

CURE for Cubes: Cubing Using a ROLAP Engine^{*}

Konstantinos Morfonios
Dept. of Informatics and Telecom. Univ. of Athens
kmorfo@di.uoa.gr

Yannis Ioannidis
Dept. of Informatics and Telecom. Univ. of Athens
yannis@di.uoa.gr

ABSTRACT

Data cube construction has been the focus of much research due to its importance in improving efficiency of OLAP. A significant fraction of this work has been on ROLAP techniques, which are based on relational technology. Existing ROLAP cubing solutions mainly focus on “flat” datasets, which do not include hierarchies in their dimensions. Nevertheless, the nature of hierarchies introduces several complications into cube construction, making existing techniques essentially inapplicable in a significant number of real-world applications. In particular, hierarchies raise three main challenges: (a) The number of nodes in a cube lattice increases dramatically and its shape is more involved. These require new forms of lattice traversal for efficient execution. (b) The number of unique values in the higher levels of a dimension hierarchy may be very small; hence, partitioning data into fragments that fit in memory and include all entries of a particular value may often be impossible. This requires new partitioning schemes. (c) The number of tuples that need to be materialized in the final cube increases dramatically. This requires new storage schemes that remove all forms of redundancy for efficient space utilization. In this paper, we propose CURE, a novel ROLAP cubing method that addresses these issues and constructs complete data cubes over very large datasets with arbitrary hierarchies. CURE contributes a novel lattice traversal scheme, an optimized partitioning method, and a suite of relational storage schemes for all forms of redundancy. We demonstrate the effectiveness of CURE through experiments on both real-world and synthetic datasets. Among the experimental results, we distinguish those that have made CURE the first ROLAP technique to complete the construction of the cube of the highest-density dataset in the APB-1 benchmark (12 GB). CURE was in fact quite efficient on this, showing great promise with respect to the potential of the technique overall.

1. INTRODUCTION

Modern data analysis “mines” knowledge from data stored in database systems discovering trends useful for decision making. To achieve this, analysts pose complex queries that extensively use aggregation in order to group together “similarly behaving tuples”. The response time of such queries over extremely large fact

^{*} The project is co-financed within Op. Education by the ESF (European Social Fund) and National Resources.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.

VLDB '06, September 12–15, 2006, Seoul, Korea.

Copyright 2006 VLDB Endowment, ACM 1-59593-385-9/06/09

tables in modern data warehouses can be prohibitive. This inspired Gray et al. [6] to propose the pre-computation of the data cube, which is a data structure that consists of the results of group-by aggregate queries on all possible combinations of the dimension-attributes over a fact table in a data warehouse.

A common representation of the data cube that captures the computational dependencies among different group-by queries is the cube lattice [9]. Figure 1 illustrates the cube lattice of a fact table R with three dimensions (A, B, and C). Every node in the cube lattice represents a group-by query and is labeled with its grouping attributes, which consist of the subset of dimensions that participate in the group-by clause of the corresponding query. If we denote the number of dimensions of a fact table with D , then the number of all cube lattice nodes is 2^D . Hence, a naive implementation method that computes each node separately and stores the result has exponential time and space complexity.

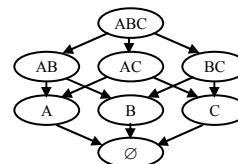


Figure 1. Example of a cube lattice

To overcome this problem, implementation of the complete data cube has been studied using various data structures to construct and store the cube. On one hand, ROLAP and MOLAP methods use materialized views and multidimensional arrays, respectively, focusing mainly on the efficient sharing of computational costs (like sorting or hashing) during cube construction. On the other hand, more recent approaches exploit specialized tree-like data structures in order to compute and store cubes more efficiently. In this paper, we focus on ROLAP methods and ignore the other categories for the following reasons: (a) MOLAP methods are poor performers when data is sparse, which is the case in most real-life applications. Although challenged by some, this has been observed by many researchers [2, 18]. (b) Complex tree-like data structures appear to have superior performance for cube construction and storage, but are currently not supported by any widely used product, hence, requiring nontrivial implementation effort.

ROLAP methods appear to strike the right balance on several fronts. They are based on materialized views and can be incorporated into any existing relational engine with minimal cost. Moreover, it has been shown that several ROLAP methods behave well in most kinds of datasets, including sparse ones, and some of them are capable of condensing the final cube by removing redundancy.

Unfortunately, current ROLAP cubing methods have focused mainly on supporting “flat” data, while many real-life applications deal with fact tables with each dimension consisting of several attributes organized hierarchically. For example, a dimension

“Region” may contain values at different levels of detail, forming the hierarchy “City” → “Country” → “Continent”. Hierarchies offer greater flexibility to analysts, since they describe the data at different granularities and form the basis for common operations, like roll-up and drill-down. On the other hand, hierarchies introduce several complications into cube construction that cannot be handled by straightforward extensions of existing techniques. (a) The number of nodes in a cube lattice increases dramatically and its shape is more involved. These require new forms of lattice traversal for efficient execution. (b) The number of unique values in the higher levels of a dimension hierarchy may be very small; hence, partitioning data into fragments that fit in memory and include all entries of a particular value may often be impossible. This requires new partitioning schemes. (c) The number of tuples that need to be materialized in the final cube increases dramatically. This requires new storage schemes that remove all forms of redundancy for efficient space utilization.

In this paper, we propose CURE (Cubing Using a ROLAP Engine), a novel ROLAP cubing method that addresses the issues above and constructs complete data cubes over very large datasets with arbitrary hierarchies. We demonstrate the effectiveness of CURE through experiments on both real-world and synthetic datasets. Among the experimental results, we distinguish those that have made CURE the first ROLAP technique to complete the construction of the cube of the highest-density dataset in the APB-1 benchmark (12 GB) [17]. CURE was in fact quite efficient on this, showing great promise with respect to the potential of the technique itself and of ROLAP in general. CURE stretches ROLAP to its limits, for the first time in the face of hierarchies, indicating that it may not be inherently inferior.

CURE contributes a novel lattice traversal scheme, an optimized data partitioning method, and a suite of relational storage schemes for all forms of redundancy. The last two are useful to “flat” datasets as well, but they are mostly necessary in the presence of hierarchies. In more detail:

- **Lattice Traversal with Dimension Hierarchies:** To the best of our knowledge, CURE is essentially the first comprehensive ROLAP solution capable of constructing a complete cube not only at the leaf level of each dimension hierarchy, but also at all higher levels, pre-computing group-by queries at all granularities. To achieve this, CURE uses an efficient way of traversing an extended lattice that includes dimension hierarchy levels (first proposed elsewhere [9]), which enables great cost sharing of sorting operations through pipelining.
- **External Partitioning:** We propose an efficient algorithm for partitioning fact tables that store hierarchical data of any size into memory-fitting segments, while computing a very small subset of the cube using inexpensive additional resources. Exploiting this early-computed data, CURE accelerates the construction of the final cube significantly, making it feasible even when the original fact table is extremely large. Existing techniques, partition data according to values in a single dimension and require that segments of tuples with the same value in this dimension fit in memory. However, as shown in Section 4, this is not always possible in cases that include hierarchies, due to small domain sizes at coarse granularities.
- **Efficient Storage:** Unlike previous ROLAP methods that rely only on avoiding redundant-tuple storage for cube size reduc-

tion, we further study alternative schemes for storing non-redundant data efficiently as well. To the best of our knowledge, CURE is the only ROLAP method that condenses the cube both by rejecting all kinds of redundancy and by further exploiting appropriate data representations.

The rest of this paper is organized as follows: After summarizing related work in Section 2, in Sections 3, 4, and 5, we study the problem of handling hierarchies and revisit external partitioning and efficient storage, respectively, under the new perspective. In Section 6, we combine everything and present CURE in pseudo-code. In Section 7, we describe the results of our experimental evaluation and finally, we conclude in Section 8.

2. RELATED WORK

Data cube construction has been the focus of much research due to its importance in improving the performance of OLAP tools. After Gray et al. [6] proposed the data cube structure, a plethora of papers has been published in this area.

There are several ROLAP cubing methods proposed so far [1, 2, 6, 12, 13, 15, 18, 19, 24], which are well-documented in the existing literature and their detailed description exceeds our purpose. BUC [2] is the most influential method in the ROLAP context attributing its success to a very efficient execution plan that enables sharing sorting costs during construction of different nodes. Both BU-BST [24] and QC-Tables [13] are BUC-based, i.e., they use the same execution plan. However, they do not support hierarchies, they have not been tested over very large data sets, and they do not store cube tuples efficiently. CURE is BUC-based as well, while also dealing with all of these problems.

Among ROLAP cubing techniques, only PipeSort and PipeHash [1, 19] have (superficially) discussed supporting hierarchies. Both of them, however, represent rather straightforward and non-scalable solutions, they have already been outperformed by all subsequent ROLAP methods, and neither handles efficient storage. Hence, CURE appears to be the first ROLAP method that studies the problem comprehensively and proposes a practical solution.

Furthermore, to the best of our knowledge, all results published so far for ROLAP cubing assume that the original fact table fits in memory. Disk-based extensions have been discussed rarely [2, 18], but only for “flat” data and without any accompanying performance results. On the contrary, CURE’s partitioning is applicable over very large hierarchical data, which is also shown experimentally even in cases that data sizes far exceed memory resources.

With respect to cube size reduction in ROLAP, Key [12], BU-BST [24], and QC-Tables [13] study the effect of removing redundant tuples from the cube. They only focus on what to avoid storing but not on how to store the data finally materialized. Like existing methods, CURE removes all kinds of redundancy but also employs efficient storage schemes that further compress the final result. Orthogonal to the above is the ability of BUC [2] to construct iceberg cubes, i.e., cubes that do not store data produced by aggregation of a small number of tuples. Being BUC-based, CURE is able to construct iceberg cubes as well.

Regarding MOLAP methods, they use multidimensional arrays for cube construction and storage [20, 26] as an alternative to relational materialized views. None of them handles hierarchies or redundancy, however, hence they are considered impractical.

The inability of existing ROLAP and MOLAP methods to solve all aspects of the cubing problem efficiently has given rise to a third category of algorithms that use sophisticated, complex tree-like data structures for cube construction and storage [5, 8, 14, 22, 25]. Among them, Dwarf [22] is the most promising, being able to deal with hierarchies [21] while removing several kinds of redundancy from cube data, which gives it polynomial scaling [23]. QC-Trees [14] have similar redundancy-reduction capabilities as well. As we explained earlier, the methods of this category cannot be directly used currently, since to the best of our knowledge, the complex data structures they exploit are not supported by any widely-used product and their implementation is far from straightforward. Hence, CURE is the only solution that shares common properties with such sophisticated methods, including polynomial storage requirements, while still being ROLAP compatible and, hence, easily put into an existing server. A comparison among CURE, Dwarf and QC-Trees would be interesting, since it would reveal the foundational strengths and weaknesses of the two underlying philosophies, but it exceeds the purpose of this paper and is left for future work. The aim of this paper is to find the best, easy-to-implement (i.e., ROLAP), comprehensive cubing method that is a potentially viable competitor of the elite of the existing methods that exploit more sophisticated (and costly) data structures.

Finally, apart from the methods that construct complete cubes, there are methods that select subsets of nodes for partial construction (e.g., [9]) and others that compute a small number of predefined nodes [4, 16]. Clearly, the functionality of such methods is orthogonal to that of CURE, since selecting *what* to materialize is orthogonal to deciding *how* to materialize it efficiently. Hence, although they are interesting and their combination with CURE seems possible, their study exceeds the scope of this paper.

3. HIERARCHICAL EXECUTION PLANS

Consider a fact table with D dimensions. If we denote the number of levels of the i -th dimension with \mathcal{L}_i , the number of all cube nodes is given by the product $\prod_{i=1}^D (\mathcal{L}_i + 1)$, which is greater than or equal to 2^D (equality holds when all dimensions are flat, i.e. when $\mathcal{L}_i = 1, \forall i \in [1, D]$). Assume, for example, that the dimensions of the fact table R (whose lattice appears in Figure 1) are organized in hierarchies as follows: $A_0 \rightarrow A_1 \rightarrow A_2, B_0 \rightarrow B_1$, and C_0 . The number of nodes in this example is equal to $(3+1) \cdot (2+1) \cdot (1+1) = 24 > 2^3 = 8$. Clearly, finding an efficient execution plan becomes more complex when using hierarchies. Taking the importance of hierarchies into account [10, 11], we propose below an efficient in-memory method for the construction of a hierarchical cube (a cube whose dimensions form hierarchies).

3.1 Comparison of Alternative Plans

Cubing algorithms that compute the cube using only the most detailed level of every dimension are not viable in practice, since then, for common roll-up and drill-down operations, the underlying system must further aggregate materialized aggregates on the fly, which is computationally expensive. Moreover, executing the cubing operation several times, once for every possible combination of the hierarchy levels of the cube's dimensions, is not practical either. Efficiency of all cubing methods depends on their ability to share computational costs among as many cube nodes as possible. Hence, independent construction of different sub-cubes can never be optimal overall.

The arguments above make it clear that the ideal cubing method must construct all nodes of the hierarchical data cube in a pipelined fashion using as few data passes as possible. Hence, a cube lattice that includes hierarchies must be found, together with an efficient way to traverse it and prune it into a tree that shows the order of execution, i.e., creating the so-called execution plan of the corresponding cubing method.

With respect to pruning a lattice into a tree, BUC [2] has been found to be the winner among all ROLAP cubing algorithms that compute complete and flat cubes. Its efficiency is primarily due to the way it traverses the cube lattice, namely bottom-up and depth-first. Such a traversal is ideal for flat cubes and can easily be extended properly to efficiently handle hierarchies as well.

With respect to identifying an appropriate lattice that incorporates hierarchies, there are two main alternatives: A straightforward solution is to consider every level of each dimension as a separate dimension and construct the lattice as if it were flat. In this case, nodes including more than one levels of the same dimension (e.g., A_1A_2 or $A_0B_0B_1$) should be omitted, since they repeat trivial information. The second alternative is the lattice introduced for hierarchies in the context of view selection [9], which natively reflects relationships between different levels of the same dimension.

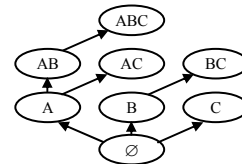


Figure 2. The “flat” execution plan of BUC (P_1)

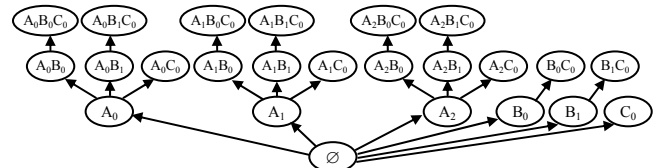


Figure 3. A straightforward hierarchical execution plan (P_2)

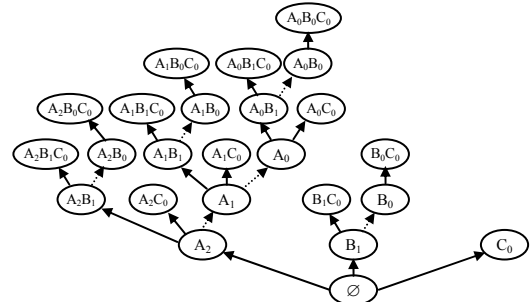


Figure 4. CURE's hierarchical execution plan (P_3)

To compare the two alternatives, consider the example of R , which is a flat fact table whose lattice appears in Figure 1 and one BUC-based execution plan (called P_1) appears in Figure 2. For R with hierarchies, a BUC-based traversal over the first and second types of lattices produces execution plans P_2 and P_3 , illustrated in Figure 3 and Figure 4, respectively. Note that P_2 is actually the shortest possible extension of P_1 (height remains equal to 3), while P_3 is the tallest possible (its height is equal to 6), since it pushes the computation of any node as high as possible.

To compare the quality of P_2 vs. P_3 let us first study P_1 , the execution plan of the standard BUC algorithm. According to it, BUC

