Pre-Aggregation for Irregular OLAP Hierarchies with the TreeScape System

Torben Bach Pedersen Christian S. Jensen
Department of Computer Science
Aalborg University
{tbp,csj}@cs.auc.dk

Curtis E. Dyreson
Department of Computer Science
Washington State University
dyreson@eecs.wsu.edu

Abstract

We present the TreeScape system which, unlike any other system known to the authors, enables the reuse of pre-computed aggregate query results involving the kinds of irregular dimension hierarchies that occur frequently in practice. The system establishes a foundation for obtaining high-performance query processing while pre-computing only few aggregates. It is demonstrated how this reuse of aggregates is enabled through dimension transformations that occur transparently to the user.

1 Introduction

In order to improve query performance, modern On-Line Analytical Processing (OLAP) systems use a technique known as *practical pre-aggregation*, where combinations of aggregate queries are materialized *selectively* and re-used when computing other aggregates; full pre-aggregation, where all combinations of aggregates are materialized, is infeasible, as it typically causes a blowup in storage requirements of 200–500 times the size of the raw data [3, 6]. Normally, practical pre-aggregation requires the dimension hierarchies to be regular, i.e., to be balanced trees, but this is often not the case in real-world systems.

The TreeScape system presented here enables practical pre-aggregation even for irregular hierarchies, based on techniques described previously by the authors [4]. We show how to achieve practical pre-aggregation through transformations of the dimensions and how the transformations can be accomplished transparently to the user. The system enables the achievement of fast query response time while saving huge amounts of storage compared to current OLAP systems and techniques.

The prototype implementation of TreeScape demonstrates that these benefits may be achieved with standard technology. While this demonstration uses a particular RDBMS, it's ODBC driver, and particular relational OLAP tool, TreeScape is not dependent on any specific products[1], making the solution flexible and generally applicable.

2 Normalizing Hierarchies

We use a small case study concerning patients and their diagnoses for illustrating the workings of the system. Diagnoses have three different levels of precision, depending on how accurate a patient's condition can be described. The most precise diagnoses are *low-level diagnoses*, which are grouped into *diagnosis families*, which, in turn, are grouped into *diagnosis groups*. The example data consists of 9 diagnoses and their hierarchical relationships, along with patient counts. The data can be seen in Table 1 and to the left in Figure 1.

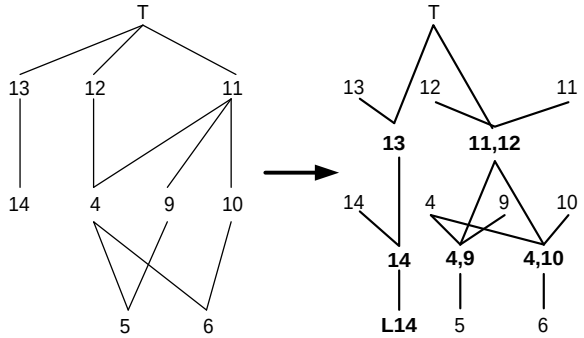

Figure 1: Dimension Transformations

The hierarchy is irregular. It is unbalanced because the diagnosis "Lung cancer" (14) has no low-level diagnoses associated with it and non-strict because, e.g., diagnosis 4 ("Diabetes during pregnancy") has several parents.

With this data, problems occur when pre-aggregated data at lower levels is used to compute new values at higher levels. For example, if we pre-aggregate the counts of patients at the low-level diagnosis level and want to aggregate to the diagnosis family level, we cannot deduce what the value should be for "Lung Cancer" (14). If we pre-aggregate at the diagnosis family level, patients with diag-

[1] The solution assumes that an ODBC interface is available for the RDBMS, a requirement met by all major commercial RDBMSs.

ID	Text	Type
4	Diabetes during pregnancy	Family
5	Insulin dependent diabetes during pregnancy	Low-Level
6	Non insulin dependent diabetes during pregnancy	Low-Level
9	Insulin dependent diabetes	Family
10	Non insulin dependent diabetes	Family
11	Diabetes	Group
12	Pregnancy related	Group
13	Cancer	Group
14	Lung cancer	Family

Diagnosis

ParentID	ChildID
4	5
4	6
9	5
10	6
11	9
11	10
12	4
13	14

Grouping

DiagID	Count
5	1

Patient

Table 1: Case Study Tables

noses 5 or 6 will be counted for both of the diagnoses 4 and 9, and 4 and 10, respectively, leading to wrong results when we aggregate to the diagnosis group level.

Our solution to the problems with reusing aggregates is to render the hierarchies well-behaved by *normalizing* them. Informally, the normalization process introduces new *placeholder* values where the hierarchy is unbalanced, and introduces *fused* values that represent *sets of* parent values when child values have multiple parents. The result of normalizing the hierarchy in our example is given to the right in Figure 1. For example, value "**L14**" representing "Lung Cancer" at the low-level diagnosis level, and value "**4,9**" representing the set of diagnoses $\{4, 9\}$ are introduced by the normalization. In the figure, all values and links in boldface have been added by the normalization process, which is described in detail elsewhere [4].

The normalized hierarchy supports practical pre-aggregation. For example, it is possible to store counts of patients at the low-level diagnosis level, and then re-use these to compute the counts for diagnosis families and diagnosis groups. With the example data (one patient with diagnosis 5), this will only require the storage of the one value versus six values being required for *full* pre-aggregation (one value for low-level diagnosis 5, two values for diagnosis families 4 and 9, two values for diagnosis groups 11 and 12, and one value for $\top$, which represents the total for all diagnoses).

The example is somewhat indicative of the storage savings achieved within a single dimension. When several dimensions are combined, the total space saved (with respect to full pre-aggregation) grows quickly with the number of dimensions. The savings occur because of *multidimensional sparseness* [3, 6], the phenomenon of the multidimensional space being very sparse for the lower levels in the dimensions, while quickly becoming more dense at higher levels. A study of the benefits of normalization are presented in Section 4.

3 System Architecture

While the hierarchy transformations enable practical pre-aggregation, they also have the undesired side-effect of introducing new values into the hierarchies that are of little meaning to the users. Thus, the transformations should remain invisible to the users. This is achieved by working with two versions of each user-specified hierarchy and by using a query rewrite mechanism, as described in detail elsewhere [4, 5]. The overall system architecture is seen in Figure 2.

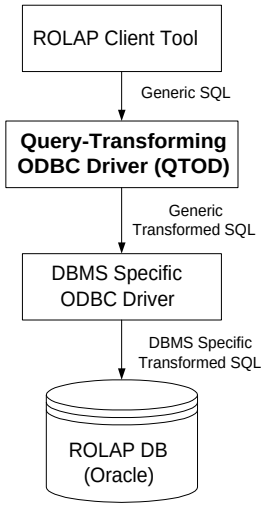

Figure 2: System Architecture

The ROLAP client tool, in this case the ROLAP tool Synchrony, which originates from Kimball's Startracker tool [1], makes SQL requests to the ROLAP DBMS, in this case the Oracle8 RDBMS, using the ODBC standard. We have implemented a special, query-transforming ODBC driver (QTOD) that, based on case-specific metadata, transforms the SQL requests into requests that hide the transformations from the users, returning the query results that users would expect based on the original hierarchies. A transformed request is submitted to the OLAP DB using a DBMS-specific ODBC driver. The QTOD component is common to all DBMSs, so Oracle8 may be replaced by another DBMS, such as IBM DB2, Informix, or MS SQL Server. Another ROLAP tool may also be used, making the solution quite general and flexible.

The prototype is based on an RDBMS (Oracle8) since RDBMSs are the most commonly used platform for Data Warehouse and OLAP applications. Additionally, the major RDBMSs now, like dedicated multidimensional DBM-

Dim.	1	2	3	4
No P-A	1.0	1.0	1.0	1.0
TP P-A	1.0	1.0	1.1	1.8
Full P-A	3.5	7.3	17.0	41.2

Total Data Size in Multiples of the Base Data Size

Dim.	1	2	3	4
No P-A	6.5 KB	10 MB	17 GB	25 TB
TP P-A	6.5 KB	10 MB	19 GB	48 TB
Full P-A	22 KB	75 MB	279 GB	1.1 PB

Absolute Data Size

	1	2	3	4
No P-A	207	4.3×10^4	8.9×10^6	1.8×10^9
TP P-A	2.6	5.3	8.4	9.8
Full P-A	1	1	1	1

Average-Case Performance in Multiples of Optimal

	1	2	3	4
No P-A	2.1 secs	11 hours	103 days	57 years
TP P-A	26 ms	53 ms	84 ms	98 ms
Full P-A	10 ms	10 ms	10 ms	10 ms

Absolute Average-Case Performance in Time Units

Table 2: Comparison of The Three Alternatives

Ses (MDDBs), use pre-aggregated data for faster query responses [7]. However, the approach could also be implemented using multidimensional technology, e.g., the Microsoft OLE DB for OLAP technology [2].

The transformation algorithms are implemented in Oracle's PL/SQL programming language. The transformations are relatively fast, taking at most a few minutes, even for large dimensions. Once the dimension hierarchies have been transformed, the QTOD transforms queries and results between the original and transformed hierarchies. The QTOD is a thin layer and adds very little overhead to queries. It is implemented using the GNU Flex++/Bison++ scanner/parser generators and the MS Visual C++ compiler.

4 Experimental Results

Next, we compare our technique, Transformed Practical Pre-Aggregation (TP P-A), to the two alternatives, namely *no* pre-aggregation (No P-A), which gives very long query response times, and *full* pre-aggregation (Full P-A), which requires unrealistically large amounts of storage for pre-aggregated data. We assume that an answer can be fetched using 1 I/O in the optimal case, and that 1 I/O takes 10 ms to perform.

The comparison has been done analytically using a combination of real and synthetic data. The dimension data is based on the British "Read Codes" diagnosis classification. The initial dimension hierarchy has 22570 values (diagnoses), while the transformed hierarchy has 51695 nodes, both have eight levels. The calculation of storage use is based on the assumption that the fact data has 10% density per dimension, i.e., .01 density for two dimensions, .001 density for three dimensions, etc., which corresponds to real-world cases where the multidimensional space gets very sparse when the number of dimensions increases. For the TP P-A technique, materialized aggregates where chosen so that the average-case performance was at most ten times worse than the optimal performance obtained with full pre-aggregation. The results of comparing the three alternatives for 1–4 dimensions are seen in Table 2.

We see that the benefit of our technique increases dramatically with the number of dimensions. For four dimensions, the average response time with no pre-aggregation is *57 years*, which clearly makes this alternative unusable. Even with a speed increase of a factor of 1000 due to the use of parallel and sequential I/O, the average response time is still *20 days*. Full pre-aggregation, on the other hand, requires storage that is *41 times* the size of the base data. This is equal to 1.1 *petabytes* for four dimensions, which is far beyond the capacity of current disk systems. We note that the problems with these techniques will only get worse for more dimensions. In comparison, our technique achieves an average response time of *98 ms* using only 80% more storage than the base data, making it very attractive.

5 Demonstration

Based on the case study described above, the demonstration initially shows snapshots that illustrate the hierarchy normalization process. Next, query processing is demonstrated, including a description of how the queries are transformed to hide the hierarchy transformations from the user, as well as the evaluation of the queries on concrete data. Finally, the demonstration compares the query execution times for the queries and the amount of storage required for pre-aggregated data with the two alternatives to our approach. Supporting material in the form of slides and posters are used in the demonstration.

References

[1] R. Kimball. *The Data Warehouse Toolkit*. Wiley Computer Publishing, 1996.

[2] Microsoft Corporation. OLE DB for OLAP Version 1.0 Specification. Microsoft Technical Document, 1998.

[3] The OLAP Report. *Database Explosion*. <www.olapreport.com/DatabaseExplosion.htm>. Current as of August 29, 2000.

[4] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. Extending Practical Pre-Aggregation in On-Line Analytical Processing. In *Proc. of VLDB*, pp. 663–674, 1999.

[5] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. The TreeScape System: Re-Use of Pre-Computed Aggregates over Irregular OLAP Hierarchies. In *Proc. of VLDB*, 2000 (demo track).

[6] A. Shukla et al. Storage Estimation for Multidimensional Aggregates in the Presence of Hierarchies. In *Proc. VLDB*, pp. 522–531, 1996.

[7] R. Winter. Databases: Back in the OLAP game. *Intelligent Enterprise Magazine*, 1(4):60–64, 1998.