope definitions corresponding to classes in a classification hierarchy this problem can be reduced to the determination of ancestor and sibling relationships.

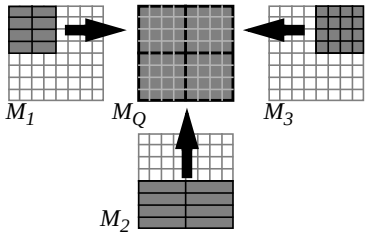


Figure 2: Patch-Working

Definition 9: A multidimensional object M = (M, S, G) is *partially derivable* from a multidimensional object M' = (M', S', G'), i.e. M' „ M, if and only if

- the conditions (1) and (3) of the total derivability hold for M and M'
- M and M' are overlapping, i.e. S ∩ S' ≠ ∅

In this case, M' is said to support M, and $M \cap M'$ is called the *support* of M' for M.

From the set of supporters, i.e. the potential candidates, an appropriate set of partitions, or *patches*, has to be selected in order to answer a query. Such a set is called a *substitution* for the query. .

Definition 10: A set $S = \{ \langle M_1, M_1' \rangle, ..., \langle M_k, M_k' \rangle \}$ with $M_i = (M_i, S_i, G_i)$ and $M_i' = (M_i', S_i', G_i')$ is called a *substitution* for a multidimensional object M_Q, if and only if

- ∀i (1≤i≤k): $M_i = M_Q$, $G_i = G_Q$, and
- ∀i (1≤i≤k): M_i' „ M_i,
- $S_Q = \cup_i S_i$,
- ∀i, j (1≤i,j≤k) with i≠j: $S_i \cap S_j = \emptyset$.

In this case M is said to be *set-derivable* from { M_1', ..., M_k' }

Each patch in S has two components $\langle M_i, M_i' \rangle$ denoting that partition M_i of M_Q is to be computed from M_i'. All partitions must be disjoint and their union must result in M_Q.

The Query Optimization Problem

In general, the cache contains several possibly overlapping aggregates with different granularities which support a query. Therefore, the problem of the multidimensional query optimizer is to choose a substitution that allows the computation of the query at minimal cost. This problem is very similar to the well-known knapsack problem ([CoLR90]), where a thieve tries to select the most valuable set of things with different values and weights. A common method is to use a greedy strategy to approximate the optimal solution for the NP-complete 0/1 knapsack problem. In the case of the fractional knapsack problem where the thieve may take arbitrary fractions of objects the greedy approach actually yields the optimum. The problem of finding an optimal substitution can be reduced to the fractional knapsack, if the cost to compute a fraction of one multidimensional object from another one is monotonous in the size of the support.

Definition 11: A cost function Cost(M, M') determining the cost to compute the support of a multidimensional object M from M' is *monotonous*, if for any two arbitrary multidimensional objects M' and M'' holds:

$$| M \cap M' | < | M \cap M'' | \Rightarrow \text{Cost}(M, M') < \text{Cost}(M, M'').$$

In the case of a monotonous cost function, an algorithm selecting the cheapest substitution for each data cell in the multidimensional object M automatically finds the optimal substitution ('Greedy-Choice property'; [CoLR90]). However, especially in the context of relational databases monotony of the cost function may not be given. In a relational backend the computation of a multidimensional object results in range queries on the affected tables representing the supporting multidimensional objects. In the absence of indexes, range queries on relational tables usually result in full table scans destroying the property of monotony.

In order to find a substitution for a query we use the greedy strategy outlined in algorithm 1. The algorithm gets the query M_Q and the set C of all materialized multidimensional objects as input parameters. It returns a substitution S and the estimated cost to compute M_Q. As long as there are still uncomputed fragments remaining (line 5), the candidate with the least access cost per cell for each fragment is selected (lines 9-21). After the selected patch is added to the solution S (line 24) it must be removed from the remainder R using the difference operator (line 28). Since M may

Algorithm: Substitution (Patch Working)

Input: set of existing multidimensional objects $C = \{M_1, ..., M_n\}$
multidimensional object representing the query M_Q,

Output: substitution S for M_Q

```
1   // initialize remainder, patch-work and cost
2   R := { M_Q }; S := ∅; cost_total := 0
3
4   // iterate over all
5   While ( R ≠ ∅)
6       M_Best := ∅; cost_rel := ∞;
7
8       // find cheapest substitution for each remaining patch
9       Foreach M ∈ R
10          // loop through all candidates
11          Foreach M_cand ∈ C
12              // next candidate if this one is not supporting M
13              If ( Not ( M_cand „ M ) )
14                  Next;
15              End If
16              // if relative cost of new candidate is less
17              If ( cost_rel > Cost(M, M_cand) / |M ∩ M_cand| )
18                  M_best := M_cand;
19                  cost_rel := Cost(M, M_cand) / |M ∩ M_cand|;
20              End If
21          End Foreach
22
23          // add the best patch to the result
24          S := S ∪ { ⟨M∩M_best, M_best⟩ };
25          cost_total := cost_total+Cost(M, M_best);
26
27          // remove the new patch from the remainder
28          R := R \ { M } ∪ (M-M_best);
29      End Foreach
30  End While
31  Return (S, cost_total)
32  End
```

Algorithm 1: Substitution of a multidimensional object (‘patch-working’)

only be partially derivable from M_{best} the uncovered difference M-M_{best} must remain in R. It is worth to mention here that a solution is always possible if C contains also the raw data objects

Cache Replacement

The knowledge of the OLAP application domain provides not only a lot of semantic information about the data model but also about the users behavior. In order to incorporate reference density as well as semantic knowledge the benefit in our approach is defined by the four factors of the weighted relative reference density, the absolute benefit, the degree of relationship, and the reconstruction cost ([ADB+99]):

- Weighted Relative Reference Density D_M
  The weighted relative reference density is used to approximate the traditional least reference density (LRD) in the multidimensional context. A reference counter is increased only by the fraction of M that was actually referenced.
- Absolute Benefit $A_{M:}$
  The absolute benefit is based on the size, the granularity, and the scope of the multidimensional object and therefore generalizes the benefit definitions from [HaRU96] as well as [DRSN98].
- Degree of Relationship $R_M(M_Q)$:
  In the OLAP context, it is very likely that an object in the cache that can be reached within a few navigational operations will be used with a high probability. Therefore, the number of navigation operations which are necessary to transform the last query M_Q into a materialized, i.e. cached multidimensional object M defines the degree of relationship for M.
- Reconstruction Cost $C_M(M_Q, C)$:
  Since cached multidimensional objects may be computed from the set of multidimensional objects being currently an element of the cache, it may prove beneficial if those multidimensional objects are replaced which can be reconstructed most easily.

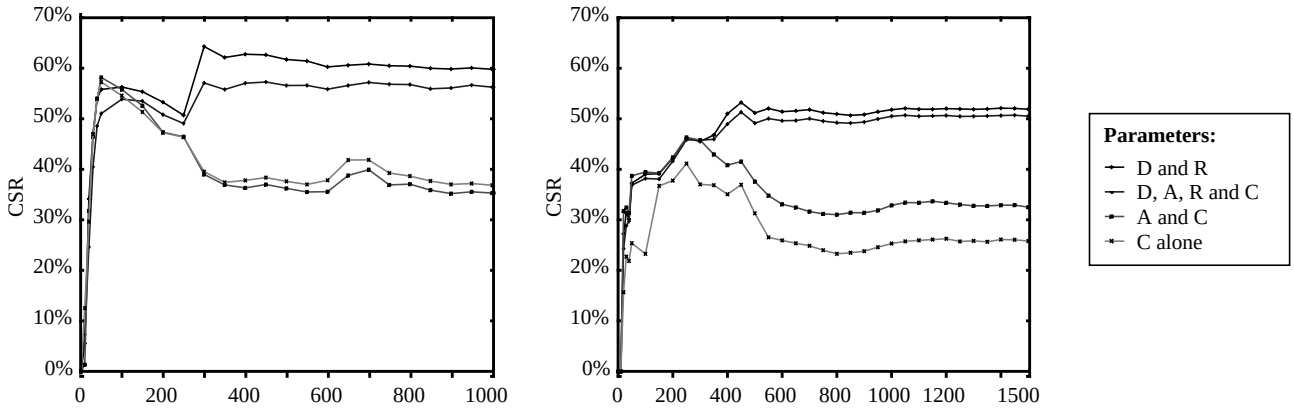
Since the factors have very different orders of magnitude, they are normalized by their average values before the overall benefit $B_M(M_Q, C)$ of a multidimensional object M for a cache configuration C after query M_Q is computed by a linear combination. The overall benefit is used as the indicator, whether there is a change of the set of cached multidimensional objects or not.

5 Experimental Results

The following section illustrates the optimization potential of the presented algorithms for caching multidimensional objects. Our proposed dynamic caching strategy is analyzed according to different cache sizes and influence factors for the computation of the overall benefit. Furthermore, we compare our strategy with the static precomputation approach of [HaRU96].

Experimental Setup

The proposed data structures and algorithms are implemented in the multidimensional OLAP server CUBESTAR. All query processing and cache management is based on algebraic operations on multidimensional objects. However, multidimensional objects are physically stored in a relational backend (Oracle 8.0.4; DEC Alpha 3000/800, 1CPU, 192MByte RAM) in a star-schema-like manner (ROLAP system). For each patch in the optimal substitution for the query (algorithm 1), the query processor gener-



a) single user, 1000 steps

b) multiple users, 1500 steps

Figure 3: Cost saving ratio for different combinations of cache parameters

ates an SQL statement, which are put together by UNION ALL operations to yield the final result. We could not completely eliminate side-effects of the system, especially of the database buffer, but multiple runs confirmed the following results.

Multidimensional Database Schema and Query Generation

The schema of the multidimensional database consists of three dimensions with five, four and six levels, and a set of up to 14 dimensional attributes, respectively. The corresponding raw data cube was randomly filled with a sparsity of 3% resulting in about 250.000 tuples in the relational representation. To simulate the cache replacement behavior, a new query is constructed from the previous query by applying a single operation in a certain dimension with a predefined probability. Since a typical OLAP user does not switch very often between dimensions, in 80% of the cases the operation was executed on the same dimension and in 20% of the cases the operation involved a different dimension as the previous one. The set of possible operations included slice (scope changes to a child node), unslice (scope changes to a parent node), drill-down (granularity changes to the next lower level), roll-up (granularity changes to the next higher level), split (add a dimensional attribute) and merge (remove a dimensional attribute).

Performance Metric

Since in general the cache management aims at maximizing the cost saving ratio ([ShSV99], [DRSN98]), we defined this metric in the context of multidimensional objects. The cost saving ratio CSR on cached multidimensional objects is defined by the division of the sum of the costs of the queries using the cache content and the total cost to compute the queries from the non-aggregated raw data.

In the experimental scenario, two user traces, one simulating 1000 operations of single user and another one 1500 steps of five independent users, were used to investigate the behavior of the cache for varying cache sizes and configurations.

Influence of the Cache Parameters

The first set of graphs in figure 3 illustrates the impact of the four cache parameters using a cache size of 20% of the raw data volume. The figures show the development of the cost saving ratios for the single and the multi user trace. If all parameters are enabled, cost savings of about 56% in the single user case and 50% in the multi user case were obtained. By trying all possible combinations of parameters it turned out that for both the single and the multi user traces the usage of the relative reference density D and the relationship degree R yields the best results of 60% and 52% respectively. On the contrary, the worst results of 35% and 28%, respectively, were achieved using the absolute benefit A and the reconstruction cost C.

It is interesting to note that the number of potential users of a materialized aggregate, reflected by the absolute benefit A, is not suitable for the determination the overall benefit. Algorithms like [HaRU96] are based solely on this factor. Furthermore, it turns out that the general critics according to query-based caching schemes namely the possibly redundant storage of the same part of data cube in different objects is of minor interest, because the influence factor of the reconstruction cost also does very badly.

Influence of the Cache Size

The diagram in figure 4 shows the simulation results for different cache sizes varying over 10.000, 25.000, 50.000, 100.000, and 500.000 tuples. Compared to the raw data volume these values result in an additional storage overhead of 4%, 10%, 20%, 40% and 200%, respectively. For these simulations the best combination of cache parame-

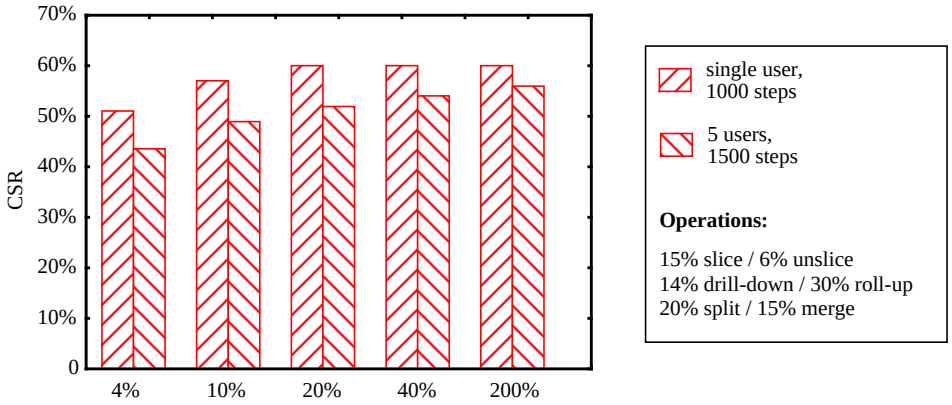


Figure 4: Query trace configurations and cost savings with varying cache sizes

ters, i.e. D and R, was used. As can be seen, already with a relatively small cache of 4% of the raw data size, cost reductions of over 50% in the single user mode and over 40% in the multi user mode are possible. However, the cost savings in the single user mode do not increase much over 60% for any cache size beyond 50.000 tuples. The figure illustrates that even for multiple users a cache of a fairly small size yields cost reductions not much worse than for a single user, i.e. above 50%.

Comparison to Static Precomputation of Aggregates

In order to compare our method to other materialization methods of aggregations, we implemented a static precomputation algorithm based on [HaRU96][‡]. This algorithm computes a set of aggregates with an unrestricted scope based on a benefit definition similar to the absolute benefit. Figure 4 shows the cost saving ratio for different cache sizes. If only 4% of additional storage are provided, there is almost no cost reduction possible for the static algorithm (therefore not in the figure), because the size of almost all useful aggregates with an unrestricted scope is larger than the cache size. Although with increasing cache size the cost saving ratio of the static approach rises, the values for the dynamic strategies remain unrivaled. Even for a cache size twice as large as the raw data volume the cost saving ratios for the single and multi user cases are only about 40% and 30%, respectively. Note that the dynamic strategy yields over 50% with 1/50 of that cache size.

The reason for the bad results of the static approach is that the size of the aggregation lattice grows exponentially in the number of possible group-by combinations. In most application scenarios this number is high, because several evaluation criteria on a single dimension are used, like dimensional attributes or multiple hierarchies. Using a reasonable amount of additional storage, static methods can only supply an insufficient number of aggregates. Moreover, because general static approaches do not take into account specific partitions, i.e. regions of interest or hot spots, much of the additional storage space is wasted for regions of aggregates which are never or seldom requested.

‡. Unfortunately, we could not compare our approach with the work of [DRSN98], since they require a very special ‘chunk-based’ file system in the context of their Paradise database system.

6 Summary

In this paper, we present a novel method for the dynamic selection of materialized aggregates in the context of multidimensional database systems. Our proposed solution is based on the notion of multidimensional objects which provide a consistent view to queries as well as to materialized aggregates. The use of multidimensional objects drastically simplifies the determination of intersections and differences of queries. Therefore, we are able to present a query-based caching strategy overcoming the limitations of query containment by the concept of set-derivability. Set-derivability is achieved by a patch-working mechanism, which (in the case of a monotonous cost function) computes the cheapest substitution of a query by a set of materialized aggregates. The cache replacement strategy is based on the overall benefit computation for each element in the aggregate cache which consists of a linear combination of four different influence parameters. At last we give the results of various experiments. Average cost reductions of 50% to 60% may be obtained by only 10% additional storage capacity.

The proposed optimization technique must not be seen as an alternative to traditional physical optimization methods like indexes or fragmentation, but as an addition. Our method not only explicitly supports fragmentation of the fact table but makes it even more beneficial. If aggregates are only materialized for hot spots, a single query may utilize both fragments of materialized aggregates and fragments of raw data in one operation. Our proposed strategy may easily be integrated into most relational OLAP servers, because all optimizations involve only algebraic operations on common multidimensional structures.

Currently, we are exploring the integration of indexes on the multidimensional objects and the benefit of using special linearizations. Another research is the use of the fragment-based patch-working algorithm in a parallel server environment.

References

ABD+99 Albrecht, J.; Bauer, A.; Deyerling, O.; Günzel, H.; Hümmer, W.; Lehner, W.; Schlesinger, L.: Management of multidimensional Aggregates for efficient Online Analytical Processing. In: *International Database Engineering and Applications Symposium* (IDEAS'99, Montreal, Canada, August 1-3), 1999

AlGL99 Albrecht, J.; Günzel, H.; Lehner, W.: Set-Derivability of Multidimensional Aggregates. In: *First International Conference on Data Warehousing and Knowledge Discovery* (DaWaK'99, Florence, Italy, August 30 - September 1), 1999

BaPT97 Baralis, E.; Paraboschi, S.; Teniente, E.: Materialized Views Selection in a Multidimensional Database. In: *23rd International Conference on Very Large Data Bases* (VLDB'97, Athens, Greece, Aug 25-29), 1997, pp. 156-165

ChMM88 Chen, M. C.; McNamee, L.; Melkanoff, M.: A Model of Summary Data and its Applications in Statistical Databases. In: *Proceedings of the 4th International Working Conference on Statistical and Scientific Database Management* (4SSDBM, Rome, Italy, June 21-23), 1988, pp. 356-372

ChSh95 Chaudhuri, S.; Shim, K.: An Overview of Cost-based Optimization of Queries with Aggregates. In: *IEEE Data Engineering Bulletin 18(1995)3*, pp. 3-9

CoLR90 Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.: *Introduction to Algorithms*, Cambridge (MA), London: The MIT Press; New York u.a.: McGraw-Hill Book Company, 1990

DRSN98 Deshpande, P.M.; Ramasamy, K.; Shukla, A.; Naughton, J.F.: Caching Multidimensional Queries Using Chunks. In: *Proceedings of the 27th International Conference on Management of Data* (SIGMOD'98, Seattle (WA), USA, June 2-4), 1998, pp. 259-270

Fink82 Finkelstein, S.: Common Expression Analysis in Database Applications. In: *Proceedings of the International Conference on the Management of Data* (SIGMOD'82, Orlando (FL), USA, June 2-4), 1982, pp. 235-245

GHRU97 Gupta, H.; Harinarayan, V.; Rajaraman, A.; Ullman, J.D.: Index Selection for OLAP. In: *Proceedings of the 13th International Conference on Data Engineering* (ICDE'97, Birmingham, Great Britain, April 7-11), 1997, pp. 208-219

GuHQ95 Gupta, A.; Harinarayan, V.; Quass, D.: Aggregate-Query Processing in Data Warehousing Environments. In: *Proceedings of the 21th International Conference on Very Large Data Bases* (VLDB'95, Zürich, Schwitzerland, Sept. 11-15), 1995, pp. 358-369

Gupt97 Gupta, H.: Selection of Views to Materialize in a Data Warehouse. In: *Proceedings of the 6th International Conference on Database Theory* (ICDT'97, Delphi, Greece, Jan. 8-10), 1997, pp. 98-112

HaRU96 Harinarayan, V.; Rajaraman, A.; Ullman, J.D.: Implementing Data Cubes Efficiently. In: *Proceedings of the 25th International Conference on Management of Data*, (SIGMOD'96, Montreal, Quebec, Canada, June 4-6), 1996, pp. 205-216

Klug82 Klug, A.: Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions. In: *Journal of the ACM 29(1982)3*, pp. 699-717

KoRo99 Kotidis, Y.; Roussopoulos, N.: DynaMat: A Dynamic View Management System for Data Warehouses. In: *International Conference on Management of Data* (SIGMOD'99, Philadephia (PA), U.S.A., June 1-3), 1999, pp. 371-382

LeAW98 Lehner, W.; Albrecht, J.; Wedekind, H.: Normal Forms for Multidimensional Databases. In: *Proceedings of the 10th International Conference on Scientific and Statistical Data Management* (SSDBM'98, Capri, Italy, July 1-3), 1998, pp. 63-72

LeSh97 Lenz, H.-J.; Shoshani, A.: Summarizability in OLAP and Statistical Data Bases. In: *Proceedings of the 9th International Conference on Statistical and Scientific Database Management* (9SSDBM, Olympia (WA), USA, Aug. 11-13), 1997, pp. 132-143

SDJL96 Srivastava, D.; Dar, S.; Jagadish, H.V.; Levy, A.Y.: Answering Queries with Aggregation Using Views. In: *Proceedings of 22th International Conference on Very Large Data Bases* (VLDB'96, Bombay, India, Sept. 3-6), 1996, pp. 318-329

ShSV99 Shim, J.; Scheuermann, P.; Vingralek, R.: Dynamic Caching of Query Results for Decision Support Systems. In: *Proceedings of the 11th International Conference on Scientific and Statistical Database Management* (SSDB'99, Cleveland (OH), U.S.A., July. 28-30), 1999, pp. 254-263

SuKN89 Sun, X.-H.; Kamel, N.; Ni, L.M.: Solving Implication Problems in Database Applications. In: *Proceedings of the 18th International Conference on Management of Data* (SIGMOD'89, Portland (OR), USA, May 31-June 2), 1989, pp. 185-192

YaKL97 Yang, J.; Karlapalem, K.; Li, Q.: Algorithms for Materialized View Design in Data Warehousing Environment. In: *Proceedings of the 23rd International Conference on Very Large Data Bases* (VLDB'97, Athens, Greece, Aug. 25-29), 1997, pp. 136-145