

Answering OLAP Queries via Subcubes

Huei-Huang Chen¹ Kuo-Wei Ho²

¹Department of Information Management, Tatung University
hhchen@ttu.edu.tw

²Department of Electronic Engineering, De-Lin Institute of Technology
kwho@dlit.edu.tw

Abstract

OLAP queries are complex and time-consuming and hence materializing data cube is a commonly used technique to reduce response time. To our knowledge, most previous OLAP cube implementation techniques apply a static view selection algorithm on the search lattice. These static methods first treat each node in the lattice as an undividable unit and then pick some of them for materialization. Pre-computing some nodes without being aware of which nodes are actually accessed at run time would seriously impact both response time and available space. We propose to further partition nodes in the lattice into *subcubes* into each of which multiple OLAP queries via a *dynamic* materialization algorithm can be mapped. When a query is posed, the entire subcube containing multiple results of related queries is materialized to take potential *locality* effects into consideration. Experiments show that the locality effects do exist in OLAP queries, and our dynamic method keeps a reasonable performance even though the available space is very limited and is practical for OLAP query processing.

Keywords: OLAP, data cube, subcube

1. Introduction

Data warehousing and On-line Analytical Processing (OLAP) technologies [4] have been one of the most important decision support systems in recent years. Inherently, OLAP queries are complex and time-consuming; hence materialization is a commonly used technique to reduce response time. To the best of our knowledge, applying view selection algorithms on the search lattice in advance is a common practice in most previous data cube implementation techniques. These static methods first treat each node in the lattice as an undividable unit and then pick some nodes for materialization. Furthermore, they depend heavily on some sampling techniques [11] to estimate the view size which is not practical in implementation.

Typically, generation of an OLAP cube can be accomplished by repeatedly computing group-bys based on the dimension levels, and the result forms a search lattice [12]. Note that the OLAP cube generated this way does not cover all aggregations it may have because a cube operation [8] can be further applied on the nodes in the lattice. Our experiments [6] on the APB-1 benchmark database show that the OLAP cube grew from 8 to 37 times as the size of the base fact table. In addition, the OLAP queries access only a small portion of the OLAP cube (especially when data cubes are quite sparse). Not only few nodes in the lattice are accessed but a small portion within each accessed node is used. Therefore, careless materialized view selection strategies (or algorithms) may result in exhausting available system space with useless aggregations. Our experiments [6] also exhibit that only 13% nodes in the lattice are accessed in APB-1 Benchmark queries, and the accessed regions are much smaller than their corresponding nodes.

With those issues mentioned above, we propose defining a *finer* but *not too fine* partition, *subcube*, for OLAP implementations. A *finer* partition contributes to more efficient space utilization than the whole node in the lattice. Experiments show that OLAP queries cluster only on some nodes in the lattice and hence it is critical that we select the right set to materialize. A partition that is *not too fine* allows potential *locality* effects, i.e. multiple OLAP queries can be mapped to the same subcube.

We also propose a dynamic view selection algorithm to materialize subcubes from existing ones instead of pre-computing some nodes for materialization (i.e. *static* methods) without being aware of which views are actually accessed at *run time*.

2. Related work

Previous studies related to OLAP implementations deal with two major problems at different levels. Research investigating the problem of how to *compute* an OLAP cube efficiently belongs to *memory* level, and has developed two major approaches including *top-down* and *bottom-up* [1, 15, 2, 3, 10], while those

research on the problem of how to *store* an OLAP cube efficiently belongs to *storage* level, and has designed many algorithms to select the right set of view to materialize [12, 2, 7, 9, 14]. The most representative one is the greedy algorithm introduced in [12] choosing a near-optimal subset of views, while their heavy dependence on some sampling techniques [11] to estimate the view size which we think is not practical in implementation. In this paper, we propose a better *partition* for materialization and a new technique for estimation of a view size. Through materializing subcubes, queries benefited from the results of previous are made possible.

3. The subcube framework

3.1. The query cell notation

Typically OLAP queries examine the aggregations (measures) in several different contexts (via slicer attributes) and from several different angles (by group-by attributes). We use the following example to demonstrate what information is specified in an OLAP query and how an answer to a query can be viewed as a result of drill-down operation to a cell.

Example 3.1 Consider the query description of *Channel Sales Analysis* (i.e. *Query 1*) defined in APB-1 benchmark queries [13]. For the sake of clarity, the query has been slightly modified to omit some details not directly related to our discussion. This query shows *units sold* and *dollar sales* for a given *channel* by product, customer and time dimensions. The functional query definition is listed below.

```
get UNITS SOLD, DOLLAR SALES
by PRODUCT = children_of(prod)
by CUSTOMER = children_of(cust)
by TIME = children_of(time)
where CHANNEL = chan
```

Note that the *member_name* in parentheses is a parameter denoting a data member regarding a certain dimension, and *children_of()* denotes its child data members. The above query is represented as a 4-tuple (chan, prod, cust, time) in our notation. The underlined elements are *group-by attributes* used to specify along which dimensions the measures are analyzed in the query. The element not underlined is *slicer attribute* used to restrict which data member of a certain dimension is extracted. We assume that all measures are analyzed in each query, so there is no need for us to specify measures in our query notation.

3.2. The subcube

Although the terms “data cube” and “OLAP cube” are

commonly used and even interchangeable in the literature investigating multidimensional databases or OLAP systems, we take a different view. Basically, we regard the data cube as the result of a cube operation [8], while the OLAP cube is the union of applying cube operation on each node in a cube lattice [12].

Definition 3.1 [The Data Cube] A *data cube* is the result set of the *cube operator*. For an *n*-dimensional data cube, the *cell* is an $(n+m)$ -tuple in the form of $(d_1, d_2, \dots, d_m, f_1(*), f_2(*), \dots, f_m(*))$, where

- Each d_i is either a value from the domain of dimension levels of the *i*-th **Dimension** or an **ALL** values;
- Each $f_i(x)$ is a value generated by the aggregate function defined on the data cube.

If the cardinality of the *n* attributes are C_1, C_2, \dots, C_m , then the number of cells of the *n*-dimensional data cube is at most $\prod_{i=1}^n (C_i + 1)$, and the number of super-

aggregates (i.e. those cells containing **ALLs**) is

$$\prod_{i=1}^n (C_i + 1) - \prod_{i=1}^n C_i$$

In real-life applications, *hierarchies* exist in dimensions and underlie two important querying operations: *drill-down* and *roll-up*. Based on the *dependency relation* \leq [12], each *GB* form a cube lattice.

Definition 3.2 [The Cube Lattice] Given *n* dimensions D_1, D_2, \dots, D_n and each with a set of dimension levels (*hierarchy*)

$$\begin{aligned} L_1 &= \{l_1^1, l_1^2, \dots, l_1^{h_1}\}, \\ L_2 &= \{l_2^1, l_2^2, \dots, l_2^{h_2}\}, \\ &\dots \\ L_n &= \{l_n^1, l_n^2, \dots, l_n^{h_n}\} \text{ respectively.} \end{aligned}$$

Then the set $L = \{L_1 \times L_2 \times \dots \times L_n\}$ and *dependence relation* \leq forms a **Cube Lattice** $\langle L, \leq \rangle$ with $|L_1| * |L_2| * \dots * |L_n|$ nodes.

Definition 3.3 [The Node Operations] Given an *n*-dimensional fact *f* and the corresponding cube lattice $\langle L, \leq \rangle$, for each element $(l_1, l_2, \dots, l_n) \in L$, the set of its constituent elements $\{l_1, l_2, \dots, l_n\}$ can be used to compute *group-by* and *cube* operations from *f* denoted by $GB(l_1, l_2, \dots, l_n)$ and $CB(l_1, l_2, \dots, l_n)$.

Each node of an *n*-dimensional cube lattice is an *n*-tuple, and its constituent elements can be used as group-by attributes to generate the node value. The computation is first joining the fact table with each dimension tables then grouping it by its constituent elements.

Definition 3.4 [The Node Value] Given an n -dimensional cube lattice $\langle L, \leq \rangle$, for each element $(l_1, l_2, \dots, l_n) \in L$, the corresponding node in the lattice is denoted by $Node_{(l_1, l_2, \dots, l_n)}$ or $scClass_{(l_1, l_2, \dots, l_n)}$, and its value is the result of cube operation applying on it.

Since each node of the cube lattice is further partitioned into subcubes, to better express the concepts regarding the subcube, a node in the lattice is usually referred to as a subcube class represented by $scClass$ in our discussion.

Definition 3.5 [The OLAP Cube] Given an n -dimensional cube lattice $\langle L, \leq \rangle$ with q nodes, the OLAP cube is the union of q data cubes.

$$CB_{OLAP}(L) = \{CB(s_1) \cup CB(s_2) \cup \dots \cup CB(s_q) \mid \forall s_i \in L\}$$

Generally, an OLAP query is a subset of a subcube. To be benefited from potential locality effects, not only the result of an OLAP query is materialized but the entire subcube. We demonstrate how a query cell is mapped into a subcube, and how a materialized subcube can be used to answer multiple queries in following example. Before we formally define the subcube, some useful functions are listed below.

- $level(x)$: returns the level name of x .
- $child_level(x)$: returns the child level name of x .
- $parent(x)$: returns the parent name of x .

Definition 3.6 [The Subcube] Given an $CB_{OLAP}(L)$ and a query cell (c_1, c_2, \dots, c_n) , the corresponding subcube is represented by $sc(s_1^{l_1}, s_2^{l_2}, \dots, s_n^{l_n})$, where

- $s_i = \begin{cases} c_i & \text{if } c_i \text{ is underlined} \\ parent(c_i) & \text{if } c_i \text{ not underlined} \end{cases}$
- $l_i = \begin{cases} child_level(c_i) & \text{if } c_i \text{ is underlined;} \\ level(c_i) & \text{if } c_i \text{ not underlined.} \end{cases}$

and its values is a subset of a data cube $CB(l_1, l_2, \dots, l_n)$, where each cell share the same parent-child relationship.

$$sc(s_1^{l_1}, s_2^{l_2}, \dots, s_n^{l_n}) = \{(c_1', c_2', \dots, c_n', a_1, a_2, \dots, a_m) \mid c_i' = parent(s_i), \forall i = 1, 2, \dots, n\}$$

3.3. The SUBCUBING algorithm

We develop a dynamic algorithm Subcubing based on our subcube framework. Generally, Algorithm Subcubing materializes each accessed subcube as long as space is available. If the remaining space is not sufficient, the algorithm repeatedly replaces the existing subcube that has the least reuse frequency

Algorithm SUBCUBING(S)

Given: S is a queue of query cells, r is the result set of the query and M is the set of materialized subcubes.

```

BEGIN
  M = ∅;
  r = ∅;
  while S ≠ ∅
    pick the first cell c in S;
    s = CELLMAPPING(c);
    if (s ∈ M) then /* Existing subcube */
      s.frq = s.frq + 1;
      r = slice(s);
      return r;
    else /* NEW subcube */
      while sp < size of(s)
        pick the subcube s' ∈ M that
          has the least frq and bnf;
        drop s';
        sp = sp + size of(s');
      end while
      compute s from M;
      M = M ∪ s;
      s.frq = 1;
      sp = sp - size of(s);
      r = slice(s);
      return r;
    end if
    S = S - c;
  end while
END

```

Procedure CELLMAPPING(c)

Given: $c = (c_1, c_2, \dots, c_n)$ is an n -dimensional query cell.

```

BEGIN
  for i = 1 to n
    if  $c_i$  is underlined then
      /* group-by attributes */
       $l_i = child\_level(c_i)$ ;
    else /* slicer attributes */
       $l_i = level(c_i)$ ;
       $c_i = parent(c_i)$ ;
    end if
  end for
  s = ( $c_1^{l_1}, c_2^{l_2}, \dots, c_n^{l_n}$ );
  return s;
END

```

until space become sufficient again to materialize a newly accessed subcube. There are some variables used in the algorithm to facilitate the materialization processing: The variable sp denotes the size of available space allocated and it shrinks with time. To determine the least recently used subcube, we use the variable frq to keep track of the reuse frequency for each materialized subcube. In addition, the variable bnf is used to denote the potential benefit it may has.

4. Experimental results

We have implemented the dynamic algorithm we developed and the greedy algorithm developed in [12] for comparison. The system used is a Pentium 4 2.8G Hz with 2GB DDR 400 SDRAM, running Microsoft Windows 2000 Server and SQL Server 2000. The algorithm was implemented using Microsoft Visual Basic 6.0 and ActiveX Data Model (ADO). We also implemented the greedy algorithm [12] for

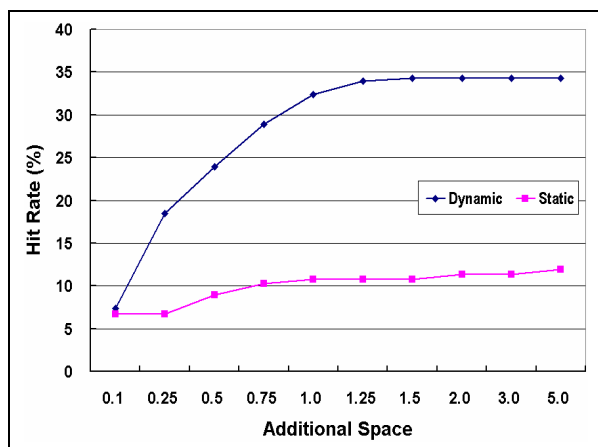


Fig. 1: The comparison between dynamic and static algorithms

comparison. The sample data used in our experiments is produced by APB-1 OLAP Benchmark Release II File Generator. The common parameters used in these experiments are: channel = 10, number of users = 100 (for enough query streams). Experimental results show that our method of partitioning subcube classes into subcubes is almost *immune* from density change.

We compare the hit rates with the static method, as shown in Fig.1. The results show that pre-computing some views without being aware of the actual usage they made at run time performs poorly. Note that once additional space exceeds 1.5 times the size of base fact table the hit rate no longer increases.

5. Conclusions and future works

We have investigated the problem of further partitioning the *subcube class* into *subcubes* in order to raise its reusability. We experimented on the APB-1 Benchmark database and emphasized the need to handle the serious situation that most OLAP queries focus only on certain *subcube classes* and sometimes even a small portion within a subcube class. We also developed one dynamic view selection algorithm to rapidly determine frequently used subcubes. Through moderately enlarge the size of aggregations queried to subcube(s) potential locality effects are taken into consideration, and at the same time reusability is made possible. We believe that the subcube framework will also apply to other OLAP data models. We are currently committed to developing subcube-based query processing, and part of the work appeared in [5].

Acknowledgement

This work has been partially sponsored by the National Science Council under grants NSC93-2213-E-036-007.

6. References

- [1] S. Agrawal, R. Agrawal, P.M. Deshpande, J.F. Naughton, R. Ramakrishnan, and S. Sarawagi. "On the Computation of Multidimensional Aggregates," In *VLDB*, pp. 506-521, 1996.
- [2] E. Baralis, S. Paraboschi, and E. Teniente, "Materialized View Selection in a Multidimensional Database," In *VLDB*, pp. 156-165, 1997.
- [3] K. Beyer and R. Ramakrishnan, "Bottom-up Computation of Sparse and Iceberg Cubes," In *SIGMOD*, pp. 359-370, 1999.
- [4] S. Chaudhuri and U. Dayal, "An Overview of Data Warehousing and OLAP Technology," In *SIGMOD Record*, Vol. 26, No. 1, pp. 65-74, 1997.
- [5] H.-H. Chen, K.-W. Ho, and C.-L. Shiou, "An Implementation for Subcube-based Query Processing," *Proceedings of International Computer Symposium*, 2004.
- [6] H.-H. Chen and K.-W. Ho, "Implementation Data Cubes via Subcubes," *Proceedings of the International Database Engineering and Applications Symposium*, 2004.
- [7] H. Gupta, "Selection of Views to Materialize in a Data Warehouse," In *ICDT*, pp. 98-112, 1997.
- [8] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh, "Data Cube: A Relational Aggregation Operator Generalizing Group By, Cross-Tab, and Sub-Totals," In *ICDE*, pp. 152-159, 1996.
- [9] H. Gupta, V. Harinarayan, A. Rajaraman, and J.D. Ullman, "Index Selection for OLAP," In *ICDE*, pp. 208-218, 1997.
- [10] J. Han, J. Pei, G. Dong, and K. Wang, "Efficient Computation of Iceberg Cubes with Complex Measures," In *SIGMOD*, pp. 1-12, 2001.
- [11] P.J. Haas, J.F. Naughton, S. Seshadri, L. Stokes, "Sampling-Based Estimation of the Number of Distinct Values of an Attribute," In *VLDB*, pp. 311-320, 1995.
- [12] V. Harinarayan, A. Rajaraman, and J.D. Ullman, "Implementing Data Cubes Efficiently," In *SIGMOD*, pp. 205-216, 1996.
- [13] OLAP Council, "OLAP Council APB-1 Benchmark Specification," White Paper, 1998.
- [14] A. Shukla, P. M. Deshpande, and J. F. Naughton, "Materialized View Selection for Multidimensional Datasets," In *VLDB*, pp. 488-499, 1998.
- [15] Y. Zhao, P. Deshpande, J. F. Naughton, "An Array-based Algorithm for Simultaneous Multidimensional Aggregates," In *SIGMOD*, pp. 159-170, 1997.