J. Albrecht, A. Bauer, P. O. Deyerling, H. Günzel, W. Hümmer, W. Lehner, L. Schlesinger

Management of multidimensional aggregates for efficient online analytical processing

Erstveröffentlichung in / First published in:

International Database Engineering and Applications Symposium (IDEAS'99). Montreal, 4.08.1999. IEEE, S. 156-164. ISBN 0-7695-0265-2

DOI: https://doi.org/10.1109/IDEAS.1999.787264

Final edited form was published in "International Database Engineering and Applications Symposium (IDEAS'99)", Montreal, 1999. ISBN 0-7695-0265-2.
https://doi.org/ 10.1109/IDEAS.1999.787264

MANAGEMENT OF MULTIDIMENSIONAL AGGREGATES FOR EFFICIENT ONLINE ANALYTICAL PROCESSING

J. Albrecht, A. Bauer, O. Deyerling, H. Günzel, W. Hümmer, W. Lehner, L. Schlesinger
Department of Database Systems, University of Erlangen-Nuremberg
{albrecht, asbauer, ordeyerl, guenzel, wghuemme, lehner, lzschles}@immd6.informatik.uni-erlangen.de

Abstract

Proper management of multidimensional aggregates is a fundamental prerequisite for efficient OLAP. The experimental OLAP server CUBESTAR*, which concepts are described in this paper, was designed exactly for that purpose. All logical query processing is based solely on a specific algebra for multidimensional data. However, a relational database system is used for the physical storage of the data. Therefore, in popular terms* CUBESTAR *can be classified as a ROLAP system. In comparison to commercially available systems,* CUBESTAR *is superior in two aspects: First, the implemented multidimensional data model allows more adequate modeling of hierarchical dimensions, because properties which apply only to certain dimensional elements can be modeled context-sensitively. This fact is reflected by an extended star schema on the relational side. Second,* CUBESTAR *supports multidimensional query optimization by caching multidimensional aggregates. Since summary tables are not created in advance but as needed, hot spots can be adequately represented. The dynamic and partition-oriented caching method allows cost reductions of up to 60% with space requirements of less than 10% of the size of the fact table.*

1. Introduction

The area of statistical analysis of empiric or business data has been boosted significantly in recent years by the concepts of data warehousing and online analytical processing (OLAP, [7]). This paper focuses on the concepts of the experimental OLAP server CUBESTAR, which differs essentially in two aspects from commercial systems and research prototypes. First, the used multidimensional data model provides a more very flexible modeling of classification hierarchies resulting in an extended star schema on the relational side. Second, the system implements a dynamic and adaptive aggregate cache which allows much more efficient multidimensional query processing.

The structure of the paper is as follows: The next section gives a short introduction to the multidimensional data model and the query language. Section three outlines the fundamental ideas of the dynamic and partition-oriented aggregation cache. The fourth section explains the relational representation of the multidimensional structures and queries. The general architecture of CUBESTAR is outlined in section five. The last section illustrates the performance potential of the aggregate cache by some simulation results. The paper concludes with a summary and an outlook of the current research activities.

2. Data Model and Query Language

A structural deficiency of many multidimensional modeling approaches like [1], [5] or [18] is the missing distinction of classifying and characterizing information. Consider the following SQL statement for the statistical table in figure 1 assuming a two-dimensional scenario:

```
SELECT    Product_Family, Product_Brand,
          Shop_Region, Shop_Type,
          SUM(Sales)
FROM      ...
WHERE     Product_Group = 'Video' AND
          Shop_Country = 'Germany'
GROUP BY  Product_Family, Product_Brand,
          Shop_Region, Shop_Type
```

SUM(Sales)			Video			
			Camcorder		HomeVCR	
			Sony	JVC	Sony	JVC
Germany	North	Hyper Market				
		Cash&Carry				
		Retail				
	South	Hyper Market				
		Cash& Carry				
		Retail				

Fig. 1: Statistical table

Both the relational mapping and the multidimensional equivalent know only grouping attributes resulting in an actually four-dimensional output table, respectively data cube. However, like the prefixes of the attributes demonstrate, the example is about a two dimensional problem where Product_Family is a *classification attribute* (CA) in a hierarchical classification structure of the product dimen-

Final edited form was published in "International Database Engineering and Applications Symposium (IDEAS'99)", Montreal, 1999. ISBN 0-7695-0265-2.
https://doi.org/ 10.1109/IDEAS.1999.787264

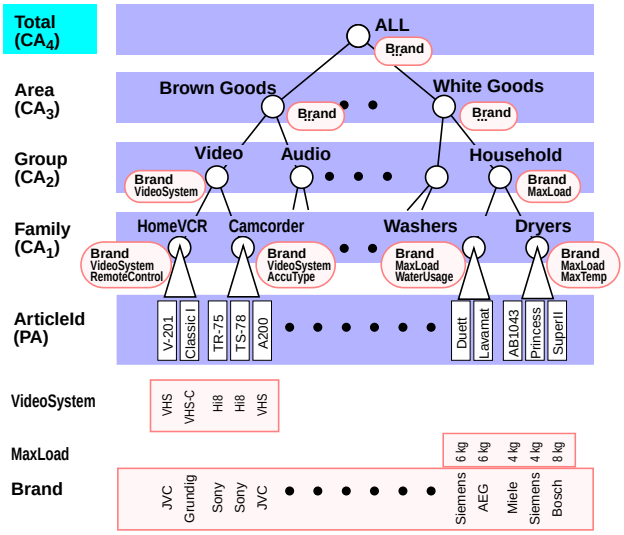


Fig. 2: Classification hierarchy of a product dimension with property attributes

sion (figure 2). The attribute Brand however is not part of the classification but serves as a *property attribute* for an additional characterization of single items. These in turn - generally called dimensional elements - are instances of the *primary attribute* (PA=CA_0) of a dimension, which functionally determines all other attributes.

At the instance level, each class of dimensional elements inherits some property attributes from its parent class where others are specific to that node ([21]). For example, only Video products show a property attribute VideoSystem. A property VideoSystem for the product group Washers, which is also element of the product dimension, would result in a structural schema defect ([15]). Other multidimensional data models are not able to express this fact. The modeling paradigm of CUBESTAR distinguishes between classification attributes (figure 2, upper part) and property attributes (figure 2, lower part). The classification hierarchy defines the overall framework for the analysis, specific property attributes are used for detailed evaluations.

2.1. Multidimensional Objects

The implementation of a multidimensional query processor requires a formal notion for multidimensional data cubes. Like relations in the relational model, such a formalism should be able to express descriptive queries and stored data as well as intermediate results. The basic data structure of CUBESTAR is the multidimensional object ([16]).

A multidimensional object describes a partition of a possibly aggregated multidimensional data cube. The example of figure 1, the sum of the Sales of Video recorders in Germany for each (product) family and (shop) region further detailed by Brand and ShopType may be specified as a multidimensional object in the following manner:

M = ([Sales, SUM, FLOW],
(Product.Group = 'Video', Shop.Country = 'Germany'),
[(Product.Family, Shop.Region),{ Brand, ShopType }])

Formally a *multidimensional object (MO)* is defined as a triple (M, S, G). The measure M = [N, O, A] consists of a name N (e.g. Sales), the applied operation O∈ {NONE, SUM, AVG, COUNT, MIN, MAX, ...} and the aggregation type A∈ {FLOW, STOCK, VALUE_PER_UNIT} specifying the permitted aggregation operations for that measure ([17]). The second component S is a tuple of one classification node per dimension describing scope of the represented data partition ((Product.Group = 'Video', Shop.Country = 'Germany')). The restriction to single classification nodes instead of arbitrary predicates results in simple conditions for the derivability of MOs (section 3.1). The last component of a MO describes the granularity G = [L, P], consisting of a tuple of hierarchy levels L ((Product.Family, Shop.Region)) and a set of property attributes P ({Brand, ShopType}).

2.2. Operations on Multidimensional Objects

The application of typical interactive OLAP operators on a multidimensional object results in a simple manipulation of its components. The classification oriented operators "drill-down" and "roll-up" result in a change of the hierarchy level part of the granularity. The equivalent of the operators for property attributes are "split" and "merge" which add, respectively remove, a property attribute. "Slice" and "unslice" operations change the scope of the MO. For example, the data cube from figure 1 can be obtained from the total sum of all sales by a "drill-down" to Product.Family and Shop.Country, a "split" on Brand and ShopType and a following "slice" to Shop.Country='Germany' and Product.Group='Video'.

Besides these navigational operators, operations like explicit aggregations (e.g. AVG(G')MO or binary operations ($MO_{Turnover} = MO_{Sales} * MO_{Price}$) are defined ([16]).

2.3. Query Language

Descriptive queries can be specified in the *Cube Query Language* (CQL, [3]). The query for figure 1 would be expressed in CQL as follows:

```
SELECT      SUM(Sales)
FROM        Product, Shop
WHERE       Product.Group = 'Video',
            Shop.Country = 'Germany'
UPTO        Product.Family, Shop.Region
SPLIT BY    Product.Brand, Shop.ShopType
```

Final edited form was published in "International Database Engineering and Applications Symposium (IDEAS'99)", Montreal, 1999. ISBN 0-7695-0265-2.
https://doi.org/ 10.1109/IDEAS.1999.787264

Despite the (intended) similarity, CQL shows some substantial differences to SQL. On the one hand the FROM-clause contains no relations, but a set of dimensions spanning the actual analysis context. Queries are internally represented by multidimensional objects. The translation of such a simple query into a MO is straightforward. Note that the distinction of classification and property attributes is represented on the syntactical level by the UPTO and the SPLIT-BY clause.

But CQL has much more expressive power than the previous example suggests. Nested statements and arbitrary restrictions on the dimensional structures can easily be formulated. For example, the computation of the percental sales of each article based on its product group (range-to-total) can be expressed in the following manner:

```
SELECT   Num / Denum * 100 AS PercentalSales
FROM     Products P, Shops S, Time T
WHERE    P.Group = ´Video´,
         S.Region = ´South´,
         T.Month = ´07/99´
WITH     (SELECT SUM(Sales) AS Num
          UPTO P.Article, S.Region, Time.Month),
         (SELECT SUM(Sales) AS Denum
          UPTO P.Family, S.Region, Time.Month)
```

Therefore, in general a query is represented by a tree. Its leaves represent multidimensional objects which can be computed by a simple aggregation, the inner nodes represent binary operators, like the division and the multiplication in the example above, or complex restrictions predicates.

3. Processing Multidimensional Aggregates

In the application domain of OLAP/data warehousing an adequate and very common optimization method is to pre-aggregate some results and store them in summary tables. One approach is to precalculate summaries at all possible aggregation levels, so that queries may easily be answered during run-time through a simple lookup operation. Some commercial products use this strategy ([6], [13]). However, since the number of possible aggregates grows exponentially with the number of grouping attributes, this method is not feasible in application scenarios where there are 20 or more property attributes describing a product or a customer.

The only alternative is to select some of the most beneficial aggregates for materialization. Some queries may then be derived from the aggregates where others need to access the raw data. The problem is to select those aggregates which give the highest overall speed-up by allocating the least amount of disk space. Based on the relational data model [10] and [23] proposed algorithms to select views in a data warehouse. Of fundamental importance in the multi-dimensional context was the work [12] where a greedy algorithm based on an aggregation lattice was presented. The work was later extended by [9] and [4].

One fundamental lack of this technique is the missing support of analysis hot spots. The algorithm only investigates aggregates at different granularities but does not include the scope. For example it is not possible to pre-aggregate only the sales for the actual month; each aggregate contains data for all months.

Another central criticism of this method is that it does not adapt to the dynamics of the interactive query behavior ([22]). The selected set of aggregates is statically materialized after each data warehouse refresh and does not change with the actual query set during the analysis phase. In order to overcome this problem, [20] presented a rather restrictive data warehouse cache based on the relational data model. Recently, [8] implemented a more flexible caching method for partitions of multidimensional aggregates. They address the same problem as the CUBESTAR approach. In contrast to CUBESTAR, their solution is based on aggregate partitions of a fixed size, like buffer pages.

The strategy of CUBESTAR is to store precalculated queries for reuse in an aggregate cache. The typical draw back of query caching is that a new query must be contained in the old one in order to utilize it. In CUBESTAR a patch-working algorithm is used to combine cached multidimensional objects. The result of this algorithm in combination with a sophisticated cache strategy are substantial performance gains even for very small cache sizes (section 6).

3.1. The Patch-Working Principle

If a multidimensional query is to be answered by cached aggregates which only overlap but do not contain the query, a composition method (patch-working) is necessary. Figure 3 illustrates the fundamental idea. Several MOs or fragments of MOs are selected (dark gray shaded areas) in order to answer the query. The goal of the patch-working algorithm is to compose those fragments of multidimensional objects in the cache which allow the computation of the query at the least cost.

The restriction of convex scopes for multidimensional objects allows a very simple algorithm to determine the set of candidates to answer the query. One condition a candidate must fulfill is that its granularity must be smaller or equal than the granularity of the query. The other condition is that the query and the candidate are overlapping. Since the scope of a MO is defined by a tuple of classification nodes, this problem can be solved by simply checking intersections of nodes in the respective hierarchy trees. A detailed description of the algorithm can be found in [14].

Final edited form was published in "International Database Engineering and Applications Symposium (IDEAS'99)", Montreal, 1999. ISBN 0-7695-0265-2.
https://doi.org/ 10.1109/IDEAS.1999.787264

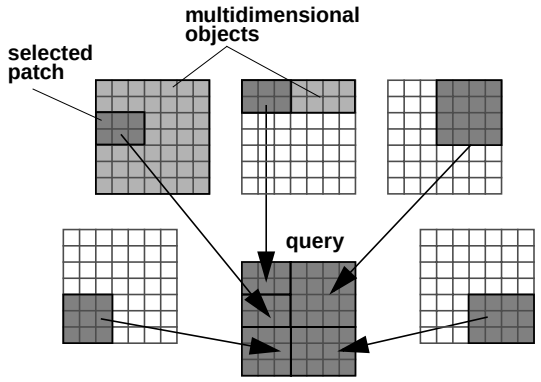


Fig. 3: Patch-Working

The problem of selecting the candidates with the least cost, however, is in general NP complete and can be reduced to the 0/1 knapsack problem. In CUBESTAR a greedy algorithm is used to find an approximate solution. The algorithm successively chooses for each part of the query which is not yet covered that multidimensional object with the least (estimated) access cost per resulting data cell. If the cost function to access a fragment of a MO is monotonous, the problem degenerates to the fractional knapsack and the greedy algorithm yields the optimal solution. Unfortunately, since monotonous means that the cost to access a fragment of an object de- or increases with the size of the fragment this is in general not the case in relational databases.

3.2. The Replacement Strategy

The fundamental part of a caching method is the replacement strategy which in this case is necessary to select multidimensional objects for removal from the cache if the cache is full and new aggregates are to be added. Due to non-uniform MO-sizes a set of several smaller MOs might be removed simultaneously in order to admit one large MO to the cache.

The knowledge about the data schema and specific properties of the data model in the OLAP application domain allows a much more sophisticated estimation of the benefit of a cached object as it is possible in traditional database systems. A simple LRU strategy can not make use of the available information about the dependencies and relationships between multidimensional objects. Therefore, the replacement strategy used for dynamic aggregate caching involves several factors some of which are strongly using the knowledge about the underlying data schema.

In the experiments (section 6) the influence of the following parameters for aggregate replacement is investigated:

- *Weighted Relative Reference Density* D_M
  The weighted relative reference density D_M is used to approximate the reference density of M in the multidimensional context. It is based on the relative reference counter R_{rel} of M, the time M was inserted into the cache T_{in}, and the time of the its last reference T_{last}[1]:

$$D_M = \frac{R_{Rel}}{T_{in} + 1} \cdot \frac{T_{in} - T_{last} + 1}{T_{last} + 1}$$

  If a cached object M is referenced by a query object M_Q, i.e. M is used to compute a part of M_Q, the reference counter is increased only by the fraction of M that was actually referenced:

$$R_{Rel} = R_{Rel} + \frac{|M \cap M_Q|}{|M|}$$

- *Degree of Relationship* $R_M(M_Q)$
  The degree of relationship $R_M(M_Q)$ of a multidimensional object M to the last query M_Q is defined as the number of OLAP navigation operations (‘drill-down’/ ‘roll-up’ etc.; section 2.2) which are necessary to transform M to M_Q. Since from the users point of view most OLAP queries are defined *relative* to one another, it is very likely that an object in the cache that can be reached within a few operations yields a higher benefit as a multidimensional object with a completely different content.
- *Reconstruction Cost* $C_M(M_Q, C)$
  In most cases there are interdependencies between the cached multidimensional objects in the sense that some objects may be computed from the set of multidimensional objects being currently an element of the cache C. In this case it may prove beneficial if those multidimensional objects are replaced which can be reconstructed most easily. Thus, multidimensional objects with a high reconstruction cost become the most valuable. The patch-working algorithm is used for the computation of the reconstruction cost $C_{Mi}(M_Q, C)$ for a multidimensional object M_i in the cache with objects $C = \{M_1, ..., M_n\}$.
- *Absolute Benefit* A_M
  The absolute benefit of a multidimensional object M reflects its usefulness independent of cache content and query behavior. It is based on the size, the granularity, and the scope of the multidimensional object and measures the cost savings of the access if M was used instead of the raw data. Therefore, it generalizes the benefit definitions from [12] as well as [8].

1. This formula corresponds to the weighted reference density as used in the classical least reference density (LRD) replacement strategy.

Final edited form was published in "International Database Engineering and Applications Symposium (IDEAS'99)", Montreal, 1999. ISBN 0-7695-0265-2.
https://doi.org/ 10.1109/IDEAS.1999.787264

The overall benefit per unit space $B_M(M_Q, C)$ of a multidimensional object M for a cache configuration C after query M_Q is a linear combination of these factors[2] divided by the size of M:

$$B_M(M_Q, C) = \frac{\alpha \dot{D}_M + \beta \dot{R}_M + \gamma \dot{C}_M(M_Q, C) + \delta \dot{A}_M}{|M|}$$

The weights $\alpha, \beta, \gamma, \delta$ can be used to parametrize the factors. However, in the experiments (section 6.2) they were only used as switches to enable or disable factor completely in order to measure its influence on the buffer. Finding a good parametrizations is still an open issue.

4. Relational Mapping of Multidimensional Structures

Of fundamental importance for the structured and efficient relational processing of multidimensional operators is a suitable relational representation of multidimensional objects. Today, most relational data warehouses are based on the star schema (figure 4), where the data cube containing the measures is represented by a central fact table. Its primary key is a composition of foreign keys to the according dimension tables. The dimension tables contain - in a denormalized fashion - the classification structures as well as the property attributes for each dimension.

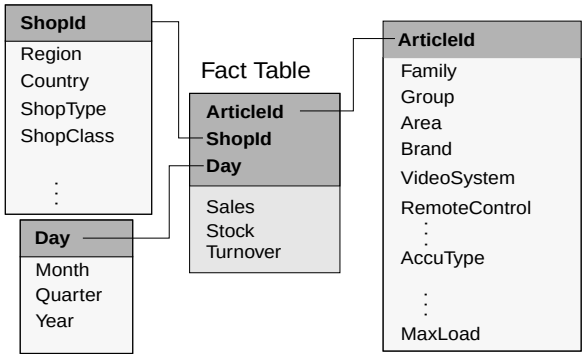


Fig. 4: Sample Star Schema

The use of a single dimension table for each dimension turns out to be problematic in the context of class-specific property attributes since different classes, like Camcorder and Washers, have also different schemata with regard to their properties (e.g. attributes VideoSystem and MaxLoad). In order to cope with these problems, the global dimension tables of the traditional star schema are replaced (basically partitioned) by class-specific dimension tables (scope tables), each of which having a possibly different schema.

2. The parameters are normalized by their averages (indicated by the dot).

The following example illustrates this method for the product dimension. Foundation for the hierarchy are the base classes, i.e. the classes of the lowest level in the hierarchy like Camcorder and HomeVCR, with their specific property schemata:

Products_Camcorder(ArticleId, Brand, VideoSystem, AccuType)

ArticleId	Brand	VideoSystem	AccuType
TR-75	Sony	HI8	Li
TS-78	Sony	HI8	NiCd
A200	JVC	N8	NiCd

Products_HomeVCR (ArticleId, Brand, VideoSystem, RemoteControl)

ArticleId	Brand	VideoSystem	RemoteControl
V-201	JVC	VHS	Yes
ClassicI	Grundig	VHS-C	No

All classes at higher levels are then recursively defined as views to the lower classes. The names of the child classes as well as all common attributes are propagated to the parent. For the product group Video the resulting scope table (view) is defined as follows:

```
create view
Products_Video (ArticleId, Brand, VideoSystem, Family) as
  select ArticleId, Brand, VideoSystem, 'Camcorder'
  from Products_Camcorder
union all
  select ArticleId, Brand, VideoSystem, 'HomeVCR'
  from Products_HomeVCR
```

Products_Video (ArticleId, Brand, VideoSystem, Family)

ArticleId	Brand	VideoSystem	Family
TR-75	Sony	HI8	Camcorder
TS-78	Sony	HI8	Camcorder
A200	JVC	N8	Camcorder
V-201	JVC	VHS	HomeVCR
ClassicI	Grundig	VHS-C	HomeVCR

Hence, the information for each class node of a dimension is stored in a class-specific relation or view, containing all dimensional elements below that class node with information about their parents in the classification hierarchy and the subset of property attributes visible at that node.

This construction has a positive side effect. Since the scope of a MO is defined by a classification node (section 2.1) which is represented by a specific scope table, the SQL queries for the computation of a multidimensional object do not need any restrictions beside the join conditions. The following example illustrates this approach. Suppose for the computation of the multidimensional object M from section 2, the patch worker picks up the following aggregates from the cache:

```
M1 = ([ Sales, FLOW, NONE ],
      ( Products.Group = 'Video',
        Shops.Country = 'Germany'),
      [ (Products.ArticleId, Shops.ShopId), {} ] ) // raw data
```

Final edited form was published in "International Database Engineering and Applications Symposium (IDEAS'99)", Montreal, 1999. ISBN 0-7695-0265-2.
https://doi.org/ 10.1109/IDEAS.1999.787264

M_2 = ([Sales, FLOW, SUM],
(Products.Group = 'Video', Shops.Region = 'South'),
[(Products.Family, Shops.State), {Brand, ShopType}])

M_3 = ([Sales, FLOW, SUM],
(Products.Family = 'Camcorder',
Shops.Country = 'Germany'),
[(Produkte.Family, Shops.Region),
{Brand, VideoSystem, ShopType}])

In order to compose M based on these MOs CUBESTAR generates the following SQL query:

```
CREATE TABLE M AS
  SELECT    Family, Region, Brand, ShopType,
            SUM(Sales) AS Sales
  FROM      M1, Products_HomeVCR P, Shops_North S
  WHERE     M1.ArticleId = P.ArticleId AND
            M1.ShopId = S.ShopId
  GROUP BY Family, Region, Brand, ShopType
UNION ALL
  SELECT    Family, Region, Brand, ShopType,
            SUM(Sales) AS Sales
  FROM      M2,
            (SELECT DISTINCT(State, Region)
             FROM Shops_Germany) S
  WHERE     M2.State = S.State
  GROUP BY Family, S.Region, Brand, ShopType
UNION ALL
  SELECT    Family, Region, Brand, ShopType, Sales
  FROM      M3
  WHERE     Family = 'HomeVCR'
```

The example clearly shows that in the context of multidimensional objects all restrictions are implicitly applied by joins with the according scope tables. The view mechanism selects with the scope tables Products_HomeVCR and Shops_North in the case of MO M_1 only those elements which are requested in the query avoiding to scan a possibly large dimension table (like customer). In the case of using the cached aggregate MO M_2, the summary data must be further aggregated from the state to the regional level. Therefore, the according scope table containing *only* the State-Region relationships must be dynamically generated by a SELECT DISTINCT statement applied to scope table Shops_Germany. No aggregation is necessary for the MO M_3, because its granularity is already at the requested level. Moreover no join with a scope table is needed, because the selection attribute corresponds to the group-by attribute.

5. The Architecture of CUBESTAR

Like typical ROLAP-systems, the CUBESTAR prototype is designed according to a three-tier-architecture. CUBESTAR consists of a client tier, the multidimensional CUBESTAR server and a relational backend (figure 5). The task of the CUBESTAR server is twofold: On the one hand, based on the patch-working algorithm a set of cached MOs to accelerate the execution of the query must be determined and reflected in an SQL statement. On the other hand, these MOs must be created and maintained by the server as well. Therefore, the CUBESTAR server itself has a three-layer architecture consisting of the query processor, the aggregate cache management and the partition directory service.

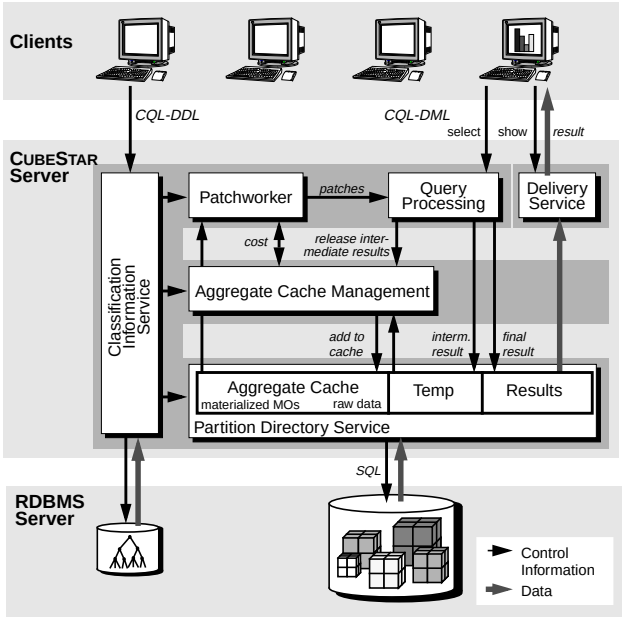


Fig. 5: Architecture of CUBESTAR

Another component which is used by all three layers is the classification information service. Basically, the classification information service is an API providing easy access to the dimensions and classification hierarchies. Calls to the classification directory request for example the children or properties of a certain class node or relationships between class nodes in the hierarchy.

The task of the multidimensional query optimizer is to find for each leaf of the tree of a set of physically existing MOs. All relevant information about materialized multidimensional objects are stored and managed by the partition directory. Based on the decision of the patch-worker, in the first phase, an SQL query is generated and executed for each leaf of the query tree. The intermediate result, a newly materialized MO, is added to a temporary storage area. In a second phase, an SQL statement is generated recursively for the whole query tree in a bottom-up manner. The intermediate results for inner nodes are not materialized, because the operators applied during the computation of a nested query may become very complex. Thus, the compatibility checks necessary to determine the reusability of such an aggregate are very complicated. The final result, however, is materialized and asynchronously delivered to the client.

Final edited form was published in "International Database Engineering and Applications Symposium (IDEAS'99)", Montreal, 1999. ISBN 0-7695-0265-2.
https://doi.org/ 10.1109/IDEAS.1999.787264

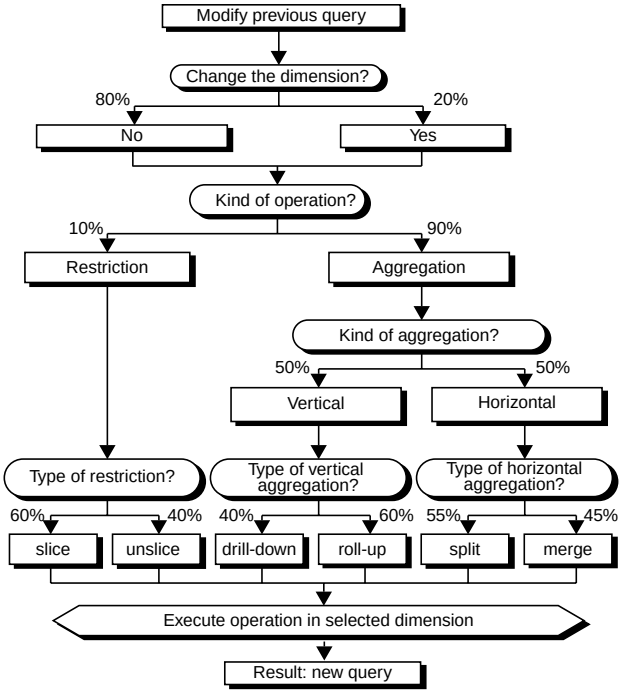


Fig. 6: Query generation

After the complete computation of the query, each of the newly materialized MOs for the leaves is handed over to the aggregate cache. Based on the replacement strategy (section 3.2) one or more cached MOs might be chosen for replacement. Depending on the outcome of the benefit computation, either the new multidimensional object or one of those in the cache might be deleted in order to satisfy the space constraint.

6. Experimental Results

In order to simulate a typical OLAP query behavior we implemented a query generator, which constructs a new query in modifying the previous one based on the decision path in figure 6. Each alternative can be parametrized with a certain probability. Since it is very hard to get actual query statistics from real world OLAP applications, we experimented with different parametrizations and got very good results for most of the query sequences. For simplicity, all simulations in this chapter are based on query sequences generated with the annotated probabilities.

The scenario consists of three dimensions with a four-, five- and six-level hierarchy, respectively. Each class at the base level of the hierarchies had between 10 and 15 property attributes. With a sparsity of 97% the data cube consisted of approximately 250.000 tuples.

The relational database management system used for the physical storage of the multidimensional objects was Oracle Enterprise Server 8.0.4. on a DEC Alpha 3000/800 server (1 CPU, 192 MByte RAM). The average query execution time for queries which could not utilize the cache was between 10 and 20 seconds. Therefore, the time for multidimensional processing can be neglected.

The following figures show the cost saving ratio (CSR; [14], [20], [8]), i.e. the relative savings resulting from accessing the cache instead of raw data, for a set of simulations. The values of the abscissa are determined by the step counter.

6.1. Static versus Dynamic Pre-aggregation

Figure 7 compares the dynamic aggregate cache as explained in section 3 to the static pre-aggregation strategy from [12]. The different graphs in each diagram show the cost saving ratio for a number of different cache sizes.

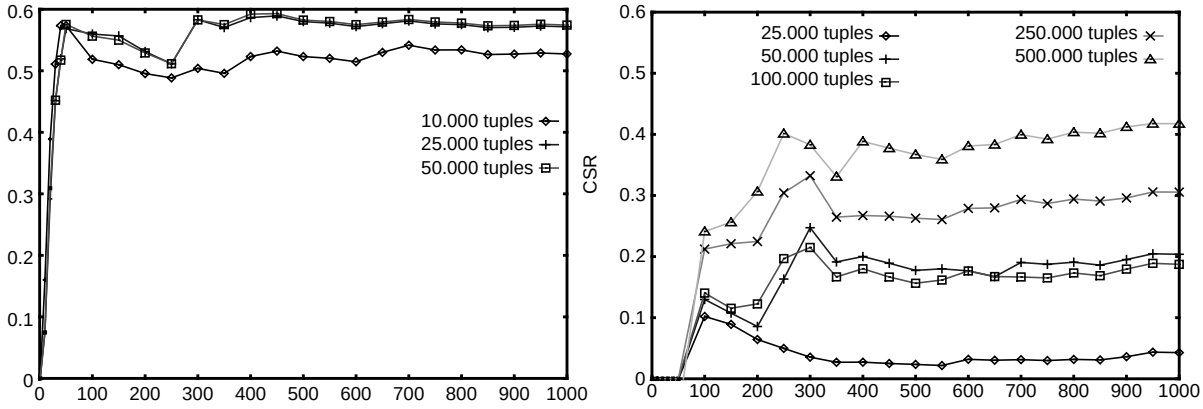
As illustrated in figure 7a, with the dynamic caching method cost reductions of more than 50% are possible with a cache size of only 10.000 tuples (all cache parameters were enabled; compare section 6.2). Note, that this is only about 4% of the fact table size, i.e. much less than a simple B-Tree index on the fact table would require. With a cache size of 25.000 tuples the speed up can be increased to about 60%. Adding more space does not yield any benefit, since there are always queries which can not be answered using the cache.

Figure 7b shows the cost savings if a fix set of aggregates was computed a priori using the algorithm proposed in [12]. Although the time to compute the aggregates was not even taken into account, the cost saving ratio is much lower. An additional space of already 25.000 tuples yields only cost savings of about 5%. This is because each aggregate contains the full scope of the data cube. Therefore, only aggregates which are summarized to very high aggregation levels are smaller than 10.000 tuples. For cost savings comparable to the dynamic method an additional space for summary tables of more than 200% of the size of the fact table would be necessary. Note that especially if there are many independent grouping attributes, like the 10 or 15 property attributes for each dimension, static algorithms are not able to select a proper set of summary tables because of the exponentially growing number of possibilities.

6.2. Influence of the Cache Parameters

In figure 8a the influence of each of the cache parameters on the cost saving ratio was investigated separately. The cache size was 50.000 tuples. In the experiments, the weights α, β, γ, δ (section 3.2) were only used to enable or disable the parameters, i.e. they were either set to 0 or 1. Experiments for fine tuning of the parameters have yet to

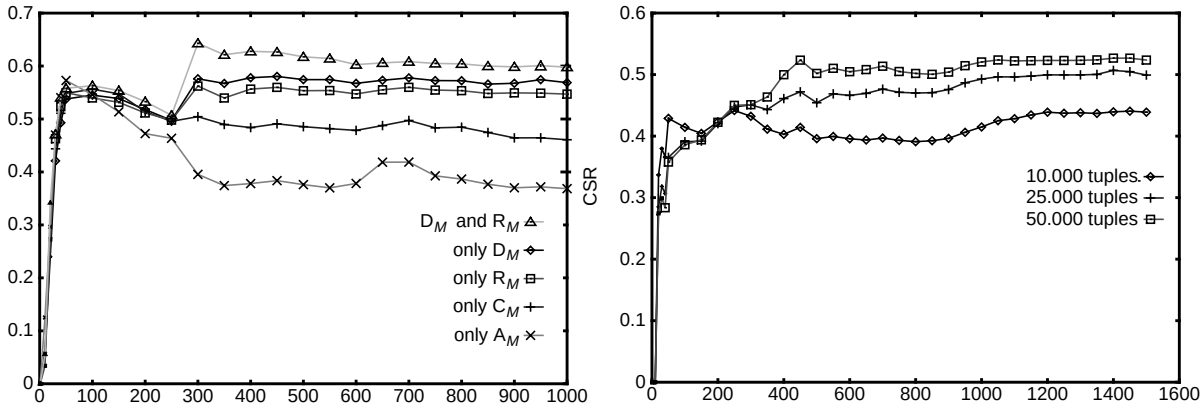
Final edited form was published in "International Database Engineering and Applications Symposium (IDEAS'99)", Montreal, 1999. ISBN 0-7695-0265-2.
https://doi.org/ 10.1109/IDEAS.1999.787264



a) CSR for dynamic aggregate cache; one user, all cache parameters enabled

b) CSR for static pre-aggregation; one user

Fig. 7: Development of the cost saving ratio (CSR) for 1.000 simulated single user queries; Comparison of the the dynamic aggregate and static pre-aggregation



a) CSR for dynamic aggregate cache; separate examination of cache parameters

b) CSR for dynamic aggregate cache; five users with approx. 300 queries each

Fig. 8: Development of the cost saving ratio (CSR); Influence of the cache parameters and multiuser simulation

be done. It turns out that the reference density D_M and the degree of relationship R_M had a more positive effect than the reconstruction cost C_M and the particularly bad absolute benefit A_M, which is the only parameter being absolutely independent of the query behavior and current cache content. The best result, cost savings of over 60%, were obtained using the combination of R_M and D_M. Simulations with other query sets confirmed that only these two parameters yields the highest cost savings.

6.3. Influence of Multiple Users

Figure 8b shows the cost saving ratio in a multi-user simulation with five independent users working with the same cache. Each user issued about 300 queries resulting in a set of altogether 1500 queries. The parameter configuration was the same as in figure 7a. The cost saving ratio is only about 10% less than in the single user simulation. The reason for the high cost savings of the small cache even for multiple users is twofold. First, each user needs only a very small cache. Second, some queries may access aggregates in the cache which were requested by another user.

7. Summary and Conclusion

This article introduces the general concepts of the multidimensional OLAP server CUBESTAR. Both, the multidimensional data model and the dynamic management and composition of aggregates based on the patch-working algorithm are an extension of previously published work. It was demonstrated that an adaptive strategy for pre-aggregation in general yields much better results than static algorithms. It was shown that performance increases of up to 60% are possible with cache sizes of only 5-10% of the size of the fact table. Except for the patch-working, no addi-

Final edited form was published in "International Database Engineering and Applications Symposium (IDEAS'99)", Montreal, 1999. ISBN 0-7695-0265-2.
https://doi.org/ 10.1109/IDEAS.1999.787264

tional pre- or post-processing is necessary. The cache can be used in conjunction with other optimizations like indexes or also static aggregates. In order to make the results comparable to other systems we will adapt the data of the APB-1 Benchmark of the OLAP Council ([19]).

All components of the CUBESTAR server are completely implemented in Java. Current work is focussed on good parametrizations of the replacement strategy. Other research activities involves the exploitation of parallelization and distribution ([2]). On the data modeling side we are currently exploring object-relational techniques in order to reflect the classification structures, or classes, in a more natural manner.

References

[1] Agrawal, R.; Gupta, A.; Sarawagi, S.: Modeling Multidimensional Databases, in: *13th International Conference on Data Engineering* (ICDE'97, Birmingham, U.K., April 7-11), 1997

[2] Albrecht, J., Lehner, W.: On-line Analytical Processing in Distributed Data Warehouses, in: *International Database Engineering and Applications Symposium* (IDEAS'98, Cardiff, Wales, U.K., July 8-10), 1998

[3] Bauer, A.; Lehner, W.: The Cube-Query-Language for Multidimensional Statistical and Scientific Database Systems, in: *5th International Conference on Database Systems For Advanced Applications* (DASFAA'97, Melbourne, Australia, April 1-4), 1997

[4] 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, August 25-29), 1997

[5] Cabibbo, L.; Torlone, R.: Querying Multidimensional Databases, in: *6th International Workshop on Database Programming Languages* (DBPL'97, Estes Park (CO), USA, August 18-20), 1997

[6] N.N.: Powerplay, Product Information, Cognos Incorporated, http://www.cognos.com/busintell/products/powerplay_overview.html, 1998

[7] Codd, E.; Codd, S.; Salley, C.: *Providing OLAP (On-line Analytical Processing) to User Analysts: An IT Mandate*, White Paper, Arbor Software Corporation, 1993

[8] Deshpande, P.; Ramasamy, K.; Shukla, A.; Naughton, J.: Caching Multidimensional Queries Using Chunks, in: *27th International Conference on the Management of Data* (SIGMOD '98, Seattle, USA, June 2-4), 1998

[9] Gupta, H.; Harinarayan, V.; Rajaraman, A.; Ullman, J.D.: Index Selection for OLAP, in: *13th International Conference on Data Engineering* (ICDE'97, Birmingham, U.K., April 7-11), 1997

[10] Gupta, H.: Selection of Views to Materialize in a Data Warehouse, in: *6th International Conference on Database Theory* (ICDT'97, Delphi, Greece, January 8-10), 1997

[11] Gyssens, M.; Lakshmanan, L.V.S.: A Foundation for Multi-Dimensional Databases, in: *23th International Conference on Very Large Data Bases* (VLDB'97, Athens, Greece, August 25-29), 1997

[12] Harinarayan, V.; Rajaraman, A.; Ullman, J.D.: Implementing Data Cubes Efficiently, in: *25th International Conference on Management of Data*, (SIGMOD'96, Montreal, Canada, June 4-6), 1996

[13] N.N.: Hyperion Essbase OLAP Server, Product Information, Hyperion Solutions Corporation, http://www.hyperion.com/essbaseolapprodInfo.cfm, 1998

[14] Lehner, W.; Albrecht, J.: Divide and Aggregate: Caching Multidimensional Objects, Technical Report, University of Erlangen, 1999

[15] Lehner, W.; Albrecht, J.; Wedekind, H.: Multidimensional Normal Forms, in: *10th International Conference on Scientific and Statistical Data Management* (SSDBM'98, Capri, Italy, July 1-3), 1998

[16] Lehner, W.: Modeling Large Scale OLAP Scenarios, in: *6th International Conference on Extending Database Technology* (EDBT'98, Valencia, Spain, March 23-27), 1998

[17] Lenz, H.-J.; Shoshani, A.: Summarizability in OLAP and Statistical Data Bases, in: *9th International Conference on Statistical and Scientific Database Management* (SSDBM'97, Olympia (WA), August 11-13), 1997

[18] Li, C; Wang, X.S.: A Data Model for Supporting On-Line Analytical Processing, in: *5th International Conference on Information and Knowledge Management,* (CIKM'96, Rockville (MD), November 12-16), 1996

[19] N.N.: APB-1 OLAP Benchmark, Release II, OLAP Council, http://www.olapcouncil.org/research/bmarkly.htm, 1998

[20] Scheuermann, P.; Shim, J.; Vingralek, R.: WATCHMAN: A Data Warehouse Intelligent Cache Manager, in: *22nd International Conference on Very Large Data Bases* (VLDB ´96, Bombay, India, September 3-6), 1996

[21] Smith, J.M.; Smith, D.C.P.: Database Abstractions: Aggregation and Generalization, *ACM Transactions on Database Systems*, Vol. *2*, No. 2, *1977*

[22] Theodoratos, D.; Sellis, T.: Data Warehouse Configuration, in: *Proceedings of the 23rd International Conference on Very Large Data Bases* (VLDB´97, Athens, Greece, August 25-29), 1997

[23] Yang, J.; Karlapalem, K.; Li, Q.: Algorithms for Materialized View Design in Data Warehousing Environment, in: *23rd International Conference on Very Large Data Bases* (VLDB'97, Athens, Greece, August 25-29), 1997